> Война — это путь обмана.
> Вершина искусства войны — побеждать без боя.
> Быть невидимым, быть неслышимым —
> Хитростью привести противника к поражению.
>
> – Сунь-цзы, «Искусство войны»

Монография знакомит читателя с современными угрозами операционных систем. В центре внимания — перспективные атаки на ядро: техники сокрытия, обход средств защиты и скрытые гипервизоры, использующие аппаратную виртуализацию.

Автор исследует актуальную проблему защиты операционных систем в условиях активного противодействия со стороны нарушителей и предлагает практические пути её решения.

В работе обобщены результаты многолетних исследований автора, прошедшие апробацию на международных конференциях и подтверждённые экспериментальными данными.

Для специалистов по защите информации, преподавателей, студентов и аспирантов, обучающихся по направлению «Информационная безопасность».

Ключевые темы:

- Анализ угроз безопасности ядра Windows: руткиты, вредоносные драйверы, атаки класса BYOVD и гипервизоры-руткиты.
- Практические примеры новых атак на ядро Windows: техники обхода Windows Defender и EDR/AV-решений.
- Описание авторских гипервизорных решений для мониторинга и защиты памяти ядра от атак со стороны драйверов.
- Детали среды для запуска драйверов в изолированных областях памяти (анклавах).
- Математические методы и алгоритмы обнаружения скрытых гипервизоров в условиях противодействия.

Информация об авторе

Игорь Юрьевич Коркин — кандидат технических наук (05.13.19), системный программист и технический эксперт в области информационной безопасности операционных систем и гипервизоров. Автор более 50 научных работ, включая статьи, патент и доклады на международных конференциях в США, Европе, Азии и России: BlackHat, LABScon, HITB, REcon, EKOPARTY, ROOTCON, Texas Cyber Summit, CDFSL, IEEE S&P и РусКрипто. Научный руководитель студентов МИФИ.

ЗАЩИТА ЯДРА ОПЕРАЦИОННЫХ СИСТЕМ В УСЛОВИЯХ ПРОТИВОДЕЙСТВИЯ

Игорь Коркин

Игорь Коркин

ЗАЩИТА ЯДРА ОПЕРАЦИОННЫХ СИСТЕМ В УСЛОВИЯХ ПРОТИВОДЕЙСТВИЯ

И.Ю. КОРКИН

ЗАЩИТА ЯДРА ОПЕРАЦИОННЫХ СИСТЕМ В УСЛОВИЯХ ПРОТИВОДЕЙСТВИЯ

МОНОГРАФИЯ

znanium
электронно-библиотечная система

Москва
ИНФРА-М
2026

УДК 004.45+004.49+004.056.5(075.4)
ББК 16.8:32.97
К66

Автор:

Коркин Игорь Юрьевич, кандидат технических наук (05.13.19), выпускник кафедры № 42 Национального исследовательского ядерного университета «МИФИ»; ведущий исследователь международных проектов по кибербезопасности в компаниях Huawei Technologies (Китай) и Binarly (США); системный программист в области информационной безопасности операционных систем, гипервизорных технологий (VT-x/EPT) и противодействия перспективным киберугрозам; автор более 50 научных работ, включая статьи, патент и доклады на международных конференциях по кибербезопасности в США, Европе, Азии и России; научный руководитель дипломных и научно-исследовательских работ студентов МИФИ по направлению «Информационная безопасность»

Рецензенты:

Запечников Сергей Владимирович, доктор технических наук, доцент, профессор кафедры криптологии и кибербезопасности Института интеллектуальных кибернетических систем Национального исследовательского ядерного университета «МИФИ»;

Ключарёв Петр Георгиевич, доктор технических наук, профессор кафедры информационной безопасности Московского государственного технического университета имени Н.Э. Баумана

Дизайн обложки:

Устинова Анна Олеговна (vk.ru/girafenoir)

Коркин И.Ю.

К66 Защита ядра операционных систем в условиях противодействия : монография / И.Ю. Коркин. — Москва : ИНФРА-М, 2026. — 488 с. — DOI 10.12737/2228735.

ISBN 978-5-16-021527-3 (print)
ISBN 978-5-16-114374-2 (online)

Монография посвящена решению одной из наиболее актуальных проблем информационной безопасности — защите ядра Windows от сложных низкоуровневых угроз. Исследование выполнено в двух взаимосвязанных направлениях: защите памяти ядра от манипуляций со стороны драйверов и обнаружении скрытых гипервизоров, использующих технологию аппаратной виртуализации.

Выполненный анализ эволюции руткит-техник и защитных механизмов ОС показывает, что многие атаки на ядро на практике не обнаруживаются штатными средствами защиты, включая Kernel Patch Protection (PatchGuard). В ответ на эти вызовы автор предлагает два взаимодополняющих подхода. Первый — гипервизорная защита памяти ядра, воплощенная в решениях MemoryMonRWX, AllMemPro и MemoryRanger. Эти системы используют Intel VT-x и EPT для запуска драйверов в изолированных анклавах и для контроля обращений к памяти. Второй подход направлен на выявление скрытых гипервизоров с помощью статистического анализа характеристик длительности выполнения инструкций. Предложенная методика позволяет обнаруживать такие угрозы даже при активном противодействии со стороны нарушителя.

Работа основана на многолетних исследованиях автора, отраженных в его многочисленных научных публикациях, патенте и докладах на международных конференциях.

Для специалистов по информационной безопасности, преподавателей, студентов и аспирантов. Книга также будет полезна разработчикам и экспертам, занимающимся анализом вредоносного программного обеспечения и созданием систем защиты информации для операционных систем.

УДК 004.45+004.49+004.056.5 (075.4)
ББК 16.8:32.97

Данная книга доступна в цветном исполнении в электронно-библиотечной системе Znanium

ISBN 978-5-16-021527-3 (print)
ISBN 978-5-16-114374-2 (online)

© Коркин И.Ю., 2026

IGOR KORKIN

KERNEL PROTECTION OF OPERATING SYSTEMS UNDER COUNTERMEASURES

MONOGRAPH

znanium
электронно-библиотечная система

Moscow
INFRA-M
2026

Author:

Korkin Igor Yurievich, PhD in Cybersecurity (05.13.19), is an alumnus of Department No. 42, National Research Nuclear University MEPhI, Moscow, Russia; Lead Researcher in international cybersecurity projects at Huawei Technologies (China) and Binarly (USA), with expertise in operating system security, hypervisors (Intel VT-x/EPT), and advanced cyber threat mitigation.

With a prolific publication record of over 50 scholarly works, he has authored numerous research papers, holds a patent, and has presented his research at major international cybersecurity conferences worldwide.

He serves as an invited academic supervisor at MEPhI, mentoring undergraduate research and helping to develop the next generation of cybersecurity experts.

Reviewers:

Zapechnikov Sergey Vladimirovich, Doctor of Technical Sciences, Associate Professor, Professor of the Department of Cryptology and Cybersecurity at the Institute of Intelligent Cybernetic Systems of the National Research Nuclear University MEPhI;

Klyucharev Peter Georgievich, Doctor of Technical Sciences, Professor of the Department of Information Security at the Bauman Moscow State Technical University

Korkin I.Yu.

К66 Kernel protection of operating systems under countermeasures : monograph / I.Yu. Korkin. Moscow : INFRA-M, 2026. – 488 p. – DOI 10.12737/2228735.

ISBN 978-5-16-021527-3 (print)
ISBN 978-5-16-114374-2 (online)

This monograph addresses one of the most serious challenges in information security—protection of Windows OS kernel against sophisticated low-level threats.

The research focuses on two interrelated directions: prevention of unauthorized access to the kernel memory and detection of hidden hypervisors that leverage hardware-assisted virtualization technologies.

The presented analysis of rootkit techniques and of built-in OS security mechanisms demonstrates that many kernel-level attacks remain undetected.

Two complementary approaches are proposed to address these challenges.

The first approach introduces hypervisor-based kernel memory protection, implemented in the systems MemoryMonRWX, AllMemPro, and MemoryRanger. These solutions employ Intel VT-x and Extended Page Tables (EPT) to execute drivers within isolated enclaves and to enforce fine-grained memory access control.

The second approach focuses on the detection of stealth hypervisors through statistical analysis of instruction execution time characteristics.

The proposed methodology enables reliable detection of such threats even under active adversarial countermeasures.

The monograph is based on the author's long-term research, reflected in numerous scientific publications, a patent, and presentations at international conferences.

The book is intended for researchers, security practitioners, and students specializing in malware analysis and the development of operating system security mechanisms.

ISBN 978-5-16-021527-3 (print)
ISBN 978-5-16-114374-2 (online)

© Korkin I. Yu., 2026

Список принятых сокращений

ACG — Arbitrary Code Guard (механизм Windows, запрещающий создавать или модифицировать исполняемый код в памяти, что затрудняет эксплуатацию уязвимостей)

ACL — Access Control List (список управления доступом)

ACE — Access Control Entry (запись управления доступом в списке ACL)

ACPI — Advanced Configuration and Power Interface (интерфейс расширенной конфигурации и управления питанием)

AMD SEV — Secure Encrypted Virtualization (технология аппаратного шифрования памяти виртуальных машин)

AMT — Intel Active Management Technology (технология администрирования удаленных компьютерных систем)

APIC — Advanced Programmable Interrupt Controller (расширенный программируемый контроллер прерываний)

APT — Advanced Persistent Threat (целенаправленная и скрытая кибератака)

ASLR — Address Space Layout Randomization (механизм защиты ОС, благодаря которому области памяти пользовательского пространства (код приложения, загружаемых библиотек и структуры данных) размещаются в памяти с использованием случайных смещений, что затрудняет эксплуатацию уязвимостей

AV — Antivirus software (антивирусные средства защиты)

BIOS — Basic Input/Output System (базовая система ввода вывода)

BSOD — Blue Screen of Death (BugCheck, сообщение о системной ошибке (сбое) ядра Windows ОС)

BYOVD — Bring Your Own Vulnerable Driver (атака на основе эксплуатации уязвимостей легитимных драйверов, содержащих цифровую подпись)

CET — Control-flow Enforcement Technology (технология контроля потока исполнения)

CFG — Control Flow Guard (технология контроля целостности потока исполнения в Windows)

DAC — Discretionary Access Control (дискреционный контроль доступа)

DACL — Discretionary Access Control List (дискреционный список управления доступом)

DACR — Domain Access Control Register (ARM-регистр управления доступом к доменам памяти)

DEP — Data Execution Prevention (технология защиты, предотвращающая исполнение кода из страниц памяти, помеченных как «неисполняемые»)

DKOM — Direct Kernel Object Manipulation (прямая манипуляция объектами ядра)

DLL — Dynamic-Link Library (динамически подключаемая библиотека)

DMA — Direct Memory Access (прямой доступ к памяти)

DoS — Denial of Service (атака типа «отказ в обслуживании»)

DPAPI — Data Protection Application Programming Interface (интерфейс Windows API для защиты данных)

DSE — Driver Signature Enforcement (механизм проверки цифровой подписи для драйверов ядра)

EDR — Endpoint Detection and Response (системы обнаружения и реагирования на угрозы на оконечных устройствах пользователя)

ELAM — Early Launch Anti-Malware (технология раннего запуска антивирусного драйвера)

EPT — Extended Page Table (механизм расширенных таблиц страниц памяти)

ETW — Event Tracing for Windows (подсистема трассировки и логирования событий в Windows)

GART — Graphics Address Remapping Table (механизм отображения памяти для видеоподсистемы)

HAL — Hardware Abstraction Layer (слой аппаратных абстракций между оборудованием и ОС)

HPET — High Precision Event Timer (высокоточный таймер событий)

HVCI — Hypervisor-protected Code Integrity (механизм защиты целостности кода в памяти, использующий возможности гипервизора)

IDT — Interrupt Descriptor Table (таблица дескрипторов прерываний)

IOCTL — I/O control code (механизм управляющих кодов-запросов для взаимодействия приложений и драйверов)

IOMMU — Input-Output Memory Management Unit (блок управления отображением памяти для устройств ввода-вывода)

IoC — Indicator of Compromise (индикатор компрометации, технический артефакт, указывающий на факт или признак атаки)

IRP — I/O Request Packet (структура, описывающая запрос ввода-вывода в Windows)

IRQL — Interrupt Request Level (уровень запроса прерывания)

ISA — Instruction Set Architecture (архитектура набора/системы команд)

IUM — Isolated User Mode (изолированный пользовательский режим)

JIT-ROP — Just-In-Time Return-Oriented Programming (разновидность ROP-техники, при которой цепочка гаджетов не подбирается заранее статически, а динамически строится во время атаки «на лету»)

KASLR — Kernel Address Space Layout Randomization (механизм защиты ОС, благодаря которому области памяти ядра (код ядра, драйверы, таблицы и структуры данных) располагаются в случайном порядке при каждой загрузке системы)

KDBG — Kernel Debugger Data Block (структура данных ядра Windows)

KDP — Kernel Data Protection (механизм защиты целостности содержимого данных в памяти, использующий возможности гипервизора)

KMCI — Kernel Mode Code Integrity (часть Device Guard/VBS, компонент проверки целостности кода ядра, блокирующий выполнение неподписанного кода)

KMCS — Kernel-Mode Code Signing (Требование обязательной цифровой подписи для кода, выполняемого в режиме ядра; реализовано в Windows через DSE)

KOH — Kernel Object Hooking (техника перехвата, основанная на изменении пути выполнения путем модификации указателей в структурах данных ядра ОС)

KPCR — Kernel Processor Control Region (структура, в которой ядро хранит данные о процессоре)

KPP — Kernel Patch Protection, PatchGuard (механизм защиты Windows ОС, контролирующий целостность кода и структур памяти ядра без использования гипервизора)

LLM — Large Language Model (большая языковая модель)

LKM — Loadable Kernel Module (загружаемый модуль ядра в контексте Linux)

LSASS — Local Security Authority Subsystem Service (системный сервис, обеспечивающий аутентификацию, авторизацию и управление учетными данными пользователей)

MBR — Master Boot Record (главная загрузочная запись)

MDL — Memory Descriptor List (структура, описывающая физические страницы буфера в памяти, используется в IRP/Direct I/O)

ME — Management Engine (вспомогательный микроконтроллер/подсистема управления на платформах Intel)

MIC — Mandatory Integrity Control (механизм контроля целостности)

MTF — Monitor Trap Flag (флаг в структуре VMCS, установка которого приводит к передаче управления гипервизору после исполнения каждой инструкции кода гостевой ОС)

NCSC — National Computer Security Center (Национальный центр компьютерной безопасности США)

NPT — Nested Page Table (механизм «второй ступени» трансляции адресов для случая аппаратной виртуализации процессоров AMD)

NTFS — New Technology File System (журналируемая файловая система)

NTP — Network Time Protocol (сетевой протокол синхронизации времени)

OS (ОС) — операционная система (Operating System)

PAE — Physical Address Extension (режим расширения физических адресов)

PE — Portable Executable (формат исполняемых файлов, используемый в Windows ОС)

PEB — Process Environment Block (блок окружения процесса)

PFN — Page Frame Number (номер фрейма физической страницы)

PID — Process Identifier (идентификатор процесса)

PIT — Programmable Interval Timer (программируемый интервальный таймер)

PnP — Plug and Play (технология автоматического обнаружения и конфигурирования оборудования)

PPL — Protected Process Light (механизм защиты системных процессов Windows)

PTP — Precision Time Protocol (протокол высокоточной синхронизации времени)

ROP — Return-Oriented Programming (техника эксплуатации уязвимостей, при которой нарушитель не внедряет новый код, а строит цепочку из уже существующих фрагментов кода (гаджетов), заканчивающихся инструкцией возврата)

RSB — Return Stack Buffer (стековый буфер адресов возврата, БСВ)

RSM — Resume From System Management Mode (инструкция возврата управления из режима SMM, ИВУ)

RTC — Real Time Clock (часы реального времени)

SCM — Service Control Manager (диспетчер управления службами Windows)

SGX — Software Guard Extensions (технология создания доверенной среды исполнения)

SID — Security Identifier (идентификатор безопасности)

SKPG — Secure Kernel Patch Guard (HyperGuard, механизм защиты памяти Windows ОС, использующий гипервизор для защиты памяти ядра)

SLAT — Second Level Address Translation (аппаратная поддержка вложенного страничного преобразования адресов)

SMAP — Supervisor Mode Access Prevention (механизм защиты, запрещающий коду в режиме ядра обращаться к памяти пользовательского режима)

SMEP — Supervisor Mode Execution Prevention (механизм защиты, запрещающий коду в режиме ядра выполнять инструкции из памяти пользовательского режима)

SMI — System Management Interrupt (прерывание режима системного управления, ПСУ)

SMM — System Management Mode (режим системного управления процессора)

SMBR — System Management Mode based rootkit (ВПО на основе режима SMM)

SRM — Security Reference Monitor (монитор безопасности Windows)

SSDT — System Service Descriptor Table (таблица дескрипторов системных сервисов)

TEE — Trusted Execution Environment (аппаратно изолированная среда, обеспечивающая защищенное выполнение кода и хранение данных)

TEB — Thread Environment Block (блок окружения потока)

TF — Trap Flag (системный флаг трассировки, установка которого приводит к генерации отладочного исключения #DB после исполнения каждой инструкции программы)

TGT — Kerberos Ticket Granting Ticket (мандат, используемый центром Kerberos)

TLB — Translation Lookaside Buffer (буфер ассоциативной трансляции)

TOCTOU — Time-of-Check to Time-of-Use (тип уязвимости, при котором состояние ресурса может измениться между его проверкой и использованием)

TPM — Trusted Platform Module (аппаратный криптопроцессор/модуль доверенной платформы)

TSC — Time Stamp Counter (счетчик тактов, высокоточный таймер в ЦП)

TXT — Intel Trusted Execution Technology (технология Intel, обеспечивающая безопасность загрузки и работы компьютера)

UAC — User Account Control (механизм защиты Windows ОС, контролирующий запуск действий, требующих административных прав)

UAF — Use-After-Free (тип уязвимости, основанный на использовании освобожденной области памяти)

UDK — UEFI Development Kit (набор для разработки UEFI-компонентов)

UEFI — Unified Extensible Firmware Interface (единый расширяемый микропрограммный интерфейс)

VBS — Virtualization-Based Security (подход к построению защищенной архитектуры Windows, использующий аппаратную виртуализацию)

VMCS — Virtual Machine Control Structure (управляющая структура виртуальной машины)

VMI — Virtual Machine Introspection (механизм анализа виртуальной машины с уровня гипервизора, без активного агента «внутри» гостевой ОС)

VMM — Virtual Machine Monitor (монитор виртуальных машин, МВМ, гипервизор)

VMX — Virtual Machine Extensions (расширения виртуализации в процессорах Intel VT x)

VMX non-root — режим выполнения кода гостевой ОС

VMX root — режим выполнения кода гипервизора

VPN — Virtual Private Network (виртуальная частная сеть)

VPID — Virtual Processor Identifier (виртуальный идентификатор процессора)

VSM — Virtual Secure Mode (виртуальный безопасный режим; изолированная защищенная среда на базе гипервизора Hyper V)

VTL — Virtual Trust Level (виртуальный уровень доверия)

WDCG — Windows Defender Credential Guard (механизм безопасности Windows)

WDK — Windows Driver Kit (набор средств разработки драйверов Windows)

WHQL — Windows Hardware Quality Labs (программа сертификации Microsoft для проверки драйверов устройств)

WinAPI — Windows Application Programming Interface (программный интерфейс приложений Windows)

WMI — Windows Management Instrumentation (технология Microsoft для централизованного управления и мониторинга компьютерных систем)

WSL — Windows Subsystem for Linux (подсистема Windows для запуска Linux утилит)

XFG — eXtended Flow Guard (технология контроля целостности потока исполнения)

xFG — Extreme Flow Guard (технология контроля целостности потока исполнения)

ВПО — вредоносное программное обеспечение (malicious software, malware)

СВТ — средство вычислительной техники

ЦП — центральный процессор

Посвящается моей семье

Предисловие

Монография основана на совокупности результатов независимой научно-исследовательской и практической деятельности автора, кандидата технических наук по специальности 05.13.19 «Методы и системы защиты информации, информационная безопасность», выполненной на протяжении более чем 10 лет. В работе систематизированы материалы, представленные в более чем 40 рецензируемых публикациях, патенте, а также в докладах на 30 международных конференциях по кибербезопасности, что подчеркивает актуальность темы и практическую ценность достигнутых результатов.

Практическая значимость полученных результатов подтверждается их широкой апробацией на ведущих профессиональных площадках:

- HITB Security Conference Phuket 2023 (HITBSecConf2023), Таиланд;
- EKOPARTY, 2022 г., Аргентина;
- ROOTCON, 2022 г., Филиппины;
- LABScon, 2022 г., США;
- Black Hat USA, 2022 г., США;
- Black Hat Europe, 2021 г., Великобритания;
- Texas Cyber Summit, 2021–2022 гг., США;
- Huawei Trustworthy Workshop, Chong Ming Technology Center, 2020-2022 гг., Китай;
- IEEE Symposium on Security and Privacy Workshops, 2021 г., США;
- Hack In The Box Security Conference, 2020 г., Сингапур;
- Black Hat Europe, 2018 г., Великобритания;
- REcon, 2016 г., Канада;
- Управление информационной безопасностью в современном обществе, НИУ ВШЭ «Высшая школа экономики», 2019 г., г. Москва, Россия;

- Annual Conference on Digital Forensics, Security and Law (ADFSL), 2014–2022 гг., (семь конференций), Дейтона-Бич, Флорида, США;
- Skolkovo Cybersecurity Challenge, 2016 г., деревня Сколково, Россия;
- XVI, XVII, XVIII Всероссийская научно-практическая конференция «Проблемы информационной безопасности в системе высшей школы» Инфофорум, 2009–2011 гг., г. Москва, Россия;
- XIV Международная телекоммуникационная конференция молодых ученых и студентов «Молодежь и наука», 2011 г., г. Москва, Россия;
- XIX и XXI Общероссийская научно-техническая конференция «Методы и технические средства обеспечения безопасности информации», 2010–2012 гг., г. Санкт-Петербург, Россия;
- XV конференция «Телекоммуникации и новые информационные технологии в образовании», НИЯУ МИФИ, 2011 г., г. Москва, Россия;
- семинары в Институте системного программирования Российской академии наук, 2011 г., г. Москва, Россия;
- семинар в Московском Государственном Техническом Университете им. Н.Э. Баумана, 2011 г., г. Москва, Россия;
- XIII Международная конференция «РусКрипто», 2011 г., Россия.

Результаты автора представлены в ведущих российских и зарубежных академических и научно-практических журналах, среди которых:

- Журнал «Безопасность информационных технологий»;
- Журнал «Системы высокой доступности»;
- Журнал «Вопросы кибербезопасности»;
- Журнал «Хакер»;
- Journal of Digital Forensics, Security and Law;
- IOSR Journal of Computer Engineering (IOSR-JCE);
- Hakin9 Extra Magazine;
- Proceedings of the 2021 IEEE Security and Privacy Workshops (SPW).

Профессиональный опыт автора охватывает роли исследователя, системного программиста и технического эксперта в ве-

дущих российских и международных компаниях в области информационной безопасности: предприятие системы ФСТЭК России, Центр специальной системотехники (Россия), Binarly (США), Huawei Technologies Co. Ltd. (Китай), Positive Technologies (Россия). Кроме того, на протяжении многих лет автор успешно руководит дипломными проектами студентов НИЯУ МИФИ по направлению «Информационная безопасность».

Благодарности

Выражаю глубокую благодарность **Тихонову Андрею Юрьевичу**, открывшему для меня мир системного программирования, исследования ядра Windows и механизмов безопасности, за неоценимый вклад в мое профессиональное становление как технического эксперта.

С особой признательностью отмечаю вклад заместителя заведующего кафедрой № 42 «Криптология и кибербезопасность» МИФИ (1997–2012 гг.), кандидата технических наук, доцента **Петровой Тамары Васильевны**, в формирование моего научного мировоззрения, эффективное научное руководство и многолетнее наставничество в области информационной безопасности. Научная школа Тамары Васильевны внесла существенный вклад в развитие методов и средств защиты информации и во многом определила мои научные интересы.

Введение

И, чтобы жертвою не стать самообмана, бди!
Козьма Прутков
Собрание сочинений,
1884 г.

Актуальность и контекст

Первая четверть XXI в. ознаменовалась бурным развитием и массовым внедрением информационно-коммуникационных технологий во все сферы жизни и бизнеса, что определило технический облик цифровой глобализации, открыв новые возможности и породив новые угрозы.

Последствия пандемии COVID-19 ускорили глобальный переход к повсеместному использованию удаленных рабочих мест. Эти факторы существенно размыли границы традиционного периметра безопасности организаций и создали новые векторы компьютерных атак.

Бурное развитие больших языковых моделей (Large Language Models, LLMs) предоставило нарушителям принципиально новые инструменты для автоматизации и совершенствования вредоносного программного обеспечения (ВПО), что осложняет задачу обеспечения кибербезопасности и требует адаптации современных средств защиты информации.

Широкое внедрение анонимных пиринговых платежных систем не только облегчает нарушителям скрытое обогащение, но и формирует благоприятные условия для развития и монетизации ВПО.

Центральная роль Windows в современной бизнес-инфраструктуре

Операционная система (ОС) Windows продолжает занимать доминирующее положение в сегменте настольных компьютеров как в глобальном масштабе, так и в российской ИТ-инфраструктуре. По данным *Microsoft*, летом 2025 г. более 1,5 млрд

устройств во всем мире продолжают работать под управлением Windows ОС.

Согласно данным сервиса StatCounter за 2024 г., представленным в отчете «Desktop Operating System Market Share Worldwide», доля операционных систем Windows в сегменте настольных компьютеров составила более 70% на международном уровне. Сервис StatCounter отмечает, что в России этот показатель еще выше — по итогам 2024 г. доля Windows превысила 85%, что подтверждает ее ключевую роль в корпоративной, государственной и частной сферах.

Альтернативные платформы, такие как macOS, Linux и отечественные операционные системы (например, Astra Linux), постепенно укрепляют свои позиции, особенно в сегменте импортозамещения. Однако их совокупная доля пока остается крайне ограниченной, и они не оказывают существенного влияния на технологическое доминирование Windows ОС в бизнес-среде.

Значимость своевременного обнаружения атак

Защита ядра ОС является одной из центральных задач информационной безопасности в современных вычислительных системах. Ядро ОС Windows, как важнейшая часть ОС, выступает одновременно объектом для атак злоумышленников и областью применения различных защитных инструментов.

Для обеспечения более длительного присутствия ВПО на зараженном (целевом) компьютере нарушители используют различные техники, затрудняющие или исключающие обнаружение и анализ ВПО, к которым можно отнести следующие: сокрытие ВПО от средств обнаружения, в том числе блокировка работы средств защиты, противодействие статическому и динамическому анализу, обнаружение изолированных виртуальных сред («песочниц») для анализа ВПО и другие техники.

В современных условиях, когда темпы эволюции угроз непрерывно ускоряются, а обеспечение цифрового суверенитета становится стратегическим приоритетом, задача обнаружения атак выходит на первый план и требует особого внимания со стороны исследователей и практиков в области информационной безопасности.

Цели и задачи монографии

Данная монография посвящена комплексному анализу существующих угроз безопасности ядра Windows, исследованию встроенных механизмов защиты и разработке новых методов мониторинга и защиты памяти ядра на базе технологий аппаратной виртуализации. Особое внимание уделено авторским гипервизорным решениям и математическим методам обнаружения скрытых гипервизоров в условиях активного противодействия.

В работе будет представлен анализ существующих техник сокрытия ВПО в ОС Windows с использованием драйверов и гипервизоров, а также анализ других примеров атак на ядро Windows. Будет выполнен анализ возможностей средства защиты на базе драйвера, позволяющего обнаружить рассмотренные атаки. Будет представлен анализ возможностей и опыт применения гипервизоров, построенных на базе технологии аппаратной виртуализации (bare-metal hypervisor) Intel VT-x и технологии виртуализации страниц памяти таблиц, Extended Page Table (EPT) для защиты памяти ядра Windows от атак со стороны драйверов.

Цели работы: предоставить читателям систематизированные сведения о текущем состоянии проблем защиты ядра Windows, возможностях гипервизорных технологий для их решения, а также о проблеме обнаружения и предлагаемом решении для выявления скрытых гипервизоров в условиях противодействия.

В монографии решаются следующие задачи:

- анализ архитектуры ядра Windows и эволюции встроенных средств защиты;
- исследование техник сокрытия кода и методов его обнаружения;
- разработка и описание гипервизорных технологий мониторинга и защиты памяти ядра;
- разработка математических моделей и алгоритмов для выявления скрытых гипервизоров в условиях противодействия;
- проведение экспериментальных исследований и оценка предложенных решений.

Структура монографии. Монография включает три раздела.

Раздел 1 рассматривает техники сокрытия и обнаружения исполнимых объектов ОС Windows: процессов, потоков, сервисов и драйверов. Представленный ретроспективный анализ техник сокрытия (руткит-техник) позволяет целостно представить картину угроз для ОС Windows. Для противодействия таким атакам разработчики Windows выпустили ряд встроенных средств защиты ядра, таких как Kernel Patch Protection (KPP, также известен как PatchGuard), Device Guard Code Integrity (HVCI), Kernel Data Protection (KDP). Эти новые средства защиты существенно изменили ландшафт угроз, в результате часть описанных атак потеряла свою актуальность. Однако принципы атак, лежащие в основе рассмотренных руткит-техник, остаются применимыми и для новых версий ОС Windows. Данные атаки могут быть направлены на штатные системы защиты, такие как антивирусные средства защиты (Antivirus software, AV), и системы для обнаружения и реагирования на угрозы на оконечных устройствах (Endpoint Detection and Response, EDR). Конкретные примеры таких атак и возможных способов противодействия им с помощью гипервизора будут представлены во втором разделе. Особое внимание уделяется рассмотрению способов обнаружения скрытого программного обеспечения (ПО) с помощью анализа содержимого дампа компьютерной памяти (memory forensics).

Раздел 2 посвящен вопросам разработки и применения гипервизоров для решения задач мониторинга и защиты памяти ядра Windows от атак со стороны драйверов. Подробно рассматриваются принципы работы технологии аппаратной виртуализации, современные гипервизорные технологии и их использование для изоляции и контроля доступа к критическим областям памяти ядра. Описаны авторские гипервизорные решения (MemoryMonRWX, AllMemPro, MemoryRanger), их архитектура, принципы работы, а также результаты экспериментального применения для предотвращения атак типа BYOVD (Bring Your Own Vulnerable Driver), защиты от несанкционированного доступа к структурам ядра, изоляции драйверов и предотвращения атак на штатные механизмы защиты ядра (PatchGuard, HVCI, KDP), в том числе атак на встроенный антивирус Windows Defender.

Раздел 3 посвящен вопросу обнаружения программного обеспечения, использующего возможности технологии аппаратной виртуализации (гипервизоров) для своего сокрытия. Представлен системный анализ методов обнаружения скрытых гипервизоров, а также техник сокрытия, которые используются нарушителями для противодействия обнаружению гипервизора. Предложенный авторский подход для обнаружения гипервизоров имеет следующие отличительные черты, ярко выделяющие его конкурентоспособность: способность обнаруживать гипервизор в условиях преднамеренного противодействия обнаружению, а также функции по обнаружению нескольких вложенных образцов гипервизоров. Приведены математические модели, алгоритмы и результаты экспериментальной апробации, а также рассмотрены перспективы дальнейшего развития методов обнаружения скрытых гипервизоров в условиях эскалации киберугроз.

Практическая апробация и признание научным сообществом

Авторские методы и гипервизорные решения прошли всестороннюю экспериментальную проверку. Результаты исследований и технических решений были неоднократно представлены на ведущих международных и российских конференциях по вопросам кибербезопасности, включая Black Hat USA, Black Hat Europe, HITB Security Conference, EKOPARTY, LABScon, Texas Cyber Summit, IEEE Symposium on Security and Privacy Workshops, а также на профильных российских форумах и научных мероприятиях. Это подтверждает не только научную новизну, но и практическую востребованность предложенных подходов среди специалистов и исследователей в области безопасности операционных систем и гипервизорных технологий.

Монография объединяет исследования в области защиты ядра Windows, направленные на анализ новых угроз, разработку гипервизоров и применение математических методов для выявления скрытых атак, что делает ее актуальной как для научного сообщества, так и для практиков в сфере информационной безопасности: разработчиков системных решений по защите информации, а также для исследователей, занимающихся вопросами безопасности операционных систем.

Раздел 1
АНАЛИЗ ВОПРОСОВ БЕЗОПАСНОСТИ ЯДРА WINDOWS ОС: МЕХАНИЗМЫ ЗАЩИТЫ И ТЕХНИКИ АТАК

明枪易躲，暗箭难防

Легко уклониться от явной атаки,
трудно защититься от скрытого удара.
Китайская пословица

Глава 1
ВВЕДЕНИЕ В АРХИТЕКТУРУ WINDOWS ОС С ПОЗИЦИИ ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ

Операционная система Windows является одной из наиболее популярных платформ как для персонального, так и для корпоративного использования в России и за рубежом. Ее широкое распространение делает систему привлекательной целью для нарушителей, стремящихся получить доступ к конфиденциальной информации, такой как персональные данные пользователей, банковские реквизиты и корпоративные документы. На компьютерах под управлением Windows обрабатывается огромный объем важной информации, что создает значительные риски безопасности.

Современные киберугрозы стремительно развиваются в направлении комплексных многовекторных атак, которые сочетают эксплуатацию уязвимостей инфраструктуры, целенаправленные методы социальной инженерии и продвинутые техники обхода систем защиты. Такие атаки часто реализуются как мно-

гоэтапные операции, где каждый этап служит конкретной тактической цели в рамках общей стратегии компрометации целевой системы или сети. Особую опасность представляют гибридные атаки, объединяющие использование эксплойтов нулевого дня (0-day), разнообразные методы обхода обнаружения EDR/AV-систем и продвинутые антифорензик-техники. Современные технологии ВПО позволяют нарушителям скрытно контролировать целевую систему длительное время.

Особое внимание в данном разделе уделено анализу техник сокрытия вредоносного программного обеспечения в ОС Windows. Исчерпывающий анализ всех потенциальных техник сокрытия вредоносного ПО не представляется возможным из-за непрерывной эволюции инструментария атакующих и высоких темпов роста компетенций нарушителей.

Представленный в разделе ретроспективный анализ техник сокрытия (руткит-техник) формирует целостное представление о ландшафте атак, нацеленных на ядро ОС Windows. Этот анализ углубляет понимание векторов атак и методов, применяемых злоумышленниками для обхода защитных механизмов на уровне ядра ОС. В ответ на такие угрозы разработчики Windows внедрили ряд встроенных средств защиты ядра, включая Kernel Patch Protection (KPP, также известный как PatchGuard), Device Guard Code Integrity (HVCI), Kernel Data Protection (KDP) и Driver Signature Enforcement (DSE). Появление этих защитных технологий существенно изменило ландшафт угроз: часть рассмотренных атак утратила актуальность, однако фундаментальные технические принципы руткит-атак сохраняют свою применимость и для новых методов компрометации других системных компонентов Windows. В разделе 2 будут представлены примеры подобных атак и возможные способы их нейтрализации с использованием гипервизорных технологий.

Одно из средств защиты, серьезно изменившее ландшафт угроз ядра Windows, — средство Kernel Patch Protection, которое было реализовано для обнаружения фактов несанкционированного изменения структур ядра и дальнейшей аварийной остановки работы ОС с помощью «синего экрана смерти» — Blue Screen of Death (BSOD), что исключает дальнейшую работу пользователя

в зараженной ОС. В условиях работающего PatchGuard задача сокрытия кода в ядре ОС сводится к его отключению, что является нетривиальной задачей, поскольку одна из отличительных черт PatchGuard состоит в следующем: он реализован таким образом, чтобы затруднить не только свое отключение, но и реверс-инжиниринговое исследование принципов своей работы.

Для имитации наиболее сложного сценария сокрытия в работе делается предположение, что нарушителям предварительно удалось отключить PatchGuard и скрыть драйвер от обнаружения с использованием как штатных средств ОС Windows, так и популярных опубликованных средств защиты, использующих остаточную информацию о загруженном драйвере. Для обнаружения такого наиболее сложного случая скрытого драйвера представлены анализ подходов и авторский способ обнаружения.

Вторая половина раздела посвящена рассмотрению способов обнаружения скрытого ВПО с помощью анализа компьютерной памяти (memory forensics). Представлены возможные техники нарушителей и описан пример наиболее сложного случая, когда эксперту необходимо обнаружить ПО в памяти путем детектирования отдельных фрагментов исполнимого кода, распределенных по памяти. В заключение приведено описание авторского инструмента для получения и анализа содержимого дампа виртуального контекста памяти.

Понимание базовой архитектуры операционной системы является ключевым аспектом обеспечения безопасности Windows. Мы детально проанализируем основные компоненты ядра Windows с точки зрения их роли в защите данных и функционировании механизмов безопасности.

Кроме того, ниже приведен комплексный обзор эволюции встроенных механизмов безопасности Windows — от базовых возможностей в ранних версиях до современных решений, таких как Virtualization-Based Security, активно применяемых в Windows 11. Этот исторический экскурс позволяет глубже понять текущее состояние и перспективы развития системы безопасности Windows в контексте постоянно усложняющихся киберугроз, демонстрируя, как архитектура ОС адаптируется к новым вызовам в сфере информационной безопасности.

1.1. ПОНЯТИЕ ЗАЩИЩЕННОГО РЕЖИМА РАБОТЫ ПРОЦЕССОРА, ТЕХНОЛОГИИ АППАРАТНОЙ ВИРТУАЛИЗАЦИИ И ИХ ВОЗМОЖНОСТИ С ПОЗИЦИИ ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ

Если рассматривать развитие архитектур компьютерных систем с позиции информационной безопасности, то можно выделить определенный тренд по изолированию программных компонентов, позволяющий строить сложные компьютерные системы в условиях неизбежных программных сбоев.

В первых поколениях микропроцессоров *Intel 8086* (1978 г.) основным режимом работы был реальный режим, позволяющий в каждый момент времени выполнять только одну программу, которая могла получить доступ к любой области памяти, включая системную память, а также прямой доступ к подключенному оборудованию. Многие ошибки в таких программах приводили к полной остановке всей системы, требуя перезагрузки компьютера.

В дальнейшем в процессоры был добавлен защищенный режим работы процессора (1982 г.), который включал в себя ряд новых механизмов защиты, таких как кольца защиты, механизм многозадачности, механизм страничной адресации, что предоставило возможность ограничить доступ прикладных программ к критическим ресурсам системы: системной области памяти и подключенному оборудованию. Популярные ОС Windows и Linux для своей работы используют два из четырех возможных колец защиты: наименее привилегированное третье кольцо (ring 3) для запуска пользовательских программ и системных служб (сервисов) и наиболее привилегированное нулевое кольцо (ring 0) для работы драйверов и ядра ОС. Такая двухуровневая модель стала де-факто стандартом и используется в настоящее время во многих корпоративных системах. Это разделение позволило изолировать уязвимые пользовательские компоненты от компонентов ядра ОС, что способствовало повышению общей защищенности компьютерных систем и открыло возможности для создания более сложных ОС с широким набором пользовательских приложений. Одним из преимуществ защищенного

режима работы процессора стала возможность изолирования страниц памяти ядра от несанкционированного доступа со стороны прикладных программ. Однако ошибки в ядре ОС все еще приводили к сбою всей системы, а также рост кибератак с использованием уязвимых драйверов ядра и подписанных вредоносных драйверов начал представлять новую угрозу безопасности.

Появление технологии аппаратной виртуализации стало логическим продолжением развития защитных механизмов со стороны процессора. Виртуализация дополнила защищенный режим работы процессора новым привилегированным режимом, изолировав ОС от оборудования. Виртуализация позволила запускать код гипервизора, работающий в режиме более привилегированном, чем ядро ОС, а также изолировать страницы памяти гипервизора от доступа со стороны драйверов и ядра ОС. Например, с помощью технологии аппаратной виртуализации стал возможным запуск драйверов ядра в изолированных областях памяти — проект MemoryRanger. Компания *Microsoft* в рамках продукта Secure Kernel и технологий Virtualization-Based Security и Hyper-V обеспечила ограничение доступа к памяти критических пользовательских приложений и драйверов путем запуска второго ядра ОС в отдельной изолированной области памяти. Технология аппаратной виртуализации позволяет создавать новые, более гибкие механизмы защиты, выходящие за рамки модели двух колец [1].

Очевидно, что каждый новый этап развития архитектуры процессоров и ОС направлен на решение проблем безопасности, унаследованных от предыдущих поколений. Это действительно эволюционный процесс, где каждое новое поколение технологий стремится обеспечить более высокий уровень изоляции и защиты. Важно отметить, что текущие тенденции в области безопасности, такие как аппаратные анклавы, например Intel's Software Guard Extensions (SGX) и технологии доверенного выполнения Trusted Execution Environment (TEE), а также контейнеризация, продолжают этот тренд, стремясь обеспечить еще более высокий уровень изоляции для различных компонентов системы. Этот тренд будет продолжаться и в будущем.

Защищенный режим работы процессора. Защищенный режим работы процессора (Protected Mode) — это режим, который обеспечивает выполнение прикладных программ в среде, изолированной от ядра ОС. В результате становится возможным предотвратить случайное или злонамеренное взаимодействие программ с критически важными компонентами системы, а также с использованием контекстов процессов изолировать память прикладных программ друг от друга.

Intel 80386 предоставлял два основных режима работы.

Режим реальной адресации: в этом режиме процессор функционировал аналогично процессору 8086, но с более высокой скоростью выполнения инструкций благодаря усовершенствованиям в архитектуре и производственных технологиях.

Режим защищенной виртуальной адресации: этот режим позволял одновременное выполнение нескольких задач, создавая эффект работы множества процессоров 8086, каждый из которых выполнял свою задачу. Это увеличивало эффективность использования ресурсов процессора и общую производительность системы, обеспечивая параллельную обработку данных и задач [2].

Защищенный режим позволяет использовать такие функции, как многозадачность, виртуальная память и кольца защиты. В отличие от реального режима защищенный режим обеспечивает защиту памяти, предотвращая несанкционированный доступ программ к данным и коду, принадлежащим другим программам и ядру ОС.

В защищенном режиме процессор позволяет выполнять код на разных уровнях привилегий, называемых кольцами защиты от 0 до 3, где прикладные программы пользователя выполняются на наименее привилегированном уровне 3 (ring 3), а уровень 0 (ring 0) с наибольшими привилегиями предназначен для ядра ОС и драйверов. Разделение режимов пользователя (ring 3) и ядра (ring 0) защищает ОС от ошибок или вредоносного кода в пользовательских программах только при соблюдении условий безопасности, например, отсутствии прав администратора, включенных механизмов проверки подписи драйверов и т.д. В случае

если вредоносная программа получает привилегии для загрузки драйвера, защита ядра может быть нарушена.

Взаимодействие программ с ядром ОС осуществляется посредством системных вызовов (system calls). Исторически для этого в Windows и Linux ОС использовались прерывания INT 0x2E и INT 0x80 соответственно. В современных версиях операционных систем применяются инструкции SYSENTER (SYSEXIT) и SYSCALL (SYSRET), обеспечивающие переключение между кольцами защиты. Этот механизм колец защиты позволяет контролировать доступ приложений к памяти ядра и аппаратным ресурсам, а также реализовывать дополнительные механизмы защиты на уровне ОС, например монитор безопасности (Security Reference Monitor, SRM).

Перечислим основные возможности защищенного режима с точки зрения информационной безопасности.

1. *Изоляция процессов*: каждый процесс имеет собственное адресное пространство, в результате сбой в работе одного из процессов не оказывает влияния на работоспособность всей ОС.

2. *Аппаратная поддержка многозадачности*: позволяет операционной системе эффективно управлять выполнением нескольких задач, минимизируя последствия сбоев. Защищенный режим работы процессора обеспечивает базовый уровень изоляции и безопасности. Реализация дополнительных уровней защиты и создание изолированной вычислительной среды достигаются благодаря применению различных технологий, в том числе аппаратной виртуализации [3].

Технологии аппаратной виртуализации. Появление технологии аппаратной виртуализации стало логическим продолжением развития защитных механизмов со стороны процессора. Примерами таких технологий являются Intel VT (Intel Virtualization Technology) и AMD-V (AMD Virtualization). Виртуализация дополнила защищенный режим работы процессора новым дополнительным уровнем изоляции между ОС и оборудованием. Технология аппаратной виртуализации позволила запускать код гипервизора в новом режиме, называемом VMX root, являющимся более привилегированным, чем ядро ОС, в то время как операционная система продолжает работу в режиме VMX non-root. Это

обеспечило возможность использования нового бита страниц для указания типа памяти: помимо памяти прикладных программ и системной области памяти, появилась память гипервизора, защищенная от попыток доступа со стороны ядра ОС.

Перечислим основные возможности аппаратной виртуализации с точки зрения информационной безопасности.

1. *Изоляция виртуальных машин*: каждая виртуальная машина (ВМ) работает в собственном изолированном окружении.

2. *Гипервизор*: специальное ПО, работающее в более привилегированном режиме и управляющее виртуальными машинами. Гипервизоры могут управлять ресурсами и контролировать их, обеспечивая дополнительный уровень безопасности, реализуя новый уровень разделения доступа к памяти.

Несмотря на преимущества, технологии аппаратной виртуализации могут быть использованы нарушителями. Одной из серьезных угроз является вредоносное ПО, использующее для своего сокрытия возможности аппаратной виртуализации, так называемые гипервизорные руткиты (Hypervisor-based rootkits), которые работают на уровне гипервизора. Эти руткиты могут перехватывать и изменять операции ОС, оставаясь при этом незамеченными для популярных антивирусных средств защиты и систем обнаружения вторжений, что делает их обнаружение и удаление нетривиальной задачей.

1.2. АРХИТЕКТУРА И ЯДРО WINDOWS ОС

Windows ОС представляет собой операционную систему с вытесняющей многозадачностью. Ядро Windows является гибридным, сочетая в себе элементы микроядерной и монолитной архитектуры. Это позволяет достичь баланса между производительностью и модульностью.

Основные компоненты ОС, такие как ядро Windows (Windows Kernel), планировщик процессов (Process Scheduler), менеджер процессов (Process Manager) и диспетчер виртуальной памяти (Virtual Memory Manager), работают в привилегированном режиме, что характерно для монолитных ядер. В то же время отдельные системные службы (сервисы ОС), например служба печати (Print Spooler),

функционируют в пользовательском режиме, что соответствует микроядерному подходу. Такая архитектура отражает гибридный характер ядра Windows, сочетающий преимущества монолитного и микроядерного дизайна.

Упрощенная схема архитектуры операционной системы Windows представлена на рис. 1. Windows ОС использует два уровня привилегий, поддерживаемые защищенным режимом работы процессора: пользовательский режим (user-mode) и режим ядра (kernel mode) [4].

В пользовательском режиме расположены прикладные процессы (приложения пользователя), подсистемы среды, такие как Win32, а также системные службы. Исторически поддерживались подсистемы POSIX и OS/2, но в современных версиях ОС, например Windows 10/11, они заменены или исключены. POSIX эмулируется через WSL, а поддержка OS/2 прекращена.

Динамические библиотеки (DLL) используются для разделения кода между процессами. Библиотека ntdll.dll служит интерфейсом взаимодействия между пользовательскими приложениями и ядром, позволяя прикладным программам вызывать системные функции, реализованные в ядре ОС. Большинство DLL, в частности user32.dll, gdi32.dll, msvcrt.dll, работают исключительно в пользовательском режиме и не взаимодействуют напрямую с ядром. Каждый процесс в Windows работает в отдельном виртуальном адресном пространстве или программном контексте, что обеспечивает изоляцию процессов друг от друга. В памяти процесса находятся следующие компоненты: загруженный исполнимый код приложения, загруженные библиотеки (DLL), блок окружения процесса (Process Environment Block, PEB), блоки окружения потоков (Thread Environment Block, TEB), различные элементы памяти: кучи (heaps) и стеки (stacks), файлы, отображенные в память (memory-mapped files), и другие ресурсы.

Код, работающий в привилегированном режиме процессора, известном как режим ядра (Kernel Mode), в отличие от кода прикладного режима имеет полный доступ к аппаратным ресурсам компьютера: системной памяти и подключенному оборудованию. Ядро обеспечивает основные функции операционной системы, такие как управление процессами, памятью, устройствами ввода-вывода, а также обеспечение безопасности [5, 6].

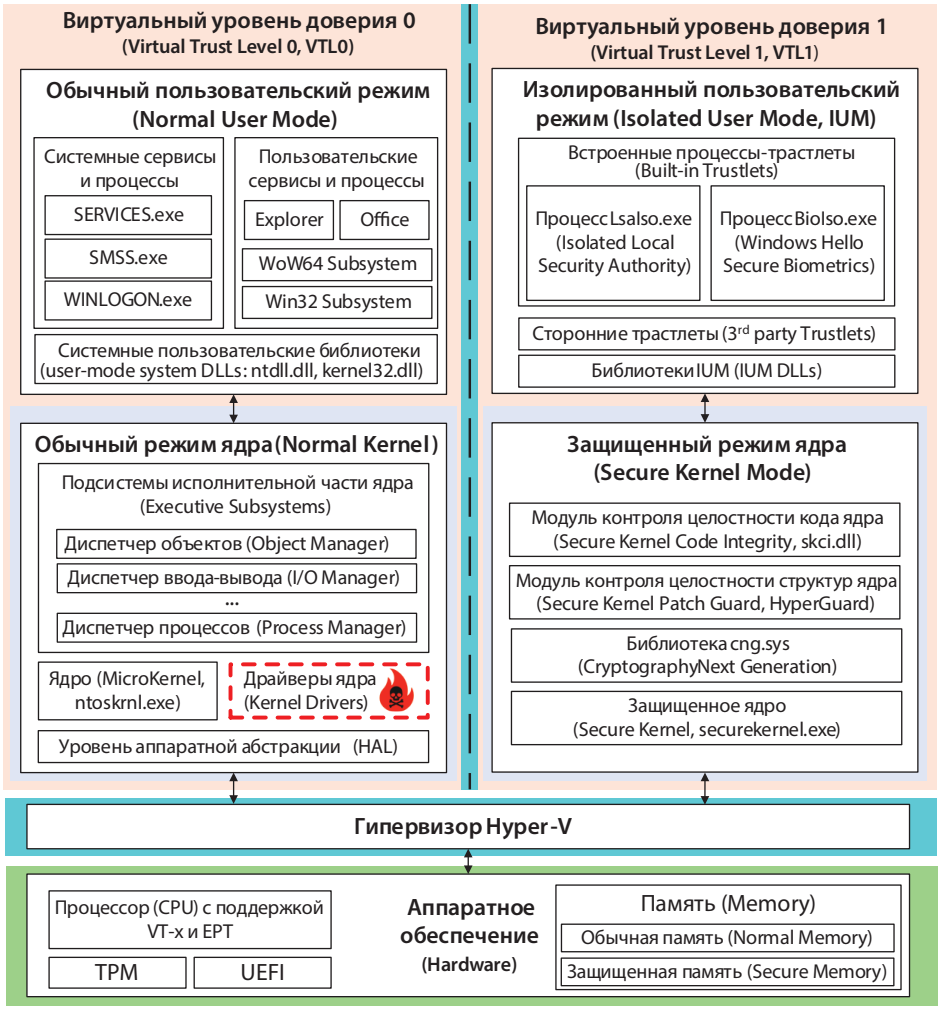


Рис. 1. Упрощенная схема современной архитектуры Windows ОС с включенной системой безопасности на основе аппаратной виртуализации

Можно выделить следующие компоненты ОС, работающие в привилегированном режиме:

- ядро ОС (Windows Kernel);
- исполнительная подсистема (executive);
- подсистема аппаратных абстракций (Hardware Abstraction Layer, HAL);
- драйверы устройств (Device Drivers).

Ядро Windows (микроядро) отвечает за множество низкоуровневых операций, включая обработку системных вызовов, исключений и прерываний, диспетчеризацию потоков, синхронизацию многопроцессорных систем, управление устройствами.

Исполнительную подсистему Windows (executive) можно условно разделить на системные менеджеры (диспетчеры), которые реализуют объектную модель Windows — диспетчер объектов (Object Manager), управление процессами (Process Manager) и потоками (Thread Manager), виртуальной памятью (Memory Manager), вводом-выводом (I/O Manager) и разграничением доступа к ресурсам (SRM).

Подсистема аппаратных абстракций (Hardware Abstraction Layer, HAL) является интерфейсом между вышележащими компонентами ядра ОС и оборудованием, обеспечивая переносимость ОС между различными аппаратными платформами.

Драйверы устройств (драйверы ядра) представляют собой исполнимые модули, своего рода DLL режима ядра, которые работают в привилегированном режиме и предоставляют интерфейсы для взаимодействия с ними. Windows использует многоуровневую модель драйверов для обеспечения расширяемости и гибкости, например для файловых систем и сетевых устройств.

Драйверы открывают новые возможности для реализации перспективных средств защиты информации. Антивирусные файловые драйверы позволяют отслеживать файловые операции и блокировать работу вредоносного ПО на ранней стадии. Прозрачное шифрование данных на уровне файловой системы можно реализовать с помощью драйвера мини-фильтра. BitLocker в Windows использует свой драйвер для шифрования дисков. Особую роль в современных системах играют драйверы, используемые в решениях класса EDR (Endpoint Detection and Response) и Sandbox. Эти драйверы позволяют собирать различную телеметрию, например информацию о процессах, сетевых соединениях, файловых операциях, что позволяет выявлять сложные и неизвестные угрозы (Advanced Persistent Threat, APT).

Все загруженные драйверы в Windows работают в общем адресном пространстве с ядром ОС. Такая архитектура обеспечивает высокую скорость доступа к памяти ядра, однако одновременно порождает новые угрозы безопасности, так как отсутствуют встроенные механизмы контроля доступа к памяти ядра, что позволяет потенциально вредоносным драйверам осуществлять несанкционированный доступ к памяти ядра.

С позиции нарушителей драйверы открывают новые возможности, поскольку имеют полный доступ к системной памяти и компонентам ядра, могут взаимодействовать напрямую с оборудованием. Для проведения атак нарушители могут как загружать собственные драйверы, так и эксплуатировать уязвимости существующих драйверов Windows ОС и сторонних компаний.

Можно выделить некоторые отдельные примеры использования драйверов нарушителями:

- обход и отключение различных защитных механизмов ОС;
- повышение привилегий в целях получения несанкционированного доступа к парольно-адресной информации пользователя;
- возможности сокрытия своего присутствия от штатных и специализированных средств обнаружения.

Существующие встроенные механизмы защиты Windows обеспечивают контроль целостности лишь некоторых списков структур и объектов в памяти (таблицы IDT (Interrupt Descriptor Table) и SSDT (System Service Descriptor Table), отдельные поля EPROCESS). В свою очередь, эти механизмы также могут быть отключены. В результате нарушители могут использовать возможности драйверов для достижения различных вредоносных целей, в том числе для активного противодействия своему обнаружению и блокировке работы средств защиты. Впоследствии мы рассмотрим атаки по сокрытию от штатных средств ОС и возможное развитие таких атак.

1.3. ИСТОРИЯ РАЗВИТИЯ ВСТРОЕННЫХ МЕХАНИЗМОВ ЗАЩИТЫ WINDOWS ОС

Совершенствование систем компьютерной безопасности тесно связано с эволюцией киберугроз. Этот процесс обусловлен тремя ключевыми факторами: 1) экспоненциальным ростом объема

цифровой информации, обрабатываемой на персональных компьютерах; 2) глобальным распространением подключений к сети Интернет; 3) появлением и популяризацией анонимных пиринговых платежных систем. Эти факторы создали благоприятную среду для развития и распространения ВПО, что, в свою очередь, стимулировало разработку новых, более совершенных защитных механизмов в ОС Windows.

В результате мы наблюдаем непрерывное техническое противостояние, своего рода «гонку вооружений» в цифровом пространстве между киберпреступниками и специалистами по информационной безопасности.

История развития механизмов защиты ОС Windows является ярким примером эволюции средств информационной безопасности, от простых защитных мер в первых версиях до использования продвинутых технологий аппаратной виртуализации в современных версиях, таких как Virtualization-based Security (VBS) в Windows 11.

Ранние версии Windows (Windows 1.0 — Windows 3.1). Основной задачей первых версий Windows 1.0–3.1 (1985–1992) было предоставление графического интерфейса поверх консольного MS-DOS. Присутствовали лишь некоторые базовые элементы защиты, такие как атрибуты файлов и контроль доступа на уровне общих ресурсов. Windows 3.1 ввела концепцию профилей пользователей (User Profiles), позволяющую разным пользователям иметь собственные настройки, хотя это было больше для персонализации, чем для безопасности. Важным техническим улучшением Windows 3.1 стал переход к работе преимущественно в защищенном режиме процессора, что было шагом вперед по сравнению с реальным режимом (real mode) ранних версий. Отсутствовали механизмы по контролю целостности ОС и встроенные защитные антивирусные системы, что оставляло систему уязвимой для многих типов атак.

Windows 95 и Windows 98. С выходом Windows 95 в 1995 г. и последующей Windows 98 в 1998 г. система безопасности получила некоторые улучшения. Был введен контроль доступа на уровне пользователей, хотя и в упрощенной форме. Появилась

поддержка файловой системы FAT32, что позволило улучшить управление дисковым пространством, но не повысило безопасность данных. Основные улучшения касались удобства пользователя, поддержки технологии Plug and Play (PnP), улучшенной работы с интернетом и виртуальными частными сетями (VPN). Несмотря на это, данные версии оставались уязвимыми для множества атак, поскольку архитектура системы все еще опиралась на MS-DOS для некоторых низкоуровневых операций. Механизмов контроля целостности ОС и антивирусных систем не было.

Windows NT и Windows 2000 (1993–2000). Релиз Windows NT 3.1 в 1993 г. ознаменовал значительный шаг вперед в области безопасности операционных систем линейки Windows.

Можно сказать, что Windows NT была фактически разработана с нуля с учетом требований вытесняющей многозадачности и информационной безопасности, что выгодно отличало ее от ОС семейства Windows 9x. Важным нововведением Windows NT стало внедрение концепции разделения режимов работы ядра (kernel mode) и пользователя (user mode), что позволило изолировать код пользовательских приложений от ядра.

Windows NT была разработана с учетом требований уровня безопасности C2, представленных в «Критериях оценки безопасности компьютерных систем» (также известных как «Оранжевая книга»), которые были разработаны Министерством обороны США и Национальным центром компьютерной безопасности (National Computer Security Center, NCSC). К основным нововведениям можно отнести следующие: внедрение многопользовательской модели с разделением прав доступа, файловой системы NTFS, механизма аудита событий безопасности (Auditing Security Events), менеджера учетных записей безопасности (Security Account Manager, SAM), модели контроля доступа на основе маркеров доступа (Access Tokens), дискреционных списков управления доступом (Discretionary Access Control Lists, DACL). Windows NT включала поддержку файловой системы NTFS, обеспечивая новые возможности за счет механизма контроля до-

ступа к объектам, а также поддерживала журналирование файловых операций [7].

Windows 2000 (также известная как Windows NT 5.0), выпущенная в 2000 г., развила концепции, заложенные в NT, и добавила ряд новых функций безопасности. Были добавлены система шифрования файлов Encrypting File System (EFS), интерфейс для защиты данных Data Protection Application Programming Interface (DPAPI). Существенно улучшилась система аутентификации с внедрением протокола Kerberos и усовершенствована система аудита событий безопасности. Групповые политики (Group Policy), впервые появившиеся в Windows NT 4.0, были значительно расширены и интегрированы с Active Directory, что было особенно важно для централизованного управления настройками безопасности в корпоративных средах.

Эти нововведения в Windows NT и Windows 2000 заложили фундамент для современных подходов к безопасности в операционных системах Windows. Они ознаменовали переход *Microsoft* к созданию более защищенных и управляемых ОС, способных удовлетворить растущие требования к безопасности.

Windows XP (Windows NT 5.1). Windows XP, выпущенная в 2001 г., стала важным этапом в развитии безопасности Windows, улучшив и расширив механизмы защиты. Ключевым нововведением стал Windows Firewall (Internet Connection Firewall), который был включен по умолчанию в Service Pack 2 (SP2) в 2004 г. В SP2 была введена функция Data Execution Prevention (DEP), предотвращающая выполнение кода в областях памяти, предназначенных для данных. Технология DEP затрудняет проведение эксплуатации уязвимостей, например атаки типа переполнения буфера. В 2005 г. в Windows XP Professional x64 Edition была добавлена функция PatchGuard (KPP) для обнаружения несанкционированных изменений структур ядра ОС, что усложняло применение драйверов нарушителями.

Windows Vista (Windows NT 6.0) и Windows 7 (Windows NT 6.1). Windows Vista, выпущенная в 2006 г., представила значительные улучшения в области безопасности. Ключевым нововве-

дением стала система контроля учетных записей пользователей (User Account Control, UAC), предотвращающая несанкционированные изменения системы путем запроса подтверждения при выполнении действий, требующих прав администратора. Также был внедрен BitLocker для полного шифрования диска, что обеспечило защиту данных даже в случае кражи устройства.

В Windows Vista была усилена защита от эксплуатации уязвимостей за счет улучшения функции предотвращения выполнения данных (Data Execution Prevention, DEP); ее сделали активной по умолчанию для всех процессов. Этот механизм предотвращает запуск кода, размещенного в неисполняемых страницах памяти. Тем не менее атакующие продолжали использовать эксплойты, основанные на техниках обхода этих защит. Например, они применяли методы построения цепочек возвратов (Return-Oriented Programming, ROP), которые позволяют выполнять вредоносный код, используя уже существующие в памяти легитимные фрагменты программ, обходя тем самым DEP. Кроме того, в Vista была внедрена технология рандомизации адресного пространства (Address Space Layout Randomization, ASLR), которая случайным образом изменяет расположение ключевых системных компонентов в памяти при каждой загрузке, что значительно усложняет злоумышленникам задачу поиска и эксплуатации уязвимостей. Совместное применение DEP и ASLR значительно повысило уровень защиты Windows Vista, однако появление сложных техник эксплуатации, таких как ROP, показало необходимость дальнейшего развития механизмов защиты в последующих версиях ОС.

Технология Kernel-Mode Code Signing (KMCS) была впервые введена *Microsoft* в Windows Vista, выпущенной в 2007 г. KMCS стала обязательным требованием для 64-битных версий операционной системы, нацеленным на повышение стабильности и безопасности путем обеспечения выполнения только проверенного кода в режиме ядра. Эта технология требует, чтобы все драйверы режима ядра были подписаны цифровым сертификатом, что усложняет внедрение вредоносного кода на уровне ядра. С момента своего по-

явления KMCS постоянно развивалась: *Microsoft* ужесточала требования к подписи драйверов и расширяла применение этой технологии в последующих версиях Windows (EV Code Signing).

Windows 7, выпущенная в 2009 г., продолжила совершенствование функций безопасности. Она представила инструмент управления приложениями AppLocker, позволяющий администраторам задавать правила запуска приложений на основе различных атрибутов, включая цифровые подписи. Это повысило контроль над пользовательским ПО, запускаемым в корпоративных средах.

Windows 8 (Windows NT 6.2) и Windows 8.1 (Windows NT 6.3). Windows 8, выпущенная в 2012 г., представила ряд значимых улучшений в области безопасности. Одним из нововведений стала технология Secure Boot, которая защищает процесс загрузки путем контроля целостности компонентов, загружаемых при старте системы.

В Windows 8 была добавлена поддержка аппаратной технологии Supervisor Mode Execution Prevention (SMEP). Технология SMEP позволяет предотвратить атаки, использующие выполнения кода из пользовательского пространства памяти, когда процессор находится в режиме ядра.

Windows 8 также включила встроенную поддержку технологии виртуализации Hyper-V. Это позволило создавать виртуальные машины на пользовательских системах и управлять ими, что открыло новые возможности для изоляции и безопасности. Еще одним важным дополнением стала технология Device Guard, которая позволяет создавать правила для ограничения запуска приложений и драйверов.

В области защиты ядра ОС Windows 8 сделала значительный шаг вперед, введя первые элементы технологии Kernel Address Space Layout Randomization (KASLR). Она представляет собой механизм рандомизации адресного пространства ядра, который затрудняет проведение атак, основанных на предсказуемости расположения компонентов ядра в памяти. Эта первая реализация KASLR позволяла рандомизировать базовые адреса некоторых компонентов ядра и ознаменовала начало новой эры в защите ядра Windows от эксплойтов и атак на уровне ядра.

В Windows 8.1, выпущенной в 2013 г., продолжилось совершенствование механизмов защиты. В частности, в обновлении Windows 8.1 Update 3 (2014) была представлена технология Control Flow Guard (CFG), обеспечивающая дополнительную защиту от атак, использующих уязвимости в памяти, за счет проверки вызовов и предотвращения перехвата потока исполнения. Кроме того, в Device Guard были добавлены функции защиты кода ядра от модификации, включая запрет на выделение исполняемой памяти из неисполняемого пула (Execute Pool Type), запрет на загрузку драйвера, содержащего секцию кода, в которую разрешена запись (Execute-Write Section), а также другие меры, направленные на усиление целостности и безопасности системных компонентов.

Еще одним дополнением стала технология AppContainer Isolation, позволяющая запускать пользовательский код из недоверенных внешних источников в изолированных средах (Sandbox Environment) с ограниченными правами. Одним из основных сценариев применения AppContainer была изоляция программ из магазина приложений *Microsoft Store* (Windows Store).

Windows 10 (Windows NT 10.0) и Windows 11. Windows 10, выпущенная в 2015 г., значительно расширила и усовершенствовала функции безопасности. Среди них можно выделить встроенный антивирус (Windows Defender), а также ряд механизмов для защиты процессов пользователя: Process mitigation policies, Arbitrary Code Guard (ACG), Control-flow Enforcement Technology (CET) [1].

Microsoft внедрила поддержку технологии Shadow Stack, являющейся частью более широкой инициативы Control-flow Enforcement Technology (CET). Shadow Stack — технология защиты стека вызовов, предотвращающая подмену адресов возврата за счет хранения их копий в защищенном «теневом» стеке.

Поддержка технологии ARM TrustZone в Windows началась с выпуском Windows 10 для ARM-устройств в 2016 г. Технология ARM TrustZone предоставляет возможности запуска защищенной среды выполнения, изолированной от основной ОС. Дальнейшее развитие принципа обеспечения безопасности через

изоляцию компонентов ОС получило свое продолжение в ARM Confidential Compute Architecture (CCA).

В 2020 г. в Windows 10 20H1 была внедрена технология Microsoft eXtended Flow Guard (XFG), которая расширяет возможности Control Flow Guard (CFG). XFG обеспечивает дополнительную защиту от попыток изменения потока управления при эксплуатации уязвимостей. Дальнейшее развитие технологии получило название Extreme Flow Guard (xFG).

В Windows была добавлена поддержка аппаратной технологии SMAP (Supervisor Mode Access Prevention), которая затрудняет эксплуатацию уязвимостей ядра за счет защиты памяти пользовательского пространства от несанкционированного чтения и записи со стороны кода, выполняемого в привилегированном режиме. Дополнительно реализована поддержка технологии SMEP (Supervisor Mode Execution Prevention), которая предотвращает выполнение кода, расположенного на страницах памяти пользовательского режима, при работе в режиме ядра.

Новая технология Hypervisor-Protected Code Integrity (HVCI) использует возможности гипервизора для обеспечения целостности страниц памяти с исполняемым кодом. Это предотвращает внедрение и выполнение вредоносного кода даже в случае компрометации ядра ОС. Новая технология KDP позволяет защитить память ядра от модификации. Компонент под названием Static KDP обеспечивает защиту от записи для секций с данными (.data-секций) загруженных драйверов. Второй компонент Dynamic KDP позволяет выделять память для записи на этапе инициализации без возможностей повторной перезаписи.

Virtualization-Based Security (VBS) использует возможности технологии аппаратной виртуализации для создания изолированных областей памяти (анклавов) в целях защиты данных пользователя и системных сервисов. Технология VBS использует возможности встроенного гипервизора Hyper-V для реализации концепции виртуальных уровней доверия Virtual Trust Levels (VTLs). В настоящее время VBS поддерживает два уровня — VTL0 и VTL1. В рамках VTL0 происходит выполнение основного режима работы ОС (Normal Mode), а уровень VTL1 обеспечивает работу защищенного режима (Secure Mode). Пользова-

тельские приложения, ядро ОС и драйверы работают в рамках режима VTL0 и не могут получить доступ к памяти системных компонентов, работающих на уровне VTL1.

Одним из примеров использования механизма VBS является технология Credential Guard (CG), предназначенная для защиты различных учетных данных, таких как NTLM-хэши и Kerberos Ticket Granting Tickets (TGT). Credential Guard предотвращает атаки типа pass-the-hash и pass-the-ticket путем переноса процесса LSASS.exe из основного режима работы ОС (VTL0) в защищенный режим (VTL1), где он функционирует как процесс-трастлет LSAISO.exe (LSA Isolated). В результате гипервизор Hyper-V гарантирует, что любые процессы и драйверы, работающие в VTL0, не могут получить прямой доступ к памяти этого процесса-трастлета. Изолированный процесс LSA взаимодействует с LSA в основной ОС через удаленные вызовы процедур.

Еще одним важным элементом защиты ядра Windows является HyperGuard, который дополняет PatchGuard и работает в изолированной среде VTL1, недоступной для кода основного ядра (VTL0). Даже при компрометации VTL0 нарушители не могут модифицировать HyperGuard, так как его код и данные вынесены в VTL1. Код HyperGuard находится в SecureKernel.exe.

Можно отметить, что механизмы защиты, внедренные в Windows, в первую очередь являлись ответом на возникающие угрозы безопасности, связанные с драйверами ядра и применяемыми ими техниками [8]. Взаимосвязь между развитием методов атакующих и последующими усовершенствованиями систем безопасности ядра Windows наглядно демонстрирует табл. 1. Эта эволюция отражает реактивный подход *Microsoft* к обеспечению безопасности, при котором каждое новое поколение защитных механизмов разрабатывалось для нейтрализации появившихся векторов атак и устранения недостатков предыдущих версий ОС.

Нарушители используют возможности драйверов для достижения своего сокрытия от штатных средств ОС. Далее будет представлен ретроспективный анализ техник сокрытия (руткит-техник) для целостного понимания возможностей угроз для ОС Windows. Выпущенные новые средства защиты ядра, такие как KPP, HVCI, KDP, несколько изменили картину угроз ядра ОС. В результате часть рассмотренных техник нарушителей потеряла

Таблица 1

Влияние руткитов и вредоносных драйверов на развитие Windows

Название, год и тип	Версии Windows	Техника сокрытия	Принцип действия	Влияние на развитие безопасности Windows
NT Rootkit (1999), руткит	NT 4.0	Перехват системных вызовов	Модифицировал таблицу SSDT для перехвата системных вызовов	Мотивировал создание первой версии PatchGuard (Kernel Patch Protection) для защиты содержимого памяти ядра
FU Rootkit (2005), руткит	2000, XP	DKOM (Direct Kernel Object Manipulation)	Модифицировал структуры объектов ядра для скрытия процессов и портов	Развитие алгоритмов обнаружения модификаций в структурах объектов ядра
Hikit (2011), руткит	XP, Vista, 7	Перехват системных вызовов	Скрывал свое присутствие, перехватывая системные вызовы в обход PatchGuard	Совершенствование механизмов мониторинга системных вызовов и поведенческого анализа
Necurs (2012–2019), руткит	XP, Vista, 7, 8, 10	Комбинированные техники	Использовал различные техники обхода PatchGuard	Развитие комплексных технологий обнаружения и нейтрализации руткитов
RobbinHood (2019), вредоносный драйвер	7, 8, 10	Эксплуатация легитимных драйверов, BYOVD	Использовал уязвимый подписанный драйвер Gigabyte для отключения механизма защиты ядра (Driver Signature Enforcement)	Усиление политик подписи драйверов, улучшение защиты от атак с использованием легитимных уязвимых драйверов

Окончание табл. 1

Название, год и тип	Версии Windows	Техника сокрытия	Принцип действия	Влияние на развитие безопасности Windows
PPLdump (2021), вредоносный драйвер	10, 11	Модификация PPL (Protected Process Light)	Отключение защиты PPL для получения доступа к содержимому памяти защищенных процессов	Усиление механизмов защиты PPL, улучшение мониторинга попыток модификации защищенных процессов
BURNTCIGAR (2022), вредоносный драйвер	10, 11	Отключение EDR (Endpoint Detection and Response)	Использовался для отключения EDR-решений перед атаками программ-вымогателей (ransomware)	Совершенствование защиты EDR-решений, улучшение механизмов предотвращения отключения средств защиты
Werkernel.sys CVE-2024-26169 (2024)	10, 11	Эксплуатация легитимных драйверов, Bring Your Own Vulnerable Driver (BYOVD)	Эксплуатация уязвимости в драйвере для повышения привилегий	Усиление политик подписи драйверов, Vulnerable Driver Blocklist
CSC.sys CVE-2024-26229 (2024)				
TeamViewer CVE-2024-7479/7481 (2024)				
CrazyHunter (2025)				

свою актуальность, однако технические принципы и алгоритмы атак, лежащие в их основе, остаются применимыми и для новых атак на другие системные структуры ОС Windows. Примеры таких атак и возможные способы противодействия им с помощью гипервизора будут представлены в разделе 2.

Глава 2
АНАЛИЗ СУЩЕСТВУЮЩИХ ТЕХНИК СОКРЫТИЯ ВРЕДОНОСНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Специалисты по информационной безопасности используют индикаторы компрометации (Indicator of Compromise, IoC) для своевременного выявления фактов заражения и несанкционированного доступа к системе. К таким индикаторам относятся подозрительные файлы, аномальные процессы, неизвестные службы и драйверы, сетевые подключения, задачи планировщика и другие сущности, появление которых является нехарактерным для штатной работы ОС. Для получения информации об этих сущностях могут быть использованы различные функции ОС: встроенные или штатные функции (например, WinAPI-функции), а также другие техники, использующие недокументированные возможности ОС.

Если нарушители не используют механизмы сокрытия, то выявить подозрительный объект не составляет труда. Это ускоряет реагирование на инцидент, позволяя оперативно принимать контрмеры для защиты системы.

Нарушители стремятся обеспечить длительное и скрытное присутствие ВПО на зараженном компьютере. Это привело к появлению отдельного класса техник, предназначенных для сокрытия объектов от обнаружения, которые развивают низкоуровневые вредоносные технологии [9, 10].

Существующие техники сокрытия ВПО можно условно разделить на две основные группы и соотнести с матрицей MITRE ATT&CK, в которой систематизированы тактики и техники злоумышленников:

1) стеганографические техники (T1036 Masquerading);

2) системные техники, или руткит-техники (T1014 Rootkits).

Системные техники, в свою очередь, можно разделить на две подгруппы:

1) техники, работающие «внутри» ОС (изменение пути выполнения и модификация структур объектов);

2) техники, работающие «вне» ОС (на основе различных аппаратных технологий).

Схема систематизации техник сокрытия вредоносного ПО наглядно демонстрирует эту иерархическую классификацию (рис. 2).

Рис. 2. Схема систематизации техник сокрытия

2.1. СТЕГАНОГРАФИЧЕСКИЕ ТЕХНИКИ СОКРЫТИЯ

Стеганографические техники применяются нарушителями для маскировки ВПО или его активности, представляя их под видом легитимных объектов. Эта техника, относящаяся к типу T1036 «Masquerading» матрицы MITRE и к типу 0 по классификации J. Rutkowska [11], направлена на сокрытие самого факта присутствия ВПО в ОС путем введения пользователя или эк-

сперта в заблуждение относительно истинного назначения объекта.

Основные примеры стеганографического сокрытия включают:

- использование имен и путей процессов и файлов, сходных с системными или легитимными компонентами;
- изменение метаданных файлов и подделку цифровых подписей.

Эти приемы создают иллюзию, что вредоносные элементы происходят из доверенных источников, затрудняя их идентификацию как угрозы.

Примеры ВПО, использующего стеганографические техники:

- OSX.WindTail [12]: использует подписание кода отозванными сертификатами (техника T1036.001);
- ComRAT: разработанный группой Turla бэкдор, использующий имя WSqmCons, ассоциированное с легитимным сервисом Windows SQM Consolidator (техника T1036.004).

Можно выделить следующие особенности стеганографических механизмов:

1) данные механизмы не вмешиваются в работу ОС и не требуют прав администратора для работы своих функций по сокрытию;

2) обладают переносимостью на различные версии ОС Windows и оборудования.

Важно отметить, что повышенную опасность представляет синтез стеганографических механизмов с другими техниками сокрытия.

2.2. СОКРЫТИЕ ОБЪЕКТОВ «ВНУТРИ» ОС: ИЗМЕНЕНИЕ ПУТИ ВЫПОЛНЕНИЯ И СТРУКТУР ПАМЯТИ

Для углубленного анализа техник сокрытия и разработки новых способов обнаружения рассмотрим, как происходит работа ОС по сбору информации об объектах (структуры и списки в памяти ядра ОС). После запуска процесса и потока или регистрации драйвера и службы в памяти ОС создается структура, однозначно соответствующая объекту. Структуры объектов находятся в двунаправленных (или двусвязных) списках: для про-

цессов, потоков и драйверов структуры находятся в памяти ядра ОС, а для сервисов — в пользовательской области памяти процесса services.exe (рис. 3).

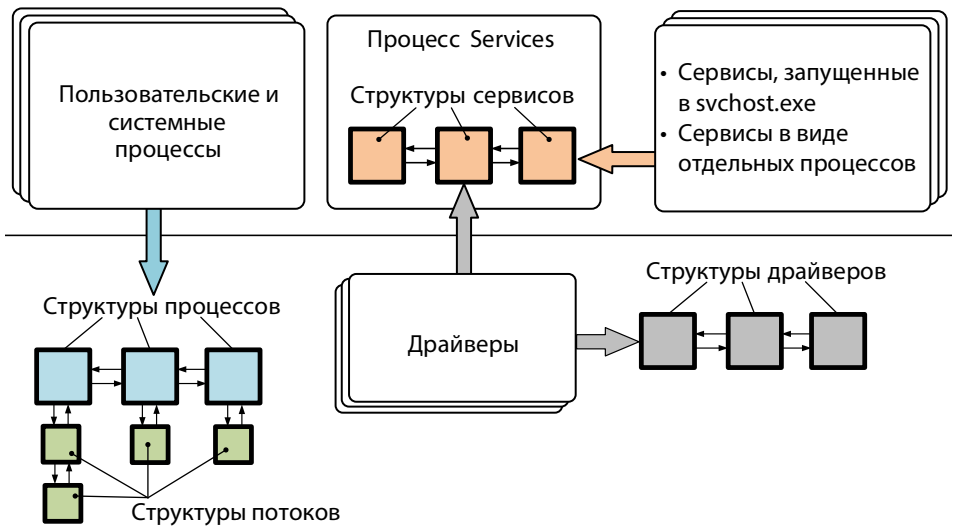


Рис. 3. Расположение в памяти исполнимых объектов и соответствующих им структур в памяти

Для получения большинства списков объектов, таких как запущенные процессы или загруженные драйверы, ОС Windows предоставляет WinAPI-функции. Данные функции представляют собой системные вызовы (syscall), которые формируют буфер с результатом на основе данных, полученных путем прохода по списку структур в памяти.

Системные техники скрывают объекты нарушителей в зараженной ОС путем воздействия на компоненты ОС. При этом выделяют два вида таких воздействий или манипуляций: изменение пути выполнения и изменение объектов в памяти. Программные средства, реализующие системные техники сокрытия (руткит-техники), называются *руткитами* [9].

Системные техники сокрытия (руткит-техники) воздействуют на работу функций ОС, которые формируют списки объектов, например списки работающих процессов и загруженных драйверов. Такие манипуляции нарушителей приводят к изменению результата, возвращаемого системными функциями ОС: информация

о скрываемом объекте удаляется из списка, при этом дальнейшее функционирование объекта и ОС не нарушается. Системные техники работают по принципу: «объект не виден, значит, его и нет». В результате штатные средства ОС Windows не позволяют обнаружить наличие скрытого объекта.

Эксперты по кибербезопасности разработали ряд подходов к обнаружению скрытых объектов. Один из наиболее популярных подходов (Cross-View based Rootkit Detection) заключается в формировании нескольких *независимых* списков объектов с использованием различных функций ОС и других низкоуровневых механизмов ОС. Последующее сравнение содержимого этих списков позволяет выявить несоответствия, которые могут указывать на присутствие скрытого объекта. В ответ на это нарушители постоянно совершенствуют свои техники сокрытия, стремясь удалить следы скрываемого объекта из всех возможных системных списков и структур данных или иным образом нарушить работу средств обнаружения.

Такое противостояние постоянно усложняет задачу обнаружения скрытых объектов, мотивируя исследовательский поиск и разработку новых алгоритмов, методов и средств защиты. Разработчики Windows ОС выпускают новые защитные механизмы для блокировки работы отдельных руткит-техник, делая целые группы руткит-техник неработоспособными в новых версиях ОС. При этом важно отметить, что алгоритмы руткит-техник могут быть использованы для решения различных задач информационной безопасности, не связанных напрямую с задачей сокрытия объектов в ОС.

В результате работы системных техник сокрытия функция ОС возвращает измененный результат. Системные техники сокрытия, работающие «внутри» ОС, можно условно разделить на две группы:

1) изменение пути выполнения;

2) изменение структур объектов.

Нарушители могут воздействовать на работу системных функций ОС как на уровне пользователя (user level), так и в привилегированном режиме или режиме ядра (kernel level). Далее будет представлен анализ техник на основе модификации структур объектов в памяти и изменения пути выполнения.

2.2.1. Изменение структур объектов

Изменение структур объектов — это техника, при которой нарушители модифицируют содержимое структур в памяти таким образом, что системная функция, генерирующая список объектов, возвращает его без скрываемого элемента. Такие техники, использующие модификацию структур объектов в памяти, получили название Direct Kernel Object Manipulation (DKOM). Изменяемые нарушителями структуры могут находиться как в привилегированной памяти (память ядра ОС), так и в пользовательской области памяти [13]. При этом нарушители могут изменять как сами структуры скрываемых объектов, так и другие структуры, задействованные в работе функций и ОС. Изменение в памяти не требует внедрения новых обработчиков в систему. В большинстве случаев нарушители модифицируют поля структуры объектов и глобальные переменные-флаги, которые находятся в динамически выделенных областях памяти. По классификации J. Rutkowska, такие техники относятся ко второму типу [11].

Одним из первых примеров, демонстрирующих возможности сокрытия путем модификации динамических структур в памяти, был руткит Fu, представленный Джейми Батлером (Jamie Butler) в 2004 г. на конференции Black Hat USA. Данный прототип позволял скрывать запущенные процессы и загруженные драйверы путем модификации связей в соответствующих двунаправленных списках PsActiveProcessHead и PsLoadedModuleList, а также повышать привилегии процессов.

2.2.2. Изменение пути выполнения

Техники первой группы (control flow modifying, hooking, intercepting) модифицируют работу системной функции путем внедрения собственного обработчика в целях изменения потока выполнения системного вызова. Перехват контроля управления заключается в установке вредоносного посредника-диспетчера, который в реальном времени получает управление при каждом вызове функций и путем изменения входных, выходных буферов, а также возвращаемого статуса обеспечивает сокрытие объекта. В результате функция возвращает скомпрометированный результат. Встраиваемый обработчик должен присутствовать в системе на протяжении всего периода сокрытия.

Установленный обработчик может реализовывать различные алгоритмы в зависимости от перехваченной функции и задач сокрытия: удалять информацию о скрываемом объекте, формировать новый (измененный) список объектов или возвращать статус об отсутствии объекта. При этом встраиваемые обработчики могут получать управление как до выполнения функции (pre-hooking / pre-processing), так и после ее завершения (post-hooking / post-processing), корректируя результат работы. Изменение пути выполнения может быть реализовано на различных этапах потока выполнения системного вызова.

По классификации J. Rutkowska эти техники относятся к первому типу [11]. Как правило, они ориентированы на модификацию исполняемой памяти, управляющих регистров, таблиц и адресов памяти, задействованных при системных вызовах.

Одним из первых примеров руткита, демонстрирующих данную технику, был драйвер NTRootkit (_root_.sys), представленный экспертом Грегом Хоглундом (Greg Hoglund) в журнале *Phrack* в 1999 г. Реализованный прототип скрывал ветки и ключи реестра с использованием модификации таблицы SSDT.

Модификация потока выполнения основана на внесении изменений в различные объекты, участвующие в управлении выполнением кода ОС. К таким объектам относятся:

- регистры MSR — модификация регистров MSR IA32_SYSENTER_EIP и IA32_LSTAR, IA32_CSTAR. (техника MSR hooking);
- исполнимый код — встраивание перехватчика в различные диспетчерские функции и системные функции модулей ядра. (техника code patching);
- содержимое глобальных таблиц — таблица дескрипторов прерываний (IDT), таблица системных вызовов (SSDT) (System Service Hijacking);
- содержимое структур в памяти, хранящих указатели на функции-обработчики, задействованные в потоке управления. В случае объектов ядра эта техника называется Kernel Object Hooking (KOH) [14] и изменяет функциональные указатели внутри структур без изменения глобальных таблиц.

Диапазон применяемых нарушителем инструментов может быть расширен: например, для сокрытия файлов на диске нарушитель может установить файловый фильтр, а также модифицировать содержимое структур IRP (I/O Request Packets), включая адреса функций IoCompleteRequest [15].

Схема, представленная на рис. 4, иллюстрирует процесс вызова системных функций в ОС Windows, а также показывает возможные точки модификации руткитами для сокрытия их присутствия.

Представленная схема демонстрирует возможности изменения потока управления на различных этапах работы потока управления при обработке системного вызова.

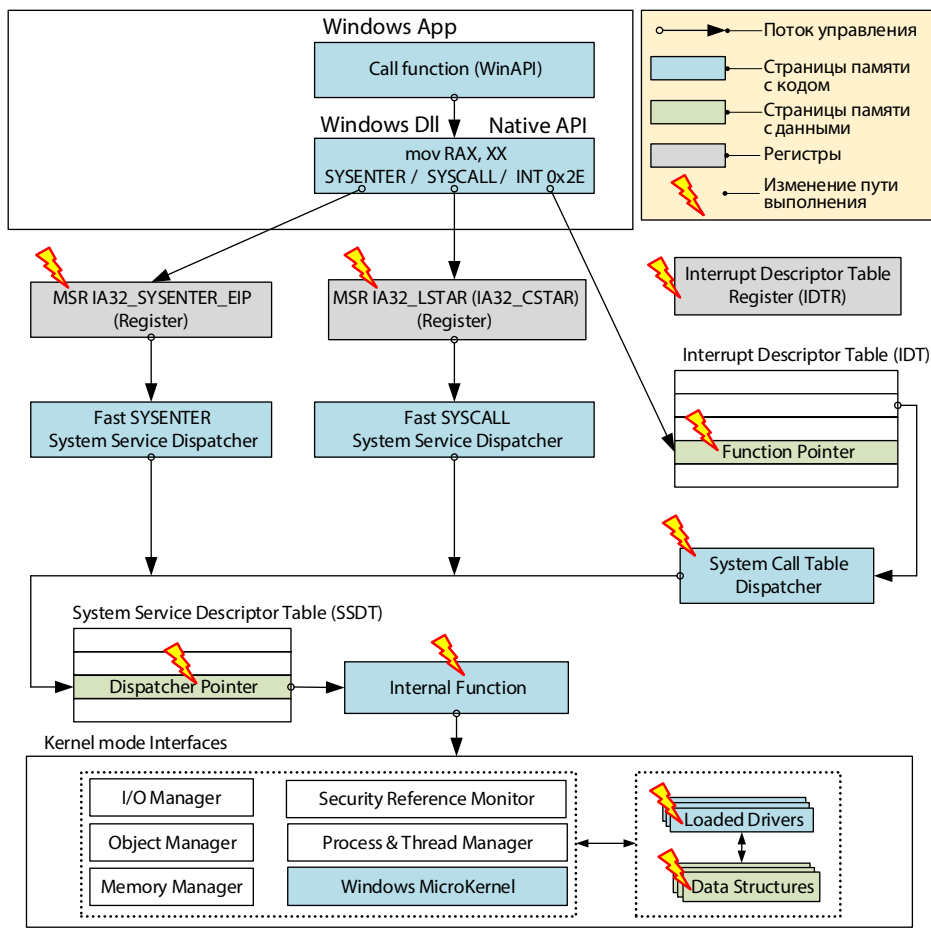


Рис. 4. Обобщенный процесс вызова системных функций в Windows ОС и возможные точки для встраивания перехватчика

Современные защитные механизмы, такие как PatchGuard и HVCI, значительно усложняют использование методов перехвата управления, делая их менее актуальными. В то же время техники изменения структур памяти остаются более скрытными и сложными для обнаружения.

Ниже перечислены основные ограничения действий техник сокрытия, основанных на изменении пути выполнения:

- *средства контроля целостности регионов памяти*. Современные системы защиты включают инструменты для контроля целостности отдельных системных областей памяти, таких как IDT и SSDT. Примером является KPP в Windows, который значительно усложняет работу руткитов, пытающихся модифицировать эти области для сокрытия своей активности;
- *средства контроля целостности исполняемого кода*. Существуют инструменты, которые предотвращают изменение страниц памяти с исполняемым кодом. Например, Device Guard с функцией HVCI предотвращает установку перехватчиков функций в ядре [16]. Это существенно ограничивает возможности руткитов по изменению пути выполнения системных функций;
- *потеря производительности при работе посредника-диспетчера*. Внедрение промежуточных обработчиков приводит к замедлению системы и может служить индикатором наличия такого диспетчера и способствовать его обнаружению [11];
- *необходимость сокрытия самого встраиваемого обработчика*. Чтобы избежать обнаружения, нарушителям приходится дополнительно скрывать внедренные перехватчики, что усложняет их реализацию и повышает риск выявления.

В современных версиях Windows использование руткитов на основе изменения пути выполнения становится менее актуальным благодаря штатным средствам защиты. Установка драйверов-фильтров файловой системы и реестра в некоторых случаях все еще встречается на практике [17], однако данные техники сокрытия также могут быть обнаружены из-за внесенных изменений.

Ретроспективный анализ показывает, что наиболее сложно обнаруживаемыми и, следовательно, наиболее опасными являются

руткиты, использующие модификацию содержимого структур памяти с данными (Direct Kernel Object Manipulation, DKOM) [18–22]. В дальнейшем основное внимание уделяется обнаружению объектов, скрытых с использованием DKOM-техник.

2.3. ГИПЕРВИЗОР-РУТКИТЫ

Появление новых технологий в компьютерном мире зачастую порождает новые возможности как для разработчиков средств защиты информации, так и для нарушителей, которые стремятся достичь долговременного и скрытого присутствия на целевом компьютере. К таким технологиям можно отнести технологию аппаратной виртуализации Intel VT-x компании *Intel* и технологию AMD-V компании *AMD*. Далее без ограничения общности будет представлен анализ для случая технологии аппаратной виртуализации VT-x процессоров семейства *Intel*.

Технология аппаратной виртуализации обеспечивает выполнение программного кода (обработчика гипервизора) в режиме с более высокими привилегиями, чем режим ядра операционной системы. Данный режим получил название VMX root, его также называют режимом монитора виртуальных машин (МВМ), или режимом «-1» кольца (как продолжение обратной нумерации колец защиты архитектуры *Intel* в сторону повышения привилегий). *Гипервизором* называется код, работающий в режиме VMX root (рис. 5). Операционная система продолжает свою работу в режиме VMX non-root, который также называется гостевым режимом (guest mode). В результате гипервизор получает прямой доступ к системной памяти и регистрам процессора, а гостевая ОС выполняется изолированно под контролем гипервизора.

Технология аппаратной виртуализации позволяет диспетчеру гипервизора перехватывать широкий спектр событий ОС, включая обращения к системным регистрам, памяти и оборудованию. При этом в гипервизоре определяется механизм «перехвата» (interception, VM-Exit и т.п.), который срабатывает всякий раз, когда гостевая ОС пытается выполнить перехватываемые операции: безусловно перехватываемые инструкции и операции, настраиваемые при загрузке гипервизора. После перехвата гипер-

визор получает управление и далее может либо предоставить ей требуемый ресурс либо скорректировать или заблокировать соответствующее действие. Гипервизор может быть как установлен на этапе загрузки системы, так и внедрен в уже работающую ОС (для этого потребуются привилегии драйвера ядра).

Прототип гипервизора, использующего возможности UEFI для загрузки, был впервые представлен на конференции Black Hat в 2008 г. экспертом Доном Бейли (Don Bailey) [23]. Возможности таких гипервизоров обсуждались в рамках конференций Hexacon и OffensiveCon на семинарах по разработке гипервизоров в системах защиты информации Hypervisor Development for Security Analysis, эксперт Сатоши Танда (Satoshi Tanda) в 2024 г. [24].

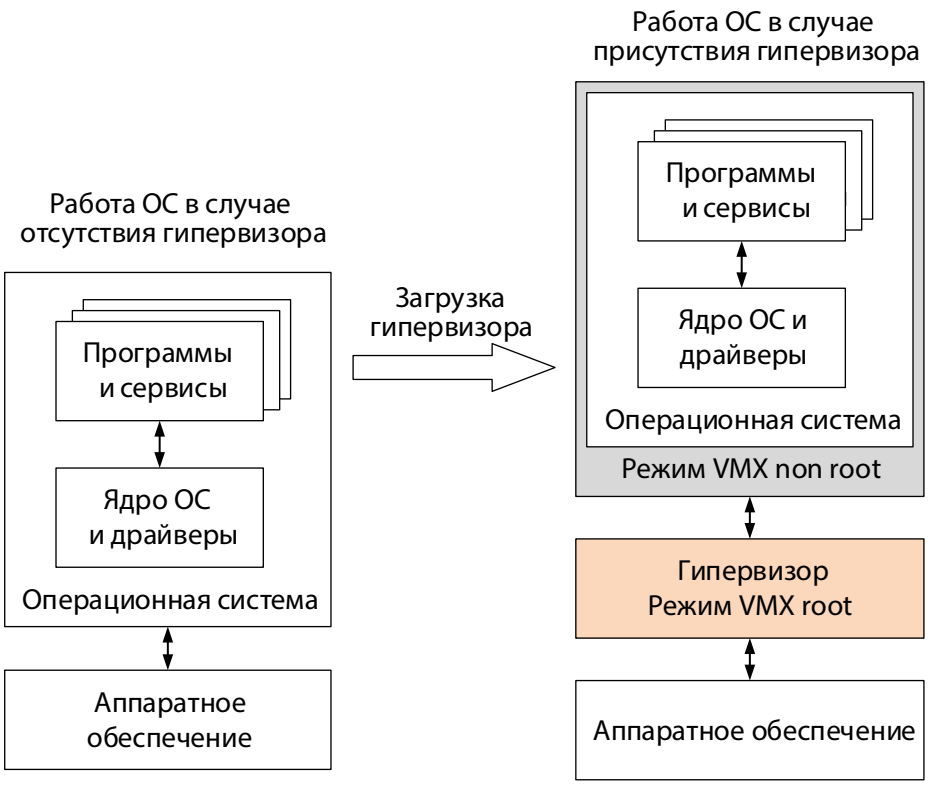


Рис. 5. Работа ОС в случаях отсутствия и присутствия гипервизора

Возможности гипервизоров активно используются в различных системах защиты информации, например в системах класса «песочница» (Sandbox), для анализа ВПО и антивирусных

системах защиты информации. Гипервизор может быть использован как самостоятельное средство или в составе программного продукта, например, в целях защиты областей памяти драйвера от несанкционированного доступа со стороны нарушителей.

Для нарушителей гипервизоры открывают новые возможности: во-первых, они обеспечивают сокрытие от антивирусных систем, работающих в гостевой ОС, поскольку отсутствуют штатные способы обнаружения гипервизоров; во-вторых, гипервизоры могут контролировать работу ОС и содержимое памяти доступной для ОС, препятствовать своему обнаружению, защищать ВПО в ОС, а также блокировать работу средств защиты в ОС. Наконец, удаление таких гипервизоров чрезвычайно сложно; если гипервизор загружается с помощью BIOS/UEFI, то такое удаление требует модификации соответствующих микропрограмм оборудования и других низкоуровневых компонентов системы.

Blue Pill и SubVirt являются примерами концептуальных гипервизор-руткитов (Hypervisor-based Rootkit, Virtual Machine based Rootkits (VMBR)), демонстрирующих возможности скрытого гипервизора для компрометации систем.

Blue Pill — один из первых прототипов гипервизор-руткитов, созданный исследователем Джоанной Рутковской (J. Rutkowska). Blue Pill использует возможности аппаратной виртуализации AMD-V для перехвата управления операционной системой и сокрытия своего присутствия [25].

SubVirt разработан исследователями из *Microsoft* и Университета Мичигана. SubVirt демонстрирует возможность создания гипервизор-руткитов, которые могут заразить систему, оставаясь незамеченными. SubVirt ориентирован на Intel VT-x и требует перезагрузки системы [26].

Гипервизор-руткиты представляют собой одну из наиболее сложных угроз для современных информационных систем, требующую продуманных и комплексных решений в области информационной безопасности.

2.4. SMM-РУТКИТЫ

Режим системного управления, или System Management Mode (SMM), — это режим работы процессора, предназначенный

для выполнения задач низкоуровневого управления системой. Он используется для обработки различных системных событий, управления питанием и аппаратными средствами, защиты от перегрева, например, при переводе системы в состояние пониженного энергопотребления, а также для эмуляции устройств.

Переключение в режим SMM происходит при возникновении прерывания SMI (System Management Interrupt). После получения прерывания SMI процессор передает управление обработчику SMM. При входе в SMM процессор сохраняет состояние регистров и отключает прерывания. В результате обработчик SMM выполняет код с привилегиями, большими, чем ОС и гипервизор, оказываясь вне их контроля. В терминах архитектуры x86_64 режим SMM можно условно назвать «кольцо-2», так как он обладает бóльшими привилегиями, чем гипервизор («кольцо-1»). Для возврата из режима SMM применяется инструкция RSM, после чего процессор восстанавливает предыдущее состояние и продолжает работу в исходном режиме: реальном, защищенном или в режиме гипервизора.

Особую угрозу представляет собой ВПО, реализованное в виде обработчика режима SMM. Благодаря изолированной памяти (SMRAM) и привилегированному доступу к аппаратным ресурсам нарушители получают новые возможности для скрытного вмешательства в работу системы. Соответствующий вредоносный обработчик получил название SMM-rootkit, System Management Mode based rootkit, SMBR.

Исследователи безопасности активно изучают возможности создания SMM-руткитов и способы их внедрения.

Одна из первых работ по исследованию возможностей SMM-руткитов была опубликована в 2008 г. автором Шоном Эмблтоном (Shawn Embleton) под названием «SMM rootkits: a new breed of OS independent malware» [27], которую он подготовил совместно с Шерри Спаркс (Sherri Sparks) и Клиффом Цзоу (Cliff C. Zou). В статье авторы продемонстрировали возможности уязвимости BIOS для внедрения кода в режим SMM, ранее исследованной экспертом Лоиком Дюфло (Loïc Duflot). Авторы показали, как внедренный SMM-обработчик может быть использован для создания клавиатурного шпиона, который передает перехваченные

нажатия клавиш на удаленный сервер, а также принимает управляющие команды от внешнего нарушителя.

В 2014 г. была представлена работа «The SMM Rootkit Revisited: Fun with USB» авторов Джошуа Шиффман (Joshua Schiffman), Дэвид Каплан (David Kaplan), описывающая SMM-руткит [28]. Авторский прототип руткита перехватывает USB-события на уровне хост-контроллера USB с помощью обработчика прерываний SMI до того, как USB-события будут обработаны ядром ОС. Представленный в статье прототип позволяет не только играть роль клавиатурного шпиона, но и подменять нажатые пользователем клавиши, эмулировать нажатия клавиш.

В 2022 г. была опубликована статья «The design of the simple SMM rootkit» [29], которая описывает технику внедрения SMM-обработчика путем модификации содержимого UEFI. Для разработки модуля авторы выбрали фреймворк UEFI Development Kit (UDK). В статье описываются возможности вредоносного SMM-обработчика, такие как чтение и модификации системной памяти в целях внедрения перехватчиков системных вызовов. Авторы отмечают, что благодаря внедрению кода с UEFI такой SMM-руткит продолжает работать после переустановки ОС, что затрудняет устранение аналогичной угрозы на практике.

Обнаружение SMM-руткитов является сложной задачей, однако существуют некоторые подходы:

- анализ микропрограмм: регулярная проверка и обновление BIOS/UEFI могут помочь предотвратить внедрение вредоносного кода;
- использование специализированных инструментов для анализа низкоуровневых системных компонентов, которые могут выявить аномалии, указывающие на присутствие SMM-руткита;
- мониторинг поведения системы: необычное поведение системы (например, снижение производительности или частые сбои) может указывать на присутствие руткита.

Защита от SMM-руткитов требует комплексного подхода [30]. Механизмы аппаратной защиты, такие как Intel Boot Guard и AMD Secure Boot, позволяют предотвратить несанкционированное изменение микропрограмм, однако они, в свою очередь, также содержат уязвимости.

2.5. AMT/ME-РУТКИТЫ

Еще одной аппаратной технологией, которой могут воспользоваться нарушители, является технология Intel Active Management Technology (AMT). Она была создана для обеспечения возможностей удаленного администрирования компьютерными системами даже в условиях отсутствия ОС, наличия различных ошибок, препятствующих включению компьютера. Для работы технологии AMT необходимо подключение питания и предоставление сетевого доступа. Технология AMT реализована на базе Intel Management Engine (ME) — микроконтроллера, встроенного в чипсет материнской платы. Технология ME работает на основе независимого микроконтроллера и специализированной прошивки на базе ОС Linux.

Компрометация технологий AMT и ME через эксплуатацию уязвимостей и внедрение вредоносного кода представляет серьезную угрозу информационной безопасности. В результате такой атаки нарушители могут получить широкие возможности по несанкционированному управлению системой, оставаясь скрытыми для популярных средств защиты, работающих на уровне ОС.

Исследователи по безопасности активно изучают внутреннее устройство этой технологии и ее недокументированные возможности. В 2009 г. была представлена одна из первых работ, описывающих угрозы компрометации Intel AMT и ME, — «Introducing Ring-3 Rootkits» авторов Александра Терешкина (Alexander Tereshkin) и Рафала Войтчука (Rafal Wojtczuk). В этой работе впервые было показано, как можно использовать уязвимости ME для создания руткитов уровня «кольцо-3» (ring-3), которые работают вне контроля ОС и гипервизора.

В 2017 г. была обнаружена уязвимость CVE-2017-5689 в технологии Intel AMT. Эта уязвимость позволяла удаленно получать привилегии администратора через веб-интерфейс в обход механизма аутентификации. Исследователи Дмитрий Евдокимов (Dmitriy Evdokimov), Александр Ермолов (Alexander Ermolov) и Максим Малютин (Maksim Malyutin) в рамках доклада «Intel AMT. Stealth Breakthrough» продемонстрировали возможности этой уязвимости.

В 2022 г. эксперты по безопасности компании *Positive Technologies* Марк Ермолов (Mark Ermolov), Дмитрий Скляров (Dmitry Sklyarov) и Максим Горячий (Maxim Goryachy) представили работу «Недокументированные инструкции x86 для управления CPU на уровне микроархитектуры в современных процессорах Intel» [31]. Авторы обнаружили две недокументированные инструкции и специальный отладочный режим для доступа к данным микроархитектуры процессора, включая микрокод. Авторы подчеркивают опасность таких возможностей с точки зрения безопасности, так как они могут быть использованы нарушителями.

Руткиты на базе технологий Intel AMT и ME представляют собой одну из самых сложных угроз современной кибербезопасности. Борьба с такими угрозами требует комплексного подхода, включающего использование аппаратных механизмов защиты, регулярное обновление прошивок и мониторинг сетевой активности.

Глава 3
ВОЗМОЖНОСТИ DKOM-ТЕХНИК: ОТ СОКРЫТИЯ ОБЪЕКТОВ ДО ПРОТИВОДЕЙСТВИЯ EDR

Далее проведем анализ техник Direct Kernel Object Manipulation (DKOM), которые используются нарушителями для сокрытия объектов, а также для нарушения работы средств EDR/AV.

Техника DKOM позволяет скрывать объект путем изменения результата, возвращаемого функциями WinAPI, за счет модификации структур объектов в памяти, к которым обращаются эти функции. Особенность DKOM-сокрытия заключается в том, что изменения в структурах памяти приводят к изменению поведения WinAPI-функций в интересах нарушителя. При этом внедрение диспетчера-посредника для реализации логики DKOM-сокрытия не требуется, что делает подход более скрытным и сложным для обнаружения по сравнению с техниками на основе изменения пути выполнения.

Нарушители могут модифицировать структуры объектов как в памяти пользовательского режима, так и в памяти ядра. Для защиты некоторых типов структур в ядре разработчики Windows ОС выпустили ряд встроенных защитных инструментов, таких как KPP и HyperGuard, которые обеспечивают контроль целостности структур. В результате применения таких средств защиты атаки нарушителей на эти структуры потеряли смысл, однако практические исследования показали, что принципы, заложенные в техники DKOM, оказались пригодными для проведения атак на другие структуры, целостность которых не контролируется штатными инструментами защиты.

В настоящее время эксперты по безопасности отмечают новые угрозы применения DKOM-техник для нарушения работы средств защиты конечных узлов (EDR) и антивирусных решений, например, путем отключения ETW- [32] и WMI-подсистем [33], а также путем преднамеренного понижения привилегий для модулей защиты [34]. Аналогичные примеры приведены в работах [35, 36].

Такие модификации структур данных в целях их сокрытия обладают рядом преимуществ по сравнению с техниками на основе изменения пути исполнения. Важно отметить, что такие модификации структур, находящихся в памяти ядра и пользовательского режима, сложно отследить на практике из-за отсутствия штатных средств мониторинга и контроля обращений к памяти. Одним из возможных решений является сохранение информации об объектах с использованием дублирующих списков; далее при обнаружении несоответствий в исходном системном списке и дублирующем происходит срабатывание средства защиты. Аналогичный алгоритм заложен в основу KPP — штатного средства защиты ядра Windows.

Проблема обнаружения усугубляется тем, что ВПО может вносить изменения лишь на очень короткое время, например, для изменений привилегий процесса для сбора парольно-адресной информации, а после восстанавливать исходные значения. В данном случае, если модификация и восстановление структур оказались в интервале времени между двумя последовательными циклами проверки, то средство защиты не сможет выявить атаку. Для обнаружения таких атак возможно использовать гипервизорные решения на базе технологии аппаратной виртуализации.

В ОС Windows информация о работающих процессах, потоках, драйверах и службах хранится в двунаправленных списках, находящихся в памяти (см. рис. 3), что позволяет операционной системе управлять ими, изменять их состояние и отслеживать выполнение. Эти списки структур обеспечивают рациональное управление и отслеживание состояния различных объектов системы. Штатные средства получают информацию о работающих объектах путем прохода по соответствующим спискам. Для процессов, потоков и драйверов такие списки находятся в памяти ядра, в то время как информация о службах (сервисах) извлекается из списка, находящегося в пользовательской области памяти процесса services.exe.

Нарушители могут несанкционированно, в обход функций и механизмов ОС, изменять содержимое этих структур в памяти для достижения вредоносных целей. Техники сокрытия DKOM

основаны на таком изменении связей между структурами, при котором структура скрываемого объекта удаляется из списка (unlinked structure). В результате функция ОС при проходе по обновленному списку пропускает удаленную структуру и возвращает результат, в котором отсутствует скрываемый объект. В ряде случаев может быть также изменено содержимое полей удаленной структуры.

3.1. СОКРЫТИЕ ПРОЦЕССОВ, ПОТОКОВ, ДРАЙВЕРОВ И СЕРВИСОВ (СЛУЖБ) С ПОМОЩЬЮ DKOM

После запуска процесса ОС Windows создает структуру EPROCESS, однозначно соответствующую этому процессу, и добавляет ее в двусвязный список. После завершения процесса его структура удаляется из списка, а память, выделенная для этой структуры, освобождается.

Штатные средства ОС, такие как taskmgr.exe, tasklist, командлет Get-Process, для получения списка процессов используют вызов функции NtQuerySystemInformation (ZwQuery SystemInformation). Данная функция использует проход по списку структур EPROCESS, который образован полем ActiveProcessLinks. Работа функции основана на последовательном прохождении списка структур EPROCESS, используя значения полей Flink и Blink. Эти указатели находятся в структуре ActiveProcessLinks, входящей в состав каждой структуры EPROCESS. Flink указывает на следующую структуру в списке, Blink — на предыдущую структуру.

На основе такого поведения функции по получению списка процессов существует следующий способ сокрытия [18]. Структура скрываемого объекта удаляется из списка путем изменения связей в списке с помощью модификации значений указателей соседних структур. У двух соседних структур EPROCESS, между которыми находится структура скрываемого процесса Target, необходимо изменить значения Flink и Blink таким образом, чтобы элемент «удалился из списка». Для скрываемого объекта значения Flink и Blink устанавливаются так, чтобы они указывали сами на себя.

Алгоритм сокрытия процесса включает несколько этапов (рис. 6).

1. С помощью функции IoGetCurrentProcess драйвера ядра получить указатель на структуру EPROCESS текущего процесса.

2. Путем обхода списка структур EPROCESS через поля Flink и Blink найти нужную структуру.

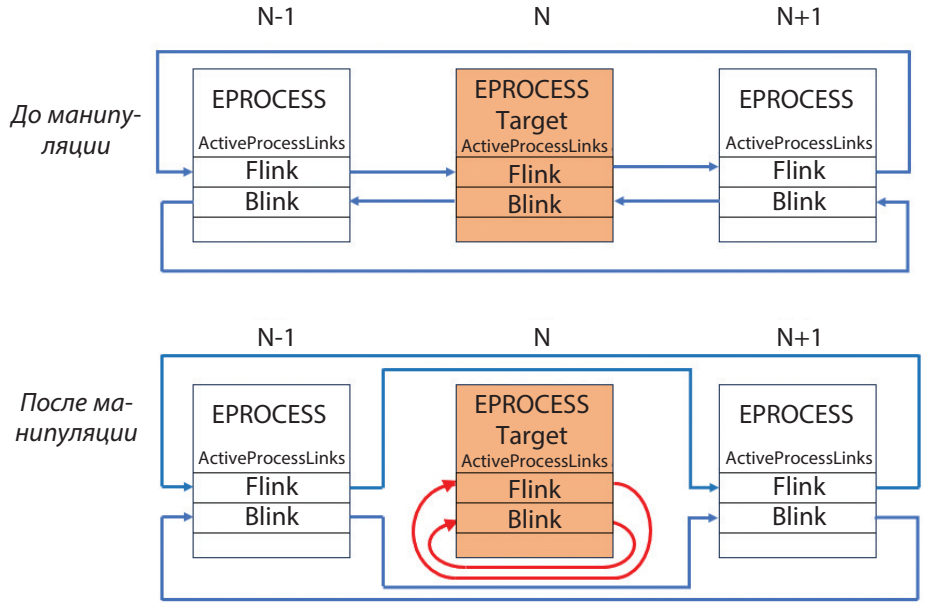


Рис. 6. Схема способа сокрытия процесса

3. Изменить значения указателей Flink и Blink соседних структур путем перенаправления ссылок соседних структур (N-1 и N+1) в обход скрываемого процесса (N) таким образом, чтобы исключить структуру скрываемого процесса из списка.

4. Для затруднения последующего анализа нарушители могут вносить и другие изменения в скрываемую структуру: перезаписать значение уникального идентификатора процесса (Process Identifier, PID), имя файла исполняемого процесса (ImageName), полный путь к исполняемому файлу процесса (ImageFileName).

После выполнения этих манипуляций вызов функции NtQuerySystemInformation вернет результат, в котором не будет информации о скрываемом процессе. Как было отмечено ранее, данная техника позволяет скрывать процессы без нарушения ра-

боты ОС и остается при этом незаметной для пользователя. Однако в современных версиях Windows было добавлено штатное средство контроля целостности этих структур — PatchGuard, которое обнаруживает внесенные изменения и останавливает работу системы для предотвращения дальнейших действий нарушителя.

Способ сокрытия работающего потока аналогичен способу сокрытия процессов. Для сокрытия необходимо, чтобы структура потока была удалена из соответствующего двусвязного списка структур потоков. Схема, иллюстрирующая процесс сокрытия потоков (ETHREAD1), в котором изменяются указатели в структуре списков, чтобы скрываемый поток исключался из видимости системных инструментов, показана на рис. 7. Схема иллюстрирует процесс сокрытия потоков путем манипуляции структурами KTHREAD и KPROCESS. Для сокрытия потока используется изменение этих ссылок. На схеме показан пример, где структура потока KTHREAD 1 удаляется путем перенаправления ссылок в структурах KTHREAD Root и KTHREAD 2. В результате поток KTHREAD 1 становится невидимым для инструментов, использующих функцию NtQuerySystemInformation, но продолжает работать.

Эта техника позволяет нарушителям скрывать свою активность и предотвращать обнаружение вредоносных потоков. Скрытые потоки продолжают функционировать, но становятся невидимыми для инструментов анализа.

Сокрытие драйверов описывалось в популярных изданиях [15, 18]. Также необходимо упомянуть списки драйверов, которые не используются встроенными инструментами: списки ObjectDirectory [37], список служб Service Control Manager (SCM) [38]. Анализ популярных списков структур, используемых для поиска скрытых драйверов, будет дан в главе 5.

Метод нахождения списка структур драйверов в памяти предложен в работе [18]. В структуре DRIVER_OBJECT по определенному смещению находится поле DriverSection, которое указывает на элемент в списке драйверов, соответствующий DRIVER_OBJECT. Все структуры DRIVER_OBJECT находятся в двунаправленном списке PsLoadedModuleList. Для сокрытия

структуры драйвера нужно изменить значения Flink и Blink у двух структур Loaded Module, между которыми находится скрываемый драйвер, так чтобы скрываемый элемент выпал из списка. Для скрываемого драйвера значения Flink и Blink должны указывать на него самого. Эти изменения не влияют на функционирование драйвера и ОС.

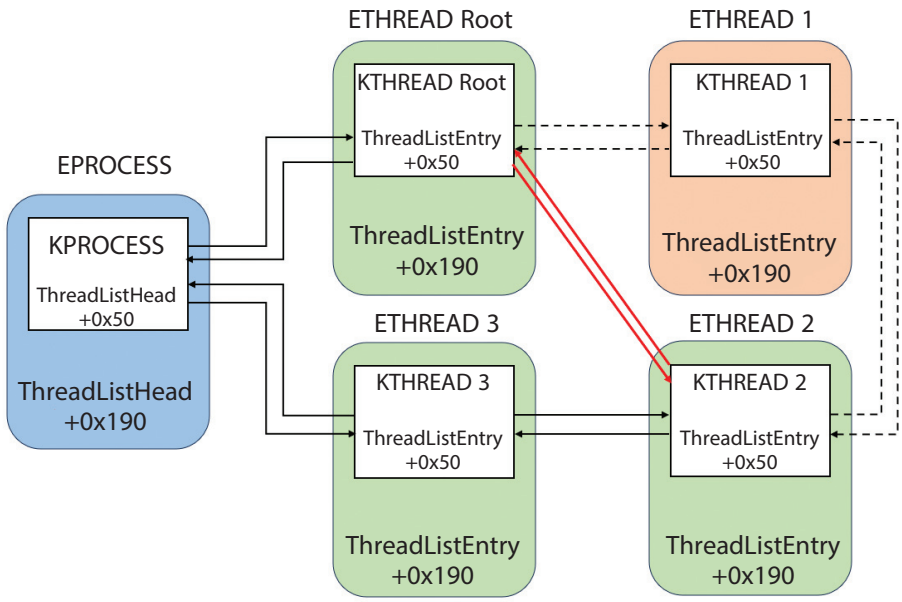


Рис. 7. Схема способа сокрытия потоков

Сокрытие служб (сервисов) реализуется аналогично. Структуры, описывающие службы, находятся в пользовательском адресном пространстве процесса SERVICE.exe. В литературе такие структуры называют Service Record List, SERVICE RECORD и SC_SERVICE_RECORD. Информация о содержимом этих структур отсутствует в отладочных символах Windows, и для ее восстановления необходимо проведение реверс-инжиниринговых исследований. Для сокрытия заданной службы необходимо найти соответствующую структуру в памяти и изменить значения указателей в соседних к ней элементах списка так, чтобы скрываемая запись была исключена из списка. При этом сама структура остается в памяти, а служба продолжает функционировать.

В результате штатные средства управления службами (например, sc query, services.msc) не обнаруживают скрытую службу. Однако ее выполнение и взаимодействие с системой остаются неизменными.

3.2. ИСПОЛЬЗОВАНИЕ DKOM ДЛЯ ОБХОДА И ПРОТИВОДЕЙСТВИЯ EDR/AV

В настоящее время нарушители активно используют драйверы ядра при проведении компьютерных атак на оконечные узлы пользователей. Несмотря на наличие штатных средств защиты для контроля загрузки драйверов, таких как проверка цифровой подписи (Driver Signing Policy) и блокировка загрузки известных уязвимых драйверов (Microsoft Vulnerable Driver Blocklist), нарушителям удается загружать драйверы с помощью различных техник: использование действующих сертификатов, добытых незаконным образом для подписи вредоносных драйверов, путем эксплуатации уязвимостей сторонних драйверов (Bring Your Own Vulnerable Driver, BYOVD), уязвимостей UEFI [39] и другими техниками. С другой стороны, существующие средства защиты объектов ядра — PatchGuard и Hypervisor-Protected Code Integrity (HVCI) — имеют функциональные ограничения и не могут контролировать целостность всех полей различных структур ядра. Особенно это касается случая, когда содержимое самой структуры, например перечень привилегий процесса и различные переменные-флаги в памяти, может меняться при работе системы.

Эти уязвимости создают возможности для атакующих нарушить работу защитных механизмов и компрометировать ОС. Многочисленные атаки подчеркивают необходимость совершенствования существующих методов защиты и разработки новых подходов к обеспечению безопасности ядра ОС.

Среди примеров DKOM атак на EDR-решения можно выделить следующие работы:

- атаки на подсистему Event Tracing for Windows;
- атаки на подсистему Windows Management Instrumentation (WMI);
- модификация привилегий для антивирусных процессов.

Атаки на подсистему Event Tracing for Windows, ETW. Подсистема трассировки событий Windows (ETW) активно используется различными EDR- и AV-решениями для выявления угроз на оконечных узлах пользователей. Поскольку подсистема ETW была разработана для решения задач по диагностике производительности и отладки приложений, разработчики не включали никаких механизмов для контроля целостности и корректной работы. В результате злоумышленники могут нарушить работу целого спектра EDR- и AV-решений путем воздействия на подсистему ETW.

Эксперты по безопасности компании Binarly, включая Клаудиу Теодореску (Claudiu Teodorescu), Игоря Коркина (Igor Korkin) и Андрея Голчикова (Andrey Golchikov), представили на конференции Black Hat [32] более 30 вариантов атак на ETW в пространстве пользователя и ядра. Было продемонстрировано, как подмена структур в памяти, например WMI_LOGGER_CONTEXT, или модификация Security Descriptor соответствующих сессий позволяют отключить логирование ETW-событий, тем самым «ослепляя» антивирус Microsoft Defender и другие EDR-решения без срабатывания штатных средств защиты ядра.

Атаки на подсистему Windows Management Instrumentation (WMI). Инструментарий управления Windows — WMI является встроенной технологией в Windows, разработанной для проведения удаленной инвентаризации компьютерных систем. Благодаря доступу к телеметрии и гибкой системе подписок на события WMI играет важную роль во многих решениях класса EDR, антивирусных системах и SIEM-платформах.

При этом механизмы WMI часто становятся целью атак, поскольку требования по безопасности изначально не были учтены при разработке этой системы. Нарушители используют различные техники для нарушения работы WMI, в том числе и DKOM-атаки на компоненты ядра и пользовательского режима. Эксперты из компании *Binarly* К. Теодореску, И. Коркин и А. Голчиков также представили на конференции Black Hat USA [33] доклад, демонстрирующий ряд техник, которые позволяют «ослеплять» EDR- и AV-продукты путем нарушения

работы ALPC-каналов WMI и модификации структур клиентской и серверной частей процесса WinMgmt. Подобные атаки не только отключают телеметрию скрытно от штатных средств защиты ядра, но и практически не порождают типичных сигнатур, что усложняет их обнаружение.

Модификация привилегий для антивирусных процессов. Проблема атак на встроенное антивирусное решение Microsoft Defender с использованием драйверов ядра была дополнительно рассмотрена в работе экспертов Дениса Погонина (Denis Pogonin) и Игоря Коркина (Igor Korkin) на конференции ADFSL [34] в 2022 г. В ходе исследования авторы провели анализ различных техник обхода защитных механизмов Windows. Особое внимание было уделено возможностям изменения содержимого полей EPROCESS, включая поля Privileges и IntegrityLevelIndex, с помощью которых можно понизить уровень доступа и другие привилегии процесса. Такой подход не только отключает защитные функции антивируса, изолируя его от файловой системы и ОС, но и не вызывает защитной реакции со стороны PatchGuard, поскольку PatchGuard не контролирует целостность изменяемых полей структуры EPROCESS. Для предотвращения подобных угроз авторы предлагают использовать гипервизорное решение Memory Ranger, изолирующее уязвимые структуры памяти ядра от несанкционированного доступа.

Примеры других атак на структуры ядра будут показаны далее в разделе 2.

Глава 4
АНАЛИЗ ТЕХНИК ПОЛУЧЕНИЯ ДАМПА ПАМЯТИ ДЛЯ ВЫЯВЛЕНИЯ СКРЫТЫХ ОБЪЕКТОВ

Один из способов обнаружения скрытого кода в ОС заключается в получении дампа памяти и дальнейшем его анализе в целях обнаружения как самого исполнимого кода, так и соответствующих структур и других индикаторов, которые могут помочь выявить потенциальные угрозы.

Получение дампа памяти (memory dump) представляет собой процесс копирования содержимого оперативной памяти (RAM) на устройство хранения в виде файла дампа памяти (формат *.dmp). Этот файл фиксирует текущее состояние системы на момент создания дампа, включая данные и программы, которые были активны в оперативной памяти [40]. Таким образом, дамп памяти — это, по сути, снимок, фиксирующий сложное содержимое памяти компьютера в определенный момент времени [41]. При этом выделяют дамп физической памяти и дамп виртуальной памяти, под которым понимается копия памяти одного из программных контекстов.

4.1. ТЕХНИКИ ПОЛУЧЕНИЯ ДАМПА ВИРТУАЛЬНОЙ ПАМЯТИ ПРОЦЕССОВ

Дамп виртуальной памяти процесса представляет собой снимок текущего состояния виртуального адресного пространства заданного процесса. Основные техники получения дампа виртуальной памяти включают использование встроенных системных утилит, отладчиков и специализированных инструментов.

Windows ОС предоставляет встроенные утилиты, такие как Task Manager, которые позволяют создавать дампы памяти процессов. Однако такие дампы часто являются неполными и в них может отсутствовать требуемая аналитику информация [41].

Отладчики, такие как WinDbg и Visual Studio Debugger, позволяют получить более полный и детализированный дамп процесса. Эти инструменты используют возможности API Windows,

такие как MiniDumpWriteDump, для создания дампов памяти, которые могут включать полный набор страниц виртуальной памяти процесса [1, 41–43].

Для получения файла с дампом памяти заданного процесса можно использовать специальные средства, такие как ProcDump от *Sysinternals* [44], userdump.exe от *Microsoft* [45], инструкцию «Лаборатории Касперского» [46], pmdump.exe от лаборатории *Arne Vidstrom* [47] и др. Эти инструменты могут быть настроены для автоматического создания дампов при возникновении различных условий. Данные решения работают с функциями OpenProcess и ReadProcessMemory или их низкоуровневыми аналогами KeStackAttachProcess и ZwReadProcessMemory [48]. В книгах [49, 50] содержится обзор указанных средств. Одним из недостатков этих решений является их уязвимость к манипуляциям ВПО, которые могут нарушить ожидаемое поведение функций OpenProcess и ReadProcessMemory (KeStackAttachProcess и ZwReadProcessMemory), например, вследствие их перехвата. Другим недостатком является то, что соответствующий файл дампа памяти может содержать недостаточно информации для проведения полноценного анализа.

4.2. ТЕХНИКИ ПОЛУЧЕНИЯ ДАМПА ФИЗИЧЕСКОЙ ПАМЯТИ НА РАЗЛИЧНЫХ УРОВНЯХ ПРИВИЛЕГИЙ

Получение дампа физической памяти крайне важно для анализа инцидентов безопасности, и соответствующие техники получения дампа памяти могут варьироваться в зависимости от доступного уровня привилегий.

4.2.1. Средства получения дампа физической памяти на уровне ядра

Средства получения дампа физической памяти на уровне ядра имеют доступ к системным ресурсам, что позволяет им получать полные дампы памяти. Для чтения памяти на уровне ядра могут быть использованы следующие функции ОС:

1) ZwOpenSection — функция открывает раздел объекта, предоставляя дескриптор для доступа к физической памяти;

2) ZwMapViewOfSection — функция отображает вид раздела объекта в виртуальное адресное пространство процесса, предоставляя доступ к физической памяти;

3) MmMapIoSpace — функция отображает физические адреса ввода-вывода в виртуальное адресное пространство ядра;

4) MmMapMemoryDumpMdl — функция используется для отображения дампов памяти в виртуальное адресное пространство.

Некоторые известные программные средства, позволяющие получить дамп памяти на уровне ядра, приведены в табл. 2. Здесь же показаны функции, которые, соответственно, используются программными продуктами.

Дамп памяти можно получать и анализировать удаленно [51, 52], и эти возможности реализованы в некоторых коммерческих продуктах [53]. Локальный агент считывает физическую память, используя указанные выше функции, и затем передает данные на сервер для последующего анализа.

Таблица 2

Программные средства получения дампа памяти

Название средства	Разработчик
FTK Forensic Toolkit	*Exterro* (США)
Belkasoft Live RAM Capturer	*Belkasoft* (Россия)
Compiled Memory Analysis Tool (CMAT)	J. Okolica, G. Peterson (США) [54]
Magnet DumpIt for Windows	*Magnet Forensics* (Канада)
Encase Forensic	*OpenText* (Канада)
Winpmem	*The Volatility Foundation* (США)
Memoryze	*FireEye Inc.* (США)
ProDiscover Basic Edition	*DotC Technologies Pvt Ltd.* (Индия)
Redline	*FireEye Inc.* (США)

Нарушители могут противодействовать средствам получения дампа памяти, например, путем перехвата функций в целях изменения их работы. Пример такого способа защиты от чтения памяти описан Л. Милковичем (Luka Milković) в работе [53]. Автор предложил перехватывать функции, которые сохраняют

страницы памяти на жесткий диск для изменения содержимого буфера. Таким образом, в итоговом файле дампа памяти будут отсутствовать страницы со скрытыми объектами.

Это наглядно показывает, что существующие средства чтения памяти на уровне ядра не всегда работоспособны в условиях противодействия со стороны современного ВПО и не в полной мере соответствуют современным вызовам времени.

4.2.2. Средства получения дампа физической памяти на уровне аппаратной виртуализации

Технология аппаратной виртуализации (см. параграф 1.2) позволяет выполнить код, называемый гипервизором, который работает в более привилегированном режиме (режим Virtual Machine Extension root, VMX root), на уровне ниже, чем ОС. Использование гипервизоров для получения дампа памяти описано в работах [55–57].

Получение дампа памяти с помощью гипервизора, в отличие от рассмотренных ранее техник, позволяет противостоять действиям ВПО, блокирующим получение дампа памяти с помощью перехвата функций. В то же время данный способ применим только в системах, поддерживающих аппаратную виртуализацию, и только в том случае, если ранее загруженный легитимный гипервизор поддерживает вложенную виртуализацию [58].

Недостатком данного подхода является его уязвимость к атаке «человек посередине» (Man-In-The-Middle), так как вредоносный гипервизор (гипервизор ВПО) может загрузиться раньше, чем легитимный. Используя технологии Shadow Page Tables (*AMD*) и Extended Page Tables (*Intel*), гипервизор ВПО может скрывать области памяти [59]. В результате легитимному гипервизору будут недоступны для чтения определенные страницы памяти [15].

Технологии Intel Trusted Execution Technology (TXT) и AMD Secure Extension Mode (SEM) обеспечивают механизмы доверенной загрузки с использованием модуля Trusted Platform Module (TPM) [60, 61]. Эти технологии могут обеспечить приоритетную загрузку доверенного гипервизора и противодействовать атакам типа «человек посередине» (Man-in-the-Middle, MITM).

Однако разработка гипервизора, загружаемого непосредственно из режима UEFI, представляет собой сложную техническую задачу и в настоящее время реализуется преимущественно в лабораторных условиях, как это показано в проектах MiniVisorPkg и Illusion.

4.2.3. Средства получения дампа физической памяти на уровне режима системного управления

Режим системного управления (System Management Mode, SMM) является более привилегированным (см. параграф 2.4), чем режим VMX root, и обеспечивает функции управления питанием и обратную совместимость; частично документирован и описан в статьях [30, 62, 63]. Анализ возможностей SMM-режима по получению дампа памяти представлен в работах [64, 65].

Однако применение этого подхода на практике затрудняется инсталляцией SMM-диспетчера на различные материнские платы [66]. Другим недостатком является необходимость перезагрузки ПК, что не всегда возможно [57]. Также нарушитель может установить собственный вредоносный обработчик SMM [67]. Такой подход может быть применен в некоторых старых моделях материнских плат. Адаптация к новым компьютерам требует серьезных и нетривиальных исследований [68].

4.2.4. Средства получения дампа физической памяти на уровне аппаратной технологии AMT

На компьютерах с технологией Active Management Technology (AMT), которая является частью Intel Management Engine (ME), можно реализовать еще один метод получения дампа памяти. Код AMT является более привилегированным, чем код уровней VMX root и SMM (см. параграф 1.2).

В работах [69, 70] описывается технология AMT с точки зрения информационной безопасности. Поскольку на уровне AMT могут выполняться вредоносные программы [66, 71], справедливо утверждать, что в этом режиме могут работать средства получения дампа памяти. Широкое применение данного метода на практике затруднительно из-за отсутствия полноценной документации по AMT и ME.

4.2.5. Средства получения дампа физической памяти на уровне аппаратного обеспечения

В 2005–2010 гг. появились технологии IOMMU (Input-Output Memory Management Unit) [72] от *AMD* и VT-d (Intel Virtualization Technology for Directed I/O) от *Intel*. Эти аппаратные механизмы обеспечивают контроль доступа устройств ввода-вывода к физической памяти, предотвращая несанкционированный доступ и обеспечивая изоляцию ресурсов [73, 74].

В работах [75, 76] отмечается преимущество аппаратных средств получения дампа памяти по сравнению с программными. Поэтому далее будут подробно рассмотрены аппаратные подходы к получению дампа памяти.

Механизмы прямого доступа к памяти (Direct Memory Access, DMA) с использованием стандартных интерфейсов PCI (Peripheral Component Interconnect) и PCIe (PCI Express) позволяют периферийным устройствам напрямую обмениваться данными с оперативной памятью без участия центрального процессора (CPU), что значительно ускоряет процесс передачи данных и снижает нагрузку на систему. Это делает DMA идеальным для задач, требующих быстрого доступа к памяти, таких как компьютерная криминалистика и выявление вредоносных программ в условиях противодействия. Возможности устройств PCI (PCIe) по получению прямого доступа к памяти DMA реализованы в работе [77].

В статье [78] предлагается процедура получения данных из оперативной памяти с использованием аппаратной карты расширения, которая устанавливается в слот PCI и активируется физическим переключателем.

Инструмент безопасности Co-Pilot, разработан компаниями *Komoku* и *Microsoft* для повышения защиты компьютеров от различных угроз. Основная цель Co-Pilot заключается в обеспечении мониторинга и анализа системной активности для выявления потенциальных атак и вредоносных программ [79, 80].

Другой инструмент RAM Capture Tool от *BBN Technologies* позволяет извлечь и сохранить данные из памяти для последующего анализа, что важно при расследовании инцидентов кибер-

безопасности и выявлении вредоносных программ [81]. Пример использования интерфейса передачи данных FireWire (IEEE 1394) для дампа памяти описан в работе [82]. Применение аппаратных интерфейсов USB, eSATA, DisplayPort, Thunderbolt и других инструментов для доступа к физической памяти представлено в работе [83].

Для получения дампа памяти можно использовать и стандартные средства: применение таблицы преобразований графических адресов (Graphics Address Remapping Table, GART) [84] или получение дампа памяти с помощью встроенного контроллера северного моста [85].

Важно отметить, что ВПО может блокировать чтение памяти даже на уровне аппаратного обеспечения. Например, технология защиты от внешнего доступа (External Access Protection Technology) от *AMD* позволяет скрыть страницы памяти от периферийных устройств [86]. В работе [87] показано, как можно перепрограммировать контроллер северного моста и скрыть области памяти от доступа периферийных устройств. Изменения адресов таблиц диспетчеризации в контроллере северного моста позволяют скрыть физические области памяти.

Несмотря на то, что подходы на уровне аппаратного обеспечения в общем случае способны противостоять скрытому ВПО, такие аппаратные техники применимы в основном в лабораторных условиях.

4.3. ТЕХНИКА ПОЛУЧЕНИЯ ДАМПА ПАМЯТИ С ПОМОЩЬЮ ДЕРЕВА СТРУКТУР СТРАНИЦ ПАМЯТИ

4.3.1. Подход к получению дампа памяти

Рассмотрим способ получения дампа виртуальной памяти программного контекста процесса, основанный на обходе дерева структур таблиц памяти. Данный способ является, с одной стороны, программным, т.е. не требует дополнительно подключаемого оборудования, а с другой стороны, он позволяет получать дамп памяти даже в условиях противодействия со стороны ВПО в виде перехвата функций, задействованных для получения дампа памяти.

Полученный дамп памяти представляет собой набор файлов, позволяющих работать с ним так, как будто он был получен на локальной системе: получать содержимое памяти по заданному виртуальному адресу, а также рассчитывать виртуальный адрес в памяти, соответствующий смещению в файле дампа.

Получение дампа памяти виртуального контекста основано на использовании структур страничной памяти, таких как таблицы страниц (page tables) и каталоги страниц (page directories). Эти структуры участвуют в процессе страничной адресации и управляют отображением виртуальной памяти в физическую.

Алгоритм получения дампа памяти процесса включает в себя следующие шаги:

1) переключение контекста выполнения текущего потока на заданный процесс. Для этого можно использовать функцию KeAttachProcess() или KeStackAttachProcess(), а также их низкоуровневые аналоги для случая противодействия ВПО, использующего перехват этих функций [88];

2) последовательный проход по структурам страничной памяти. В процессе получения дампа памяти создаются два файла: файл со страницами дампа памяти 'dump.log' и файл 'struct.log' с информацией о виртуальных адресах страниц и их смещениях в файле 'dump.log';

3) в отдельных файлах сохраняется дополнительная информация, например, о списках EPROCESS, KDBG, KPCR, полученных с использованием WinAPI на исходной системе.

Полученные файлы могут быть использованы для решения различных задач информационной безопасности, таких как обнаружение вредоносных программ и драйверов, проведение обратной разработки (reverse-engineering), расследование компьютерных инцидентов. Подробное описание работы с этими файлами будет рассмотрено далее.

4.3.2. Процесс отображения виртуальной и физической памяти

Процесс отображения виртуальной и физической памяти при анализе дампа памяти на примере сохранения страницы с но-

мером 3 (page 3) в 32-битовом формате приведен на рис. 8. Он также показывает, как информация о страницах памяти сохраняется в файл с дампом памяти (Dump File). Виртуальная страница транслируется в физическую посредством таблицы страниц. Соответствующие размеры страниц совпадают, однако их порядок следования, как правило, отличается.

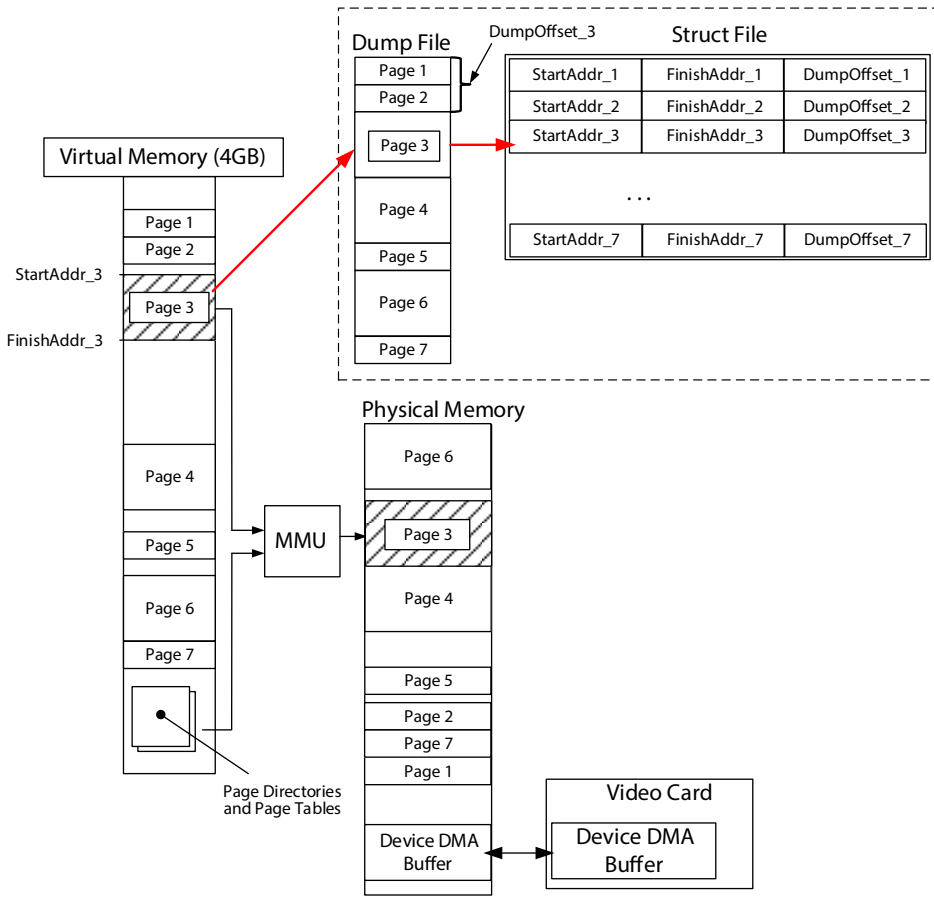


Рис. 8. Отображение виртуальной и физической памяти при анализе дампа памяти:
MMU — механизм управления памятью; Virtual Memory — виртуальная память; Physical Memory — физическая память; Dump File — файл дампа, в котором сохраняется информация о страницах виртуальной памяти

Дамп памяти получается в результате прохода по системным таблицам, таким как таблицы директорий страниц (page directory

pointer tables) и каталоги страниц (page directories), таблицы страниц (page tables) и др. Детали прохода зависят от включенного страничного режима (paging mode), режима работы встроенного блока управления памятью (Physical Address Extension, PAE), а также от разрядности ОС Windows (32 или 64 бита).

Алгоритмы прохода по таблицам для 32-битовых версий ОС Windows работают похожим образом, независимо от того, включен режим PAE или нет. Основное различие заключается в деталях формата и размера самих таблиц, так как при включенном режиме PAE используются расширенные структуры. В случае 64-битовых версий ОС Windows алгоритм прохода аналогичен, но необходимо учитывать четыре таблицы. Для новой версии архитектуры x86-64 нужно учитывать пять таблиц. Поэтому без ограничения общности остановимся на алгоритме прохода для 32-битовой версии ОС Windows.

Для получения дампа памяти выполняется следующий цикл:

1) последовательный проход по записям в таблице директорий страниц (page directory). Проверка флага присутствия P для каждой записи;

2) если флаг PDE.P равен 0, то выполняется переход к следующей записи. Если PDE.P равен 1, то проверяется флаг размера страницы PS;

3) если флаг PDE.PS равен 1, то соответствующая страница памяти сохраняется. Ее размер зависит от того, включен ли режим PAE или нет, и равен 2 или 4 Мб соответственно;

4) если флаг PDE.PS равен 0, то текущая запись соответствует таблице страниц (page table), содержащей информацию о страницах размера 4 Кб. Переход к данной таблице страниц;

5) аналогичным образом выполняется последовательный проход по таблице страниц и проверка флага P. Если флаг PTE.P равен 0, то осуществляется переход к следующей записи. Иначе сохраняется страница памяти размером 4 Кб.

Каждая страница сохраняется в буфер. Когда буфер заполнен, его содержимое архивируется, после чего результаты сохраняются в файл 'dump.log'. Это повышает общую производительность программной реализации [54].

Описанный алгоритм представлен на рис. 9.

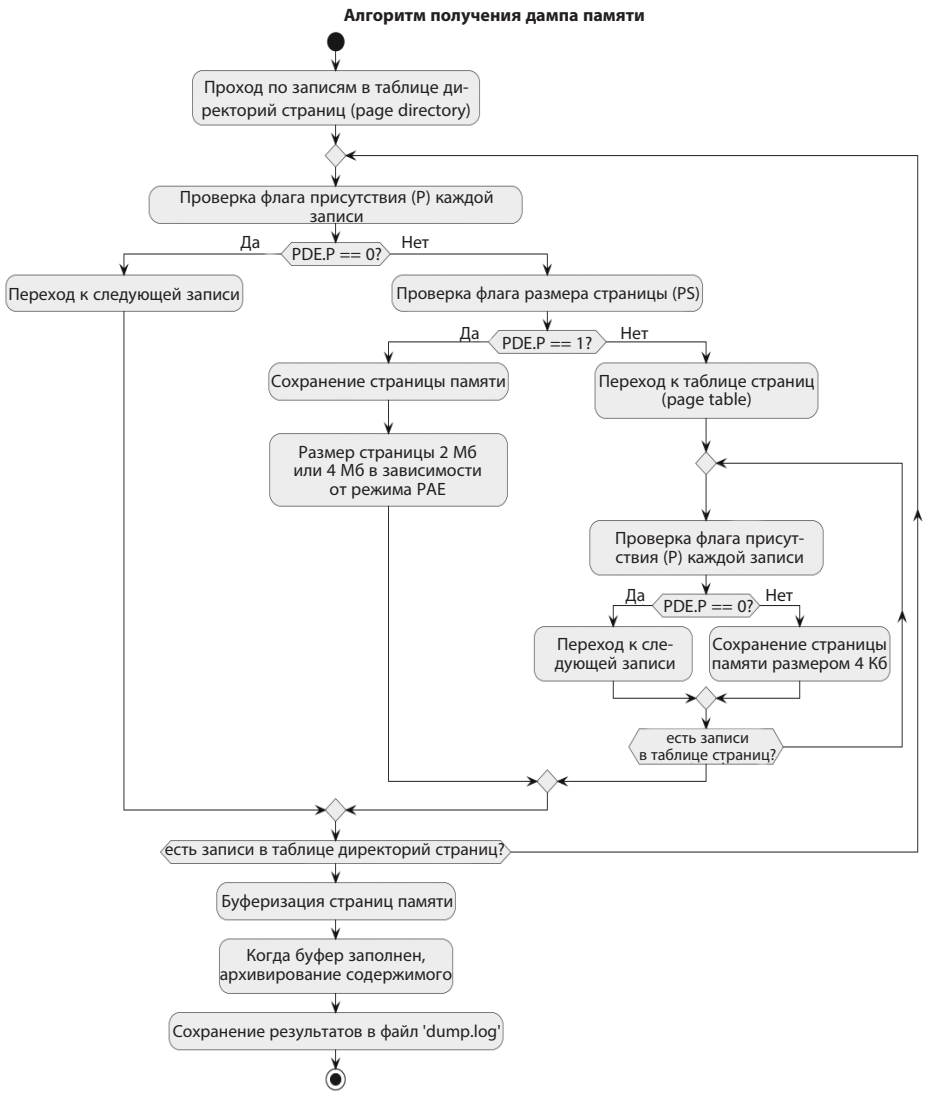


Рис. 9. Алгоритм получения дампа памяти

Главные особенности рассмотренного подхода по получению дампа памяти:

- эксперименты показали, что для исключения загрузки пустых страниц необходимо выполнять проход по таблицам страниц в направлении от старших адресов к младшим;
- алгоритм прохода по страницам памяти должен быть реализован на уровне PASSIVE_LEVEL IRQL, так как только

на этом уровне задействуется механизм подгрузки страниц с жесткого диска в память;
- при обращении к странице памяти буфера DMA, относящегося к устройству, например к видеокарте, начиная с Windows Vista происходит сбой системы. Эти критические исключения не могут быть обработаны стандартными функциями обработки исключений. Чтобы предотвратить сбой, необходимо пропускать эти страницы при обходе дерева таблиц.

4.3.3. Обработка данных, полученных из дампа памяти

Обработка данных, полученных из дампа памяти, включает несколько этапов. Первоначально при подготовке к анализу данных из дампа памяти необходимо удалить ненужную информацию (например, пустые или неиспользуемые страницы памяти), а затем применить фильтры на основе установленных критериев (адреса памяти и типы данных и т.д.).

Следующий этап — восстановление указателей и ссылок для корректной интерпретации данных, где важную роль играют сведения, содержащиеся в файлах 'dump.log' и 'struct.log', обеспечивающие восстановление соответствия между виртуальными адресами и физическими смещениями. После получения файлов 'dump.log', 'struct.log' и др., их обработка может быть выполнена локально на текущем компьютере или дистанционно после передачи этих файлов на удаленную машину.

Похожий подход с получением дампа памяти и его автоматизированным анализом реализован в таких программных средствах, как Volatility или Rekall, позволяющих идентифицировать аномалии и потенциально вредоносные программы [98].

Рассмотрим пример восстановления соответствия между виртуальными адресами в исходной памяти и значениями смещения в файле дампа памяти с использованием файлов 'dump.log' и 'struct.log'.

Введем следующие обозначения:
- VALF (Virtual Address in the Loaded dump File) — виртуальные адреса загруженного файла дампа памяти;
- ODUF (Offset in DUmp File) — соответствующие смещения в файле дампа;

- VAOM (Virtual Address in the Original Memory) — виртуальный адрес исходной памяти.

Для получения содержимого памяти по известному виртуальному адресу необходимо получить соответствующее ему значение смещения в дампе файла. Наоборот, по смещению в файле дампа можно получить соответствующий виртуальный адрес.

Процесс преобразования между различными адресными пространствами, используемыми в анализе дампа памяти, показан на рис. 10.

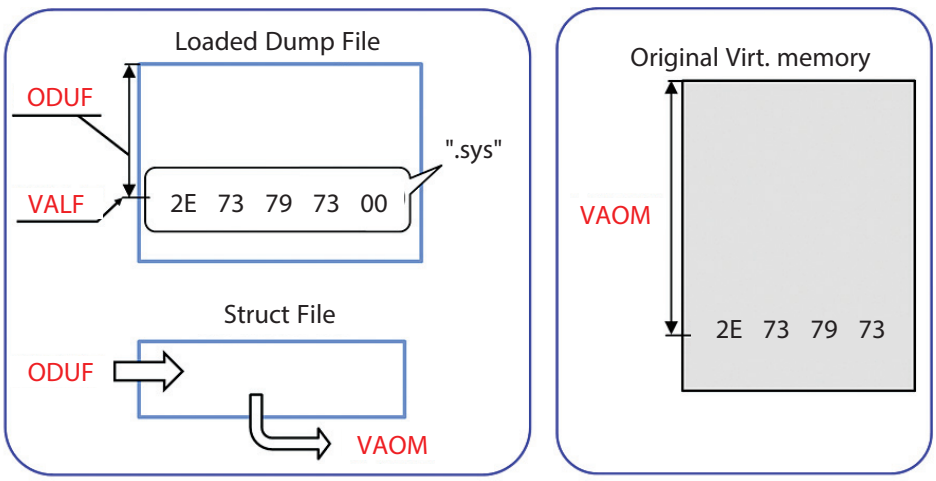


Рис. 10. Файлы 'dump.log' и 'struct.log' и преобразования 'ODUF', 'VALF' и 'VAOM'

1. Преобразование 'VAOM'→'ODUF': в файле 'struct.log' нужно найти i-ю запись, которая содержит диапазоны виртуальной памяти, такую, чтобы целевое значение 'VAOM' принадлежало ее диапазону. 'ODUF' определяется следующим образом:

$$ODUF = DumpOffset[i] + (FinishAddr[i]-VAOM),$$

где DumpOffset[i] — начальное смещение для i-й записи в файле дампа; FinishAddr[i] — конечный адрес диапазона виртуальной памяти для i-й записи; VAOM — виртуальный адрес в оригинальной памяти, который нужно преобразовать.

2. Преобразование 'ODUF'→'VAOM': в файле 'struct.log' нужно найти i-ю запись, такую, чтобы $Offset[i] \le ODUF < Offset[i+1]$, где

Offset[i+1] означает Offset следующей (i+1)-й записи. Тогда VAOM определяется так:

$$VAOM = FinishAddr[i] + (ODUF-Offset[i]),$$

где Offset[i] — начальное смещение для i-й записи в файле дампа.

3. Преобразование 'ODUF' → 'VALF' и 'VAOM' → 'VALF'. Значения 'ODUF' и 'VALF' различаются значением начального адреса загруженного файла дампа: VALF=ODUF+LoadAddr; обратное преобразование: ODUF = VALF-LoadAddr.

На рис. 10 показано, как значения адресов и смещений преобразуются между различными файлами и контекстами. Loaded Dump File — файл дампа памяти, содержащий данные в формате, доступном для анализа; также показаны: значение VALF (Virtual Address in the Loaded dump File) и соответствующее смещение ODUF (Offset in DUmp File); оригинальная виртуальная память (Original Virt. Memory), в которой VAOM (Virtual Address in the Original Memory) представляет собой виртуальный адрес исходной памяти. Файл структур (Struct File) используется для преобразования между виртуальными адресами в исходной памяти и значениями смещения в файле дампа. На указанном рисунке показано, как ODUF преобразуется в VAOM.

4.3.4. Использование инструмента MASHKA в задачах исследования памяти

Разработка методов и инструментов для анализа содержимого памяти является важным направлением в области цифровой криминалистики и информационной безопасности. Один из современных инструментов, предназначенных для такого анализа, называется MASHKA (Malware Analysis System for Hidden Knotty Anomalies) [30]. Этот инструмент способен выполнять широкий спектр задач по анализу памяти [90]. Средства антифорензики направлены на сокрытие данных, уничтожение доказательств и манипуляцию структур памяти, чтобы затруднить или сделать невозможным криминалистический анализ. MASHKA разработана для того, чтобы обеспечить анализ содержимого памяти в условиях активного противодействия.

Основные функции инструмента MASHKA: поиск текстовых и двоичных сигнатур; поиск скрытых процессов и драйверов.

MASHKA выполняет многопоточный поиск различных объектов в дампе памяти, включая однобайтовые (char) и двухбайтовые (wchar_t) строки, а также байтовые фрагменты, такие как значения адресов. Поиск может выполняться в побайтовом режиме или с шагом в 4 (8) байт для отдельных структур, таких как EPROCESS и DRIVER_OBJECT.

Когда объект найден, его VAOM (Virtual Address in the Original Memory), VALF (Virtual Address in the Loaded Dump File) и ODUF (Offset in Dump File) подвергаются детальному анализу. Возможен поиск адресов, значения которых расположены вблизи VAOM-адреса, что может использоваться при реверс-инжиниринговом анализе недокументированных структур.

Для иллюстрации принципа действия MASHKA рассмотрим один из примеров итеративного процесса обнаружения драйверов с использованием дампа памяти и отладчика WinDbg.

1. Загрузка ОС Windows в отладочном режиме под управлением WinDbg.

2. Установка драйвера с указанными 'ServiceName' и 'DisplayName', расположенными в 'BinaryPathName', например, с помощью диспетчера управления сервисами (SCM).

3. Сокрытие драйвера посредством манипуляций со списком PsLoadedModuleList. В результате вызов функции NtQuery SystemInformation не возвращает информацию об этом драйвере [34].

4. Проверка системы с помощью нескольких популярных антируткитов. Это позволит обнаружить намеренно скрытый драйвер.

5. Получение дампа памяти с помощью инструмента MASHKA. Копирование файлов 'dump.log', 'struct.log' и других важных файлов на хост-компьютер.

6. Поиск строк, содержащих 'ServiceName', 'DisplayName' и 'BinaryPathName'. Найденные строки VAOM сохраняются для дальнейшего анализа.

7. Приостановка виртуальной машины с помощью WinDbg.

8. Посредством WinDbg изменяется содержимое строк VAOM, например, происходит замена символа 'A' на 'Z' в начале каждой строки.

9. Повторная проверка системы антируткитами, которые должны показать скрытые объекты с измененными именами.

10. Анализ и внесение изменений в соответствующие списки структур и повтор шагов 4–10.

Процесс использования инструмента MASHKA в задачах криминалистического анализа драйверов представлен на рис. 11.

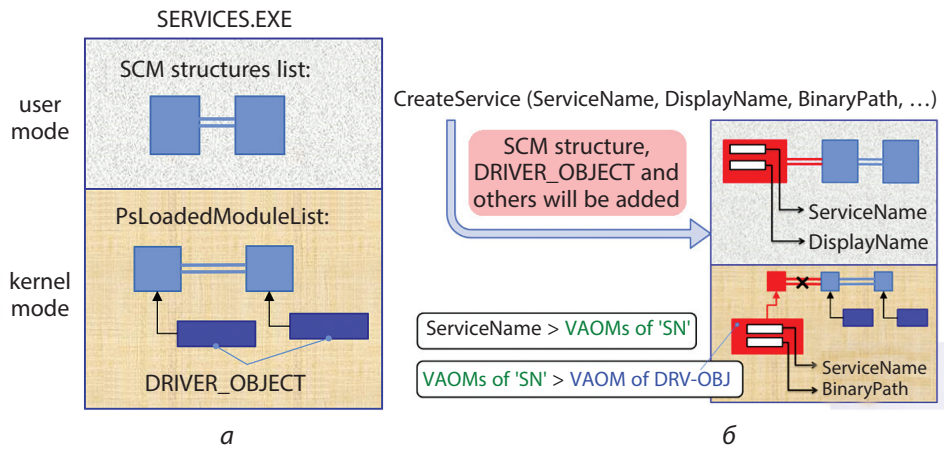


Рис. 11. Использование инструмента MASHKA в задачах исследования памяти:
а — перед запуском драйвера; *б* — изменение структур после загрузки драйвера

На рис. 11, *а* показано состояние системы до загрузки нового драйвера. Исследователь просматривает два списка: список служб в services.exe и список загруженных драйверов в памяти ядра (PsLoadedModuleList).

После загрузки драйвера в память (см. рис. 11, *б*) добавляются новые структуры, такие как SCM Structure и DRIVER_OBJECT. Эти структуры содержат ссылки на строковые значения, включая имя службы (ServiceName), отображаемое имя (DisplayName) и путь к бинарному файлу (BinaryPath). Указанные связи демонстрируют, как новые объекты добавляются в системные списки в памяти ОС.

Инструмент MASHKA служит для поиска известных строковых значений, таких как ServiceName. Найденные значения

используются для определения виртуальных адресов исходной памяти VAOM, которые указывают на конкретное расположение данных в оригинальной памяти системы.

Применяя найденные VAOM, можно далее искать и анализировать связанные структуры и списки, такие как объекты драйверов (DRIVER_OBJECT) и другие ключевые компоненты системы. Это позволяет получить полное представление о состоянии системы и идентифицировать любые скрытые или подозрительные объекты.

К числу преимуществ инструмента MASHKA можно отнести его способность функционировать в условиях активного противодействия, например, при перехвате функций для получения дампов памяти, а также широкие возможности инструмента MASHKA для выполнения задач криминалистического анализа памяти, включая выявление скрытых драйверов и реверс-инжиниринговые исследования.

Глава 5
МЕТОДЫ ОБНАРУЖЕНИЯ СКРЫТЫХ ОБЪЕКТОВ В ДАМПЕ ПАМЯТИ

Рассмотрим различные подходы к обнаружению процессов и драйверов, скрытых с использованием техники DKOM.

Метод обнаружения на основе сравнения списков (cross-view analysis). Одним из наиболее популярных способов обнаружения скрытых драйверов (руткитов) является подход кросс-анализа, который основывается на том, что информация об объектах ОС хранится в нескольких независимых списках. При обычной работе ОС без скрытых объектов эти списки совпадают. Сравнивая списки, можно выявить скрытые объекты.

Первый список обычно формируется с использованием функций WinAPI, которые применяются в штатных инструментах ОС. Для обеспечения долговременного присутствия нарушители должны скрыть объект из первого списка. Второй список может быть создан с использованием дополнительных источников данных операционной системы. Со своей стороны нарушители могут также удалить информацию и из различных дополнительных списков, что затруднит, но не исключит обнаружение.

Для формирования второго списка применяются как высокоуровневые методы (например, стандартные функции WinAPI и ядра), так и низкоуровневые техники, включая работу с недокументированными структурами данных [37, 91]. Высокоуровневые методы опираются на стандартные системные функции для перечисления объектов ОС, тогда как низкоуровневые используют эвристические анализаторы, структуры объектов и анализ содержимого памяти на основе статических и динамических сигнатур. В исследовании [14] предложена классификация механизмов обнаружения руткитов уровня ядра.

Формирование списка на основе анализа поведения объекта. Один из подходов для формирования второго списка объектов, например процессов, заключается в анализе их активности: файловых операций, сетевой активности, открытых окон [92].

Основным ограничением данного подхода является то, что обнаружение объекта становится возможным только при наличии достаточного объема данных о его активности.

Формирование списка на основе сканирования памяти. Альтернативным подходом к формированию второго списка является сигнатурный поиск как самих структур объектов, так и загруженных исполнимых модулей в дампе памяти.

В качестве сигнатур могут выступать как бинарные фрагменты исполнимых файлов, так и фрагменты структур в памяти ядра.

Сигнатурный поиск структур в памяти ядра основан на двух принципах:

- во-первых, все структуры объектов заданного типа загружены по единому шаблону и имеют сходное байтовое строение;
- во-вторых, структуры могут содержать адреса на другие структуры в ядре, таким образом, эти адреса будут превышать определенное значение, например значение 0x80000000 для 32-битных систем.

Этот метод включает побайтовый поиск фрагментов известных структур объектов в памяти и реализован в таких инструментах, как GMER, PowerTool и XueTr. Однако размеры и содержимое структур могут изменяться в новых версиях Windows, что требует адаптации метода для каждой новой версии Windows и может быть сложной задачей. Развитие этой техники получило продолжение в методе SIGIL, который использует промежуточное представление бинарных программ, позволяя создавать архитектурно независимые сигнатуры. Этот метод позволяет использовать одну сигнатуру для идентификации бинарных файлов, скомпилированных для разных архитектур из одного исходного кода [93].

Обнаружение скрытых объектов путем сканирования сигнатур также уязвимо к контрмерам со стороны вредоносного ПО. Вредоносная программа может изменить некоторые значения структур, чтобы сделать сканирование сигнатур безрезультатным [37]. Такие действия вредоносного ПО не приводят к аварийному завершению работы системы. Если хотя бы один байт не совпадает, то процесс сканирования пропустит структуру.

Нарушение работы средства анализа. Нарушители могут использовать различные техники противодействия, которые не только скрывают объект из списка, но и препятствуют проведению анализа. В результате ВПО будет оставаться необнаруженным, поскольку средство защиты не может корректно обработать дамп памяти.

Пример такой атаки был представлен Такахиро Харуяма (Takahiro Haruyama) и Хироши Судзуки (Hiroshi Suzuki); он основан на модификации байта в объектах ядра [94]. Этот метод заключается в изменении уникальных идентификационных значений, которые используются для распознавания объектов ядра. Например, для объекта DBGKD_DEBUG_DATA_HEADER64 существует уникальный тег KDBG. Если изменить всего один байт этого тега на другое значение, то некоторые инструменты анализа памяти перестают функционировать корректно. Это исследование продемонстрировало, что модификация одного байта в структуре PHYSICAL_MEMORY_DESCRIPTOR, которая используется для получения информации о физических областях памяти, может полностью нарушить процесс сбора данных этими инструментами. Если переменную NumberOfRuns этой структуры установить в 0, то инструменты не смогут получить информацию о физических областях памяти и процесс сбора данных будет остановлен [94, 95].

Проанализированные подходы демонстрируют различные методы обнаружения скрытых объектов в памяти. Однако каждый из этих методов имеет свои недостатки, которые могут быть использованы нарушителями для противодействия обнаружению.

5.1. ОБНАРУЖЕНИЕ СКРЫТЫХ ПРОЦЕССОВ И ПОТОКОВ

Когда процесс запускается, создается новый контекст процесса и информация об этом новом объекте добавляется в различные системные списки. Существует множество подходов к обнаружению процессов, и ниже описаны некоторые из них.

Часть подходов основана на дополнительных списках структур объектов, таких как список процессов из CSRSS.EXE, список планировщика на основе потоков и др. [96, 97]. Также существуют

подходы на основе эвристического анализа, использующие функции перехвата, такие как SwapContext или KiFastCallEntry. Фреймворк Volatility включает различные плагины для обнаружения скрытых процессов [98–100].

Инструмент MAS (Memory Analysis System) использует файл аварийного дампа памяти (crash dump) для обнаружения руткитов [101]. Инструмент MAS анализирует файл аварийного дампа памяти и ищет признаки руткитов путем сравнения содержимого дампа с ожидаемыми данными и структурами. Применение этого алгоритма в промышленных системах ограничено из-за необходимости наличия файла дампа памяти, что делает этот метод непригодным для постоянного мониторинга и защиты в реальном времени.

В отличие от популярных методов, основанных на перечислении списков ядра, этот инструмент для анализа дампов памяти систем на базе Windows использует паттерны поиска для сканирования всей памяти в поисках следов процессов и потоков независимо от списков, поддерживаемых ядром [102]. Инструмент способен обнаруживать скрытые и завершенные процессы и потоки даже после перезагрузки системы. Этот подход имеет ряд существенных недостатков: сложно добиться его переносимости на различные версии ОС Windows, а также он уязвим к модификациям полей.

Метод SigGraph используется для поиска экземпляров структур данных ядра в образах памяти путем побайтового сканирования памяти (brute force memory scanning) [103]. При таком сканировании весь объем данных сканируется на предмет наличия определенных структур или объектов без использования предварительных знаний о расположении этих объектов. Метод строит графовые сигнатуры на основе взаимосвязей между полями структур ядра. Авторы предложили новые графовые сигнатуры, которые позволяют обнаружить структуры в ОС Linux. Метод уязвим к модификации отдельных байтов и требует участия специалистов для подбора и тестирования графовых сигнатур для новых версий Windows.

Можно выделить следующие способы обнаружения скрытых объектов:

- прямое получение списка объектов;
- косвенное определение существования объектов.

Метод прямого получения списка объектов использует API-функции операционной системы для получения таких списков независимо от их типа. В некоторых случаях ОС предоставляет соответствующие API, а в других аналитик должен разработать решения самостоятельно [104]. Примеры таких функций для получения списка процессов включают:

- функции NtQuerySystemInformation и ZwQuerySystem Information, которые используются для получения различных системных данных, включая информацию о процессах, модулях, производительности процессора и других параметрах. Функция возвращает массив структур SYSTEM_PROCESS_INFORMATION, содержащих информацию о каждом процессе в системе, таких как количество потоков, уникальный идентификатор процесса (PID), использование памяти и другие параметры [105];
- функция CreateToolhelp32Snapshot создает снимок текущего состояния системы, включая процессы, модули и потоки. Этот снимок используется для перечисления процессов и получения информации о них с помощью функций Process32First и Process32Next [106];
- функция Process32First извлекает информацию о первом процессе в снимке, созданном с помощью CreateToolhelp32Snapshot. Информация включает имя исполняемого файла, идентификатор процесса (PID) и другие данные [107];
- функция Process32Next извлекает информацию о следующем процессе в снимке, созданном с помощью CreateToolhelp32 Snapshot. Используется в сочетании с Process32First для перечисления всех процессов в системе [108].

Для получения списка всех служб, зарегистрированных в системе, можно использовать функцию EnumServices. Она обращается к диспетчеру управления службами (SCM) и позволяет перечислять службы на локальном или удаленном компьютере [109].

Косвенные методы определения существования объектов основаны на анализе их активности. Например, для определения

списка процессов можно использовать информацию о связанных с ними окнах, портах, уникальных идентификаторах (handles), которые операционная система использует для управления и отслеживания различных ресурсов и объектов, и т.д. Такие методы зависят от активности объекта и функциональности, заложенной программистом.

Принципиальные различия между методами заключаются в том, что прямой метод позволяет обнаруживать любые работающие объекты, загруженные в память, а косвенный метод выявляет только функционирующие объекты. Таким образом, загруженный, но неактивный объект может быть обнаружен прямым методом, тогда как косвенный метод его не выявит.

5.2. ОБНАРУЖЕНИЕ СКРЫТЫХ ДРАЙВЕРОВ ЯДРА

Драйверы, выполняясь в высокопривилегированном режиме ядра, обладают широкими возможностями сокрытия и способны вмешиваться как в работу ОС, так и в функционирование средств защиты. Функции скрытых драйверов широко используются в шпионских программах [110], поэтому приоритетной задачей кибербезопасности является обнаружение скрытых драйверов.

Для создания стойких к противодействию методов обнаружения скрытых объектов важно понимать:

- как работают драйверы в ОС Windows и как их можно скрыть;
- как работают существующие способы обнаружения скрытых драйверов;
- перспективные варианты развития техник сокрытия и обнаружения.

После загрузки драйвера в память ядра происходит копирование и отображение (mapping) его исполняемого файла в память ядра. Информация о драйвере добавляется в несколько системных списков ОС, после чего управление передается функции DriverEntry загруженного драйвера [15].

Организация структур драйверов ядра в операционной системе приведена на рис. 12. В верхней части рисунка показаны два связанных системных списка структур драйверов: «Системный список № 1 структур драйверов» и «Системный список № 2 структур драйверов». В каждом списке содержатся структуры

драйверов A, B и C, но имеются различия в методах организации и (или) использования этих списков.

В нижней части рисунка показаны загруженные исполняемые файлы драйверов A, B и C в формате PE (.sys), который характерен для драйверов в Windows. Эти файлы представляют собой код, загруженный в память и выполняющийся в контексте ядра ОС.

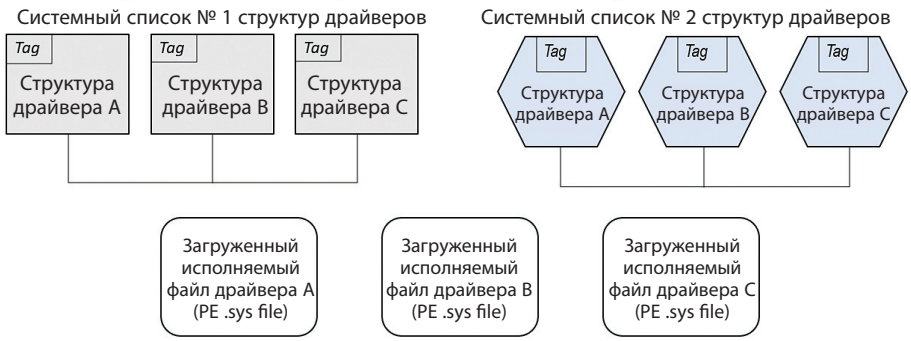


Рис. 12. Структуры драйверов ядра в дампе памяти и их загрузка в системных списках

Штатные средства ОС Windows используют один из этих списков для получения информации о загруженных драйверах [1]. Предположим, что список № 1 — это список, используемый функцией, например, NtQuerySystemInformation с входным параметром SystemModuleInformation [11]. Поскольку этот механизм основан на проходе по списку, он уязвим к методам противодействия анализу. В работе [15] показано, что путем изменения связей между структурами объектов можно скрыть драйвер от обнаружения средствами ОС.

Альтернативные подходы к обнаружению можно разделить на две категории в зависимости от субъекта поиска: первая с применением структур драйверов ОС и вторая с использованием содержимого файлов драйверов в памяти. Их классификация приведена на рис. 13. Первый подход в свою очередь можно разделить на две группы: на основе связей между структурами и на основе сигнатурного поиска структур драйверов. Второй подход к обнаружению файлов драйвера в памяти может быть также разбит на два: сигнатурный и статистический.

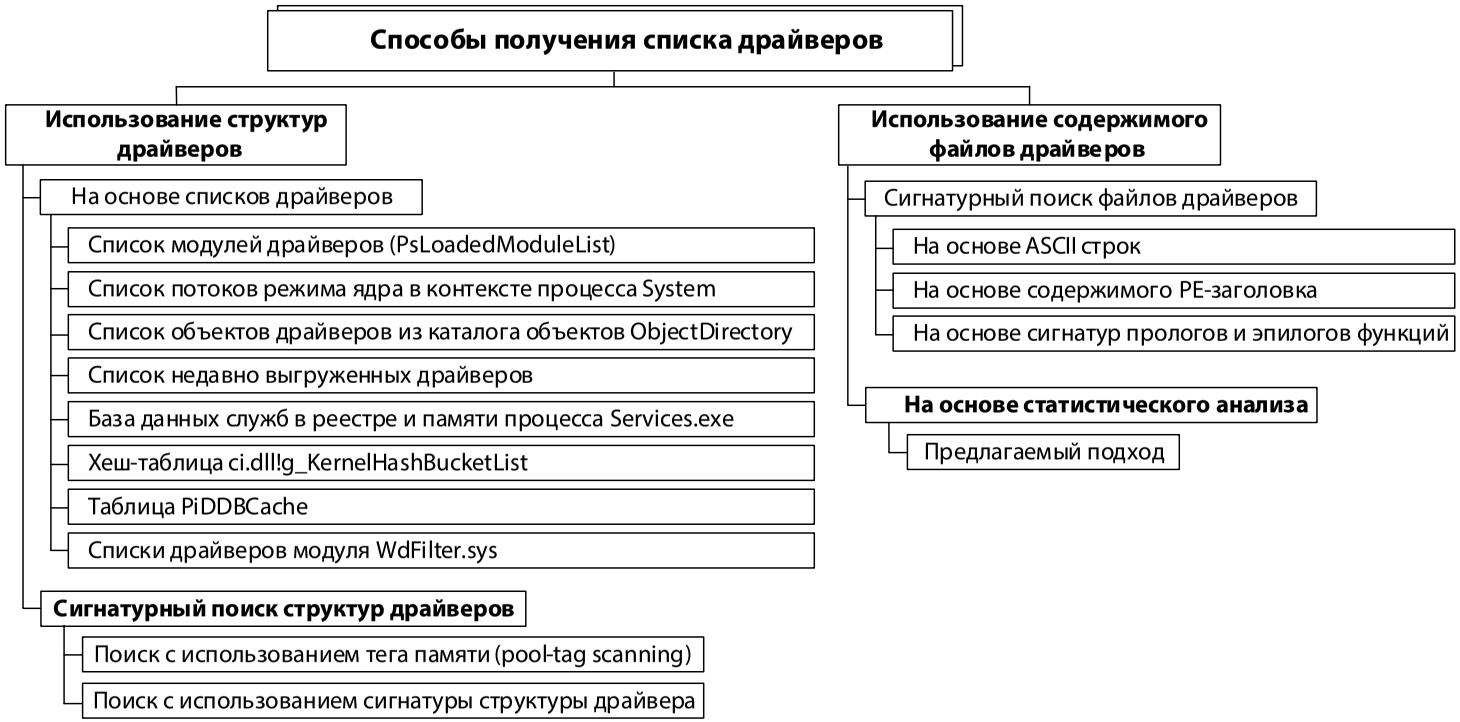


Рис. 13. Классификация методов обнаружения драйверов

Формирование второго списка для обнаружения драйверов неразрывно связано с возможными техниками противодействия нарушителей.

Для моделирования наиболее сложного случая рассмотрим последовательно сценарий применения данных техник сокрытия.

1. Нарушитель удаляет структуры драйверов из соответствующих списков (рис. 14).

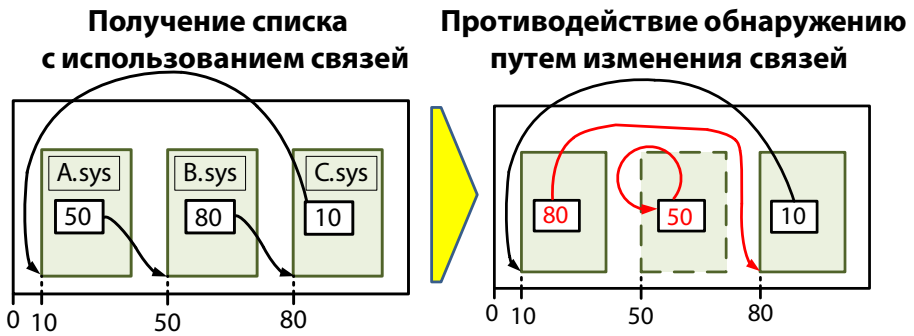


Рис. 14. Удаление связей между структурами

2. Эксперты по безопасности могут применить статические и динамические сигнатуры для обнаружения структур объектов вне зависимости от того, находятся ли эти структуры в списке или нет. В этом случае нарушители могут противодействовать сигнатурному поиску путем перезаписи полей удаленной структуры (рис. 15).

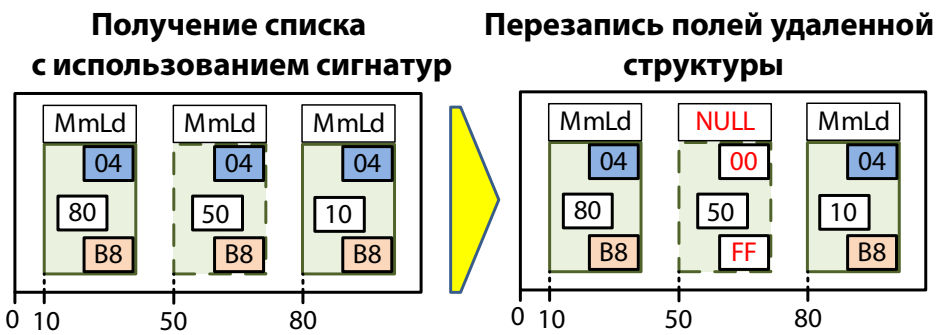


Рис. 15. Перезапись содержимого полей структуры для противодействия сигнатурному поиску структур в памяти

3. Использование содержимого PE-файла для обнаружения скрытого драйвера. В случае, когда структуры объектов становятся непригодными для поиска, эксперты по безопасности могут обратиться к анализу содержимого PE-файлов (Portable Executable) для выявления скрытых драйверов. В этом случае возможны следующие источники для формирования сигнатур:

- заголовок PE-файла. Характерные байтовые последовательности и строки, такие как 'MZ', 'PE' или 'This program cannot be run in DOS mode', могут быть индикаторами присутствия исполняемого файла;
- таблица импорта. Строки, содержащие вызовы системных функций, например 'ZwOpenFile', могут указывать на наличие секции файла драйвера;
- исполнимый код драйвера. Последовательности байтов, соответствующие прологу и эпилогу функций, также могут служить сигнатурами для обнаружения.

Однако нарушители могут использовать различные методы для противодействия этим техникам. Среди них:

- перезапись заголовка и содержимого секций. Изменение содержимого PE-заголовка затрудняет его идентификацию;
- обфускация кода драйвера. Применение методов шифрования и запутывания кода затрудняет использование известных шаблонов.

4. Использование битовой энтропии Шеннона для обнаружения обфусцированного кода. Для выявления обфусцированного кода эксперты могут использовать метод анализа битовой энтропии Шеннона. Высокая энтропия обычно указывает на то, что код был зашифрован или упакован (сжат).

Вредоносное ПО может противодействовать этому методу с помощью техник «разбавления» исполняемого кода. Злоумышленники вставляют в код дополнительные фрагменты данных с низкой энтропией, чтобы снизить рассчитываемую энтропию. При этом:

- длина и количество вставок варьируются, чтобы избежать детектирования самих вставок;
- содержимое вставок подбирается таким образом, чтобы снизить энтропию, но не достигать нулевого значения, так как это

также может быть индикатором, который помогает обнаружению.

Эти техники позволяют нарушителям усложнить задачу экспертам по обнаружению скрытого объекта. В конце раздела 1 будет представлен анализ методов для обнаружения такого сложного сокрытия.

5.3. ПОДХОДЫ НА ОСНОВЕ СРАВНЕНИЯ СОДЕРЖИМОГО РАЗЛИЧНЫХ СПИСКОВ ДРАЙВЕРОВ

Для получения списка драйверов могут быть использованы различные списки операционной системы. Можно выделить следующие наиболее популярные связанные списки ОС, которые используются для обнаружения драйверов [38]:

1) список модулей драйверов (PsLoadedModuleList);

2) список потоков режима ядра в контексте процесса System (Kernel-mode threads);

3) список объектов драйверов из каталога объектов (Object Directory);

4) список недавно выгруженных драйверов (Recently unloaded drivers);

5) список структур сервисов в контексте процесса Services.exe и (ServiceDatabase);

6) база данных установленных служб в системном реестре (Database of Installed Services in the system registry);

7) хеш-таблица драйверов ядра, реализованная в модуле ci.dll (g_KernelHashBucketList);

8) таблица PiDDBCache — недокументированная структура, используемая для хранения информации о загруженных драйверах;

9) списки драйверов RuntimeDriversArray и RuntimeDriversList модуля WdFilter.sys.

Ниже рассмотрим эти списки более подробно.

1. Список модулей драйверов (PsLoadedModuleList) содержит информацию обо всех загруженных модулях драйверов в системе. Он хранится в памяти ядра и используется для управления загрузкой и выгрузкой драйверов. Это необходимо для управления зависимостями между драйверами, обеспечения корректной загрузки драйверов.

Список PsLoadedModuleList используется встроенной WinAPI функцией ZwQuerySystemInformation. Этот двунаправленный список образован структурами KLDR_DATA_TABLE_ENTRY, которые содержат данные о каждом модуле ядра. Плагин модулей Volatility [99] использует этот список.

Метод обхода — сканирование структуры PsLoadedModuleList, которая находится в памяти ядра, для получения списка всех активных драйверов.

Для сокрытия структур драйвера применяется метод DKOM, который подробно описан в работах [37, 111]. Метод реализован в функции DhpHideModule экспериментального образца Dementia [54, 95] для демонстрации возможностей сокрытия драйвера.

2. Список потоков режима ядра от процесса 'System' (Kernel-mode threads). Процесс 'System' с PID 4 включает в себя список потоков режима ядра, содержащих структуры ETHREAD [131]. Эти структуры содержат информацию о потоках, работающих в режиме ядра.

Метод обхода — сканирование всех потоков, запущенных от имени процесса 'System'. Плагин orphan от Volatility [99] использует этот список для анализа и обнаружения потоков режима ядра.

Для сокрытия структур драйвера применяется метод DKOM-patching. В этом случае злоумышленник перезаписывает данные, используемые для работы инструмента, из соответствующей структуры ETHREAD, например поле StartAddress. В результате измененная структура становится непригодной для дальнейшего исследования, что исключает обнаружение драйвера в памяти. Идея этой атаки состоит в том, что нарушитель не может предотвратить факт обнаружения соответствующей структуры в памяти, но он может сделать найденную структуру непригодной для дальнейшего анализа [113].

3. Список объектов драйверов из каталога объектов (ObjectDirectory) содержит структуры различных объектов, включая объекты драйверов DRIVER_OBJECT, и используется для управления этими объектами в системе, предоставляя структурированную организацию структур объектов в виде дерева объектов [114].

Для получения информации об объектах драйверов служит функция NtQueryDirectoryObject, которая позволяет получать сведения о загруженных драйверах путем обхода структур DRIVER_OBJECT, находящихся в каталоге объектов [115]. Для открытия каталога перед вызовом NtQueryDirectoryObject используется функция NtOpenDirectoryObject, которая возвращает дескриптор нужного каталога, например каталога \Driver.

Для сокрытия структур DRIVER_OBJECT нарушители могут использовать метод DKOM-unlinking. Этот метод позволяет удалить соответствующие структуры из дерева объектов, делая их невидимыми для инструментов анализа [116–118]. Примером такой атаки является применение функции StealthInitializeLateMore в AntiMida [119].

4. Список недавно выгруженных драйверов (Recently unloaded drivers) содержит имена, время, начальный и конечный адреса драйверов, которые были выгружены из системы [112, 120].

Метод обхода — в Windows Research Kernel (WRK) переменная MmUnloadedDrivers хранит адрес начала этого списка, а функция MmLocateUnloadedDriver позволяет найти указанный виртуальный адрес в списке [115, 121]. Также этот адрес можно получить, вызвав функцию ReadDebuggerData с параметром DEBUG_DATA_MmUnloadedDriversAddr. Плагин Volatility unloadedmodules использует этот список для извлечения информации о недавно выгруженных драйверах.

Преимущества — позволяет отслеживать драйверы, которые были выгружены недавно, что важно для анализа инцидентов и диагностики, а также используется в инструментах для восстановления истории загрузки драйверов.

Для сокрытия информации о выгруженных драйверах злоумышленники могут использовать метод DKOM-patching для перезаписи полей, используемых для работы инструментов обнаружения, делая список непригодным для анализа. Это может включать изменения адресов и других данных, что затрудняет обнаружение драйверов [112].

5. Списки структур записей служб (ServiceDatabase) — после установки драйвера с помощью SCM информация о драйвере добавляется в несколько списков, в том числе в список структур

записей служб (SERVICE_RECORD) находящегося в пользовательской области памяти процесса Services.exe [112, 122].

Информацию о загруженных драйверах можно получить из ServiceDatabase с помощью функции EnumServicesStatus с параметром SERVICE_KERNEL_DRIVER [109], в качестве альтернативы можно использовать функцию EnumServicesStatusEx, которая предоставляет расширенную информацию, а также непосредственно обход списка структур.

Сокрытие структур драйвера — использование метода DKOM-unlinking для сокрытия записей о службах. Этот метод позволяет удалить структуру записи службы из списка, делая ее невидимой для инструментов анализа. Примеры таких атак описаны в анализе трояна Blazgel [123] и в других источниках [112].

6. База данных установленных служб в системном реестре (Database of Installed Services in the system registry). При регистрации драйвера с помощью диспетчера управления службами (SCM) информация о нем также добавляется в базу данных, которая находится в следующем разделе реестра HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services [124, 125]. Этот раздел содержит подраздел для каждой установленной службы и драйвера. Имя подраздела соответствует имени службы, указанному при установке службы с помощью функции CreateService.

Информацию о зарегистрированных драйверах можно получить с помощью RegEnumKeyEx [126]. Эта функция позволяет перечислить все ключи в указанной ветке реестра и извлечь соответствующую информацию.

В целях сокрытия информации о драйвере нарушители могут использовать различные методы для сокрытия информации о драйверах в реестре. Одним из таких методов является удаление соответствующего ключа реестра, содержащего информацию о драйвере. Другим методом является установка драйвера-фильтра, который скрывает значение реестра. Пример удаления информации о драйвере в реестре описан в публикации [117].

7. Хеш-таблица ci.dll!g_KernelHashBucketList. Отдельно стоит отметить хеш-таблицу g_KernelHashBucketList, которая используется компонентом Code Integrity в Windows, реализо-

ванным в модуле ядра ci.dll. Эта структура хранит записи о загруженных драйверах и служит для проверки их целостности и подлинности. Манипуляции с этой хеш-таблицей, например ее очистка (техника ClearKernelHashBucketList), позволяют обходить механизмы аутентификации драйверов и скрывать присутствие неавторизованных и вредоносных модулей в системе.

8. Таблица PiDDBCache. Отдельного внимания заслуживает таблица PiDDBCache, которая представляет собой внутреннюю структуру ядра Windows, используемую для хранения информации о загруженных драйверах. Эта таблица содержит записи с данными о каждом драйвере, включая имя файла, временные метки и статус загрузки. Манипуляции с этой таблицей, такие как удаление или подмена записей, позволяют скрывать факты загрузки вредоносных модулей.

9. Списки драйверов модуля WdFilter.sys. Кроме того, важную роль играют структуры, связанные с управлением загруженными драйверами, такие как RuntimeDriversCount, RuntimeDriversArrayLength, RuntimeDriversArray и Runtime DriversList. Эти структуры используются в ядре Windows, в частности драйвером безопасности WdFilter.sys из состава Windows Defender, для учета, контроля и управления состоянием динамически загружаемых драйверов во время работы системы. Манипуляции с этими структурами позволяют злоумышленникам скрывать присутствие драйверов, обходя механизмы мониторинга и защиты.

10. Система трассировки событий (ETW — Event Tracing for Windows). В ряде сценариев, особенно при проведении проактивного мониторинга, например, для систем защиты класса Sandbox и EDR, эксперты могут использовать возможности системы трассировки событий (ETW) для журналирования событий загрузки и выгрузки драйверов до активации вредоносного модуля [127]. Основным провайдером в данном контексте выступает Microsoft-Windows-Kernel-PnP, генерирующий события с идентификаторами 2003 (Driver Loaded) и 2004 (Driver Unloaded) при каждой попытке загрузки или выгрузки драйвера — независимо от способа инициации: через SCM, NtLoadDriver, PnP Manager или вызов IoCreateDriver.

Особую ценность ETW представляет в сценариях, когда трассировка запускается на раннем этапе загрузки системы, например, через boot-time tracing. Это позволяет фиксировать даже драйверы, загружаемые до инициализации диспетчера служб, включая ELAM-драйверы, а также те, что впоследствии скрываются из списков ядра (например, через DKOM). Поскольку события генерируются на уровне ядра в момент фактической загрузки, ETW обеспечивает достоверную и устойчивую к постфактум-манипуляциям запись. Сокрытие от ETW требует активного вмешательства: отключения провайдера, подмены обработчиков или фильтрации событий на уровне ядра, что само по себе требует высоких привилегий и оставляет артефакты. Важно отметить, что вредоносный драйвер может не только деактивировать ETW-сессии, но и в случае использования файловых трассировок или системного Event Log — попытаться удалить или очистить соответствующие журналы, что также может служить индикатором компрометации. Атаки на ETW были представлены в контексте противодействию работе AV/EDR в параграфе 3.2.

Систематизация различных списков и структур, используемых в системах для мониторинга и анализа драйверов и служб, а также примеры их использования и потенциальные атаки приведены в табл. 3.

Таблица 3

Обзор системных структур драйверов и методов их сокрытия

Название списков и структур	Примеры использования списков	Примеры атак
Список модулей драйверов, PsLoadedModuleList, KLDR_DATA_TABLE_ENTRY	ZwQuerySystemInformation Volatility's modules plugin	DKOM unlinking
Список объектов драйверов из каталога объектов, ObjectDirectory, DRIVER_OBJECT	ZwQueryDirectoryObject	
Список записей служб, SERVICE_RECORD	EnumServicesStatus	

Окончание табл. 3

Название списков и структур	Примеры использования списков	Примеры атак
Список системных потоков режима ядра, ETHREAD	Volatility's orphan plugin	DKOM patching
Список выгруженных драйверов MmUnloadedDrivers, UNLOADED_DRIVERS	ReadDebuggerData Volatility's unloadedmodules plugin	
Хеш-таблица ci.dll!g_KernelHashBucketList	Доступ через неэскортируемые символы	
Списки драйверов RuntimeDriversArray и RuntimeDriversList модуля WdFilter.sys		
Таблица PiDDBCache		
Service Record List в реестре	RegEnumKeyEx	Удаление ключей реестра

В результате можно сделать вывод о том, что подходы, основанные на связях между структурами, подвержены таким техникам противодействия, как DKOM-unlinking, DKOM-patching и удаление ключей реестра. Далее будут проанализированы способы обнаружения драйверов, устойчивые к рассмотренным техникам и пригодные для обнаружения в условиях противодействия.

5.4. ПОДХОДЫ НА ОСНОВЕ СИГНАТУРНОГО ПОИСКА ФРАГМЕНТОВ СТРУКТУР И МОДУЛЕЙ ДРАЙВЕРОВ В ПАМЯТИ

Как было сказано ранее, обнаружение скрытых драйверов основано на сравнении двух списков драйверов, полученных независимыми способами.

Первый список обычно получают с помощью штатных функций ОС, ожидая, что нарушители удалят информацию из этого списка в приоритетную очередь, иначе вопрос анализа драйвера не будет представлять проблемы: подозрительный драйвер будет уже присутствовать в списке.

Для формирования второго списка драйверов можно использовать ряд способов, например, на основе сигнатурного поиска.

По типу используемых сигнатур можно выделить два способа обнаружения:

1) обнаружение структур драйверов;

2) обнаружение загруженных драйверов.

Каждый из этих подходов имеет свои особенности и применимые сценарии использования.

5.4.1. Методы обнаружения структур драйверов на основе сигнатур

При загрузке драйверов ОС создает ряд структур, описывающих этот драйвер, и добавляет их в соответствующие системные списки. Поскольку ОС использует единую функцию для создания и инициализации структур драйверов одного типа, можно заключить, что все загруженные структуры драйверов одного типа имеют сходное байтовое строение, которое может быть использовано для их обнаружения.

Отличительной чертой данного способа является то, что он позволяет найти структуру драйверов вне зависимости от того, добавлена ли эта структура в список драйверов или была удалена нарушителем в целях сокрытия (DKOM-unlinking).

Поиск по сигнатурам в некоторых случаях может быть значительно ускорен за счет того, что структуры драйверов, например, DRIVER_OBJECT, зачастую расположены в памяти близко друг к другу [30].

Можно выделить два способа обнаружения структур драйверов:

1) поиск с использованием тега памяти (pool-tag scanning);

2) поиск с использованием сигнатуры структуры драйвера.

1. Сканирование по тегам пула данных (pool-tag scanning). Этот метод опирается на факт, что при выделении памяти для структур одного и того же типа операционная система использует одинаковый тег памяти, который задается параметром в функции ExAllocatePoolWithTag(). Тег памяти представляет собой последовательность из четырех байтов, добавляемую в память до начала выделенного фрагмента памяти, и служит для задач отладки кода, однако разработчики Windows используют непустые теги памяти даже в релизных версиях ОС.

Список тегов пула драйверов ОС Windows приведен в работе [128]. Например, при регистрации драйвера с использованием подсистемы SCM операционная система создает структуру SERVICE_RECORD в пользовательском адресном пространстве процесса SERVICES.EXE и задает значение тега пула, которое хранится в поле Signature структуры ServiceRecord. Значение пула тега равно 0x76724573 или sErv в ASCII. Подробности этой манипуляции приведены в исходном коде Microsoft Windows NT 4.0 [129]. В работе [130] использован поиск фрагментов данных по их тегам для обнаружения потоков Windows.

2. Сканирование с помощью сигнатуры структуры драйвера. Благодаря тому, что структуры одного типа имеют подобное байтовое строение, можно выделить способ формирования структур драйверов на основе поиска набора байтов, соответствующего структуре драйвера. В такую сигнатуру включаются только байты, значения которых совпадают для всех связанных структур в списке. Также, кроме единичных байтов, можно использовать факт того, что в структуры режима ядра входят поля, значения которых связаны с другими структурами режима ядра. Поэтому эти значения превышают значения 0x8000_0000 [131]. Данная особенность описана в работе [37], а затем улучшена в новом исследовании по обнаружению скрытых структур [30].

Системный обзор методов поиска структур драйверов на основе сигнатур представлен в табл. 4.

Таблица 4

Методы поиска структур драйверов на основе сигнатур

Название списков и структур	Примеры сигнатуры	Пример использования сигнатуры
Список модулей драйверов, PsLoadedModuleList, KLDR_DATA_TABLE_ENTRY	Тег сигнатуры MmLd	[132]
Список записей служб, SERVICE_RECORD	Тег сигнатуры sErv или serH	[38, 112]

Окончание табл. 4

Название списков и структур	Примеры сигнатуры	Пример использования сигнатуры
Таблица объектов драйверов, ObjectTable, DRIVER_OBJECT	Тег сигнатуры Driv	[112]
	Фиксированная байтовая сигнатура	[37]
	Динамическая битовая сигнатура	[30]

Обнаружение структур драйверов на основе сигнатур является важным шагом в развитии техник обнаружения, поскольку стало возможным обнаружить драйверы, которые были скрыты с помощью DKOM-unlinking. Однако предложенные способы в свою очередь также неспособны обнаружить структуры, которые были дополнительно перезаписаны с помощью DKOM-patching, так что все информативные поля оказались перезаписаны. В таком случае структуры драйверов больше непригодны для обнаружения.

Далее будут рассмотрены подходы к обнаружению драйверов, которые не используют структуры драйверов и, таким образом, способны обнаружить скрытый драйвер даже в условиях работы двух упомянутых выше техник сокрытия.

5.4.2. Методы обнаружения загруженных драйверов в памяти на основе сигнатур

В случае если структуры драйверов были скрыты с использованием DKOM-unlinking или DKOM-patching, для построения альтернативного списка (второго списка) драйверов возможно использовать исполнимые фрагменты модулей драйверов в памяти.

Можно выделить три типа сигнатур для поиска фрагментов драйверов в памяти:

1) на основе строк ASCII;
2) на основе содержимого PE-заголовка;
3) на основе сигнатур прологов и эпилогов функций.

1. Строки ASCII. Этот тип сигнатур включает в себя различные строки из исполняемого файла. Самый распространенный пример такой ASCII-строковой сигнатуры — This program cannot be run in DOS mode из PE-заголовка. Эта строка была использована при анализе образца ВПО Stuxnet. Также применяются имена функций из таблицы импорта в виде ASCII-строк. Пример использования данной сигнатуры для анализа упакованных PE-файлов включает поиск заданных функций в таблице импорта.

Техника противодействия. Нарушители в целях сокрытия могут перезаписать соответствующие участки памяти и также динамически получать адреса экспортированных функций, что уменьшит размер таблицы импорта.

2. Сигнатуры PE-заголовка. Второй тип сигнатур основан на содержимом PE-заголовка. Эта сигнатура представляет собой байтовый шаблон, включающий значения полей PE-заголовка и их смещения от начала файла [133–135]. PE-заголовок начинается с DOS-заголовка, содержащего сигнатуру MZ (0x4D 0x5A) и поле e_lfanew — смещение до основного PE-заголовка, который начинается с подписи PE\0\0 (0x50 0x45 0x00 0x00). Далее идут заголовок файла COFF и опциональный заголовок, содержащие информацию о целевой архитектуре, количестве секций, времени компиляции, размерах и характеристиках образа, а также другие параметры, необходимые для корректной загрузки и выполнения модуля.

Техника противодействия. Для обхода обнаружения на основе сигнатуры PE-заголовка нарушители могут перезаписывать содержимое PE-заголовка, поскольку после загрузки эти данные уже не используются загрузчиком и системой. Такая модификация затрудняет анализ и обнаружение драйверов по их PE-заголовкам в памяти [117].

3. Сигнатуры прологов и эпилогов функций. Третий тип сигнатур основан на содержимом исполнимого файла. Драйверы содержат исполнимые инструкции машинного языка или операционные коды (опкоды). Архитектура инструкций (ISA) включает спецификацию набора опкодов. Различные драйверы имеют одинаковые фрагменты опкодов, так как они выполняются на одной и той же архитектуре центрального процессора. Наиболее попу-

лярным примером использования опкодов в качестве сигнатуры является начало и конец функций. Соответствующие наборы опкодов также известны как пролог и эпилог функции и зависят от соглашения о вызовах [136]. Прологи и эпилоги функций — код, который выполняется в начале и в конце функции соответственно. Пролог обычно включает в себя код, сохраняющий регистры и выделяющий память на стеке, а эпилог — код, восстанавливающий регистры и освобождающий память. Прологи функций обычно содержат инструкции для подготовки стека и регистров. Эпилоги функций отвечают за очистку стека и восстановление контекста перед возвратом управления вызывающей функции.

Существует три популярных соглашения о вызовах, которые используются с языками программирования C/C++: stdcall, cdecl и thiscall [137]. Для детектирования различных вредоносных программ используются сигнатуры, основанные на опкодах [138–140].

Техника противодействия. Нарушители могут применять возможности обфускации [141] для сокрытия прологов и эпилогов функций от обнаружения, а также вставлять дополнительные инструкции в эти участки, в том числе для уменьшения энтропии [142].

В результате можно утверждать, что методы поиска загруженных драйверов по сигнатурам, несмотря на устойчивость к изменениям в структуре самих драйверов, уязвимы к другим способам сокрытия. Нарушители могут противодействовать обнаружению путем изменения данных, используемых для выявления сигнатур (например, строки ASCII и сигнатуры заголовков PE-файлов). Более того, для сокрытия содержимого драйвера в памяти могут использоваться шифрование, сжатие и обфускация. Далее будут представлены методы обнаружения, пригодные для выявления драйверов даже в таких сложных случаях сокрытия.

5.5. МЕТОДЫ ОБНАРУЖЕНИЯ ДРАЙВЕРОВ С ИСПОЛЬЗОВАНИЕМ АТРИБУТОВ СТРАНИЦ ПАМЯТИ

Рассмотрим методы, которые можно использовать для обнаружения скрытых драйверов без использования сигнатурного поиска структур драйверов или связанных с ними артефактов исполняемых модулей драйверов в памяти.

Один из подходов к обнаружению скрытых драйверов заключается в анализе атрибутов страниц памяти. Каждый раз, когда драйвер загружается в память, ОС выделяет память для исполняемого файла и помечает эти страницы как исполняемые с помощью флага, хранящегося в каталоге страниц памяти.

Алгоритм обнаружения включает в себя следующие шаги:

1) получение списка адресов страниц памяти с установленным атрибутом «выполнение» путем последовательного обхода каталога страниц;

2) получение диапазонов адресов для загруженных драйверов ядра, включая само ядро ОС;

3) выявление скрытых драйверов осуществляется путем сопоставления диапазонов адресов, полученных на шагах 1 и 2. Любые страницы исполняемой памяти, не входящие в диапазоны драйверов, могут указывать на наличие скрытого исполнимого кода драйверов.

Нарушители со своей стороны могут противодействовать такому обнаружению. Один из возможных вариантов сокрытия был реализован авторами руткита Shadow Walker, который устанавливал свой обработчик Page Fault и сбрасывал соответствующий атрибут для страниц памяти, содержащих скрываемый код.

В качестве техники обнаружения такого сокрытия можно использовать измерение длительности обращения к страницам памяти. Когда определенный диапазон адресов демонстрирует значительно большее время доступа по сравнению с другими страницами памяти, это может указывать на то, что эти страницы преднамеренно скрыты руткитом. При этом необходимо учитывать, что часть страниц может быть выгружена из памяти на диск, а также использоваться для прямого доступа оборудования, например видеокарты. Кроме того, руткит может размещать фрагменты своего кода в незанятых областях исполняемой памяти, выделенной для других драйверов, код которых не заполняет страницы целиком. Таким образом, руткит может заполнить эти свободные байты в страницах памяти, не выделяя отдельные страницы исполнимой памяти для себя.

Альтернативным вариантом является использование гипервизорных технологий защиты для обнаружения руткитов в ОС и попыток установки сторонних обработчиков Page Fault.

5.6. СТАТИСТИЧЕСКИЕ МЕТОДЫ ОБНАРУЖЕНИЯ ЗАГРУЖЕННЫХ ДРАЙВЕРОВ В ПАМЯТИ

Рассмотрим наиболее сложный случай обнаружения скрытого драйвера в дампе памяти, когда применение сигнатурного поиска структур и артефактов драйверов, а также атрибутов страниц оказывается безрезультатным. В данном случае эксперты по безопасности могут использовать определенные статистические закономерности, которыми обладают фрагменты памяти, содержащие исполнимый код, в отличие от остальных страниц памяти с данными.

Статистическое обнаружение файлов драйверов происходит в два этапа:

1) метод «скользящего окна» — анализ содержимого памяти и расчет различных статистических показателей для каждого смещения памяти или для каждой страницы в памяти;

2) поиск областей памяти с аномальными значениями статистических характеристик, так как именно эти области, вероятнее всего, содержат исполнимый код или шифрованные данные.

5.6.1. Применение энтропии для поиска сжатых и зашифрованных данных

Метод использования статистического анализа памяти для поиска исполняемого кода основан на понятии информационной, или бинарной, энтропии Клода Шеннона [160], рассчитываемой следующим образом:

$$S = -\sum_{i=1}^{255} p_i \cdot \log_2(p_i),$$

где p_i — частота каждого значения байта в файле.

Энтропия — это мера неопределенности или случайности данных, позволяющая различать их типы. Например, с ее помощью можно отличить текстовые данные от содержимого исполняемого (неупакованного) кода, а также выявить упакованные

или зашифрованные данные, характеризующиеся высокой степенью случайности (табл. 5 [143]; рис. 16).

Таблица 5

Среднее значение энтропии для разного типа данных

Тип данных	Среднее значение энтропии
Текстовые данные (plain text)	4,3
Неупакованный исполняемый код (original executable code)	5,1
Упакованный исполняемый код (packed executable code)	6,8
Зашифрованный исполняемый код (encrypted executable code)	7,2

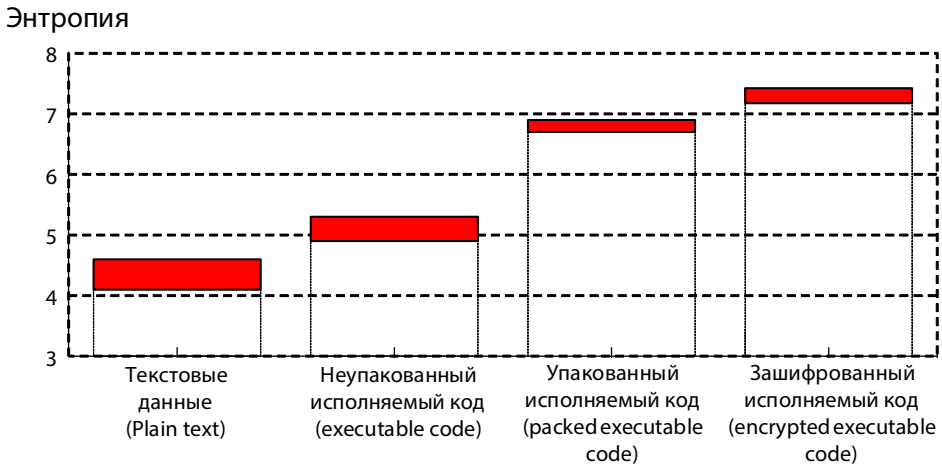


Рис. 16. Применение статистического анализа памяти для поиска исполняемого кода

Анализ значений энтропии позволяет классифицировать содержимое памяти без опоры на фиксированные сигнатуры:

- высокая энтропия указывает на зашифрованные или сжатые данные, а также на машинный код;
- низкая энтропия, напротив, свидетельствует о наличии обычного текста или структурированных данных;
- нулевая энтропия соответствует страницам, заполненным одинаковым байтом, а также пустым страницам.

В результате проведенного обзора открытых источников не найдено фактов использования энтропийного анализа для обнаружения скрытых драйверов в памяти. В то же время анализ энтропии применяется в различных задачах кибербезопасности [144, 145], анализе неизвестных бинарных файлов [146], поиске криптографических ключей в дампе памяти [147], а также для поиска исполняемого кода в следующих случаях [148]:

- в файлах офисных документов [149–152];
- в PDF-файлах [150, 153];
- в сетевом трафике [154];

Анализ значений энтропии может быть применен и для поиска скрытого содержимого драйвера в памяти.

Энтропийный анализ файлов часто используется для классификации файлов, например, чтобы обнаружить зашифрованные или упакованные файлы [155]. Он помогает определить потенциально вредоносные файлы по таким признакам, как высокая энтропия [156].

Энтропийный анализ широко применяется для раннего обнаружения программ-вымогателей (ransomware). Одна из распространенных техник злоумышленников — полное шифрование содержимого файлов в целях блокировки доступа к ним. Обнаружение таких атак базируется на отслеживании изменений значений энтропии, вычисляемой как для всего файла целиком, так и для его отдельных частей, с использованием методов построения цифровых отпечатков файла (File Fingerprinting) и нечеткого хеширования (Fuzzy Hashing).

В ответ на эти методы обнаружения злоумышленники разработали технику intermittent encryption — прерывистого шифрования. При этом файлы шифруются не полностью, а с пропусками: например, шифруются каждые 100 байт, после чего идут 100 байт незашифрованных исходных данных и т.д. Такая техника нарушителей значительно усложняет анализ по энтропии, поскольку перемежающиеся зашифрованные и открытые участки снижают общую энтропию файла и затрудняют выявление внесенных изменений в файл. ВПО LockFile был одним из первых семейств ransomware, использовавших эту технику, шифруя каждые 16 байт файла. Другие группы ransomware, исполь-

зующие intermittent encryption, включают Black Basta, ALPHV (BlackCat), PLAY, Agenda и Qyick. В результате обнаружение измененного файла путем сравнения энтропии становится затруднительным и требуется разработка новых, более совершенных техник защиты.

5.6.2. Метод «скользящего окна» для обнаружения кода в памяти

Использование «скользящего окна» для анализа значений энтропии в файле дампа памяти для выявления участков кода с высокой энтропией схематически показано на рис. 17. Такой анализ помогает идентифицировать участки памяти, которые потенциально могут содержать исполнимый код или зашифрованные данные, поскольку значения энтропии, рассчитанные от соответствующих фрагментов, лежат в диапазоне высоких значений энтропии [157, 158].

Рис. 17. Использование метода «скользящего окна» для поиска исполняемого кода

Метод «скользящего окна» заключается в последовательном рассмотрении блоков данных (окон размером 256 байт) файла на предмет высоких значений энтропии. При движении окна

по файлу вычисляется энтропия для каждого блока. Участки с аномально высокой энтропией могут указывать на зашифрованный или упакованный код.

Существует несколько методов вычисления энтропии, включая формулу бинарной энтропии [143] и другие методы; например, в работе [159] использован анализ энтропии только заголовков файлов. Помимо бинарной энтропии, используются иные статистические методы и характеристики, такие как среднее арифметическое, критерий Хи-квадрат и вес Хемминга [160]. В работе [149] отмечено, что для классификации файлов могут быть полезны следующие статистики: среднее значение, коэффициент эксцесса (kurtosis), распределение средних значений, среднеквадратическое отклонение и распределение среднеквадратических отклонений. Одной из перспективных метрик для оценки структурированности данных является сложность по Колмогорову [161].

При анализе дампа памяти можно применять метод «скользящего окна» или его модификации. Например, авторы работы [162] предложили построить граф зависимости значений байта от его смещения и вычислить частотное распределение байтов для различных фрагментов. Для анализа они использовали размеры фрагментов 100×100 и 2000×50. Метод визуализирует и анализирует байты файлов, автоматически обнаруживая случайные данные (или близкие к свойствам равномерно распределенной случайной последовательности, РРСП) с помощью статистических значений и вейвлет-преобразований.

Подход к использованию метода «скользящего окна» был предложен в работе [149], где каждый блок имеет длину 256 байт и перекрывается на 4 байта. К каждым 4 байтам потока энтропии применяется оконное преобразование Фурье. Для дальнейшего анализа дампа памяти предлагается использовать оконное преобразование Фурье (Short-Time Fourier Transform, STFT) [151, 153] или вейвлет-преобразование [162].

Методы вычисления энтропии, такие как анализ биграмм и триграмм, также могут быть применены [156, 160]. Например, в исследовании [160] описывается метод автоматической классификации различных типов файлов, таких как случайные данные, текст, машинный код и изображения. Для успешного решения за-

дачи классификации авторы сначала определили пороговые значения для разнообразных типов данных, используя контрольные образцы для их вычисления. В результате была выполнена классификация с учетом соответствующих показателей ошибок I и II рода. Альтернативный пример решения классификационной задачи с использованием энтропии приведен в работе [163]. Авторы предложили метод классификации алгоритмов упаковки в сжатых исполняемых файлах, используя метод SAX (Symbolic Aggregate Approximation), который хорошо подходит для работы с большими объемами данных и упрощения сложных паттернов, а также методы машинного обучения (наивный байесовский классификатор и опорные векторы SVM). Эксперименты с 466 программами и 15 алгоритмами упаковки показали высокую точность (94,2%).

Современные методы искусственного интеллекта (ИИ) для классификации файлов демонстрируют значительное улучшение. В отличие от утверждения, что ИИ-методы недостаточно точны из-за ошибок I и II рода [149], недавние исследования показывают, что современные ИИ-алгоритмы значительно улучшили свою точность и полноту [164].

5.6.3. Техники визуализации для ручного анализа памяти

Для анализа памяти вручную часто применяют техники визуализации. Например, можно использовать побайтовую визуализацию, где байты файла окрашиваются в зависимости от их значений (например, 0x00 — черный, 0xFF — белый), что помогает визуально обнаруживать нулевые страницы и секции исполняемых файлов [149]. Пример визуализации бинарных данных файла с помощью инструмента Binvis.io представлен на рис. 18. Цветовая кодировка указана внизу рисунка. Разноцветные области указывают на данные с высокой вариацией и случайностью, такие как исполнимый код. Темные прямоугольные области в верхнем левом углу и внизу изображения могут указывать на нулевые страницы или зарезервированные области. Используя этот подход, можно легко найти в памяти нулевые страницы, а также отдельные секции исполняемого файла — .text, .data, .rsrc.

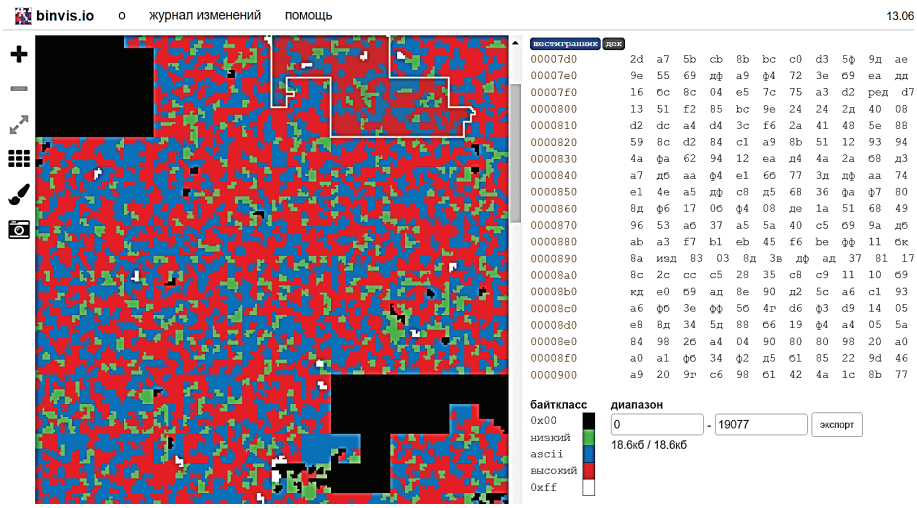


Рис. 18. Техника визуализации для анализа дампа памяти

Можно также использовать кривые, такие как зигзаг и порядок Гильберта, для визуализации и поиска исполнимого кода и зашифрованных блоков [165].

Визуализация файлов и дампов памяти с последующим анализом с помощью графических библиотек является перспективным направлением в кибербезопасности, подтвержденным различными проектами [146, 166]. Кроме того, возможна комбинированная эксплуатация различных методов, например сочетание сигнатурного и статистического анализа, что повышает точность обнаружения угроз [167].

5.6.4. Анализ техник нарушителей для противодействия статистическим методам обнаружения исполнимого кода в памяти

Нарушители применяют различные техники для сокрытия своего исполнимого кода в памяти от методов обнаружения, использующих энтропию. В целях уменьшения рассчитываемой энтропии кода нарушители могут использовать ряд техник. Наиболее популярной техникой является добавление в код блоков данных с низкой или нулевой энтропией. Она была применена в программе Zeus, которая добавляла вставки «нулевого» содержимого в код [168, 169].

В рамках исследовательских работ был реализован экспериментальный прототип ПО, который преднамеренно уменьшал энтропию программы путем вставки в код фрагментов с низкой, но не нулевой энтропией. В результате была успешно продемонстрирована возможность создания преднамеренного «разбавленного» по памяти исполнимого кода (специально структурированный код низкой энтропии).

Для обнаружения подобных специально замаскированных программ был предложен метод поиска и удаления блоков данных с уменьшенной энтропией [157]. Затем энтропию измеряли повторно с использованием байтовых гистограмм для анализа содержимого файла. Однако байтовые гистограммы также уязвимы к определенным методам, которые могут противодействовать их анализу, о чем будет рассказано далее.

Приведенный анализ показывает, что многие популярные подходы к обнаружению драйверов не всегда способны работать в условиях противодействия. Список популярных методов обнаружения драйверов и методов противодействия приведен в табл. 6.

Таблица 6

Популярные методы обнаружения драйверов и методы противодействия

Способ обнаружения драйверов		Метод противодействия
Использование структур драйверов	С помощью связей между структурами	DKOM unlinking
	Сигнатурное обнаружение структур	DKOM patching
Использование содержимого драйверов в памяти	Сигнатурное обнаружение драйверов	DKOM patching & PE packing
	Статистическое обнаружение драйверов	Memory patching

Далее будет представлен пример развития методов сигнатурного анализа для обнаружения структур процессов и драйверов в памяти. Этот метод разработан так, чтобы быть устойчивым к известным способам сокрытия драйверов.

5.7. ПРИМЕНЕНИЕ СТАТИЧЕСКИХ И ДИНАМИЧЕСКИХ СИГНАТУР ДЛЯ ОБНАРУЖЕНИЯ ПРОЦЕССОВ И ДРАЙВЕРОВ

Рассмотрим сценарий обнаружения структур процессов и драйверов, скрытых с помощью техники DKOM-unlinking. Предполагается, что данная манипуляция нарушителей осталась без реакции со стороны штатного средства защиты PatchGuard.

Для обнаружения структур процессов EPROCESS будет представлена идея применения динамических байтовых сигнатур (ДБаС) и динамических битовых сигнатур (ДБиС). Для обнаружения структур драйверов (DRIVER_OBJECT) будет предложена идея применения статической байтовой сигнатуры (СБС) на основе баллов.

Конкурентными преимуществами этой техники обнаружения являются:

- переносимость на различные версии ОС, поскольку не используются проверки на основе байтовых значений, расположенных на фиксированных смещениях;
- способность работы в условиях преднамеренно измененных нарушителями структур. Для обнаружения достаточно лишь 80% от совпадения с шаблоном, и это пороговое значение также может быть изменено аналитиком;
- возможность применения для новых и неизвестных структур.

Особенность перечисленных выше методов заключается в их способности обнаруживать все объекты заданного типа в системе, включая скрытые методом DKOM-unlinking. Основа подхода — поиск структур в памяти ядра, соответствующих шаблонам (паттернам) сигнатур. Динамические сигнатуры формируются с использованием содержимого структур, присутствующих в памяти. Сравнение текущих структур в памяти с шаблонными сигнатурами позволяет выявлять скрытые объекты.

5.7.1. Пример динамической битовой сигнатуры для обнаружения процессов

Рассмотрим метод ДБаС для обнаружения структур процессов, а затем его развитие — метод ДБиС.

Для обнаружения скрытых процессов методом DKOM необходимо проанализировать структуры EPROCESS. Метод заклю-

чается в поиске в памяти ядра структур, похожих на структуры процессов, с использованием сигнатур.

Известно, что при создании структур процессов операционная система использует единую функцию и заполняет соответствующие структуры процессов единым образом. Данная закономерность была подтверждена экспериментально и легла в основу предлагаемого метода обнаружения.

Визуальное представление различных байтов в структуре объектов представлено на рис. 19, где квадраты разного цвета символизируют байты. Квадраты идентичных цветов указывают на совпадение значений байтов, что облегчает идентификацию совпадающих байтов в структурах по цветовой маркировке.

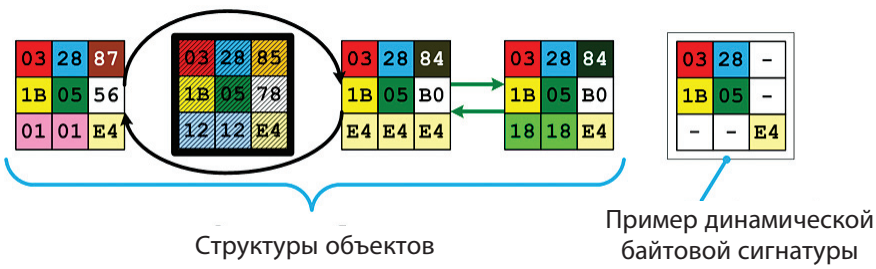


Рис. 19. Вид структуры объектов в памяти: структуры одного типа имеют сходное внутреннее байтовое строение

На рис. 19 представлены четыре блока, каждый из которых условно представляет собой структуру объекта в памяти: блоки 1, 3 и 4 (слева направо) находятся в двухсвязном списке, а блок 2 удален из этого списка путем изменения связей между структурами. При этом наглядно показано, что содержимое удаленного блока по своему байтовому строению остается аналогичным объектам, продолжающим находиться в списке. Задача — найти удаленный блок, используя факт того, что его содержимое имеет похожее строение с блоками, находящимися в списке. Каждый цветной квадрат внутри блока представляет байт данных, а цифры/буквы внутри квадратов — значения этих байтов. Из всех структур объектов выбираются общие байты, которые будут использоваться для создания динамической байтовой сигнатуры. Эти общие байты показаны на примере динамической байтовой

сигнатуры (четвертый блок справа). Сигнатура создается из одинаковых байтов во всех структурах объектов. Например, красный байт «03» и желтый байт «1B» присутствуют во всех структурах объектов, поэтому они включены в сигнатуру.

Методика обнаружения скрытого процесса включает следующие шаги:

1) создание ДБаС, которая охватывает только те байты, которые одинаковы для всех структур EPROCESS;

2) применение вероятностного подхода к поиску ДБаС в памяти ядра обеспечивает выявление всех структур EPROCESS, вне зависимости от того, находятся ли они в списке или были удалены;

3) сопоставление перечня полученного списка на шаге 2 с данными, полученными через стандартные средства ОС, например NtQuerySystemInformation.

Поиск структур EPROCESS выполняется побайтовым поиском в памяти ядра. Для каждого фрагмента памяти вычисляется количество совпадений с сигнатурой ДБаС. Если для текущего фрагмента памяти выполнено неравенство

$$\Sigma - \Delta \le i \le \Sigma,$$

то считается, что структура подобного объекта найдена. Σ — это количество байтов в сигнатуре; Δ — пороговое значение (например, Δ может быть равно $(\Sigma \cdot 0{,}8)$); i — количество совпадений для текущего фрагмента памяти с ДБаС-сигнатурой. Если для некоторой области памяти данное неравенство ложно, то эту область пропускают и продолжают анализ со следующими фрагментами памяти до тех пор, пока вся память не будет проанализирована.

В результате будет получен полный список структур процессов на основе сопоставления ДБаС-сигнатур. Сравнение списка, полученного методом ДБаС, со списком из NtQuerySystemInformation позволяет выявить скрытые процессы.

Этот подход был успешно протестирован как для случаев намеренно скрытых объектов, так и для реальных руткитов, таких как Virus. Win32.Sality.q и Trojan. Win32.VB.aqt (оба — Лаборатория Касперского).

Важно подчеркнуть, что структура EPROCESS включает в себя множество полей, значения которых связаны с другими

структурами режима ядра. Поэтому данные значения превышают значение 0x8000_0000 для 32-разрядных ОС. Этот факт частично использован в работе [102], но подход все еще уязвим к модификации байтов и нуждается в обновлении сигнатур EPROCESS при выпуске новой версии Windows.

Динамическая битовая сигнатура. Для совершенствования метода предлагается использование ДБиС, которая позволяет учитывать специфику внутреннего устройства структур процессов, например наличие указателей на другие структуры ядра. ДБиС работает аналогично ДБаС, но на уровне битовых значений, что повышает адаптивность и точность обнаружения.

Преимущества ДБиС: адаптация для новых версий Windows и сервисных пакетов (SP); обнаружение всех структур EPROCESS, даже если они изменены, с порогом совпадений 70–80%.

Этот метод применим для габаритных структур, но нуждается в доработке для небольших структур, поскольку количество ложных срабатываний увеличивается. Например, для обнаружения структуры DRIVER_OBJECT, размер которой в 4 раза меньше размера EPROCESS, предложенный подход нуждается в доработке.

5.7.2. Пример статической балловой сигнатуры для обнаружения драйверов

Статическая балловая сигнатура (СБС) представляет собой усовершенствование метода ДБиС. Различия заключаются в следующем: во-первых, СБС настраивается вручную для конкретных типов структур, таких как DRIVER_OBJECT или DEVICE_OBJECT; во-вторых, СБС использует дополнительную матрицу весов для точного учета соответствия. Вместо подсчета индивидуальных совпадений, как в ДБаС и ДБиС, СБС вычисляет суммарное количество совпадений (оценка) на основе весов. Например, если проверка верна, то добавляется 1, 2 или 3 балла к итоговой оценке.

Вывод о совпадении структуры DRIVER_OBJECT делается путем сравнения итоговой оценки с пороговым значением. Это пороговое значение определяется путем вычисления тех же

метрик для нескрытых структур DRIVER_OBJECT в списке DirectoryObject и может быть откорректировано экспертом

Рассмотрим технику обнаружения DRIVER_OBJECT:

1) получаем дамп памяти ('dump.log' и 'struct.log') и сохраняем адреса структур DRIVER_OBJECT в файле 'drvobj.log' с помощью функции ZwOpenDirectoryObject;

2) определяем значение 'min_major_function';

3) определяем значение 'global_scope';

4) определяем значение 'global_scope_deep';

5) выполняем побайтовый поиск структуры DRIVER_OBJECT с помощью функции 'is_integrated_driver', которая вычисляет количество совпадающих точек для каждой области памяти;

6) сравниваем количество совпадений с 'global_scope'. Если значение не меньше 'global_scope', то структура DRIVER_OBJECT найдена. Иначе вычисляем количество «глубоких совпадений» с помощью 'is_integrated_driver_deep';

7) сравниваем количество «глубоких совпадений» с 'global_scope_deep'. Если значение не меньше 'global_scope_deep', то структура DRIVER_OBJECT найдена. Иначе продолжаем поиск;

8) повторяем шаги 6–7 для всей области памяти, получая RPI-список структур DRIVER_OBJECT;

9) сравниваем RPI-список со списком драйверов, полученным на шаге 1.

На рис. 20 представлен алгоритм обнаружения DRIVER_OBJECT.

Шаги 6 и 7 являются конструкциями типа if-else (см. рис. 20). Опишем подробнее шаги 2, 3 и 4.

Шаг 2. Определяем 'min_major_function' с помощью функции ZwOpenDirectoryObject для получения списка структур DRIVER_OBJECT. Для каждой структуры DRIVER_OBJECT вычисляем максимальное количество совпадающих адресов функций из MajorFunction с помощью 'max_same_major_functions'. Выбираем минимальное значение — 'min_major_function'.

Шаг 3. Определяем 'global_scope' с помощью функции ZwOpenDirectoryObject для получения списка структур DRIVER_OBJECT. Для каждой структуры DRIVER_OBJECT рассчитываем количество баллов по табл. 7. Если одно из условий

ложно, то баллы не добавляются. Общая оценка рассчитывается путем проверки всех условий таблицы. Выбираем минимальное значение — 'global_scope'.

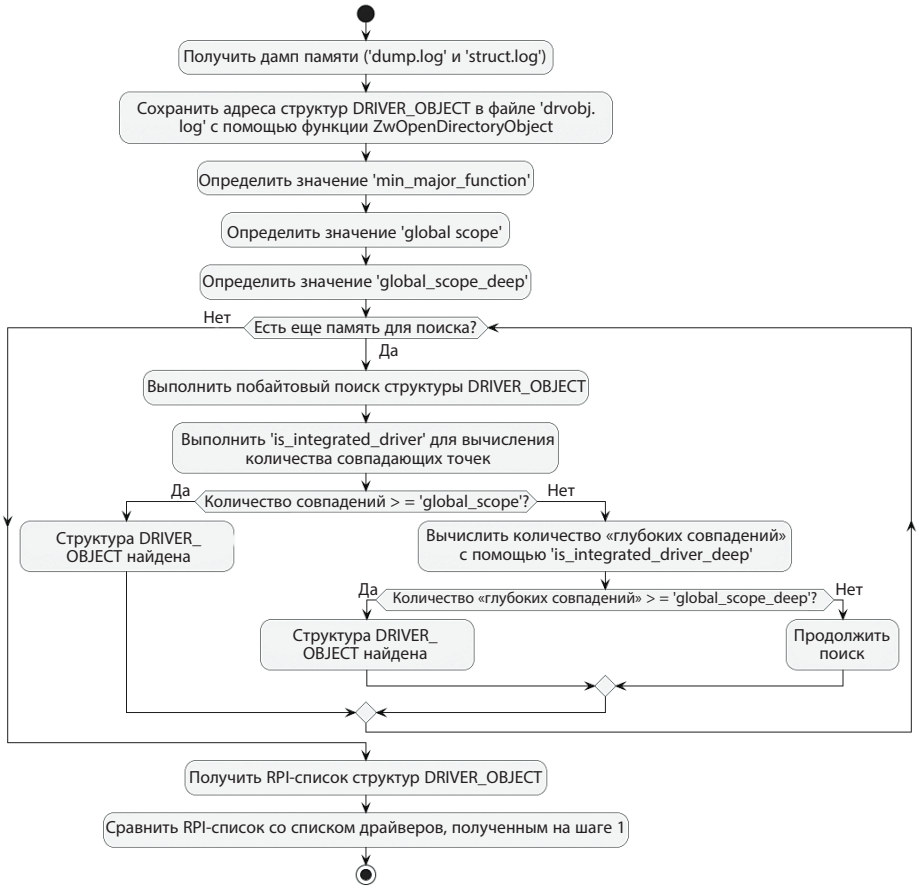


Рис. 20. Алгоритм обнаружения DRIVER_OBJECT

Таблица 7

Матрица весов для расчета 'global_scope'

Условие	Баллы
if (DRIVER_OBJECT_32.Type == 0x04)	2
if (DRIVER_OBJECT_32.Size == 0xa8)	4
if (chk_unicode_string(&DRIVER_OBJECT_32.DriverName))	2
if (chk_unicode_string(DRIVER_OBJECT_32.HardwareDatabase))	2

Окончание табл. 7

Условие	Баллы
if ((DRIVER_OBJECT_32.MajorFunction[0]) >> 31)	2
if (max_same_major_functions(&DRIVER_OBJECT_32) >= min_major_function)	2

Функция 'chk_unicode_string' проверяет корректность структуры UNICODE_STRING. Это делается путем проверки условий из табл. 8. Выражение 'iswprint(UNICODE_STRING)' означает проверку всех символов соответствующего буфера с помощью функции 'iswprint'.

Таблица 8

Функция 'chk_unicode_string'

Условие	Результат
(UNICODE_STRING.MaximumLength >= UNICODE_STRING.Length) && (UNICODE_STRING.Buffer!=NULL) && iswprint(UNICODE_STRING)	true or false

Шаг 4. Определяем 'global_scope_deep' с помощью функции ZwOpenDirectoryObject для получения списка структур DRIVER_OBJECT. Для каждой структуры DRIVER_OBJECT рассчитываем количество совпадений по табл. 9. Выбираем минимальное значение — 'global_scope_deep'.

Таблица 9

Матрица весов для расчета 'global_scope_deep'

Условие	Баллы
if (DRIVER_OBJECT_32.Type == 0x04)	2
if (DRIVER_OBJECT_32.Size == 0xa8)	2
if (DRIVER_OBJECT_32.DriverStart >> 31)	2
if (DRIVER_OBJECT_32.DriverStart % 0x1000 == 0)	2
if (DRIVER_OBJECT_32.DriverSize % 0x1000 == 0)	2
if (check_function_prologue(DRIVER_OBJECT_32.DriverStart))	4
if (DRIVER_OBJECT_32.DriverExtension >> 31)	2
K = chk_unicode_string2(&DRIVER_OBJECT_32.DriverName)	K

Окончание табл. 9

Условие	Баллы
chk_unicode_string(DRIVER_OBJECT_32.HardwareDatabase)	2
if ((DRIVER_OBJECT_32.MajorFunction[0]) >> 31)	2
if (max_same_major_functions(&DRIVER_OBJECT_32) >= min_major_function)	2

Функция 'check_function_prologue' проверяет, истинны ли условия из табл. 10. Эта проверка повторяется для первых 16 байт памяти каждой области памяти (for (int i = 0; i < 0x10 ; i++)).

Таблица 10

Функция 'check_function_prologue(addr)'

Условие	Результат
If (((addr[i+0] == 0x55) && (addr[i+1] == 0x89) && (addr[i+2] == 0xe5)) ‖ ((addr[i+0] == 0x55) && (addr[i+1] == 0x8b) && (addr[i+2] == 0xec)) ‖ ((addr[i+0] == 0x53) && (addr[i+1] == 0x56)) ‖ ((addr[i+0] == 0x56) && (addr[i+1] == 0x57)) ‖ ((addr[i+0] == 0x56) && (addr[i+1] == 0x57)) ‖ ((addr[i+0] == 0x8b) && (addr[i+1] == 0xff)))	true или false

Функция 'chk_unicode_string2' определена в табл. 11.

Таблица 11

Функция 'chk_unicode_string2(PUNICODE_STRING pDriverName)'

Условие	Баллы
if (pDriverName->MaximumLength >= pDriverName->Length)	2
if ((pDriverName->MaximumLength <= 0x50) && (pDriverName->Length <= 0x50))	4
if (chk_unicode_string(pDriverName))	2
if (_memicmp(pDriverName->Buffer, L».sys», pDriverName->MaximumLength))	2
if (wcslen(pDriverName->Buffer) <= pDriverName->Length)	2

На рис. 21 показан процесс использования СБС для поиска скрытых структур DRIVER_OBJECT.

Рис. 21. Процесс использования статической балловой сигнатуры для поиска и идентификации скрытых структур DRIVER_OBJECT

Разберем процесс по шагам (см. рис. 21).

1. Слева изображена матрица весов, которая используется для оценки структур DRIVER_OBJECT (DRV_OBJ), справа показан список DRIVER_OBJECT (A, B, C).

2. Список DRIVER_OBJECT (A, B, C) используется для определения порогового значения.

3. Пороговое значение применяется для побайтового вероятностного поиска в памяти. Результатом является полный список DRIVER_OBJECT (A, B, C, D), где D — скрытая структура.

После выполнения побайтового вероятностного поиска определяется полный список DRIVER_OBJECT (A, B, C, D). Этот список сравнивается со списком, полученным с использованием штатных средств ОС. В результате определен скрытый драйвер D.

Метод СБС включает в себя создание списка структур, определение пороговых значений и проведение вероятностного поиска для выявления всех структур драйверов.

Метод СБС можно усовершенствовать, добавив анализатор для функции 'check_function_prologue', который будет обнаруживать модифицированные прологи функций. Это полезно в случаях, когда ВПО использует различные методы защиты, такие как упаковщики или шифровальщики.

Также можно получать дополнительную информацию о драйвере, используя его хэш-значение. Подобную функциональную возможность имеет Process Explorer [170].

Метод СБС был успешно протестирован для намеренно скрытых объектов, реальных руткитов и драйверов, загруженных с помощью утилиты ATSIV (Linchpin Labs и OSR) [171]. В этих случаях существующие инструменты, такие как PowerTool, Xuetr и некоторые другие утилиты, не могли обнаружить скрытые драйверы, тогда как предлагаемый метод оказался работоспособным. С комментариями к этим инструментам можно ознакомиться в видеоматериале [172].

В работе [116] было предложено использовать связь между структурами DRIVER_OBJECT и DEVICE_OBJECT для поиска DRIVER_OBJECT. Однако эта связь не является обязательной.

Подход СБС успешно показал себя в различных сценариях, что делает его перспективным инструментом для обнаружения скрытых драйверов и других объектов в ОС.

Глава 6
ПЕРСПЕКТИВЫ РАЗВИТИЯ ТЕХНИК СОКРЫТИЯ И ОБНАРУЖЕНИЯ

Рассмотрим возможные техники для обнаружения наиболее сложного случая сокрытия кода, который был представлен в параграфе 5.2. Будут рассмотрены возможности нарушителей по повторному противодействию и контрмеры для обнаружения.

Рассмотрим пример совместного применения комплекса техник сокрытия структур драйверов и модулей в памяти, включая следующие:

1) применение техники DKOM-unlinking для сокрытия драйвера от функции NtQuerySystemInformation (см. параграф 5.6) [38, 117, 173];

2) применение техники DKOM-patching в целях сокрытия удаленной структуры DRIVER_OBJECT от обнаружения с помощью СБС (см. параграф 5.6) [112];

3) применение техники DKOM-patching для перезаписи содержимого PE-заголовка и таблицы импорта, которые являются признаками для обнаружения исполнимых модулей в памяти [174].

Следующим возможным шагом для сокрытия является противодействие методам обнаружения, использующим технику «скользящего окна» и расчет энтропии [157].

Значения энтропии исполняемого кода находятся в определенном диапазоне, что позволяет обнаружить его в памяти [143]. Однако злоумышленники могут вставлять последовательности байтов с низкой энтропией в код, чтобы снизить рассчитываемую энтропию и затруднить обнаружение модуля в памяти. Эта техника сокрытия используется, например, в ВПО Zeus. Иллюстрация техники сокрытия кода (трояна Zeus) с использованием блоков данных с нулевой энтропией, а также уязвимость этой техники сокрытия показаны на рис. 22.

Группа экспертов по безопасности разработала технику для обнаружения вставок с нулевой энтропией [157]. Авторы использовали гистограммы частот появления значений байтов, ко-

торые позволяют легко выявить значения байтов, которые были использованы для «разбавления» кода. Благодаря построению гистограммы, отражающей частоту появления значений байтов, можно выявить последовательности с низкой энтропией и игнорировать их при вычислении энтропии, что делает возможным обнаружение скрытого кода.

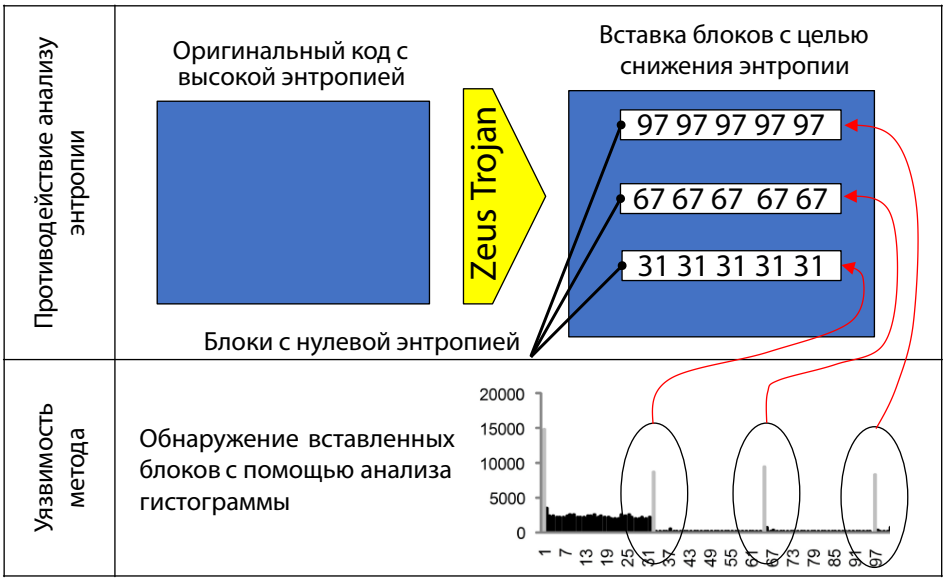


Рис. 22. Схема обхода обнаружения кода через добавление байтов с нулевой энтропией и описание уязвимости данной техники

Однако техника сокрытия, применяемая в ВПО Zeus, может быть усовершенствована. Для имитации наиболее сложных случаев для обнаружения были предложены два способа генерации последовательностей байтов для вставки в исполняемый код [175], которые, с одной стороны, уменьшают рассчитываемую энтропию, а с другой — не могут быть обнаружены с помощью гистограмм и частотного анализа.

6.1. ДЕМОНСТРАЦИЯ НОВЫХ МЕТОДОВ СОКРЫТИЯ ОБЪЕКТОВ В ПАМЯТИ

Подход, используемый для сокрытия исполняемого кода от обнаружения статистическими методами, является развитием техники, применяемой ранее в ВПО Zeus. Идея заключается во

вставке в исполняемый код специальных последовательностей байтов с низкой, но отличной от нуля энтропией. В рамках этого подхода предложены два метода генерации таких последовательностей байтов.

6.1.1. Первый способ на основе случайных диапазонов

Порядок действий:

1) разделить все возможные значения байта на диапазоны по 7–10 значений;

2) значение каждого вставляемого байта выбрать случайным образом из первого диапазона;

3) после выбора значений каждых 4096 вставленных байтов заменить диапазон, из которого выбираются значения, на следующий.

Сравнение графиков энтропии и гистограмм частоты появления значений байтов в исполняемом файле, модифицированном методом, использованным в Zeus, и первым предлагаемым способом показано на рис. 23.

Обнаружение с помощью анализа энтропии. Энтропия вычислялась для окна размером 256 байт с шагом сдвига 128 байт. Оба графика демонстрируют, что большинство значений энтропии исполняемого кода не превышают нижнюю границу диапазона значений, характерных для исполняемого кода (5 бит на байт). Это показывает, что метод вычисления энтропии «скользящим окном» не подходит для обнаружения такого кода [143]. В качестве следующего шага для обнаружения необходимо использовать частотный метод и строить гистограммы.

Обнаружение с помощью анализа гистограмм. На гистограмме исполняемого кода, модифицированного методом Zeus (см. рис. 23, *а*), значения 32, 64, 96, 128, 160, 192 и 224 встречаются наиболее часто, что позволяет выявить вставку таких последовательностей байтов и игнорировать их при вычислении энтропии. Гистограмма кода, модифицированного первым предлагаемым способом (см. рис. 23, *б*), не содержит резко выделяющихся значений, что делает невозможным определение вставленных байтов и игнорирование их при вычислении энтропии. Предложенный способ вставки позволяет противодействовать обнаружению посредством как энтропийного, так и частотного анализа.

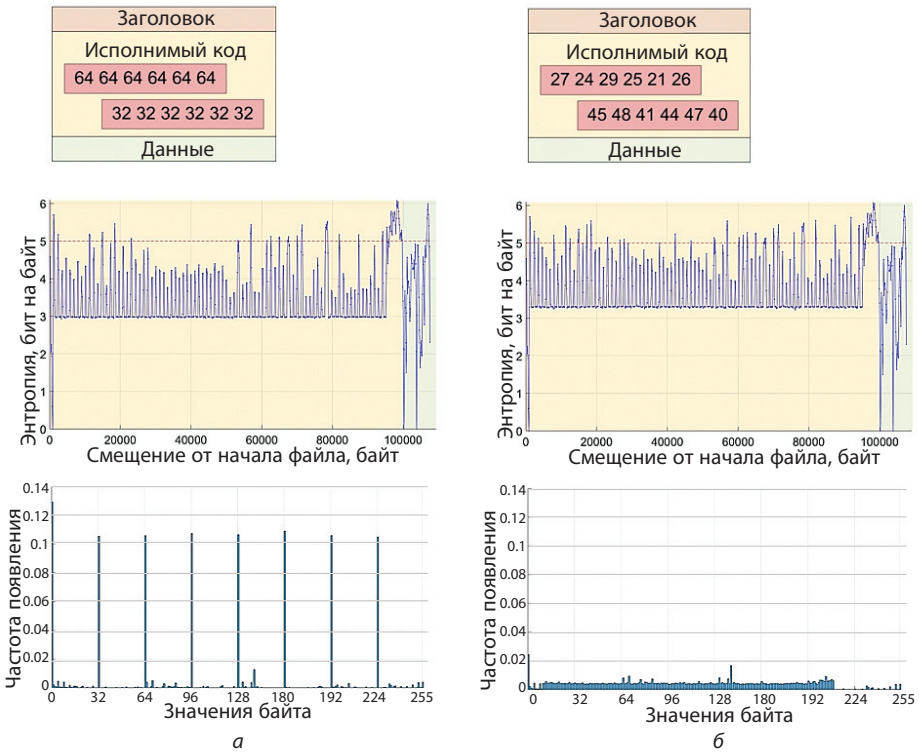


Рис. 23. Графики энтропии исполняемого файла и гистограмм частот появления значений байтов исполняемого кода и модифицированного (1-й способ):

а — способ, используемый Zeus; *б* — первый предлагаемый способ

6.1.2. Второй способ на основе циклических диапазонов

Порядок действий:

1) из всех возможных значений байта выбрать 130 и разделить на 13 диапазонов по 10 значений;

2) значение каждого вставляемого байта выбрать случайным образом из первого диапазона;

3) после выбора значений каждых 26 624 (13×2048) вставленных байтов заменить диапазон, из которого выбираются значения, на следующий.

Обнаружение с помощью анализа энтропии. Графики энтропии и гистограммы частоты появления значений байтов в ис-

полняемом файле, модифицированном методом Zeus и методом циклических диапазонов, показаны на рис. 24. Оба графика показывают, что большинство значений энтропии исполняемого кода не превышают нижнюю границу диапазона значений, характерных для исполняемого кода. Таким образом, метод вычисления энтропии «скользящим окном» не позволяет обнаружить такой код.

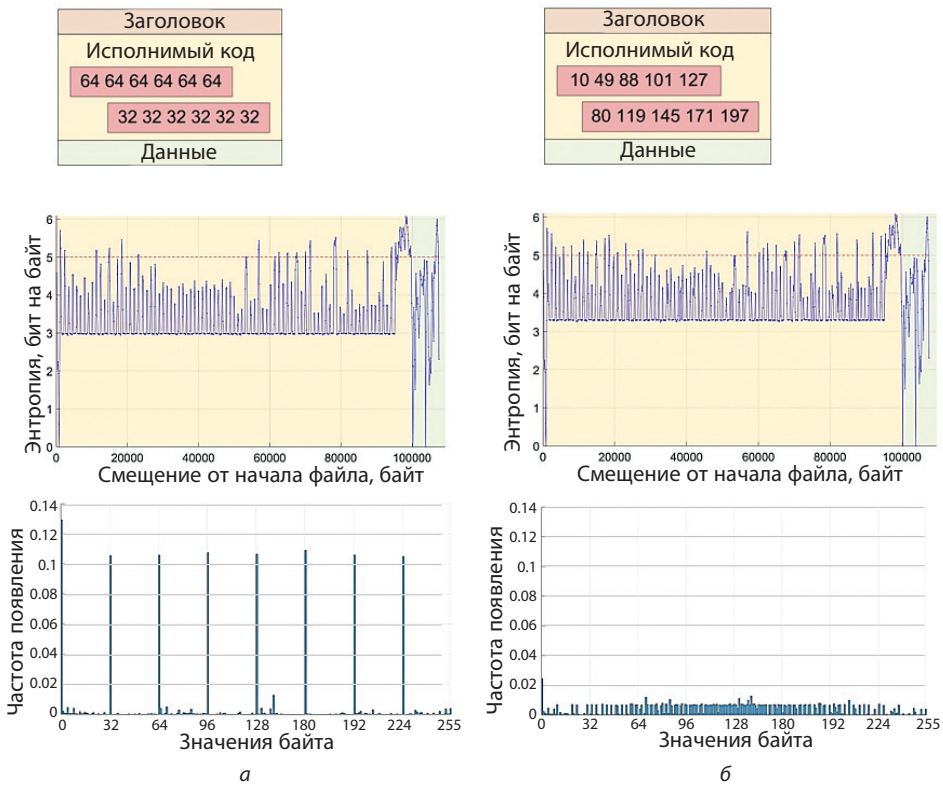


Рис. 24. Графики энтропии исполняемого файла и гистограмма частоты появления значений байтов исполняемого кода и модифицированного (2-й способ):
а — способ, используемый Zeus; *б* — второй предлагаемый способ

Обнаружение с помощью анализа гистограмм. На гистограмме исполняемого кода, модифицированного методом циклических диапазонов, выделяется 130 значений, что связано со вставкой специальных последовательностей байтов. Если при вычислении энтропии игнорировать все байты с такими значениями, то вместе со вставленными байтами будут проиг-

норированы и байты исполняемого кода с теми же значениями, что делает результат неинформативным. Предложенный способ вставки позволяет противодействовать обнаружению посредством как энтропийного, так и частотного анализа.

6.1.3. Использование ассемблерных вставок

Исследованные экспериментальные образцы ПО созданы из исходных текстов на языке C с помощью компилятора Microsoft Visual C++. Вставка последовательностей байтов в исполняемый код производится во время компиляции. Для этого в исходный текст программы необходимо предварительно добавить специального вида ассемблерные вставки, которые состоят:

- из инструкции безусловного перехода JMP, предназначенного для передачи управления на следующий после ассемблерной вставки оператор;
- конструкций _emit, предназначенных для вставки в исполняемый файл заданной последовательности байтов на этапе компиляции;
- метки, расположенной в конце вставки, по которой передается управление упомянутым выше оператором безусловного перехода.

Демонстрационный пример использования ассемблерной вставки приведен без ограничения общности для 32-разрядного случая архитектуры (рис. 25). Эта вставка предназначена для добавления в исполняемый файл при сборке последовательностей байтов, образующих фразу: «Hello, World!».

Ниже приведен фрагмент соответствующего дизассемблерного листинга (рис. 26), показывающий, что последовательности были действительно вставлены в код и после запуска программы будут находиться в оперативной памяти.

В проводимых экспериментах ассемблерные вставки были добавлены через каждые семь операторов языка C; длина последовательности байтов, вставляемой в исполняемый код при помощи каждой вставки, составляет 1024 байт. Для демонстрации возможностей была использована вставка байта с шестнадцатеричным значением 144, однако в общем случае содержимое байтовой последовательности может быть произвольным.

```
uint32_t demo_function()
{
    uint32_t state[] = { 0x12345678, 0x9ABCDEF0,
     0x12345678, 0x9ABCDEF0 };
    uint32_t result = 0;

    for (int i = 0; i < _countof(state); i++) {
        state[i] += state[(i + 3) % _countof(state)];

        // Block 1
        __asm {
            jmp LabelA
            _emit 0x48  // H
            _emit 0x65  // e
            _emit 0x6C  // l
            _emit 0x6C  // l
            _emit 0x6F  // o
            _emit 0x20  // (space)
            _emit 0x00  // Null terminator
            LabelA:
        }

        state[i] *= 0x50495021;

        // Block 2
        __asm {
            jmp LabelB
            _emit 0x57  // W
            _emit 0x6F  // o
            _emit 0x72  // r
            _emit 0x6C  // l
            _emit 0x64  // d
            _emit 0x0A  // (newline)
            _emit 0x00  // Null terminator
            LabelB:
        }
        result ^= state[i];
    }
    return result;

}
```

Рис. 25. Пример ассемблерной вставки в исходном коде на языке Си

```
                mov     [ebp+state], 12345678h
                mov     [ebp+state+4], 9ABCDEF0h
                mov     [ebp+state+8], 12345678h
                mov     [ebp+state+0Ch], 9ABCDEF0h
                mov     [ebp+result], 0
                mov     [ebp+i], 0
                jmp     short loc_4117D0
; ---------------------------------------------------------------

loc_4117C7:                             ; CODE XREF: _demo_
function+AA↓j
                mov     eax, [ebp+i]
                add     eax, 1
                mov     [ebp+i], eax

loc_4117D0:                             ; CODE XREF: _demo_
function+45↑j
                cmp     [ebp+i], 4
                jnb     short loc_41182C
                mov     eax, [ebp+i]
                add     eax, 3
                xor     edx, edx
                mov     ecx, 4
                div     ecx
                mov     eax, [ebp+i]
                mov     ecx, [ebp+eax*4+state]
                add     ecx, [ebp+edx*4+state]
                mov     edx, [ebp+i]
                mov     [ebp+edx*4+state], ecx
                jmp     short LabelA
; ---------------------------------------------------------------
aHello          db 'Hello, ',0
; ---------------------------------------------------------------

LabelA:                                 ; CODE XREF: _demo_
function+77↑j
                mov     eax, [ebp+i]
                imul    ecx, [ebp+eax*4+state], 50495021h
                mov     edx, [ebp+i]
                mov     [ebp+edx*4+state], ecx
                jmp     short LabelB
; ---------------------------------------------------------------
aWorld          db 'World!',0Ah,0
; ---------------------------------------------------------------

LabelB:                                 ; CODE XREF: _demo_
function+93↑j
                mov     eax, [ebp+i]
                mov     ecx, [ebp+result]
                xor     ecx, [ebp+eax*4+state]
                mov     [ebp+result], ecx
                jmp     short loc_4117C7
```

Рис. 26. Фрагмент дизассемблерного листинга демонстрационной программы с ассемблерными вставками

6.2. ПРЕДЛАГАЕМЫЕ ПОДХОДЫ И ЭКСПЕРИМЕНТЫ ПО ОБНАРУЖЕНИЮ НОВЫХ ТЕХНИК

Схематичное изображение фрагмента копии памяти, содержащего два исполняемых файла и произвольные данные, представлено на рис. 27. В областях, содержащих исполняемые файлы, многоточиями обозначены фрагменты исполняемого кода, стрелками — передача управления инструкциями условного и безусловного перехода, словом «блок» — последовательности байтов, вставленные в исполняемый код с целью его сокрытия. В областях фрагмента, не содержащих исполняемые файлы, многоточиями обозначены произвольные данные.

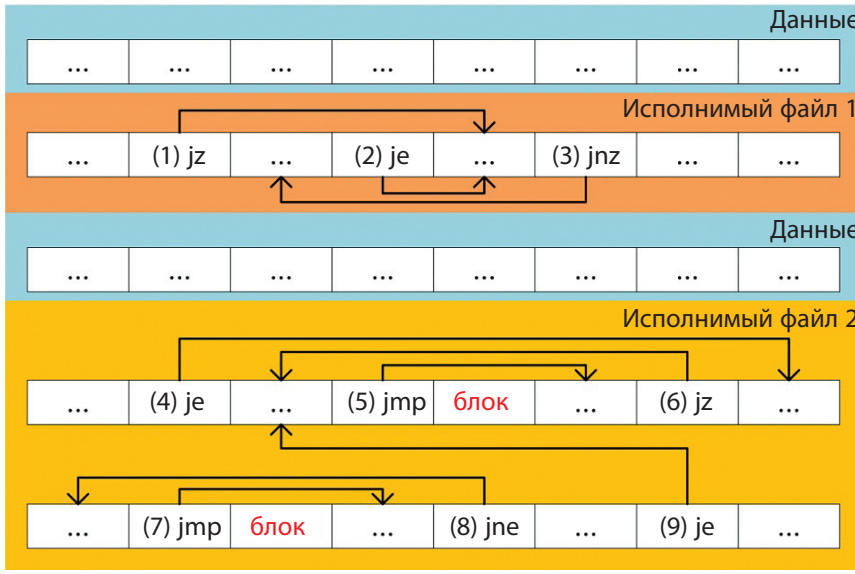


Рис. 27. Схематичное изображение фрагмента копии памяти

Копия памяти последовательно просматривается в целях обнаружения инструкций условного и безусловного перехода. В случае обнаружения инструкции перехода необходимо удостовериться, что инструкция находится в составе исполняемого кода, а не является неверно интерпретированными байтами данных. Проверка состоит из двух этапов.

На первом этапе производится дизассемблирование байтов, находящихся перед найденной инструкцией перехода, для получения 25 предшествующих ей инструкций. Поскольку длина

машинной инструкции в архитектуре x86 не фиксирована, неизвестно, каким количеством байтов представлены 25 инструкций. Поэтому необходимо произвести несколько попыток дизассемблирования, каждый раз дизассемблируя различное количество байтов перед инструкцией перехода [175]. Количество дизассемблируемых байтов для 32-разрядного исполняемого кода вычисляется из расчета от 2,7 до 4 байтов на инструкцию, что было определено экспериментально. Если будет найдено количество байтов, безошибочно дающих в результате дизассемблирования заданное число инструкций, то необходимо перейти ко второму этапу. В противном случае инструкция перехода признается неверно интерпретированными байтами данных и просмотр копии памяти в целях поиска инструкций перехода продолжается.

На втором этапе производится проверка наличия определенных инструкций среди дизассемблированных.

Исследователи из компании *FireEye* [176] составили списки инструкций, используемые для проверки точности дизассемблирования, которые включают:

- инструкции с высокими привилегиями;
- инструкции, не востребованные компиляторами;
- инструкции, встречающиеся нечасто;
- инструкции для работы с указателями на память (far-pointers);
- инструкции с специфическими префиксами.

Если обнаруживается инструкция одной из этих категорий, то соответствующая инструкция перехода игнорируется и сканирование копии памяти продолжается без остановки.

На базе выявленных инструкций перехода создается направленный граф, названный графом инструкций перехода, где вершинами выступают обнаруженные инструкции перехода. Связи между этими вершинами (ребра графа) формируются исходя из двух следующих соображений. Первое касается выявления смещения, по которому инструкция перехода передает контроль, и определения той из обнаруженных инструкций перехода, которая активируется первой после такой передачи. Эти вершины соединяются ребрами. Второе соображение применяется, если инструкция перехода, представленная вершиной, является условной: необходимо установить, какая инструкция перехода

активируется первой при невыполнении условия; если расстояние между такими инструкциями не превышает 1280 байт, то соответствующие вершины соединяются ребрами. Граф, представляющий инструкции перехода для данного фрагмента копии памяти, показан на рис. 28.

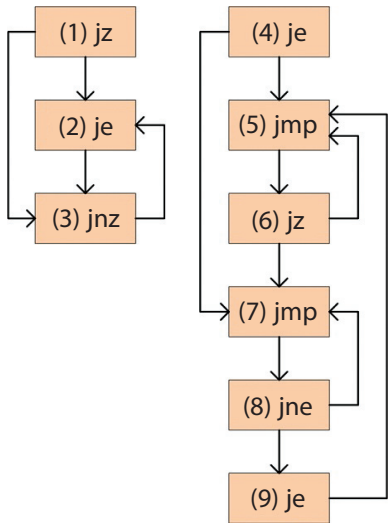


Рис. 28. Граф инструкций перехода для фрагмента копии памяти

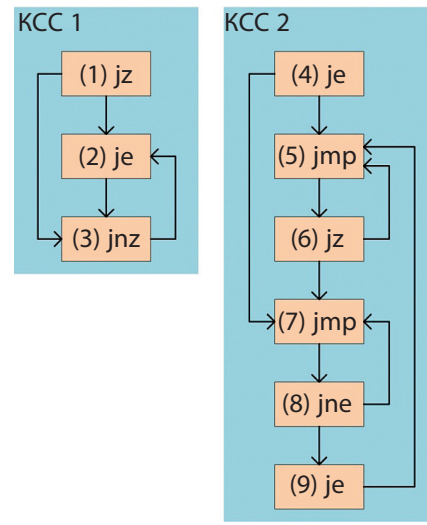


Рис. 29. Компоненты слабой связности графа инструкций перехода

В процессе анализа построенного графа инструкций перехода определяются его компоненты, обладающие свойством слабой связности, — так называемые компоненты слабой связности (КСС). Слабая связность означает, что между любыми двумя вершинами компоненты существует путь, если игнорировать направление ребер. Это позволяет выявлять логически связанные фрагменты кода. Эти компоненты включают в себя исполняемый код, который расположен в соответствующих сегментах анализируемой копии памяти и является частью исполняемого файла (рис. 29). Сегменты копии памяти, содержащие найденный исполняемый код, должны быть предметом отдельного исследования. Цель такого исследования — установить исходный исполняемый файл, включающий данный код, и провести его детальный анализ.

С использованием предложенной методики была разработана программа для выявления скрытых исполняемых кодов.

Программный прототип создан на C# и служит для анализа копии виртуальной памяти процесса, которая должна быть предварительно получена. Данный инструмент позволяет проводить анализ как целостной копии памяти, так и отдельных ее сегментов, выбор которых регулируется настройками, задаваемыми при старте программы. Для дизассемблирования кода используется библиотека SharpDisasm, а для построения и работы с графами — библиотека QuickGraph.

В ходе испытаний был проанализирован образ памяти компьютерной системы, работающей на 64-битной версии Windows 10 Pro версии 1809 (сборка 17763.194). Для этого был использован экспериментальный образец, который применялся для создания «разбавленного» исполнимого кода в памяти, после чего с помощью программы Passmark OSForensics была сделана копия физической памяти системы.

Дополнительный анализ этого образа памяти, включая запуск инструментов для обнаружения скрытых процессов или образцов, проводился на компьютере с 64-битной Windows 7 Professional SP1 (сборка 7601). С использованием The Volatility Foundation [99] и плагина memdump была извлечена копия виртуальной памяти процесса упомянутого экспериментального сэмпла. После этого файл копии виртуальной памяти и сам экспериментальный образец были открыты в программах для просмотра и редактирования в шестнадцатеричном формате, чтобы определить позицию, на которой в образе памяти располагался экспериментальный образец. Эта позиция в файле, представляющем копию виртуальной памяти, называется смещением и записывается в шестнадцатеричной форме. Как показано на рис. 30, экспериментальный сэмпл находился на смещении 2E000, а исполняемый код сэмпла — в диапазоне от 2F000 до 342F5.

При запуске средству обнаружения переданы следующие параметры: имя файла, содержащего копию виртуальной памяти процесса экспериментального образца; начальное смещение исследуемой области копии памяти — 2D000; конечное смещение исследуемой области копии памяти — 35000. Результат работы средства приведен на рис. 31. Экспериментальная проверка показала, что преднамеренно скрытый программный образец исполняемого кода был успешно обнаружен в копии памяти.

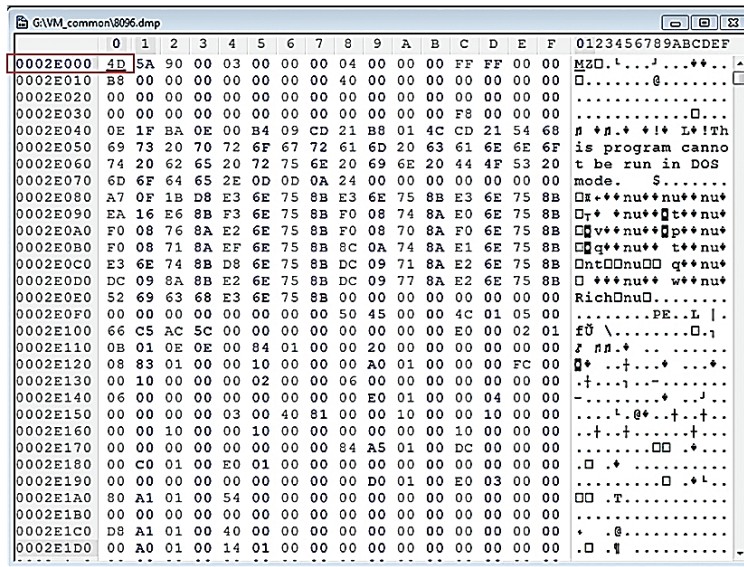


Рис. 30. Расположение экспериментального образца в копии памяти

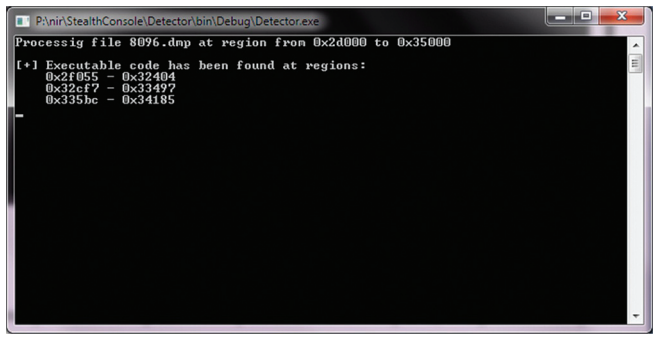


Рис. 31. Результат анализа копии памяти, содержащей экспериментальный образец

6.3. АНАЛИЗ НЕКОТОРЫХ ЭКСПЕРИМЕНТАЛЬНЫХ ТЕХНИК ОБНАРУЖЕНИЯ ДРАЙВЕРОВ В УСЛОВИЯХ СЛОЖНЫХ СЛУЧАЕВ СОКРЫТИЯ КОДА

В качестве сценария сложного сокрытия драйвера нужно взять пример из параграфа 6.2. Предположим, что нарушители скрыли драйвер с использованием комплекса популярных техник:

- удаление информации о драйвере из различных системных списков;

- перезапись содержимого структур драйверов для противодействия сигнатурному обнаружению;
- перезапись фрагментов PE-заголовка и таблицы импорта в целях противодействия сигнатурному обнаружению;
- применение техник по «разбавлению» исполнимого кода для противодействия анализу с использованием энтропии;
- применение техник по сбросу бита исполнения для страниц памяти драйвера, содержащих код отдельных функций.

По соображениям профессиональной этики и информационной безопасности детали реализации техник сокрытия и других функций скрытого драйвера не будут приведены. Будет представлен обзор возможных техник для обнаружения такого драйвера.

6.3.1. Обнаружение скрытого драйвера методом уменьшения размера «скользящего окна»

Задача обнаружения драйвера сводится к задаче распознавания исполняемого кода и других фрагментов памяти с данными. В этом контексте задача усложняется из-за «разбавления» кода по памяти. Применение существующего подхода на основе «скользящего окна» с использованием меньшего его размера может помочь найти отдельные фрагменты исполняемого кода (рис. 32), однако это, в свою очередь, приводит к росту числа ложных срабатываний.

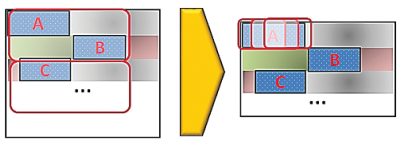


Рис. 32. Подход на основе метода «скользящего окна»

Прямоугольники A, B и C обозначают фрагменты исполняемого кода вредоносного драйвера (см. рис. 32). Эти фрагменты распределены по памяти и разбавлены данными с низкой энтропией. Метод «скользящего окна» применяется для поиска фрагментов исполняемого кода. «Скользящее окно» меньшего размера проходит по памяти, чтобы найти фрагменты исполняемого кода (A, B и C).

Метод иллюстрирует необходимость соблюдения баланса между возможностью обнаружения и количеством ложных срабатываний. Применение метода «скользящего окна» меньшего размера помогает улучшить обнаружение фрагментированного кода, но также может привести к большему количеству ложных срабатываний.

6.3.2. Альтернативные подходы к обнаружению скрытого кода в памяти в условиях противодействия

Для обнаружения исполнимого кода в памяти предлагается использовать следующие альтернативные техники.

1. *Новые методы расчета энтропии и анализа данных.*

1.1. Использование функции $P(2-P^2)$ вместо $P \cdot \log(P)$ для оценки энтропии. Предварительный анализ показывает, что эта функция растет быстрее энтропии Шеннона на интервале от 0 до 1, что делает ее более подходящей для анализа памяти.

1.2. Анализ зависимости вычисленных значений энтропии от разных длин «скользящих окон», последующий спектральный и вейвлет-анализ.

1.3. Применение методов цифровой фотограмметрии для нахождения исполняемого кода на графических диаграммах содержимого памяти.

1.4. Использование закона Ципфа — Мандельброта [177, 178], описывающего частотное распределение элементов в наборах данных, для анализа дизассемблированного листинга.

2. *Дизассемблирование содержимого памяти и анализ полученного кода.*

2.1. Частотный анализ инструкций.

2.2. Логическая оценка дизассемблированного кода.

2.3. Проектирование и анализ графа потока управления на основе дизассемблированного кода.

Методы цифровой фотограмметрии позволяют определять местоположение различных объектов на цифровых изображениях, например человеческих лиц на фотографиях, или распознавать здания и дороги на картах, используя искусственный интеллект и нейронные сети. В случае когда речь идет об обнаружении скрытых драйверов, мы сталкиваемся с аналогичной проблемой —

локализацией фрагментов памяти с высокими значениями энтропии [165], которые соответствуют исполняемому коду [179].

Аналогия между визуализацией физических объектов на карте и визуализацией данных внутри бинарного файла представлена на рис. 33. Космический фотоснимок представляет собой реальное изображение района города с высокой детализацией аналогично тому, как бинарный файл может содержать неструктурированные данные. Карта упрощает восприятие и дает больше контекстной информации, как схема бинарного файла, которая упрощает понимание его структуры. Подобно тому, как карты используются для понимания структуры и планировки городов, визуализация байтовой структуры файла позволяет исследовать его внутреннее содержание и обнаруживать аномалии.

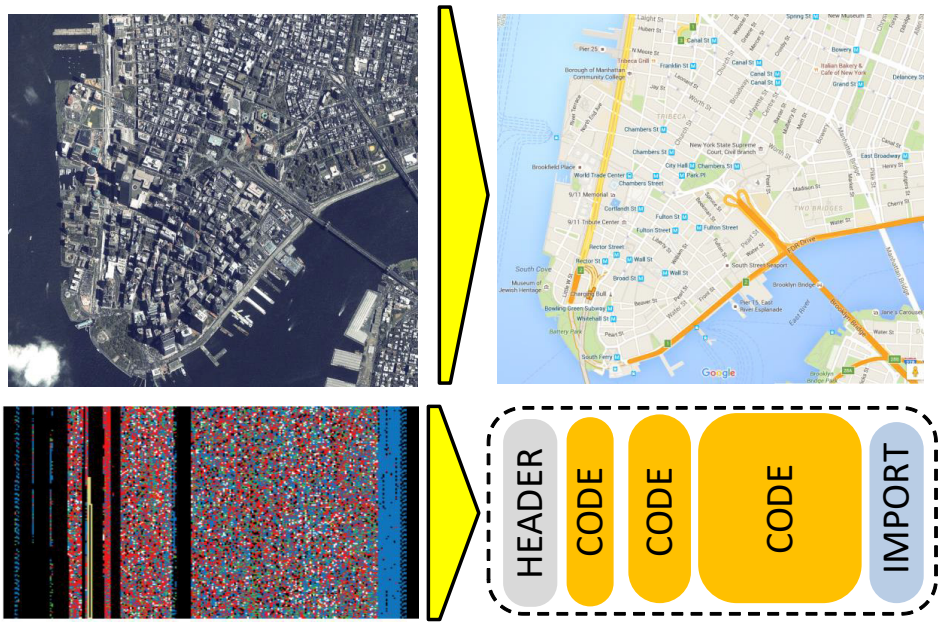


Рис. 33. Метод цифровой фотограмметрии

Визуализация байтового массива (нижняя часть рис. 33) помогает выявлять неструктурированные данные и возможные аномалии как области с очень высокими или низкими значениями. Схема файла (справа) помогает понять, как данные организованы внутри файла.

Первая идея заключается в адаптации методов цифровой фотограмметрии для анализа памяти компьютера в целях обнаружения фрагментов исполняемого кода, что представляет собой новый подход в области кибербезопасности и анализа ВПО.

Для реализации идеи по применению дизассемблирования можно использовать библиотеки с открытым исходным кодом Capstone и BeaEngine для дизассемблирования как 32-разрядного, так и 64-разрядного кода [180]. Анализ частоты появления ассемблерных инструкций включает два этапа: предварительный и этап обнаружения.

Оценка корректности дизассемблированного кода может производиться путем проверки непротиворечивости кода, например, путем проверки того, что результат выполнения инструкций не перезаписывает ранее полученные данные.

С помощью графа потока управления можно найти связи между различными фрагментами исполняемого кода и собрать вместе все три блока A, B и C [15].

Этапы процесса дизассемблирования и восстановления оригинального исполняемого файла из различных частей памяти представлены на рис. 34.

1. Исходный код каждого фрагмента (A, B, C) дизассемблируется, превращаясь в набор ассемблерных инструкций (пример дизассемблированного кода показан в верхнем правом блоке).

2. После дизассемблирования инструкции используются для построения графа потока управления, который отображает порядок выполнения инструкций и их взаимосвязи (узлы 1, 2, 3, 4, соединенные стрелками, показывают направление потока управления). Одновременно закон Ципфа — Мандельброта применяется для анализа частоты ассемблерных инструкций в каждом фрагменте кода, чтобы определить, принадлежит ли данный фрагмент к исполняемому коду или это просто данные. Если частоты инструкций соответствуют закону, то фрагмент считается частью исполняемого кода.

3. На последнем этапе фрагменты A, B и C объединяются, что позволяет аналитикам восстановить оригинальный исполняемый файл из различных частей памяти.

Похожая идея по дизассемблированию фрагментов кода используется в плагине driverirp средства Volatility с флагом '--verbose' для выявления заражения TDL3, который использует техники Stealth Hooks и Redirector Stubs для сокрытия факта перехвата [112].

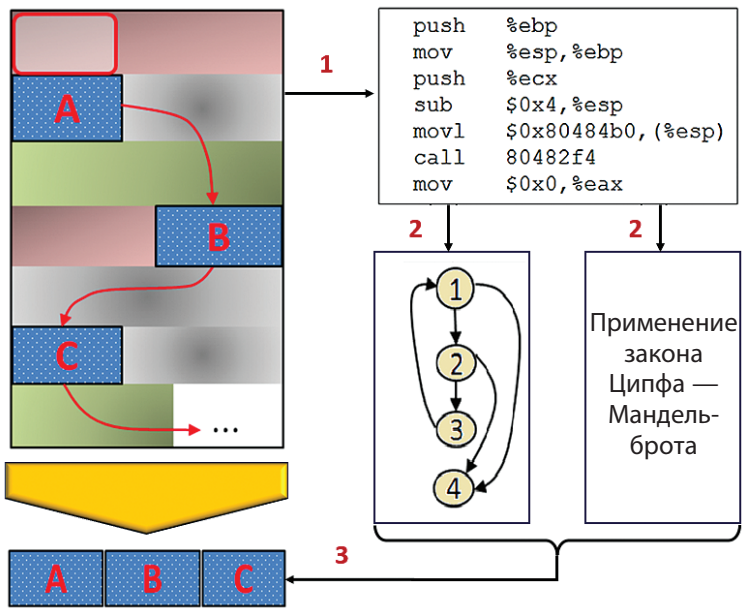
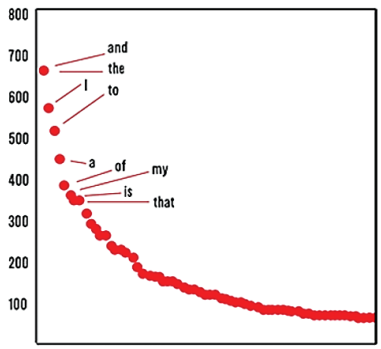


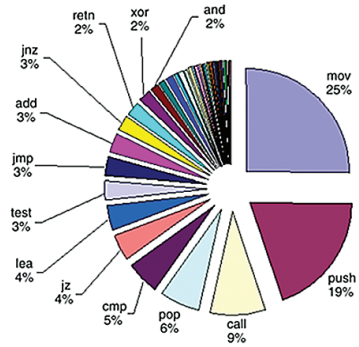
Рис. 34. Применение дизассемблирования

Закон Ципфа — Мандельброта использовался для описания распределения частот слов в естественных языках. Американский лингвист Джордж Ципф обнаружил, что частота слов в тексте обратно пропорциональна их месту в списке частот. Текст, написанный на естественном языке, почти всегда подчиняется закону Ципфа, в то время как случайный набор слов или искусственный текст — нет. Например, если отсортировать все слова в пьесе Уильяма Шекспира «Ромео и Джульетта» по частоте, то получится гиперболическая кривая (рис. 35, *а*) — наиболее часто встречающиеся слова, такие как and, the и to, будут в начале, а редкие слова — в правой части графика. В некоторых источниках применение закона Ципфа — Мандельброта связывают с оценкой гармоничности и даже «красотой» литературных произведений.



Закон Ципфа: $P(i) = C = \text{const}P(i)$;
$P(i)$ — частота i-го слова; i — его ранг

а



Закон Ципфа — Мандельброта:
$P(i) \times (B + i)^{\gamma} = C$; B, C и γ — константы

б

Рис. 35. Применение закона Ципфа — Мандельброта для литературного текста и аналогичное распределение частот для ассемблерного листинга программного обеспечения

Математик Бенуа Мандельброт расширил закон Ципфа, добавив две константы для работы с текстами высокой логической сложности. Ассемблерные инструкции в исполняемом коде имеют похожее распределение частот (рис. 35, *б*). Самая популярная инструкция — 'mov' (25%), следующие — 'push' (19%), 'call' (9%). Таким образом, можно предположить, что исполняемый код обладает свойствами, аналогичными естественному языку.

Идея заключается в том, чтобы для каждого фрагмента кода проверять частоты встречаемости инструкций на соответствие распределению по закону Ципфа — Мандельброта и использовать это соответствие для различения кода и данных в памяти.

Предложенный подход должен учитывать следующие ограничения:

- анализ памяти — исследование проводилось только для 32- и 64-битных ОС Windows и не охватывало Unix-ориентированные ОС, такие как Mac OS и Linux, а также мобильные ОС;
- методы обфускации — предполагается, что в исполняемом коде не использовались методы обфускации и полиморфизма. В случае значительного «разбавления» исполняемого кода задача обнаружения скрытых объектов усложняется, и такой анализ требует значительных вычислительных ресурсов.

Внедрение описанных выше методов и техник может существенно улучшить процесс обнаружения скрытого кода. Применение цифровой фотограмметрии, дизассемблирования и анализа по закону Ципфа — Мандельброта предоставляют новые инструменты для исследования и выявления аномалий в памяти компьютера.

Выводы

В разделе 1 монографии проведен всесторонний анализ вопросов безопасности ядра Windows, включая механизмы защиты и техники атак. Рассмотрена архитектура Windows с точки зрения информационной безопасности, а также защищенный режим работы процессора и технология аппаратной виртуализации. Эти технологии обеспечивают изоляцию программных компонентов и повышают безопасность системы, но также могут быть использованы нарушителями для достижения вредоносных целей.

Анализ существующих техник сокрытия ПО показал, что руткиты и гипервизор-руткиты остаются серьезными угрозами для безопасности Windows. Техники изменения структур памяти, такие как DKOM, представляют особую опасность из-за сложности их обнаружения. Несмотря на развитие защитных механизмов, таких как PatchGuard и HVCI, нарушители продолжают совершенствовать свои техники, используя уязвимости в драйверах и других компонентах системы.

Непрерывное развитие и совершенствование техник сокрытия, а также применение нарушителями комплекса техник противодействия обнаружению требуют от экспертов по безопасности не только анализа отдельных существующих сложных случаев сокрытия, но и моделирования их потенциального развития. Такой подход позволяет опережать действия нарушителей и обеспечивать результативную кибербезопасность для пользователей средств защиты.

В последующих разделах будут представлены новые подходы к обнаружению и предотвращению атак на ядро Windows, включая разработку гипервизоров для контроля целостности содержимого ядра и применение математических моделей для обнаружения руткит-гипервизоров в условиях противодействия.

Раздел 2
РАЗРАБОТКА ГИПЕРВИЗОРОВ ДЛЯ МОНИТОРИНГА И ЗАЩИТЫ ПАМЯТИ ЯДРА WINDOWS ОТ АТАК ДРАЙВЕРОВ

Probleme kann man niemals mit derselben
Denkweise lösen, durch die sie entstanden sind.
Albert Einstein

Если проблема не решается на уровне ее создания,
то для ее решения нужно перейти на новый уровень.
Альберт Эйнштейн

Глава 7
ПРОЕКТ MEMORYMONRWX: ОБНАРУЖЕНИЕ И АНАЛИЗ РАБОТЫ РУТКИТ-ДРАЙВЕРОВ ПУТЕМ ЛОГИРОВАНИЯ ИХ ОБРАЩЕНИЙ К ГЛОБАЛЬНЫМ ПЕРЕМЕННЫМ ЯДРА

Современные операционные системы семейства Windows сталкиваются с постоянно растущим числом угроз безопасности, в том числе с использованием возможностей драйверов ядра, которые предоставляют нарушителям полный доступ к системной памяти и компонентам ОС, что позволяет им обходить и (или) отключать защитные механизмы ОС, скрывать свою активность, а также получать несанкционированный доступ к данным. Популярные средства защиты, реализованные на уровне ядра ОС, зачастую оказываются недостаточными для противодействия таким угрозам.

Нарушители применяют следующие техники для использования возможностей драйверов Windows в своих атаках:

- BYOVD (Bring Your Own Vulnerable Driver) — это популярная тактика кибератак, при которой нарушители загру-

жают в систему легитимные, но уязвимые драйверы, подписанные действующими сертификатами. Такие драйверы, с одной стороны, благодаря цифровой подписи не вызывают подозрений у ОС и средств защиты, однако содержат уязвимости, позволяющие атакующим получить привилегии уровня ядра. Используя эти уязвимости, нарушители могут отключать защитные механизмы, скрывать свою активность, модифицировать память ядра и выполнять другие вредоносные действия. Поскольку драйвер формально считается доверенным, стандартные средства защиты зачастую не способны предотвратить его загрузку и эксплуатацию, что делает BYOVD одной из наиболее опасных техник для обхода современных защитных систем Windows;

- подпись вредоносного кода сертификатами *Microsoft*. В некоторых случаях нарушителям удается подписать вредоносные драйверы сертификатами WHQL от компании *Microsoft*. Это происходит, когда злоумышленники используют учетные записи разработчиков, которые были скомпрометированы или созданы для этих вредоносных целей;
- использование скомпрометированных сертификатов подписи кода (Compromised code-signing certificate). Получив доступ к действующему сертификату — через кражу, компрометацию инфраструктуры или покупку на черном рынке, злоумышленники могут подписывать вредоносный код, который будет восприниматься операционной системой и антивирусами как легитимный, что существенно затрудняет его обнаружение и блокировку [181];
- манипуляции с датой цифровой подписи. Злоумышленники могут изменять дату цифровой подписи, используя механизмы timestamping [182, 183]. Это позволяет вредоносному коду успешно проходить проверку цифровой подписи даже после истечения или отзыва сертификата;
- покупка сертификатов для подписи программного кода на подпольных рынках, включая даркнет, является одной из популярных тактик киберпреступников. На этих площадках нарушители могут приобрести как поддельные, так и легитимные сертификаты, выданные действующими удостоверяющими центрами, включая крупные компании, такие как *Comodo*, *Thawte* и *Symantec* [184].

В табл. 12 представлены примеры недавних атак с использованием драйверов ядра.

Популярные встроенные средства защиты Windows нередко оказываются недостаточными для противодействия этим сложным угрозам, что требует разработки новых решений на базе гипервизоров для мониторинга и защиты памяти ядра от атак драйверов.

Таблица 12

Примеры недавних атак с использованием драйверов ядра

Название ВПО (драйвера)	Год	Принцип действия
BlackCat Ransomware [185]	2023	Применение легитимных сертификатов для подписи вредоносных драйверов
EDRKillShifter, Gh0stRAT, RealBlindingEDR [186]	2024	BYOVD. Эксплуатация уязвимых драйверов для отключения средств защиты
Qilin Ransomware [187]	2025	BYOVD. Эксплуатация уязвимого драйвера
HookSignTool (RedDriver), FuckCertVerifyTimeValidity	2023	Изменение даты подписи драйверов с использованием утекших сертификатов, для загрузки вредоносных драйверов
BYOVD Ransomware Attacks via Paragon Driver (BioNTdrv.sys) [188]	2025	BYOVD. Эксплуатация уязвимого драйвера для повышения привилегий

В работе [189] показано, что современное ВПО подготовлено достаточно тщательно, что затрудняет его обнаружение даже высококвалифицированными антивирусными экспертами. Механизм Driver Signature Enforcement не может предотвратить установку ВПО, подписанного легитимным сертификатом. PatchGuard уязвим к контрмерам со стороны ВПО, а современные антивирусные продукты не могут обнаружить ВПО даже в течение нескольких лет (одним из примеров является кибератака «Проект Саурон», которая более пяти лет собирала информацию из различных организаций стран Ближнего Востока [190]).

В данном разделе представлен подход к решению этой проблемы с использованием гипервизоров на базе аппаратной

виртуализации. Рассмотрим ряд проектов — начиная от мониторинга и анализа поведения драйверов (MemoryMonRWX) и заканчивая продвинутыми системами защиты памяти ядра (AllMemPro и MemoryRanger). Особое внимание уделим отдельным модификациям MemoryRanger, направленным на защиту различных структур ядра Windows, таких как FILE_OBJECT, EPROCESS, и полей, соответствующих Protected Process Light (PPL), в том числе защите компонентов Microsoft Defender от атак. Каждый проект подробно рассмотрим с точки зрения архитектуры, реализации и практического применения, демонстрируя возможности предложенных решений в противодействии современным угрозам безопасности на уровне ядра ОС.

7.1. СРАВНИТЕЛЬНЫЙ АНАЛИЗ МЕТОДОВ МОНИТОРИНГА ДОСТУПА К ПАМЯТИ

Мониторинг попыток доступа к памяти ядра является крайне важным для обеспечения безопасности операционных систем, таких как Windows. Современные методы мониторинга доступа к памяти можно разделить на две основные категории: методы на базе операционной системы (OS-based) и методы на базе гипервизора (hypervisor-based). Классификация методов контроля доступа к памяти показана на рис. 36.

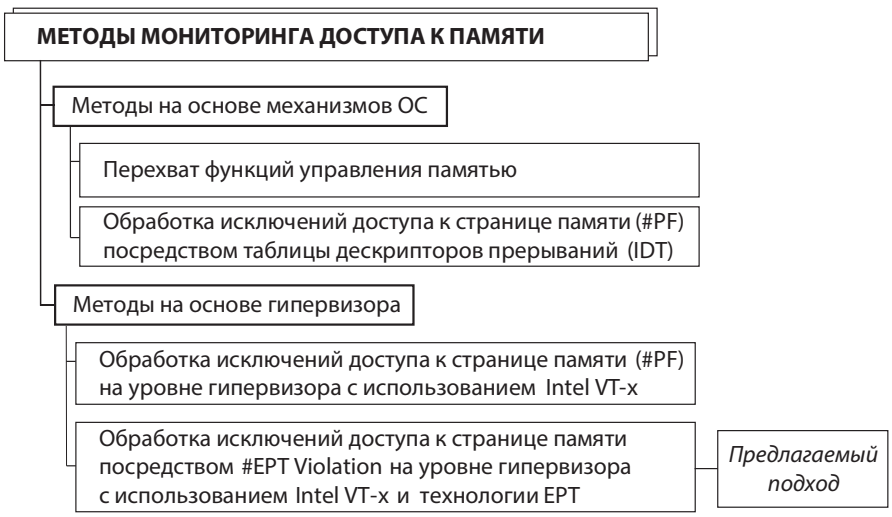


Рис. 36. Классификация методов мониторинга попыток обращения к памяти

7.1.1. Методы на основе операционной системы

Эти методы используют встроенные механизмы ОС, не требуют поддержки новейших технологий со стороны процессора. Они включают два основных подхода.

1. *Перехват функций управления памятью*: осуществление мониторинга путем перехвата системных вызовов, таких как ExAllocatePoolWithTag и MmLoadSystemImage. Это позволяет отслеживать выделение и освобождение памяти драйверами. Примеры таких решений описаны в работах [191, 192]. Метод перехвата функций режима ядра с помощью встроенных перехватов работает на многоядерных процессорных системах, он был предложен в работе [53]. Также можно перехватывать функции режима ядра, применяя известные техники модификации значений указателей в таблице диспетчеризации системных служб [193] и используя техники Stealth Hook и Redirector Stubs для скрытия перехвата [112]. Однако данные методы применимы только на 32-битных версиях Windows, поскольку 64-битные системы защищены механизмом PatchGuard, предотвращающим несанкционированные изменения в ядре. Согласно официальному блогу PatchGuard «он предназначен для защиты критических структур ядра от легкого изменения неавторизованными субъектами» [194]. PatchGuard также контролирует целостность ядра ОС Windows, включая ntoskrnl.exe [195].

2. *Обработка исключений при доступе к странице* (*#PageFault*) *с помощью IDT*: основана на манипуляции таблицами страниц и перехвате исключений доступа к страницам памяти через Interrupt Descriptor Table (IDT). Основная идея заключается в добавлении своего обработчика исключений и модификации записей в таблице страниц (например, очищении бита Present). Такие подходы описаны в работах [196, 197].

Процесс отображения памяти (memory paging) с использованием страничной организации памяти описан в документации Intel [198] и включает несколько фаз. Когда происходит обращение к странице памяти, процессор начинает обход таблицы страниц (page table), чтобы найти физический адрес, соответствующий запрашиваемому виртуальному адресу. Таблица страниц представляет собой структуру данных, которая хранит сведения о соответствии виртуальных адресов физическим. В процессе обхода таблицы страниц процессор считывает соот-

ветствующую запись таблицы страниц (Page Table Entry, PTE) и анализирует ее биты состояния, необходимые для управления доступом к памяти и оптимизации работы системы. Эти биты позволяют определить, присутствует ли страница в памяти (бит P, present) и разрешен ли запрашиваемый тип доступа, например чтение и (или) запись (биты R/W). Бит Execute Disable (XD) показывает, разрешено ли выполнение кода на данной странице. Кроме того, существуют дополнительные флаги, такие как Copy-on-Write (C), Global (G), Large Page (L), Cache Disabled (N), Write-Through (T), которые обеспечивают управление памятью и производительностью. Если страница присутствует и запрашиваемая операция разрешена, то процессор извлекает из PTE номер страницы (Page Frame Number, PFN), который указывает на конкретное местоположение страницы в физической памяти. Затем на основе PFN и смещения внутри страницы вычисляется искомый адрес в физической памяти. Если же страница оказывается недействительной или запрашиваемая операция запрещена, то процессор генерирует исключение Page Fault (#PF), что сигнализирует о необходимости обработки ошибки доступа к памяти.

В данном контексте при обращении к памяти может возникнуть несколько ситуаций:

- запись в таблице страниц (PTE) имеет бит Present, установленный в 0. Это указывает на то, что страница памяти либо выгружена, либо еще не загружена;
- PTE с установленным битом Dirty в 0 при выполнении операции записи, что означает доступность страницы только для чтения.

В любом из этих случаев процессор генерирует исключение PageFault (#PF). Для обработки этого исключения используется соответствующий обработчик, адрес которого хранится в таблице дескрипторов прерываний (IDT — Interrupt Descriptor Table). Пример реализации такого обработчика можно найти в функции nt!KiTrap0E, которая выполняет загрузку необходимых страниц в память, настройку записей PTE и возвращает управление операционной системе.

В работе [196] предложена техника использования механизма страничной организации памяти для решения задач по мониторингу доступа попыток обращений к памяти, а также для обеспечения конфиденциальности и целостности определенных об-

ластей памяти ядра. Данная техника заключается в изменении статуса записи в таблице PTE: бит Present устанавливается в 0, что указывает на отсутствие страницы в памяти и замене кода обработчика PageFault; это позволяет изменить штатное поведение системы и управлять интерпретацией памяти.

Пример развития данной техники для обеспечения конфиденциальности информации в памяти ядра представлен в патенте EP4231181B1 [199]. Авторы предусмотрели динамическое обновление списка защищаемых страниц на основе событий ОС и использование механизма самозащиты для обнаружения и предотвращения несанкционированных модификаций структур памяти, используемых в работе.

При реализации данной техники на практике необходимо учесть работу механизма кэширования и содержимое TLB буферов (Translation Lookaside Buffer), например принудительно сбрасывать кэш после обновления содержимого PTE записи.

Иллюстрация работы техники на основе обработки исключений при доступе к странице (#PageFault) подробно представлена на рис. 37.

Например, для защиты конфиденциальных данных, расположенных на странице C, от несанкционированного доступа можно изменить соответствующую запись в таблице страниц, сбросив бит присутствия (P-бит). Когда неизвестный драйвер, например 'Drv.sys', загруженный на страницах A и B, пытается прочитать эти данные, процессор начинает процесс трансляции адресов для получения содержимого страницы C. Обнаружив, что P-бит установлен в 0, процессор генерирует исключение #PF (Page Fault) и передает управление обработчику этого исключения через таблицу дескрипторов прерываний (IDT). После замены стандартного обработчика Page Fault на собственный можно реализовать различные алгоритмы обработки, используя сохраненный указатель инструкции (SourceAddr) и адрес, вызвавший исключение (DestinationAddr).

Возможны несколько сценариев использования модифицированного обработчика Page Fault.

1. *Обеспечение конфиденциальности данных в памяти ядра*: если очистить P-бит в PTE защищаемой страницы, то при попытке

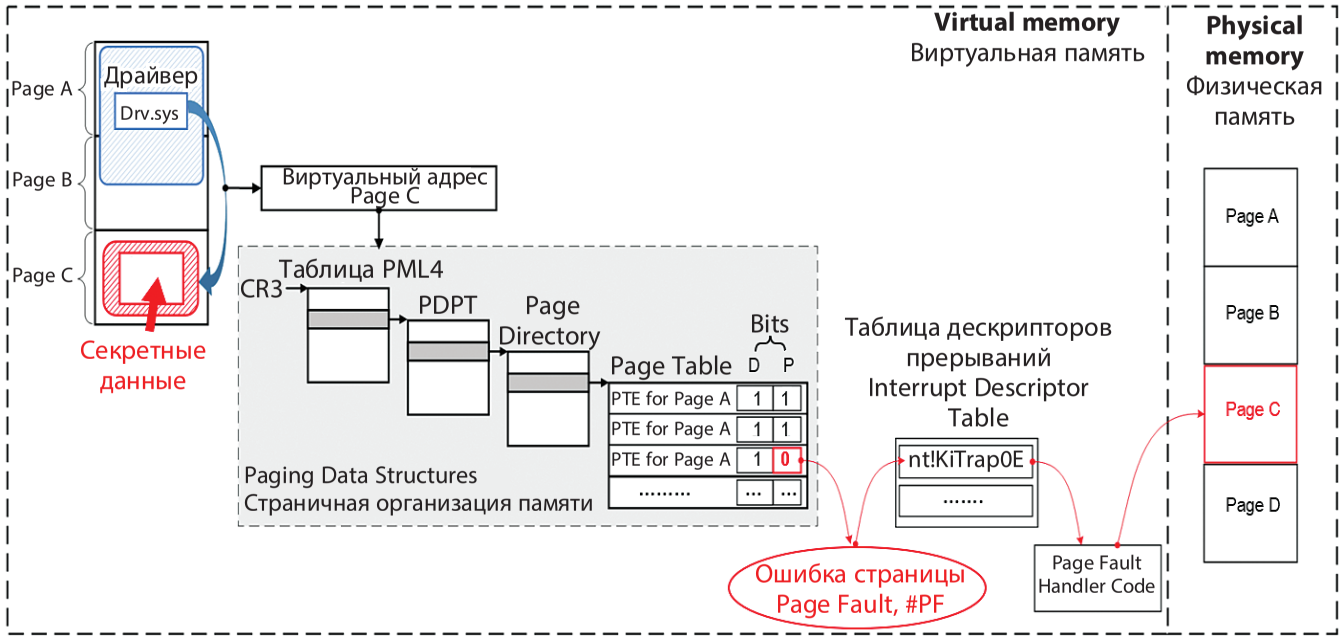


Рис. 37. Механизм сокрытия и управления доступом к памяти на основе модификации записей таблицы страниц и обработчика ошибок страниц:

PDPT — таблица указателей на каталоги страниц (Page Directory Pointer Table); PTE for PageA, B, C — запись таблицы страниц A, B, C; биты (bits): D — бит dirty (бит внесения изменений), P — бит present (бит присутствия страницы в памяти), nt!KiTrap0E — обработчик nt!KiTrap0E; Page Fault Handler Code — код обработчика ошибки страницы

чтения будет сгенерировано исключение #PF [200]. Модифицированный обработчик может перехватить это исключение и вернуть вызывающему процессу «поддельную» страницу, предотвращая доступ к фактическим данным в памяти.

2. *Обеспечение целостности данных в памяти ядра*: для обеспечения целостности системных структур можно сбросить бит Dirty (D-бит) в PTE соответствующей страницы. При попытке записи процессор сгенерирует исключение #PF. Модифицированный обработчик сможет обработать это исключение, предотвращая несанкционированные изменения.

3. *Контроль выполнения исполнимого кода в памяти ядра*: для контроля выполнения кода на странице памяти возможно использовать сброс бита NX (No eXecute) / XD (eXecute Disable). Если попытка выполнения кода происходит на странице, помеченной как non-executable, то процессор генерирует исключение, которое обрабатывается операционной системой и гипервизором. В архитектуре IA-32 отсутствует прямой способ различать операции чтения и выполнения. Однако, используя метод из первого сценария и адаптируя способ, описанный в работе [196], можно определить попытки выполнения кода, если SourceAddr и DestinationAddr совпадают. SourceAddr — сохраненный указатель инструкции (Instruction Pointer) на момент возникновения исключения. Он указывает на адрес команды, выполнение которой вызвало исключение. DestinationAddr — адрес памяти, к которому пыталась обратиться инструкция, приведшая к исключению.

4. Например, при попытке выполнения кода из страницы памяти, которая помечена как отсутствующая (бит Present в PTE сброшен), процессор генерирует исключение Page Fault, и оба адреса — SourceAddr и DestinationAddr — будут указывать на одну и ту же инструкцию, поскольку попытка выполнения происходит по адресу, вызвавшему исключение.

Ниже описаны некоторые существующие подходы, которые помечают страницы как не присутствующие и заменяют обработчик ошибок страниц.

1. Контроль доступа, основанный на повторном использовании кода, путем перехвата исключений Page Fault предложен

в работе [197]. В рамках этого подхода программам запрещено чтение исполняемой памяти, что затрудняет использование уязвимостей, связанных с выполнением кода.

2. Мониторинг доступа к памяти на уровне страниц использован в драйвере ядра Omnipack для обнаружения момента, когда программа завершает процесс распаковки. Omnipack отслеживает записанные байты и применяет политику write-xor-execute (W+X), предотвращая выполнение сгенерированного кода во время распаковки. Аналогичная политика W+X используется в системе SecVisor [201].

3. Защита адресного пространства процесса путем перехвата каждого исключения Page Fault, сигнализирующего о нарушении доступа, реализована в системе KLIMAX [202].

4. Мониторинг памяти с использованием обработчика исключений реализован в системе SPIDER [203], которая предоставляет возможность мониторинга операций чтения и записи в память. Авторы отмечают две основные проблемы применения существующих техник, основанных на механизме перехвата на уровне страниц. Во-первых, каждая инструкция, обращающаяся к отсутствующей странице, вызывает исключение Page Fault. Это приводит к значительным накладным расходам на обработку исключений и, как следствие, к заметному снижению производительности системы, особенно при интенсивном доступе к защищаемым областям памяти. Во-вторых, внесенные изменения в таблицы страниц (PTE) и обновленный обработчик исключений могут быть обнаружены драйверами ядра, что позволяет злоумышленникам идентифицировать факт применения защитных механизмов и попытаться им противодействовать. Эти ограничения существенно сужают область практического применения подобных техник в условиях противодействия современным угрозам.

Ограничения методов, основанных на модификации PTE и замене обработчика Page Fault, представлены в работе [196], где показано, что эти методы не всегда поддерживают страницы размером 4 Мб и 1 Гб, а замененный обработчик Page Fault может быть легко обнаружен. Кроме того, использование обработчика Page Fault в качестве посредника при мониторинге памяти су-

щественно влияет на производительность системы. Также возникают сложности при переносе этого метода на многоядерные процессоры.

Основным недостатком методов, реализованных на уровне ОС, является их уязвимость: они могут быть обнаружены и отключены вредоносным ПО, работающим в режиме ядра. Методы, основанные на гипервизорах, лишены этих недостатков, однако требуют наличия процессора с поддержкой аппаратной виртуализации.

7.1.2. Методы на основе гипервизора

Методы, описанные в этом подпараграфе, обеспечивают более высокий уровень изоляции и безопасности, но требуют наличия технологии аппаратной виртуализации в процессоре, которая поддерживается во всех современных центральных процессоров (ЦП). Первый метод использует технологию Intel Virtualization Technology (VT-x) без поддержки технологии EPT. В результате этот метод будет работать даже на устаревших процессорах поколения Intel Core 2. Второй метод применяет Intel VT-x с технологией EPT, которая может быть использована начиная со 2-го поколения процессоров Intel — семейства i3, i5 и i7. Технология Intel Processor Trace (PT) была добавлена в процессоры начиная с 5-го поколения.

Обработка исключений ошибки страницы (Page Fault) гипервизором. Гипервизор отслеживает обращения к памяти путем настройки срабатывания VM-Exit при возникновении исключений #PF (рис. 38).

Для того чтобы настроить гипервизор на обработку исключения #PF, необходимо задать соответствующую конфигурацию в структуре управления виртуальной машиной (Virtual Machine Control Structure, VMCS). Для этого нужно установить 14-й бит в поле Exception Bitmap, которое находится в разделе VM-Execution Control Fields структуры VMCS. Это позволит гипервизору перехватывать все случаи возникновения исключений типа Page Fault и обрабатывать их согласно заданным правилам.

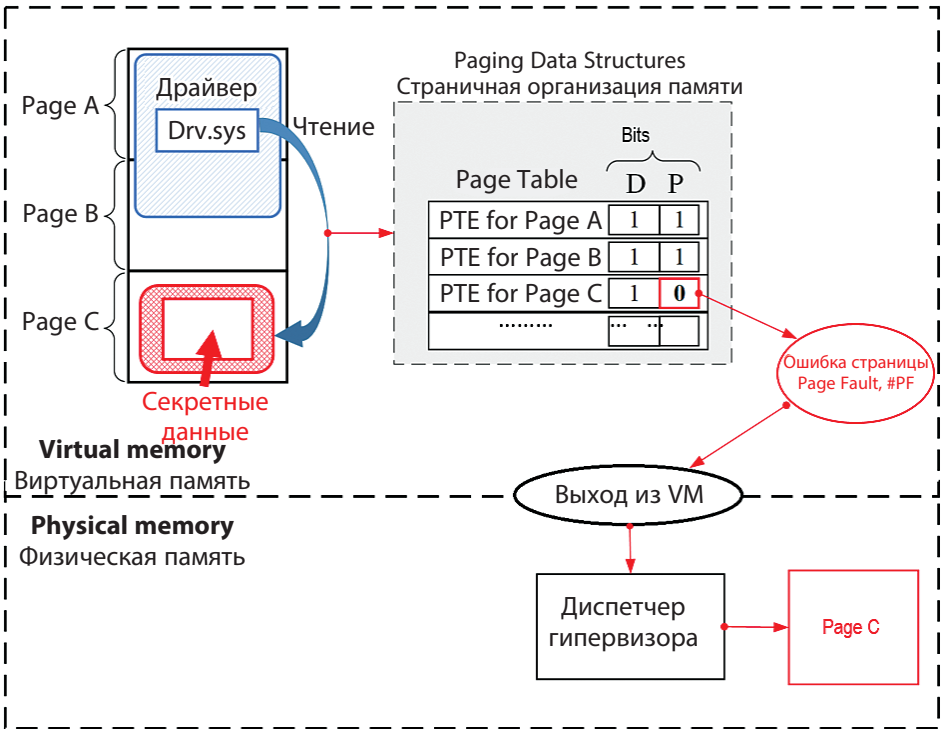


Рис. 38. Контроль доступа к данным гостевой ОС с использованием гипервизора

Применяя данный метод, гипервизор способен фиксировать как код, осуществляющий попытку доступа к памяти «адрес источника» (SourceAddr), так и адрес памяти, к которой было обращение «адрес назначения» (DestinationAddr). Это позволяет реализовывать различные сценарии по защите памяти. При обработке PageFault гипервизор извлекает адрес инструкции, вызвавшей исключение, из регистра EIP [204].

Авторы работы [205] предложили гипервизор DriverGuard для защиты промышленных инфраструктурных систем от целевых атак (Advanced Persistent Threat, APT). Они отмечают, что большинство таких угроз представляет собой «атаки нулевого дня», а инструменты безопасности, основанные на сигнатурах, не способны их обнаружить. Гипервизор DriverGuard обеспечивает защиту данных и кода виртуальной машины от несанкционированного доступа.

DriverGuard управляет таблицами страниц (PTE) и изменяет бит Present (P). В результате любой доступ к странице вызывает исключение ошибки страницы (PageFault), которое перехватывается DriverGuard. Далее гипервизор анализирует, является ли этот доступ легитимным. Если доступ выполняется легитимным кодом, то DriverGuard применяет новую технику точек останова, используя аппаратные точки останова в режиме пошагового выполнения. Этот режим активируется установкой бита Monitor Trap Flag (MTF) в структуре VMCS.

Гипервизор DriverGuard распознает области памяти, содержащие конфиденциальные данные, с помощью специального «тега» (tag). Области памяти, которые выделяются с использованием функции ExAllocatePoolWithTag с данным «тегом», защищаются. Однако нарушители могут выявить этот «тег» и использовать его в своем ВПО. Авторы подтверждают, что обработка исключений происходит довольно медленно, что может привести к значительному снижению производительности системы при частых срабатываниях PageFault.

В 2015 г. была разработана система защиты MOSKG, предназначенная для противодействия руткитам ядра с использованием механизма страничного доступа [206]. Основной целью этого исследования являлось предотвращение работы руткитов путем защиты данных в памяти ядра (структур объектов) от манипуляций типа DKOM.

Авторы отмечают важную проблему: динамические данные могут быть как легитимно изменены операционной системой, так и нелегитимно модифицированы с использованием руткитов. Крайне важно гарантированно отличать законные операции от незаконных. В работе [207] рассмотрена проблема защиты целостности гипервизора. Авторы предположили, что в модели угроз нарушители могут эксплуатировать уязвимости в ПО для перезаписи данных в памяти. Особое внимание было уделено гипервизорам, таким как Xen и KVM, а также ОС семейства Windows и Linux, где таблицы страниц остаются доступными для записи. Эксперименты показали, что изменение даже одного бита в записи таблицы страниц может быть достаточным для нарушения работы.

Для решения этой проблемы авторы предложили метод HyperSafe, который представляет собой подход к защите кода и данных гипервизора от компрометации. Для обеспечения целостности на основе W+X (запись или исполнение, но не одновременно) HyperSafe помечает таблицы страниц как доступные только для чтения и включает бит защиты от записи (WP — Write Protect) в регистре CR0. В результате любая попытка записи в защищенные страницы во время выполнения перехватывается гипервизором, что предотвращает несанкционированные изменения. Однако HyperSafe имеет ограничение: он может защищать только продукты с открытым исходным кодом. Поддержка закрытых драйверов сторонних производителей остается проблемой, которую еще предстоит решить.

Существуют патенты, которые охватывают идеи отслеживания памяти путем маркировки записей таблицы страниц ядра гостевой системы как отсутствующих и управления исключениями ошибок страниц (Page Fault). В патенте № EP2766844A1 [203] описывается обработчик исключений гипервизора, который может разрешать или запрещать доступ к страницам ядра гостевой системы или выполнение кода на этих страницах. Учитывая, что правообладателем данного патента является компания *McAfee*, Inc. из Санта-Клары, Калифорния (США), можно предположить, что этот метод применяется в антивирусных продуктах McAfee или их внутренних системах.

Механизм защиты от страничных сбоев используется для контроля доступа к страницам режима ядра, а также к страницам пользовательского режима. Прототипом защиты памяти пользовательского режима является проект U-HIPE [208].

Метод мониторинга выполнения кода в режиме ядра путем разделения таблиц страниц для данных и драйверов описан в работе [209]. Это разделение заставляет любой поток управления, пересекающий интерфейс «ядро — драйвер», вызывать исключение, которое обрабатывается кодом гипервизора, обеспечивающего контроль легитимности потока управления. В результате была создана система на основе гипервизора Gateway, которая отслеживает поведение вредоносного кода в ядре посредством мониторинга функций API ядра, вызываемых драйверами.

Похожий метод обработки исключений ошибок страниц был применен в гипервизорной системе HyperSleuth для трассировки системных вызовов [210]. Поскольку все обращения к системным вызовам проходят через общий шлюз, адрес которого определяется регистром SYSENTER_EIP, исследователи создали теневую копию этого регистра и установили ее значение на адрес несуществующей области памяти. В результате все системные вызовы приводят к возникновению исключения ошибки страницы. HyperSleuth перехватывает и сохраняет все системные вызовы в журнал, который затем передается по сети на удаленный хост для экспертного анализа.

Другой подход заключается в перехвате системных вызовов с использованием механизма интроспекции виртуальной машины [211]. Система Nitro работает с такими системными вызовами, как пользовательские прерывания и инструкции SYSCALL/SYSENTER. Однако эта система не способна мониторить вызовы функций в режиме ядра.

Гипервизорная система HIMA, предназначенная для контроля целостности памяти виртуальных машин, описана в работе [212]. В HIMA также используется идея защиты памяти гостевой системы через настройки доступа к страницам. Система применяет бит No eXecute (NX) в записях таблицы страниц. Если этот бит установлен в 1, то страница считается содержащей только данные. Любая попытка выполнения инструкций на этой странице вызывает исключение ошибки страницы, которое перехватывается гипервизором.

В диссертационной работе [213] показано, что данный метод мониторинга памяти неустойчив к атакам типа «отказ в обслуживании» (DoS). Проблема заключается в том, что ВПО может вызвать большое количество исключений ошибок страниц и каждое из них приводит к выходу в гипервизор. Этот поток исключений инициирует DoS-атаку на записываемый журнал, что затрудняет его последующий анализ.

Помимо уязвимости к DoS-атакам, этот метод имеет следующие недостатки:

- *отсутствие скрытности*: модификация атрибутов страниц или флагов PTE видна из гостевой системы и может быть легко обнаружена ВПО;

- *высокая нагрузка на систему*: обработка каждой ошибки страницы требует времени, что приводит к значительному снижению производительности;
- *неполная поддержка многоядерных процессоров*: поскольку запись таблицы страниц (PTE) существует только для одной страницы и используется всеми ядрами, ее модификация влияет на контексты других процессоров.

7.1.3. Использование технологии Extended Page Tables

Новая технология Extended Page Tables (EPT) расширяет возможности гипервизора для контроля доступа к памяти на уровне страниц, устанавливая ограничения на операции чтения, записи и выполнения.

Для расширения возможностей гипервизора по работе с памятью гостевой системы была разработана технология Second Layer Address Translation (SLAT), также известная как двухмерная страничная организация (Two-Dimensional Paging) [214, 215]. Цель SLAT — снижение накладных расходов гипервизора при управлении памятью за счет аппаратной оптимизации перевода виртуальных адресов в физические.

Технология SLAT была интегрирована в процессоры *Intel* начиная с микроархитектуры Nehalem в первых моделях Core i3, i5 и i7. В терминологии *Intel* эта технология называется Intel VT-x with Extended Page Tables (EPT). Аналогичная технология была разработана *AMD* под названием Nested Page Table (NPT). В данном подпараграфе основное внимание уделяется реализации EPT в процессорах *Intel*, однако полученные выводы в целом применимы и к другим реализациям технологии SLAT.

Механизм работы EPT и его использование для мониторинга памяти. Технология EPT позволяет виртуализировать физическую память гостевой системы, тем самым расширяя возможности процессора при использовании структур данных страничной организации, известных как EPT layout. Алгоритм работы структур данных EPT, преобразующий физический адрес гостевой системы в физический адрес хоста, аналогичен алгоритму структур данных страничной организации в защищенном режиме, который преобразует виртуальный адрес го-

стевой системы в ее физический адрес. Содержимое и организация структур EPT подобны таблицам страниц в защищенном режиме или таблицами страниц архитектуры x86-64. Однако между структурами EPT и таблицами страниц гостевой системы существуют некоторые различия [216].

Структуры EPT включают следующие таблицы (рис. 39):

- Page Map Level 4 (EPT PML4);
- Page-Directory-Pointer Table (EPT PDPT);
- Page Directory (EPT PD);
- Page Table (EPT PT).

Гипервизор должен выделять память для всех этих таблиц и управлять их содержимым. Используя различные конфигурации структур EPT, гипервизор может реализовывать различные сценарии страничной организации памяти. В работе [217] рассматривается простой сценарий с отображением памяти 1 : 1 (взаимно-однозначное отображение памяти), где физический адрес гостевой системы преобразуется в тот же физический адрес на хосте (см. рис. 39). При каждом доступе к памяти «внутри» гостевой ОС (Guest OS) сначала используются структуры данных страничной организации гостевой системы. Затем, на последнем этапе, структуры EPT преобразуют физический адрес гостевой системы в физический адрес хоста.

Доступ к памяти можно перехватывать, изменяя биты в соответствующей записи таблицы страниц EPT Page Table, в то время как записи в других таблицах, таких как EPT PML4, EPT PDPT и EPT PD, сохраняют свои исходные значения.

Запись EPT Page Table содержит следующие биты, управляющие доступом к соответствующей странице:

- бит#0 — Read Access: указывает, разрешено ли чтение с 4 Кб-страницы;
- бит#1 — Write Access: указывает, разрешена ли запись на 4 Кб-страницу;
- бит#2 — Execute Access: показывает, разрешено ли выполнение инструкций с 4 Кб-страницы.

Согласно документации *Intel* (глава 28) [218] «любая попытка недопустимого доступа вызовет нарушение EPT (EPT violation) и приведет к VM-Exit». Гипервизор перехватывает каждое нарушение

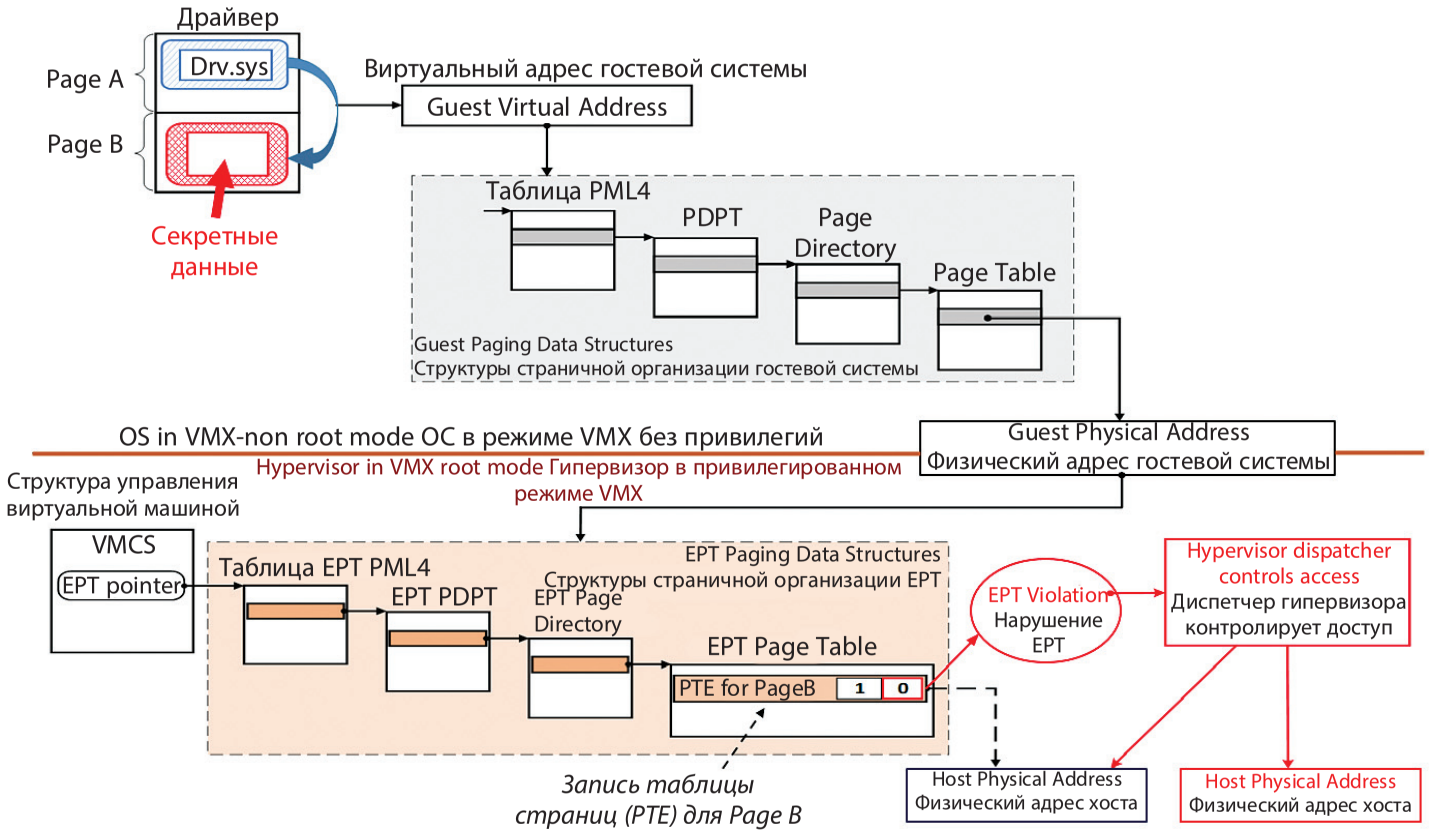


Рис. 39. Управление доступом к виртуальной памяти гостевой ОС с помощью трансляции адресов второго уровня

EPT (VM-Exit) и может применять различные алгоритмы, в том числе логирование попыток обращений к памяти и другие алгоритмы по защите памяти ядра.

Гипервизор может отслеживать любые операции чтения и записи в память, выполняемые драйвером ВПО, сбрасывая первые два бита в записи таблицы EPT PT. Для защиты содержимого страниц памяти от чтения он может сбросить бит#0 в записи EPT PT и заменить требуемую физическую страницу, например, на пустую. Для обеспечения целостности страницы памяти гипервизор может сбросить бит#1, перехватывая каждую попытку записи в эту страницу.

Руткит-гипервизор способен защищать вредоносный драйвер от антивирусов и встроенных механизмов безопасности Windows, таких как PatchGuard. Вопросы обнаружения руткит-гипервизоров будут рассмотрены в разделе 3.

Анализ решений по кибербезопасности на основе EPT. Система динамического анализа DRAKVUF позволяет отслеживать выполнение кода и обнаруживать атаки на ядро с использованием техники DKOM [219]. Эта система основана на технологиях VT-x и EPT и построена на основе гипервизора Xen с использованием библиотеки LibVMI. Одна из техник DRAKVUF основана на встраивании инструкций прерывания (код опкода 0xCC) в места кода, представляющие интерес. Эта техника, называемая внедрением точек останова, перехватывается гипервизором DRAKVUF, что позволяет автоматизировать отслеживание выполнения кода в ОС и перехватывать различные вызовы функций. DRAKVUF также применяет EPT для защиты памяти, что обеспечивает механизм анализа виртуальной машины (VMI). Еще одна техника DRAKVUF заключается в мониторинге доступа к файлам путем отслеживания структуры _FILE_OBJECT. Для этого страница памяти, на которой размещена структура, помечается как недоступная для записи с использованием EPT. Однако система поддерживает только Windows 7 в 32- и 64-битных версиях (Drakvuf, 2016), и портирование системы DRAKVUF на новые версии ОС требует работы эксперта.

Другой системой мониторинга является CXPInspector, представленная в работе [240]. Эта система позволяет анализиро-

вать состояние виртуальной машины, используя концепцию Currently eXecutable Pages (CXP) [221]. Данная система применяет EPT для управления памятью и наблюдения за выполнением кода путем маркировки определенных страниц как неисполняемых. CXPInspector также обогащает генерируемые логи, фиксируя вызовы функций и их параметры с помощью информации отладочных символов. Главной особенностью системы является ее способность записывать адреса памяти, с которых происходят вызовы или возвраты, с использованием технологии Intel Processor Trace (Intel PT), реализованной в процессорах *Intel* начиная с 5-го поколения. CXPInspector построена на гипервизоре KVM с интерфейсом к QEMU и поддерживает Windows 7.

Фреймворк на уровне гипервизора для мониторинга системных событий HyperTap предложен в работе [222]. HyperTap способен отслеживать переключение контекста, системные вызовы, выполнение инструкций и обращения к памяти. Фреймворк использует механизмы защиты памяти, устанавливая права доступа к областям памяти Memory Mapped I/O для вызова событий EPT_VIOLATION. Система реализует алгоритмы для выявления скрытых процессов (Hidden RootKit Detection, HRKD) и обнаружения повышения привилегий (Privilege Escalation Detection, PED). HyperTap была протестирована на Windows XP, Vista и 7.

Использование гипервизора для внедрения невидимых inline hooks описано в работе [217]. В этом случае гипервизор используется для контроля попыток доступа к страницам памяти с установленными перехватчиками (хуками). Так, при чтении страницы гостевая система видит исходную страницу, а при выполнении кода гипервизор использует перехватчик и передает управление обработчику. Эта схема позволяет устанавливать перехватчики на функции ядра ОС без срабатывания средств защиты ядра, таких как PatchGuard, которые должны были контролировать целостность этих страниц памяти. Это достигается благодаря тому, что гипервизор работает на более низком уровне, чем ядро ОС, и может обойти механизмы защиты, предназначенные для обнаружения изменений в памяти ядра.

Основные ограничения решений на основе EPT:

- *гранулярность управления* — технология EPT работает на уровне страниц памяти, без встроенной возможности контроля попыток обращений к памяти на уровне байтов (битов);
- *ограниченный контроль доступа* — нет открытых решений, которые одновременно контролируют все типы доступа (чтение, запись и выполнение в памяти ядра);
- *зависимость от платформы* — большинство решений основаны на крупных платформах, таких как Xen, KVM, QEMU, и используют Linux в качестве хостовой ОС, что непригодно для сценария с Windows ОС.

Технология EPT предоставляет широкие возможности для мониторинга и управления доступом к страницам памяти, что может быть использовано в качестве основы для разработки новых систем безопасности.

Сводный сравнительный анализ основных работ и проектов приведен в табл. 13, где ЦП 2-го поколения поддерживают технологии VT-x и EPT — микроархитектура Nehalem. CPU 5-го поколения поддерживают технологии VT-x, EPT и PT — микроархитектура Broadwell.

В параграфе 7.2 будет представлено решение MemoryMon RWX, которое лишено всех перечисленных выше недостатков.

Таблица 13

Сводная таблица проектов мониторинга памяти

Название, год выпуска	Управление типом доступа			Поддерживаемая ОС	Требуемое поколение процессоров
	чтение	запись	исполнение		
HIMA, 2009	–	–	+	Linux	2-е
HyperSleuth, 2010	+	–	–	Windows 7, 8	2-е
CXPInspector, 2013	–	–	+	Windows 7	5-е
SPIDER, 2013	+	+	–	Linux	2-е
DRAKVUF, 2014	–	–	+	Windows 7	2-е

Окончание табл. 13

Название, год выпуска	Управление типом доступа			Поддерживаемая ОС	Требуемое поколение процессоров
	чтение	запись	исполнение		
HyperTap, 2014	–	+	+	Windows Vista, 7	2-е
Memory MonRWX, 2017	+	+	+	Windows 7-10 x64	2-е

7.2. АРХИТЕКТУРА И РЕАЛИЗАЦИЯ ГИПЕРВИЗОРА MEMORYMONRWX ДЛЯ МОНИТОРИНГА ОБРАЩЕНИЙ ДРАЙВЕРОВ К ПАМЯТИ ЯДРА

MemoryMonRWX (Memory Monitor of Read, Write, and eXecute access) — это гипервизор, разработанный для мониторинга и контроля доступа к памяти ядра Windows. Это bare-metal-гипервизор, разработанный для мониторинга и контроля попыток чтения, записи и исполнения памяти ядра Windows в реальном времени, в том числе со стороны драйверов. Он использует технологии аппаратной виртуализации, такие как Intel VT-x с EPT, для перехвата всех типов обращений к памяти: чтения, записи и исполнения. Система поддерживает многоядерные процессоры и, соответственно, запускается на каждом ядре. Это первая система мониторинга памяти, которая позволяет контролировать даже однобайтовые модификации гостевой ОС. MemoryMonRWX прошел успешное тестирование на Windows 10 14393 x64.

Основные компоненты архитектуры MemoryMonRWX включают:

- *модуль мониторинга* — отслеживает обращения к областям памяти ядра, обеспечивая защиту от несанкционированного доступа;
- *модуль логирования* — записывает все обращения к заданным областям памяти, что позволяет анализировать поведение драйверов и выявлять потенциально вредоносные действия;
- *модуль управления* — обеспечивает взаимодействие с пользователем и настройку параметров мониторинга.

7.2.1. Применение EPT-технологии для отслеживания и контроля доступа к памяти

Современные вредоносные драйверы и руткиты ядра используют различные техники для сокрытия своей активности, включая чтение и модификацию данных и кода в памяти. Для защиты от таких угроз в системе MemoryMonRWX реализовано два основных сценария мониторинга и контроля доступа к памяти:

- *мониторинг доступа из подозрительного драйвера* — MemoryMonRWX отслеживает доступ к памяти, осуществляемый драйверами, загруженными из заданных (известных) диапазонов виртуальных адресов (SRC range). MemoryMonRWX содержит список имен подозрительных драйверов и при загрузке драйвера из списка добавляет соответствующий диапазон адресов в список SRC range;
- *контроль доступа к защищаемым адресам памяти* — система контролирует доступ к памяти в диапазоне защищаемых адресов (DST range), чтобы предотвратить несанкционированный доступ.

Система MemoryMonRWX контролирует доступ исключительно между диапазонами SRC и DST, игнорируя все остальные обращения, относящиеся к OTH:

- SRC range включает виртуальные адреса, с которых осуществляется доступ и которые должны быть перехвачены;
- DST range включает виртуальные адреса, к которым осуществляется доступ и которые также подлежат перехвату.

Все операции, которые происходят вне диапазонов SRC и DST, не перехватываются системой мониторинга и обрабатываются ОС без вмешательства со стороны гипервизора. Такие диапазоны объединяются в так называемый OTH range (other range).

В рассматриваемом случае диапазоны SRC, DST и OTH являются непересекающимися и их объединение охватывает весь виртуальный адресный контекст.

Для управления доступом к памяти MemoryMonRWX применяет технологию виртуализации Intel VT-x с расширенной таблицей страниц (EPT), которая существенно увеличивают

функциональные возможности bare-metal-гипервизора. Используя структуры расширенных таблиц страниц (EPT), можно реализовать контроль доступа к памяти из диапазона источника (SRC) в диапазон назначения (DST), игнорируя все остальные обращения (рис. 40, *а*). Алгоритм работы MemoryMonRWX можно описать поэтапно следующим образом:

1. *Преобразование адресов*: гостевая система обращается к памяти и происходит перевод из гостевого виртуального адреса (GVA) в гостевой физический адрес (GPA), как показано на рис. 40 стрелкой 1.

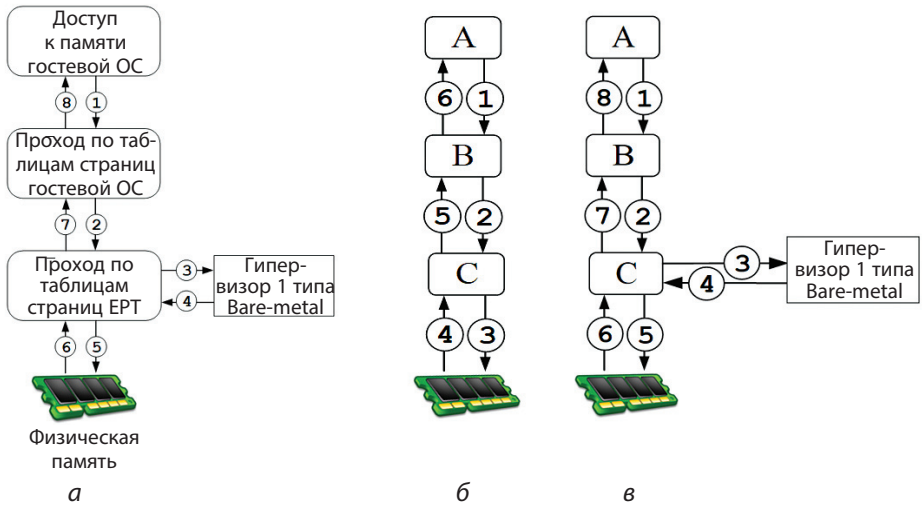


Рис. 40. Алгоритм перехвата доступа к памяти с помощью EPT:
а — общий вид; *б* — структуры EPT имеют разрешенные атрибуты;
в — структуры EPT имеют запрещенные атрибуты

2. *Преобразование гостевого физического адреса в физический адрес хоста*: после завершения обхода страниц (page walk) выполняется преобразование GPA в физический адрес хоста (HPA) с использованием структур EPT (стрелка 2).

3. *Проверка разрешений доступа*: если доступ к памяти разрешен согласно битам доступа в записях таблицы страниц EPT, то гипервизор не перехватывает управление и операция продолжается без вмешательства.

4. *Обработка нарушения EPT*: если доступ запрещен битами доступа в таблицах страниц EPT, то происходит нарушение EPT

(EPT violation), что вызывает выход из виртуальной машины (VM Exit) (стрелка 3).

5. *Логирование и контроль со стороны гипервизора*: гипервизор получает возможность регистрирования и контроля попыток доступа к данной странице памяти посредством модификации структур страничной организации EPT (стрелка 4).

6. *Доступ к физической памяти и возвращение управления*: после внесения необходимых изменений в памяти (стрелки 5 и 6) управление от гипервизора возвращается гостевой системе (стрелки 7, 8).

Таким образом, посредством использования EPT и перехвата обращений к памяти гипервизор контролирует доступ между заданными диапазонами адресов, без влияния на остальные операции в памяти.

Сценарии работы с EPT:

1) при установке разрешающих атрибутов в EPT гипервизор не перехватывает доступ, как показано на рис. 40, *б*;

2) для перехвата операций чтения с гостевой страницы атрибуты доступа в EPT модифицируются, что вызывает срабатывания EPT violation и передачу управления гипервизору, как на рис. 40, *в*.

7.2.2. Алгоритм управления доступом к памяти с использованием MemoryMonRWX

Система MemoryMonRWX позволяет отслеживать доступ к памяти и управлять им, помогая логировать обращения к памяти со стороны заданного драйвера (драйверов), а также попытки чтения и записи к данным в памяти. Метод контроля доступа к памяти использует два сценария: управление доступом из диапазона SRC и контроль доступа к диапазону DST. Этот процесс реализуется через пять шагов с использованием структур EPT.

Шаг 1: перехват выполнения кода из диапазона SRC. На первом этапе система использует структуру EPT для перехвата всех операций выполнения кода из диапазона SRC (см. рис. 40, *а*). В этой конфигурации доступ к диапазонам DST и OTH игнорируется. Любая попытка выполнения кода из диапазона SRC вызывает нарушение EPT Violation и инициирует со-

бытие VM Exit, после чего управление передается гипервизору для анализа и перехода к следующему шагу.

Шаг 2: VM Exit из-за выполнения кода в диапазоне SRC. На этом этапе гипервизор использует обновленную структуру EPT, чтобы отфильтровать доступ к диапазону DST (см. рис. 40, *б*). Любая попытка кода, перехваченного на предыдущем шаге, получить доступ к DST приводит к новому событию VM Exit. Если это происходит, то система переходит к следующему шагу.

Шаг 3: VM Exit из-за доступа к диапазону DST. Гипервизор фиксирует попытки доступа к защищенной памяти, записывая такие параметры, как:

- адреса SRC и DST;
- тип доступа (чтение, запись или выполнение);
- значения байтов, которые были прочитаны или перезаписаны.

Если необходимо защитить данные или код, то выполняются следующие процедуры:

- *замена страницы* — значение EPT.PFN для защищенной страницы изменяется, чтобы перенаправить доступ на пустую страницу;
- *разрешение доступа* — устанавливаются права на чтение и запись (EPT.DST.read = true, EPT.DST.write = true);
- *установка флага Monitor Trap Flag* (*MTF*), что позволяет генерировать событие VM Exit после выполнения каждой инструкции [223].

После этих действий система переходит к следующему шагу.

Шаг 4: VM Exit из-за установленного флага Monitor Flag Trap (MTF). При выставленном флаге MTF после выполнения каждой инструкции кода управление прерывается и передается гипервизору. Гипервизор, получив управление, выполняет следующие действия для восстановления исходной конфигурации:

- восстановление EPT.PFN на оригинальную страницу;
- ограничение прав доступа (EPT.DST.read = false, EPT.DST.write = false);
- сброс флага MTF.

Шаг 5: VM Exit из-за выполнения кода вне диапазона SRC. Система проверяет, принадлежит ли адрес VM Exit диапазону

SRC. Если адрес находится за пределами SRC, то это означает, что выполнение не требует контроля, и система возвращается к стандартной структуре EPT, готовой к следующему циклу перехвата. Если же адрес принадлежит диапазону SRC, то процесс возвращается к шагу 1.

Предложенный алгоритм представлен на рис. 41 и в табл. 14. Валидация концепции осуществлялась с использованием гипервизора MemoryMonRWX, разработанного для контроля доступа к памяти и набора тестовых драйверов, реализующих несанкционированный доступ, что подробно описано в подпараграфе 7.2.3.

EPT Normal View
(Нормальный режим EPT)
- SRC.read = true
(Чтение из диапазона SRC разрешено)
- SRC.write = true
(Запись в диапазон SRC разрешена)
- SRC.exec = false
(Любая попытка выполнения вызывает VM-Exit)
- DST.read = true
(Чтение из диапазона DST разрешено)
- DST.write = true
(Запись в диапазон DST разрешена)
- DST.exec = true
(Выполнение из диапазона DST разрешено)
- OTH. read = true
(Чтение из диапазона OTH разрешено)
- OTH.write = true
(Запись в диапазон OTH разрешена)
- OTH. exec = true
(Выполнение из диапазона OTH разрешено)

EPT Monitor View
(Мониторинговый режим EPT)
- SRC.read = true
(Чтение из диапазона SRC разрешено)
- SRC.write = true
(Запись в диапазон SRC разрешена)
- SRC.exec = true
(Выполнение из диапазона SRC разрешено)
- DST. write = false
(Любая попытка записи вызывает VM-Exit)
- DST.exec = false
(Любая попытка выполнения вызывает VM-Exit)
- OTH. read = true
(Чтение из диапазона OTH разрешено)
- OTH.write = true
(Запись в диапазон OTH разрешена)
- OTH. exec = false
(Любая попытка выполнения вызывает VM-Exit)

Рис. 41. Состояние структур EPT

Предложенная схема взаимодействия между различными режимами EPT (Extended Page Tables) для логирования и контроля доступа к памяти в системе MemoryMonRWX представлена на рис. 42.

Таблица 14

Сводная таблица манипуляций VM-Exit, если адрес доступа принадлежит диапазону SRC

Тип доступа	Текущий адрес	
	внутри диапазона DST	вне диапазона DST
Чтение/ Запись	VM-exit & Запись в журнал	Нет действий
Исполнение кода	VM-exit & Запись в журнал & Переключение в нормальный режим	VM-exit & Переключение в нормальный режим

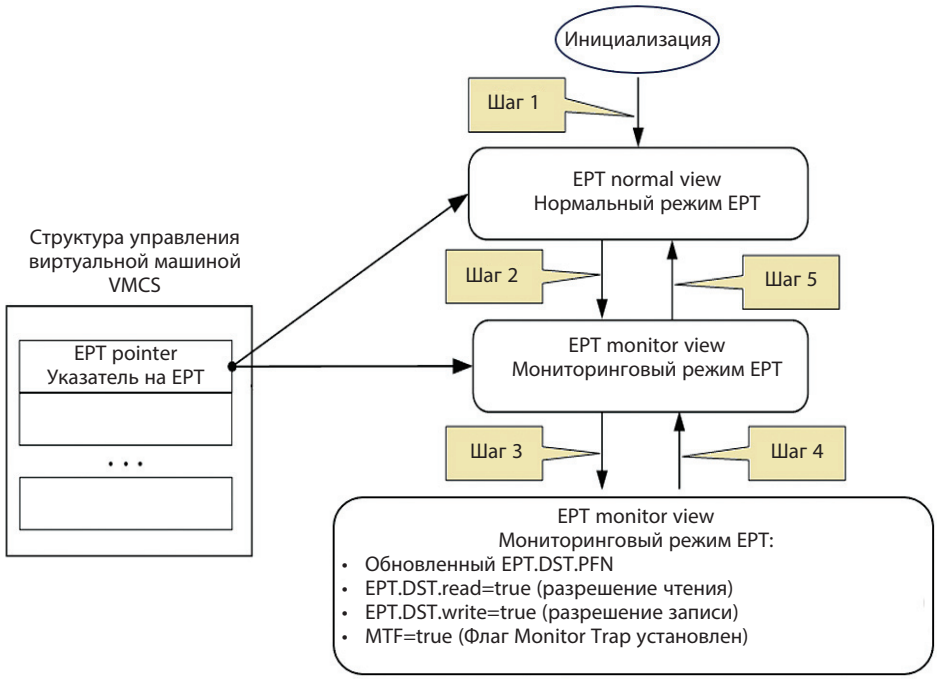


Рис. 42. Взаимодействие между представлениями EPT для регистрации и контроля доступа к памяти

Процесс начинается с начальной настройки и переключения в нормальный режим EPT (EPT normal view), который обеспечивает базовую работу. На шаге 1 система работает в нормальном режиме EPT normal view, где выполняется мониторинг выполнения кода в диапазоне SRC. На шаге 2 происходит переклю-

чение в основной мониторинговый режим EPT (EPT monitor view), если обнаружен доступ к защищенной памяти. Этот режим позволяет гипервизору перехватывать попытки доступа к памяти и фиксировать любые действия, связанные с диапазоном DST.

На шаге 3 используется расширенный мониторинговый режим EPT (EPT monitor view with), где применяется замена страницы памяти (EPT.DST.PFN) и устанавливаются права на чтение и запись. Устанавливается флаг Monitor Trap Flag (MTF), который обеспечивает передачу управления обработчику гипервизора после выполнения каждой инструкции процессора, что позволяет реализовать пошаговый контроль над действиями гостевой системы и отслеживать все обращения к защищаемой памяти.

На шаге 4 гипервизор после обработки данных восстанавливает оригинальные параметры доступа и переходит в основной мониторинговый режим EPT (EPT monitor view) для дальнейшего контроля новых попыток обращений.

Завершающий шаг 5 предполагает возврат к EPT normal view, чтобы подготовиться к обработке нового цикла мониторинга и перехвата.

Этот процесс иллюстрирует последовательное взаимодействие между режимами EPT, которое позволяет управлять доступом к памяти, фиксировать и предотвращать несанкционированные операции по доступу к памяти.

7.2.3. Архитектура и основные компоненты MemoryMonRWX

Гипервизор MemoryMonRWX [224] использует технологии Intel VT-x и EPT. MemoryMonRWX включает следующие основные компоненты: HyperPlatform, Image Load Detector, Source/Destination Range Manager (Менеджер диапазонов источника и назначения), Virtual-to-Physical Map Manager (Менеджер отображения виртуальных адресов в физические) и EPT Controller (Контроллер EPT).

Общая архитектура системы. Архитектура и работа системы MemoryMonRWX схематически представлены на рис. 43. Основным

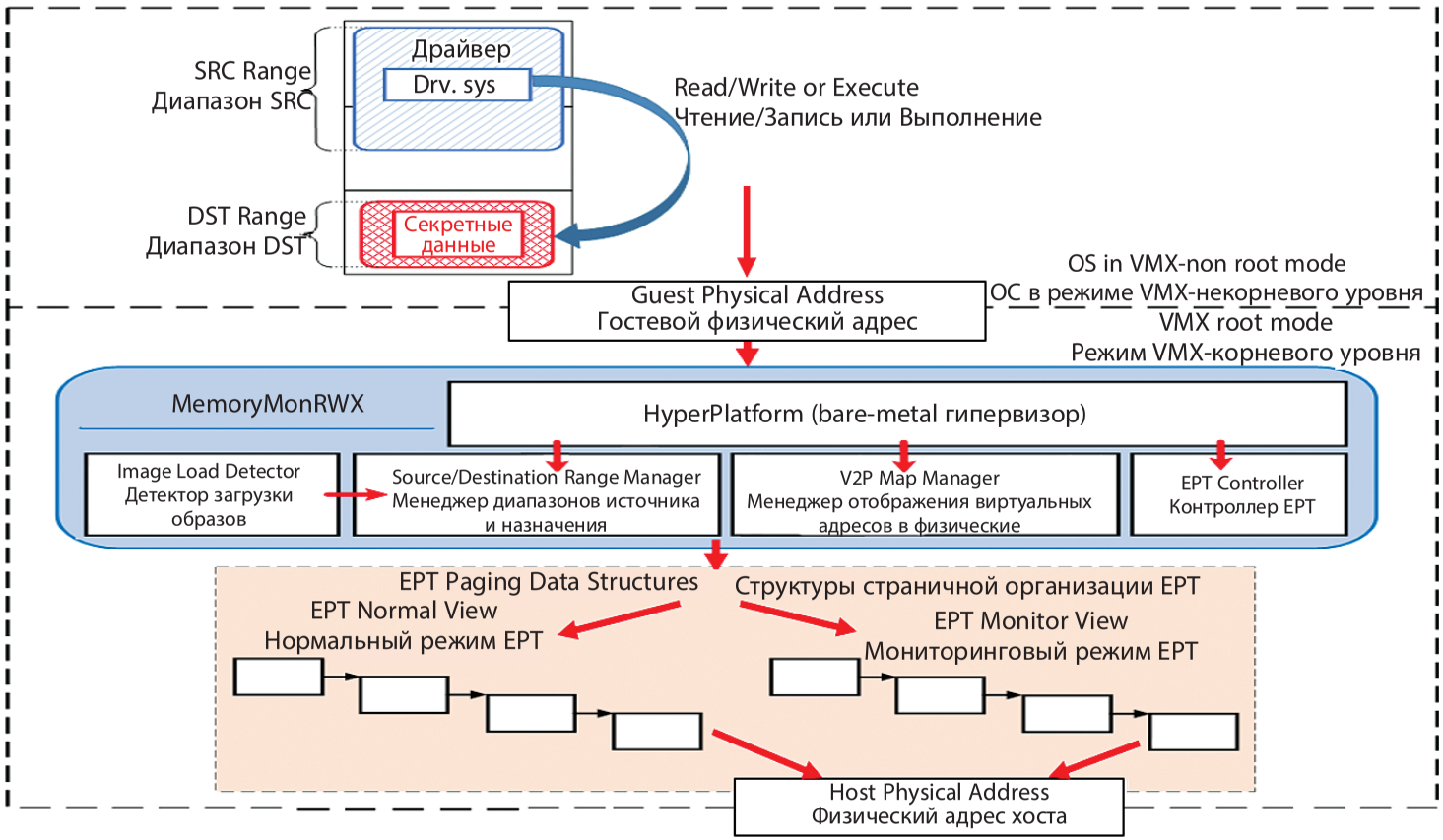


Рис. 43. Архитектура и основные компоненты MemoryMonRWX

компонентом является гипервизор HyperPlatform — монитор виртуальных машин (VMM) (bare-metal hypervisor), который был впервые представлен на конференции REcon 2016 [225]. Проект HyperPlatform предназначен для перехвата различных событий в гостевой ОС и реализует базовые функции гипервизора, без функций по мониторингу и защите содержимого памяти.

Рассмотрим работу компонентов гипервизора MemoryMon RWX.

1. Image Load Detector. Детектор загрузки драйверов формирует диапазоны памяти SRC и DST на основе виртуальных адресов гостевой ОС. SRC включает адреса заданных загруженных драйверов, а DST — области памяти, которые необходимо защищать. Диапазоны могут быть предопределены или настроены вручную путем модификации исходного кода MemoryMonRWX.

2. Source/Destination Range Manager. Менеджер диапазонов источника и назначения получает диапазоны SRC range/DST range от Image Load Detector и передает запросы в EPT Controller в целях настройки таблицы EPT таким образом, чтобы событие VM-Exit происходило при попытке драйверов из SRC обратиться к адресам из DST.

3. V2P Map Manager. Менеджер отображения виртуальных адресов в физические отвечает за отображение виртуальных адресов (VA) в физические (PA). V2P Map Manager обновляет пары VA при сбросе буфера трансляций (TLB flush) или после завершения обработки исключений #PF, связанных с доступом к отсутствующей странице.

4. EPT Controller. Контроллер EPT выполняет следующие функции:

- инициализация и обновление структур страничной организации EPT для заданных диапазонов SRC и DST;
- обработка нарушений EPT (EPT Violation) и регистрация доступа к памяти;
- взаимодействие с HyperPlatform для обработки событий VM-Exit. При возникновении VM-Exit контроллер проверяет, был ли разрешен доступ к памяти в диапазоне DST для кода из диапазона SRC.

Система MemoryMonRWX перехватывает доступ к заранее настроенным диапазонам адресов SRC и DST с использованием двух структур страничной организации EPT: EPT normal view и EPT monitor view (см. рис. 43). Эти две структуры используются на каждом процессоре, что позволяет поддерживать многопроцессорные системы:

- Normal View применяется в обычном состоянии;
- Monitor View активируется, когда гостевая система обращается к диапазону SRC.

Детальная информация о конфигурации и взаимодействии между этими двумя режимами описана в подпараграфе 7.2.2.

Логирование событий: во время работы MemoryMonRWX сохраняет журнал в файл C:\Windows\MemoryMon.log. Пример такого журнала приведен на рис. 44, *а*.

```
[EXEC] *** VA = FFFFFA800194A468, PA = 000000007fe89468, Return = FFFFF80002AD8C1C, ReturnBase = FFFFF80002A5A000
[EXEC] *** VA = FFFFFA8003D46007, PA = 000000007db46007, Return = FFFFFA800194A4AD, ReturnBase = 0000000000000000
[EXEC] *** VA = FFFFFA8003D47580, PA = 000000007db47580, Return = FFFFFA8003D460B0, ReturnBase = 0000000000000000
[EXEC] *** VA = FFFFFA8003D4AE1C, PA = 000000007db4ae1c, Return = FFFFF80002AD7B69, ReturnBase = FFFFF80002A5A000
[EXEC] *** VA = FFFFFA8003D4856B, PA = 000000007db4856b, Return = 0000000000000004, ReturnBase = 0000000000000000
```

а

```
executed fffffa800194a468, will return to fffff80002ad8c1c nt!KiRetireDpcList+0x1bc
executed fffffa8003d46007, will return to fffffa800194a4ad
executed fffffa8003d47580, will return to fffffa8003d460b0
executed fffffa8003d4ae1c, will return to fffff80002ad7b69 nt!ExpWorkerThread+0x111
executed fffffa8002626629, will return to                4
```

б

Рис. 44. Фрагменты журнала MemoryMonRWX:
а — необработанные данные; *б* — разобранные данные с расшифровками значений

Первая строка журнала указывает, что виртуальный адрес FFFFFA800194A468 был выполнен, а его возможный адрес возврата — FFFFF80002AD8C1C. При передаче управления с помощью инструкций, отличных от CALL (например, через JMP, PUSH/RET и многие другие подобные приемы), значение, читаемое в качестве «адреса возврата» из стека, может оказаться неверным или вовсе не соответствовать ожидаемому адресу возврата. Например, в последней строке журнала адрес возврата указан как 0000000000000004, что неверно.

Адрес возврата рассчитывается следующим образом:

$$*\text{ReturnAddr} = \text{RSP},$$

где RSP — содержимое регистра указателя стека в момент возникновения нарушения EPT Violation. Содержимое стека вызовов функций в этом случае определить сложно, поэтому для более точного анализа можно использовать механизм Intel Processor Trace (PT).

Для подстановки отладочных символов в журнал был разработан парсер [226]. Пример результата его работы приведен на рис. 44, *б*.

Разработанная система MemoryMonRWX обладает рядом конкурентных преимуществ:

1) контроль доступа на уровне байтов памяти. Система фиксирует любые операции доступа (чтение, запись, выполнение) вплоть до одного байта в памяти. Это достигается благодаря использованию технологии EPT, которая работает на уровне страниц, и последующей обработке, которая выявляет доступ к конкретным байтам;

2) поддержка многопроцессорных систем. MemoryMonRWX активирует режим VMX на каждом ядре процессора, что обеспечивает совместимость с многоядерными архитектурами;

3) совместимость с последними версиями Windows. Система успешно работает на Windows 10 и 11;

4) компактность кода и гибкость архитектуры. Memory MonRWX имеет прозрачную архитектуру и обладает возможностями расширения. Как показано на рис. 45, кодовая база системы составляет менее 12 тыс. строк, что составляет меньше 3% от объема кода гипервизора Xen.

Базовая платформа MemoryMonRWX может быть адаптирована для различных задач в области кибербезопасности. Например, DDIMon — мониторинг активности интерфейсов драйверов устройств [227], EoPMon — обнаружение несанкционированного повышения привилегий [228], GuardMon — исследование безопасности механизма PatchGuard [229].

Таким образом, система MemoryMonRWX предоставляет мощные инструменты для мониторинга и защиты памяти, что делает ее востребованной платформой для разработки решений в области информационной безопасности.

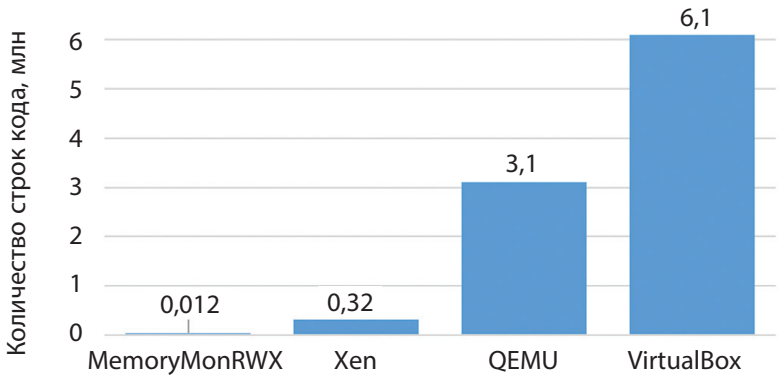


Рис. 45. Сравнение по количеству строк кода проектов гипервизоров

7.3. ПРАКТИЧЕСКОЕ ПРИМЕНЕНИЕ MEMORYMONRWX ДЛЯ ОБНАРУЖЕНИЯ И АНАЛИЗА РУТКИТОВ

Руткиты, работающие в режиме ядра, представляют серьезную угрозу безопасности, так как они могут скрывать свое присутствие и вмешиваться в работу системы. MemoryMonRWX позволяет обнаруживать такие руткиты путем:

- логирования обращений к глобальным переменным ядра — это позволяет выявлять подозрительные действия драйверов, которые могут указывать на наличие вредоносной активности;
- анализа поведения драйверов — собранные логи позволяют детально изучить последовательность действий драйверов и выявить аномалии;
- блокировки несанкционированного доступа — MemoryMonRWX может предотвращать выполнение подозрительных операций.

Рассмотрим три сценария применения системы MemoryMonRWX. В первом примере MemoryMonRWX останавливает активность эксплойта режима ядра с повышением привилегий, обнаруживая запись и вызывая BSOD. Второй пример демонстрирует способность MemoryMonRWX защищать от чтения, предотвращая отключение PatchGuard. В последнем примере изучается применение MemoryMonRWX для обнаружения выполнения подозрительного кода на примере руткита Turla.

7.3.1. Обеспечение целостности памяти. Предотвращение модификации кода и данных

Руткиты режима ядра могут использовать ряд техник, в том числе перехват функций и модификация структур в памяти с помощью техники DKOM. Схема таких атак представлена на рис. 46. Эти атаки также известны как Semantic Value Manipulation (SVM), или манипуляции семантическими значениями. Они вводят в заблуждение средства защиты путем изменения значений в структурах данных ядра. Аналогичные методы атак описаны в работе [191].

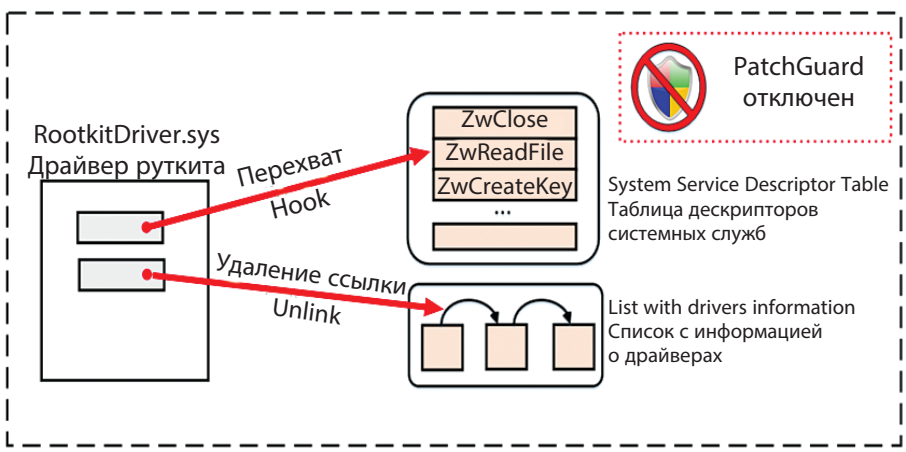


Рис. 46. Атаки на модификацию памяти режима ядра

Система MemoryMonRWX способна обнаруживать и предотвращать такие атаки. В качестве примера можно рассмотреть эксплуатацию уязвимости CVE-2014-0816, которая использует эксплойт режима ядра [230]. Данный эксплойт изменяет значение в таблице HalDispatchTable. Для защиты от этого MemoryMonRWX предварительно настраивает диапазоны и задает адрес HalDispatchTable в качестве целевого (DST). После загрузки эксплойта MemoryMonRWX фиксирует попытку модификации и блокирует ее, что позволяет остановить работу гостевой ОС и предотвратить дальнейшую эксплуатацию уязвимости.

MemoryMonRWX может использоваться для обеспечения целостности секций режима ядра, системных списков и других на-

строенных диапазонов памяти. Это позволяет защищать как системные данные, так и данные проприетарных программ. Таким образом, MemoryMonRWX выступает в качестве перспективного аналога для встроенного средства защиты HyperGuard, существенно расширяющего его возможности.

7.3.2. Обеспечение конфиденциальности памяти. Предотвращение чтения данных

Память ядра может содержать значительное количество конфиденциальной информации: журналы нажатий клавиш, пароли пользователя, хэш-значения аутентификационных данных, закрытые криптографические ключи и фрагменты расшифрованного трафика [231]. Различные атаки нарушителей направлены на компрометацию памяти на уровне ядра. Схема этих атак приведена на рис. 47.

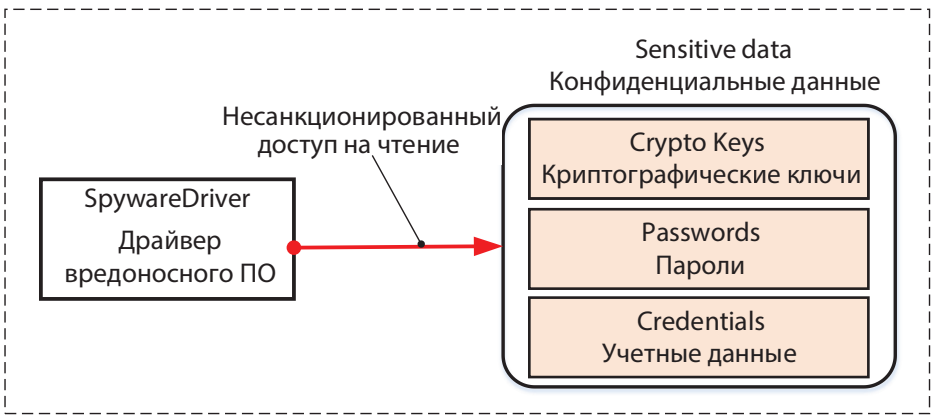


Рис. 47. Программное обеспечение крадет конфиденциальные данные

Описание атак раскрытия криптографических ключей в Open SSH, сервере Nginx и CryptoLoop рассмотрено в работе [232]. Кейлоггер на основе памяти, который перехватывает нажатия клавиш, читая содержимое DEVICE_EXTENSION драйвера kbdhid.sys, был предложен в работе [233].

Для демонстрации способности MemoryMonRWX предотвращать несанкционированный доступ к конфиденциальным данным в памяти был использован эксплойт режима ядра под названием Disarms PatchGuard (DisPG), предложенный в ра-

боте [234]. Один из компонентов DisPG считывает значение из переменной nt!PoolBigPageTableSize, которая содержит адрес системной таблицы страниц [132].

Чтобы предотвратить такие несанкционированные операции чтения, в MemoryMonRWX предварительно был настроен диапазон DST с использованием адреса переменной nt!PoolBigPageTableSize. Далее логика перехвата была изменена так, что любая попытка несанкционированного чтения этой области памяти перенаправляется на поддельную нулевую страницу. В результате эксплойт DisPG считывает замененное фиктивное значение, состоящее из нулей, и не может отключить PatchGuard. Таким образом, работа эксплойта не приводит к отключению PatchGuard и ОС остается в безопасном состоянии.

Система MemoryMonRWX предотвращает несанкционированный доступ к конфиденциальной информации, используя переопределение доступа и подмену содержимого памяти. Этот механизм показывает свою работоспособность не только в защите отдельных данных в памяти, но и в предотвращении атак, направленных на отключение штатных системных средств защиты, таких как PatchGuard.

7.3.3. Применение MemoryMonRWX для анализа руткита Turla

Одной из техник, применяемых руткитами, является перемещение вредоносного исполняемого кода за пределы секции драйвера. В результате этого в памяти ядра появляются неизвестные страницы, содержащие исполняемый код. Такая техника затрудняет обнаружение ВПО известными средствами защиты.

Система MemoryMonRWX обладает возможностью выявления такого исполняемого кода. Более того, она предоставляет средства для его анализа с использованием дизассемблера. Для демонстрации этих возможностей был использован руткит Turla (также известный как Uroburos).

Тестирование MemoryMonRWX проводилось на 64-битной версии Windows 7 с установленным руткитом Turla. В ходе эксперимента было подтверждено, что MemoryMonRWX способна обнаруживать выполнение кода из неперемещаемого пула (non-

paged pool). Также было зафиксировано, что область кода содержала распакованный код руткита [235].

Таким образом, использование MemoryMonRWX позволяет выявлять и анализировать вредоносный код, который перемещается за пределы стандартных секций драйверов. Это делает систему важным инструментом для обеспечения безопасности и анализа угроз в среде операционных систем.

7.3.4. Оценка производительности

Измерение производительности проводилось на 64-разрядной версии ОС Windows 10, работающей на Macbook Air с процессором Intel Core i7-4650U, 8 Гб оперативной памяти. В этом эксперименте были выполнены тесты Novabench и PCMark8 Home для двух случаев с включенным MemoryMonRWX и без него. Результаты показали (табл. 15), что снижение производительности системы MemoryMonRWX не превышало 10% во всех тестах, кроме графических тестов Novabench Graphics Tests. Можно предположить, что причиной более значительного падения производительности в этих тестах является частая очистка TLB, вызванная активным доступом к памяти. Тем не менее данный аспект требует более детального изучения и анализа.

В повседневной работе (открытие/сохранение документов, веб-серфинг) пользователи вряд ли заметят влияние. Анализ подтверждает, что MemoryMonRWX защищает память при минимальных накладных расходах.

Таблица 15

Накладные расходы системы MemoryMonRWX

	PCMark8 Home	Novabench RAM Speed	Novabench CPU Tests	Novabench Graphics Tests	Novabench Drive Write Speed
Накладные расходы, %	5	5	2	18	7

В рамках данного раздела представлен гипервизор MemoryMonRWX, обеспечивающий логирование и контроль операций чтения, записи и исполнения в памяти в режиме реального вре-

мени. Система обеспечивает целостность и конфиденциальность кода и данных, а также выявляет скрытое ПО, работающее в режиме ядра. Гипервизор MemoryMonRWX предоставляет широкие возможности: контроль активности драйверов, динамический анализ и обнаружение вредоносного ПО, инструмент обратной разработки, а также проверку и защиту штатных средств самозащиты, таких как PatchGuard, от попыток отключения. Практическая применимость системы подтверждена результатами тестирования, которые показывают незначительное снижение производительности. Компактность и открытость проекта позволяют использовать MemoryMonRWX в исследовательской и практической деятельности. Дальнейшее развитие этих исследований может быть направлено на улучшение производительности и функциональности системы, а также на разработку дополнительных механизмов защиты и мониторинга памяти.

Глава 8
ПРОЕКТ ALLMEMPRO: ПРИМЕНЕНИЕ ГИПЕРВИЗОРА ДЛЯ ЗАЩИТЫ КОНФИДЕНЦИАЛЬНОСТИ И ЦЕЛОСТНОСТИ ДИНАМИЧЕСКИ ВЫДЕЛЯЕМОЙ ПАМЯТИ В ЯДРЕ ОТ НЕСАНКЦИОНИРОВАННОГО ДОСТУПА

8.1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

8.1.1. Угрозы для выделенной памяти в ОС Windows

Защита данных в операционной системе становится крайне важной задачей, особенно с учетом роста интеграции Windows в промышленные системы управления. В ядре Windows динамически выделяемая память используется для хранения данных системных драйверов, а также конфиденциальных пользовательских данных и данных, обрабатываемых сторонними драйверами. Эти данные уязвимы для атак, направленных на их чтение или изменение. Нарушители могут модифицировать и подделать эти данные, установив собственный драйвер ВПО или используя уязвимости установленных модулей режима ядра [236].

Например, уязвимости присутствуют в таких драйверах, как NVIDIA Windows GPU Display Driver [237], драйвер аудио [238], драйвер клавиатуры [239] и драйвер устройства Schneider Electric UnitelWay [240]. Другой пример: уязвимость CVE-2017-0155 в компоненте Win32k позволяет нарушителю выполнить произвольный код с привилегиями уровня ядра [241].

Одной из наиболее опасных уязвимостей является уязвимость драйвера VirtualBox (VBoxDrv.sys), которая использовалась руткитом Turla. Эта уязвимость позволяет записывать произвольные значения в любую область памяти ядра [242]. Кроме того, уязвимость драйвера CPU-Z была использована в проекте HandleMaster для изменения прав доступа к дескрипторам [243].

Исследователи международной компании *ESET* в своей работе «Windows exploitation in 2016» подчеркивают, что уязвимости

сторонних драйверов становятся реальным вектором эксплуатации, предоставляя нарушителям доступ к данным и функциям системы [244].

Вредоносный код, работающий с привилегиями уровня ядра, обладает теми же правами, что и ядро Windows. Как показано на рис. 48, вредоносный код ядра может получать доступ к памяти, выделенной легитимными драйверами, с помощью уязвимостей. Такие атаки могут привести к повышению привилегий процесса, краже конфиденциальной информации и даже к нарушению работы промышленных систем.

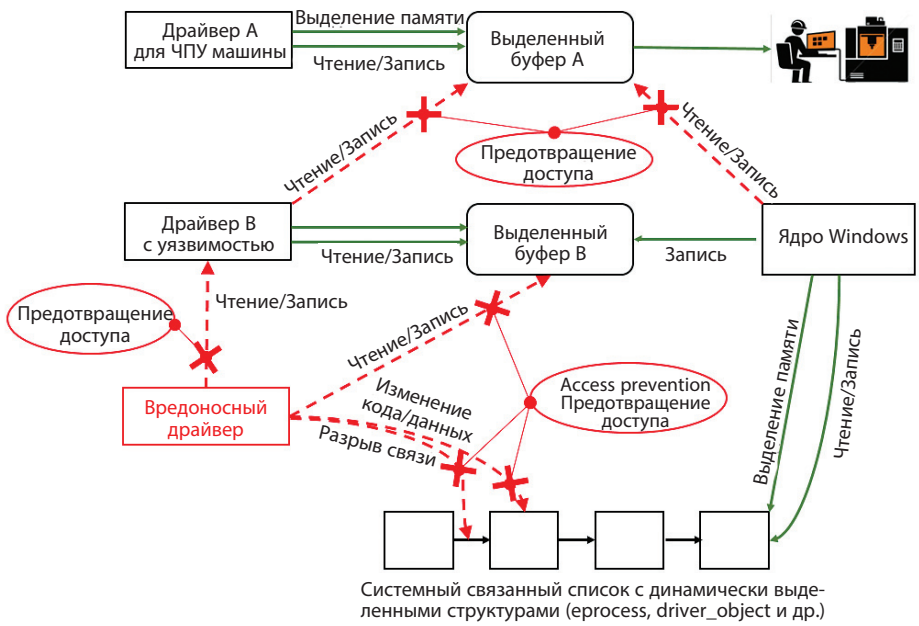


Рис. 48. Примеры попыток доступа драйвера к выделенной памяти: зеленые прямые стрелки — легитимные попытки доступа; красные пунктирные стрелки — несанкционированные попытки доступа

Уязвимости драйверов и их использование нарушителями создают серьезную угрозу целостности и конфиденциальности данных в ОС Windows, что требует внедрения новых средств защиты, которые будут отвечать вызовам времени.

На рис. 48 представлена схема взаимодействия различных компонентов системы, демонстрирующая угрозы для выделенной памяти в ОС Windows. Зеленые стрелки указывают на легитимные

запросы чтения и записи памяти, выполняемые драйверами. Красные пунктирные стрелки демонстрируют попытки несанкционированного доступа.

Рассматриваются три основные области:

1) память, выделенная ядром Windows, содержащая важные данные;

2) сторонние драйверы — легитимные модули, которым предоставлен доступ к памяти;

3) система управления ЧПУ — пример промышленного оборудования, управляемого через выделенные драйверы, данные которых также могут быть скомпрометированы.

8.1.2. Современные кибератаки: анализ, примеры и стратегии защиты для ядра и промышленного программного обеспечения

Угрозы для данных ядра и их защита. Нарушители модифицируют системные данные, выделенные в памяти ядра Windows, чтобы предотвратить обнаружение установленных вредоносных драйверов и повысить привилегии процессов. Информация о загруженных драйверах хранится в нескольких системных списках, которые состоят из связанных между собой динамически выделенных структур. Хакеры могут удалить структуру вредоносного драйвера из этих списков, что делает его невидимым для системного мониторинга. Дополнительно байты данных этих структур в памяти могут быть преднамеренно изменены, чтобы скрыть следы вредоносной активности. Эти техники руткитов известны как DKOM-unlinking и DKOM-patching и были приведены в главе 3. Кроме того, руткиты могут считывать недокументированные значения в режиме ядра. Например, программный прототип DisPG отключает защиту ядра PatchGuard, используя недокументированное значение nt!PoolBigPageTableSize [245]. Контроль доступа к этому значению в памяти может противодействовать работе DisPG по отключению PatchGuard.

Обычно процессы ВПО выполняются с низкими привилегиями, но получение высоких привилегий может позволить ВПО выполнять больше операций. Привилегии процесса могут быть повышены путем использования ошибок в драйверах режима

ядра с помощью модификации списка: Token Stealing SID List Patching, Privileges Patching и Token Stealing [246]. Примером является фреймворк mimikatz, который загружает собственный драйвер и изменяет значение Token в структуре EPROCESS [247].

Эксплойт CVE-2016-7255 использует уязвимость в компоненте win32k.sys для повышения привилегий, модифицируя структуру EPROCESS [248]. Подобные атаки на повышение привилегий (EOP) были использованы в эксплойте Duqu 2.0 [249]. Еще один эксплойт модифицирует структуры соединений Server Message Block (SMB) для получения сессий с правами администратора или системного аккаунта [250].

Нарушители могут атаковать и другие объекты Windows. Например, TDL bootkit скрывает свое присутствие, модифицируя поле StartIo целевого драйвера устройства и удаляя устройство из связанного списка DRIVER_OBJECT [251]. Аналогично Win32/Gapz и ProxyBox используют перехват обработчиков IRP из массива DRIVER_OBJECT.MajorFunction, чтобы защитить себя от удаления [252].

Подробный обзор примеров угроз для данных ядра и способов защиты приведен в работе [253].

Угрозы для пользовательских данных и необходимость их защиты. Нарушители также атакуют данные пользователя, хранящиеся в памяти ядра, такие как приватные ключи, используемые в криптографических драйверах. Кроме того, вредоносное ПО может получить доступ к данным телеметрии Windows, что ставит под угрозу приватность пользователей.

Угрозы для промышленного ПО автоматизации. Вредоносное ПО может нанести значительный ущерб, атакуя промышленное ПО управления. Примером является Stuxnet, который уничтожил около 1000 центрифуг, воздействуя на драйверы Windows, используемые в системах промышленного управления Siemens PLC [254].

Аналогичные атаки возможны на программное обеспечение для станков с числовым программным управлением (ЧПУ). ЧПУ-станки широко используются в передовом производстве, например на предприятиях NASA, Boeing, SpaceX [255]. Исследования показывают, что кибератаки на ЧПУ-станки и системы

удаленного администрирования таких станков представляют собой значительную угрозу [256].

В отчете немецкой ИТ-ассоциации говорится, что более половины компаний в Германии «стали жертвами промышленного шпионажа, саботажа или кражи данных за последние два года», и эти «компании понесли убытки в размере около 55 млрд евро в год», что составляет около 64 млрд долл. США [257].

Другой отчет, опубликованный Trend Micro Threat Research и Политехническим университетом Милана, демонстрирует реализацию кибератак на реальных промышленных роботах [258].

Подробный обзор примеров угроз для промышленного ПО автоматизации и способов защиты приведен в работе [253].

Защитное ПО должно предотвращать нелегитимный доступ к динамически выделяемым буферам в памяти ядра.

8.1.3. Модель угроз: описание и анализ

Для оценки и предотвращения действий ВПО предлагается модель угроз, которая обобщает действия вредоносных драйверов. На рис. 48 представлены основные активности ВПО:

- нарушители обходят меры защиты и устанавливают ВПО, работающее в режиме ядра;
- вредоносный драйвер находит содержимое памяти с конфиденциальными данными и кодом;
- вредоносный драйвер считывает и модифицирует данные в памяти, выделенной как ядром Windows, так и драйверами сторонних разработчиков;
- вредоносное ПО читает и изменяет секции кода модулей ядра без срабатывания средств защиты.

Операционная система Windows включает в себя несколько механизмов для предотвращения несанкционированного доступа к памяти. Однако эти механизмы обеспечивают только целостность секций кода модулей ядра и проверяют целостность отдельных полей системных связанных списков. Они не обеспечивают целостность каждой структуры целиком из этих списков и не предотвращают чтение какой-либо памяти ядра.

Существует несколько исследовательских проектов, которые частично решают проблему с защитой выделенных данных в ядре.

В работе [259] авторы предложили использование гипервизора для контроля выполнения инструкций, пытающихся записать защищенные данные ядра. Система предотвращает перезапись только отдельных системных данных ОС Windows, таких как список привилегий процессов, а также обнаруживает несанкционированное удаление структур из связанных списков. Однако система не предотвращает утечку конфиденциальной информации и кода ядра, а также не отслеживает динамическое выделение пулов памяти сторонних драйверов для их защиты.

В статье [260] авторы анализируют руткиты, которые встраивают вредоносный код в процессе установки. HACS (Hypervisor-Based Access Control Strategy) поддерживает «белый» список модулей, и только модули из этого списка имеют право модифицировать защищенные области. Однако система не предоставляет возможность применять различные политики доступа к памяти для различных модулей ядра.

Таким образом, проблема обеспечения целостности и конфиденциальности динамически выделенных данных в режиме ядра остается нерешенной. Несанкционированный доступ к этим данным может привести не только к сокрытию вредоносных драйверов и утечке персональной информации пользователей, но и к компрометации промышленных процессов, а также к значительным репутационным потерям для организаций.

В данном подпараграфе представлены разработка, реализация и оценка новой системы на основе гипервизора, которая защищает динамически выделенные данные в ядре Windows от несанкционированного доступа, а также предложены новые правила доступа к памяти для борьбы с ВПО в режиме ядра. Основные принципы включают:

- перехват каждого доступа к защищаемой памяти и предоставление полного доступа только к данным, которые ранее были выделены соответствующим драйвером;
- предотвращение несанкционированного доступа даже к одному байту памяти, выделенному другим драйвером;
- обеспечение целостности и (или) конфиденциальности выделенных данных согласно принципу минимальных привилегий;
- восстановление содержимого памяти после ее модификации.

Система защиты должна адаптироваться в реальном времени к изменениям в ОС и поддерживать следующие обновления:

- отдельные правила доступа к памяти для каждого драйвера режима ядра;
- автоматическое обновление списка защищаемых областей памяти при загрузке нового драйвера или вызове функций выделения/освобождения памяти каждым драйвером.

8.1.4. Анализ существующих методов и инструментов защиты памяти ядра

Проведем сравнение популярных исследовательских проектов по их способности обеспечивать целостность и конфиденциальность динамически выделенной памяти и кода как для ядра ОС, так и для драйверов сторонних разработчиков.

Встроенные средства защиты в Windows по защите памяти ядра. В Windows 10 можно отметить следующие штатные компоненты, созданные для защиты памяти ядра: Device Guard и PatchGuard.

Device Guard включает три основных компонента, одним из которых является Kernel Mode Code Integrity. KMCI предотвращает изменение исполняемых страниц в памяти ядра. Страницы и секции памяти не могут быть одновременно доступными для записи и выполнения, а исполняемый код не может быть изменен напрямую [261].

PatchGuard защищает целостность отдельных динамических и статических структур ядра Windows, сохраняя и периодически проверяя контрольные суммы определенных областей памяти. При обнаружении несоответствия система вызывает Blue Screen of Death (BSOD), что делает такие методы неприменимыми на производстве, например в системах ЧПУ [262, 263].

Эти встроенные средства защиты обеспечивают только целостность секций кода модулей ядра и внутренних списков с выделенными структурами, но они не поддерживают защиту выделенной памяти сторонних драйверов и не предотвращают несанкционированные модификации. BSOD, вызванный PatchGuard, неприемлем для использования в промышленности.

Гипервизорные решения. Для решения проблем были выделены две группы гипервизорных подходов, различаемых по технологии перехвата доступа к памяти [245]:

1) технология на основе генерации исключений (PageFault, #PF) при доступе к защищенной странице;

2) технология Extended Page Tables (EPT), позволяющая перехватывать операции чтения, записи и выполнения с точным контролем доступа.

Метод EPT оказывается быстрее, чем #PF, но работает только на процессорах *Intel* с микроархитектурой Nehalem или более новых, в отличие от #PF, поддерживаемого более широким спектром компьютерных систем.

Сравнительный анализ существующих гипервизорных систем. ***Sentry*** защищает динамически выделенные данные ядра путем разделения памяти на защищенные и незащищенные области [259]. Система обеспечивает целостность, но не защищает конфиденциальность данных и требует исходного кода ядра, что делает ее неприменимой для Windows. Sentry разработали с использованием Linux и гипервизора Xen. Система обладает следующими ограничениями: отсутствует возможность гибкого управления политикой доступа к памяти, что ограничивает защиту новых выделенных данных от сторонних драйверов; не обеспечивается конфиденциальность данных, а также отсутствуют механизмы контроля целостности кода в режиме ядра. Кроме того, Sentry требует исходный код ядра, что неприменимо для ОС Windows.

HACS использует EPT для защиты данных ядра, применяя «белый» список доверенных модулей [260]. «Белый» список, который составляется на основе опыта пользователя, содержит только надежные модули ядра. Однако система обеспечивает всего два уровня доверия: легитимные или нелегитимные модули. В результате HACS не может предотвратить доступ легитимного модуля к области памяти, выделенной другим легитимным модулем.

LKMG (Loadable Kernel Module Guard) относится ко второй подгруппе и использует технологию EPT. LKMG изолирует драйверы с помощью политики доступа, обеспечивая целостность

и конфиденциальность данных [264]. Система применяет общую политику безопасности для динамического доступа к данным: драйвер может обращаться только к собственным выделенным областям режима ядра. LKMG основана на гипервизоре Xen и защищает только ОС Linux. Как и другие решения, LKMG требует исходного кода.

HUKO обеспечивает целостность кода и данных ядра ОС. Система использует технологии VT-x и EPT и способна отслеживать динамическое содержимое, такое как динамические данные ядра, а также загружаемые расширения ядра [265]. HUKO рассматривает три различные категории субъектов доступа к памяти: ядро ОС, доверенные и недоверенные расширения. Однако HUKO не обеспечивает конфиденциальность и целостность данных режима ядра сторонних драйверов от вмешательства.

LKRG можно назвать аналогом PatchGuard для Linux, проверяющим хэши отдельных областей ядра. LKRG представляет собой загружаемый модуль ядра, который проводит проверку целостности ядра Linux во время выполнения. LKRG поддерживает этап ранней загрузки и «защищает систему, сравнивая хэш-значения, которые вычисляются от важных областей/секций/структур ядра, с хэш-значениями внутренней базы данных, заполняемой данными при загрузке системы и обновляемой в течение работы» [266]. Текущая версия LKRG обеспечивает целостность кода и обнаружение эксплойтов. Однако она не защищает память, выделенную сторонними драйверами.

Сравнительный анализ существующих решений представлен в табл. 16. Большинство систем требуют исходного кода драйвера для защиты данных. Средства защиты ОС Windows в большинстве случаев выявляют только факт удаления динамической структуры объекта из списка, но они не предотвращают изменение содержимого других полей этих структур. Системы HUKO и LKMG не блокируют несанкционированные попытки доступа к памяти со стороны ядра ОС, в результате чего они лишь частично защищают данные, которые были выделены сторонними драйверами.

Таблица 16

Сводная таблица проектов по защите памяти

Название, год	Режим ядра ОС		Сторонние драйверы				ОС
	Целостность		Целостность		Конфиденциальность		
	данные	код	данные	код	данные	код	
Device Guard PatchGuard Windows 10, 2017	±	+	–	+	–	–	Windows
Sentry, 2009	+	–	+	–	–	–	Linux
HUKO, 2011	+	+	±	–	–	–	Windows Linux
HACS, 2017	+	–	–	–	–	–	Linux
DADE, 2017	+	–	+	–	–	–	Linux
LKMG, 2018	+	+	±	+	+–	–	Linux
LKRG, 2018	+	+	–	+	–	–	Linux
AllMemPro, 2018	+	–	+	–	+	–	Windows

Далее будет представлена система AllMemPro, которая устраняет указанные недостатки. Текущая версия AllMemPro защищает только выделенные данные в режиме ядра. Обеспечение целостности и конфиденциальности кода будет реализовано в дальнейшем.

8.2. АРХИТЕКТУРА И РЕАЛИЗАЦИЯ ГИПЕРВИЗОРА ALLMEMPRO ДЛЯ ЗАЩИТЫ ПАМЯТИ

В данном параграфе описывается предложенная система AllMemPro на основе гипервизора, направленная на защиту динамически выделяемых данных от несанкционированного доступа и модификации [253]. Основной задачей является предотвращение атак на память, используемую модулями в режиме ядра. Для этого предлагается применять технологии расширенных таблиц страниц (EPT) в комбинации с правилами доступа к памяти.

8.2.1. Основные сценарии атак на динамически выделяемую память

Три основных сценария атак на динамически выделяемую память показаны на рис. 48 и включают:

- чтение и модификацию данных, выделенных драйверам сторонних разработчиков;
- чтение и модификацию секций кода драйверов, включая драйверы ядра Windows;
- удаление и модификацию структур данных в системных списках Windows.

Для защиты от указанных угроз предложены отдельные правила доступа к динамически выделяемой памяти, которые гарантируют целостность и конфиденциальность данных. Особенностью правил доступа к памяти является то, что они позволяют избежать несанкционированного доступа к защищенной памяти без преднамеренной генерации BSOD как встроенные системы безопасности ОС Windows [267].

Технология EPT используется для перехвата и управления доступом к памяти на уровне 4-килобайтных страниц. EPT дает возможность контролировать операции чтения, записи и выполнения, вводя правила, которые ограничивают доступ к защищаемым областям памяти.

Основные сценарии атак на динамически выделяемую память в режиме ядра и предложенные меры их предотвращения с использованием гипервизора и технологий EPT представлены в табл. 18 и подробно описаны в работе [253]. Таблица систематизирует три сценария атак, направленных на компрометацию данных в памяти, используемой драйверами и системными структурами в режиме ядра. Каждый сценарий сопровождается описанием методов атаки и предложенных мер защиты, основанных на использовании гипервизора и технологий EPT.

Сценарий 1: кража и модификация выделенных данных сторонних драйверов. Нарушители пытаются получить доступ к данным, выделенным сторонними драйверами, или модифицировать их. Для предотвращения таких атак гипервизор устанавливает правила доступа, контролируя обращения к памяти, которая была выделена этими драйверами, путем мониторинга

вызовов функций выделения и освобождения памяти. Пример правил доступа к памяти приведен в табл. 17. Это позволяет разрешать операции только легитимному драйверу, инициировавшему выделение памяти, и предотвращать несанкционированный доступ со стороны других модулей.

Таблица 17

Пример правил доступа к памяти

DriverStartAddr	Driver Size	AllocatedStartAddr	Allocated Size	Shared Access
fffff8016f670000 (mem_allocator_driver.sys)	0000B000	FFFFA400AF3C3F80	40	R = 0, W = 0
fffff80170201000 (ntoskrnl.exe)	008D2000	FFFFA400AF3C3F80	40	R = 0, W = 0

Сценарий 2: кража и модификация участков кода. Нарушители стремятся получить доступ к коду драйверов для его анализа или внедрения вредоносных инструкций. Несмотря на встроенную в Windows защиту от модификации кода (KMCI), чтение содержимого кода остается уязвимым местом. Для защиты предлагается динамическое переключение между двумя EPT-структурами: первая позволяет выполнение только защищtнного драйвера и блокирует выполнение из остальной памяти, а вторая предотвращает несанкционированный доступ к коду.

Сценарий 3: модификация системных структур данных. Данный сценарий включает атаки, такие как сокрытие процессов посредством прямой манипуляции объектами ядра (DKOM). Гипервизор устанавливает правила доступа к системным структурам, таким как EPROCESS, предотвращая их модификацию со стороны неавторизованных модулей ядра. Это достигается посредством мониторинга создания и удаления процессов и контроля попыток изменения данных, что обеспечивает целостность системных структур.

Систематизированный обзор угроз и соответствующих мер защиты, которые легли в основу архитектурных решений, реализованных в гипервизоре AllMemPro, предложен в табл. 18. Далее будет подробно описано, как AllMemPro применяет эти методы

для обеспечения безопасности динамически выделяемой памяти в режиме ядра.

Схема защиты с использованием гипервизора AllMemPro представлена на рис. 49. Она демонстрирует применение правил доступа к памяти для предотвращения несанкционированного чтения или модификации. Схема включает два уровня:

1) VMX non-root (режим с ограниченными привилегиями гостевой ОС), где работают драйверы ядра и выполняются операции с памятью;

2) VMX root (режим гипервизора), где функционирует AllMemPro.

Процесс состоит из пяти шагов.

Шаги 1 и 2: инициализация гипервизора и создание структур EPT с полным доступом к памяти. Гипервизор загружается прежде, чем анализируемый драйвер будет загружен в память. Гипервизор будет получать уведомления о загрузке образа в память с помощью PsSetLoadImageNotifyRoutine и выбирать, какой драйвер режима ядра будет защищен, используя заданный список имен и масок. Далее гипервизор перехватывает каждый вызов выделения памяти и выбирает только те функции, которые были вызваны из кода, принадлежащего защищенному драйверу. Здесь рассматривается использование функции ExAllocatePoolWithTag для выделения пула памяти (аллокации) и ExFreePoolWithTag для освобождения выделенной памяти (деаллокции). Когда защищенный драйвер вызывает процедуру выделения/освобождения, соответствующие правила добавляются/удаляются в реальном времени (см. пример правил в табл. 17). Применяя технологию EPT, гипервизор может перехватывать, обрабатывать и контролировать каждый доступ к памяти [245].

После добавления нового правила доступа к памяти гипервизор обновляет содержимое таблиц EPT соответствующим образом для отражения добавленного правила. Перед освобождением памяти драйвером гипервизор дважды заполняет блок памяти нулями и удаляет соответствующее правило доступа, восстанавливая разрешения в EPT. В результате любые попытки чтения или записи в защищенную память вызывают нарушение EPT.

Шаг 3: при легитимном доступе гипервизор временно разрешает чтение и запись, используя оригинальный номер физи-

ческой страницы (PFN), и предоставляет доступ только защищенному модулю, выделившему эту память (см. пример правил в табл. 17). Для этого гипервизор устанавливает разрешение на запись/чтение и выставляет флаг MTF, чтобы генерировать VM Exit после выполнения каждой инструкции кода. После завершения обращения управление возвращается гипервизору, который восстанавливает разрешения страницы и очищает MTF.

Шаг 4: при несанкционированном доступе гипервизор изменяет номер кадра страницы (PFN) в соответствующем элементе таблицы страниц (PTE) для защищенной памяти и временно предоставляет доступ к замененной памяти, устанавливая разрешение на чтение и (или) запись. Гипервизор также устанавливает MTF. После того как неавторизованный модуль прочитает или запишет память замененной страницы, управление переходит к гипервизору, поскольку происходит выход из VM Exit. Далее гипервизор восстанавливает начальную конфигурацию: устанавливает исходное значение PFN для защищенной памяти, очищает биты доступа и сбрасывает флаг MTF.

Шаг 5: после освобождения памяти гипервизор очищает данные и восстанавливает настройки EPT.

Схема (см. рис. 49) подчеркивает динамический характер системы, а именно: гипервизор не только обеспечивает защиту выделенной памяти в режиме реального времени, но и оперативно реагирует на изменения, адаптируясь для защиты новых выделенных и освобожденных буферов памяти.

Таблица 18

Сценарии атак на динамически выделяемую память и методы их предотвращения

Сценарий	Детали атаки	Предложенные меры защиты
Сценарий 1: чтение или изменение неисполнимых областей памяти, выделенных сторонними драйверами [267]	ВПО пытается получить доступ к выделенной памяти стороннего драйвера для чтения или модификации	Использование правил доступа к памяти, контролируемых гипервизором с применением технологии EPT; перехват вызовов выделения и освобождения памяти; контроль доступа через перехват EPT-нарушений; разрешение или запрет доступа в соответствии с правилами

Окончание табл. 18

Сценарий	Детали атаки	Предложенные меры защиты
Сценарий 2: чтение и изменение содержимого фрагментов кода [245]	Чтение секций кода (модификация предотвращается компонентом KMCI в Windows 10, но конфиденциальность кода остается уязвимой)	Предложение использования двух EPT-структур: первая позволяет выполнение защищенного драйвера и блокирует выполнение из остальной памяти; вторая блокирует доступ к защищенным данным и позволяет выполнение из всей памяти, кроме защищенного драйвера
Сценарий 3: изменение содержимого системных структур данных и глобальных переменных в памяти [268, 269]	Применение техник DKOM unlinking и DKOM patching для сокрытия объектов и отключение средств защиты AV, EDR, в том числе путем воздействия на ETW, WMI, Event Logging	Добавление правил доступа к памяти для каждой структуры; защита системных структур и глобальных флагов путем добавления правил доступа к памяти

8.2.2. Архитектура системы AllMemPro

Гипервизорная система AllMemPro включает три основных компонента: контроллер, коммутатор и диспетчер.

Контроллер получает нотификацию о загрузке драйверов, о выделении и освобождении буферов памяти. Для отслеживания загрузки каждого драйвера контроллер использует функцию PsSetLoadImageNotifyRoutine, с помощью которой происходит регистрация callback-функции для получения уведомления о загрузке нового драйвера. Соответствующая callback-функция извлекает три основных значения: полное имя загруженного файла драйвера, адрес загрузки драйвера (ImageBase) и размер образа драйвера в памяти (ImageSize) [270].

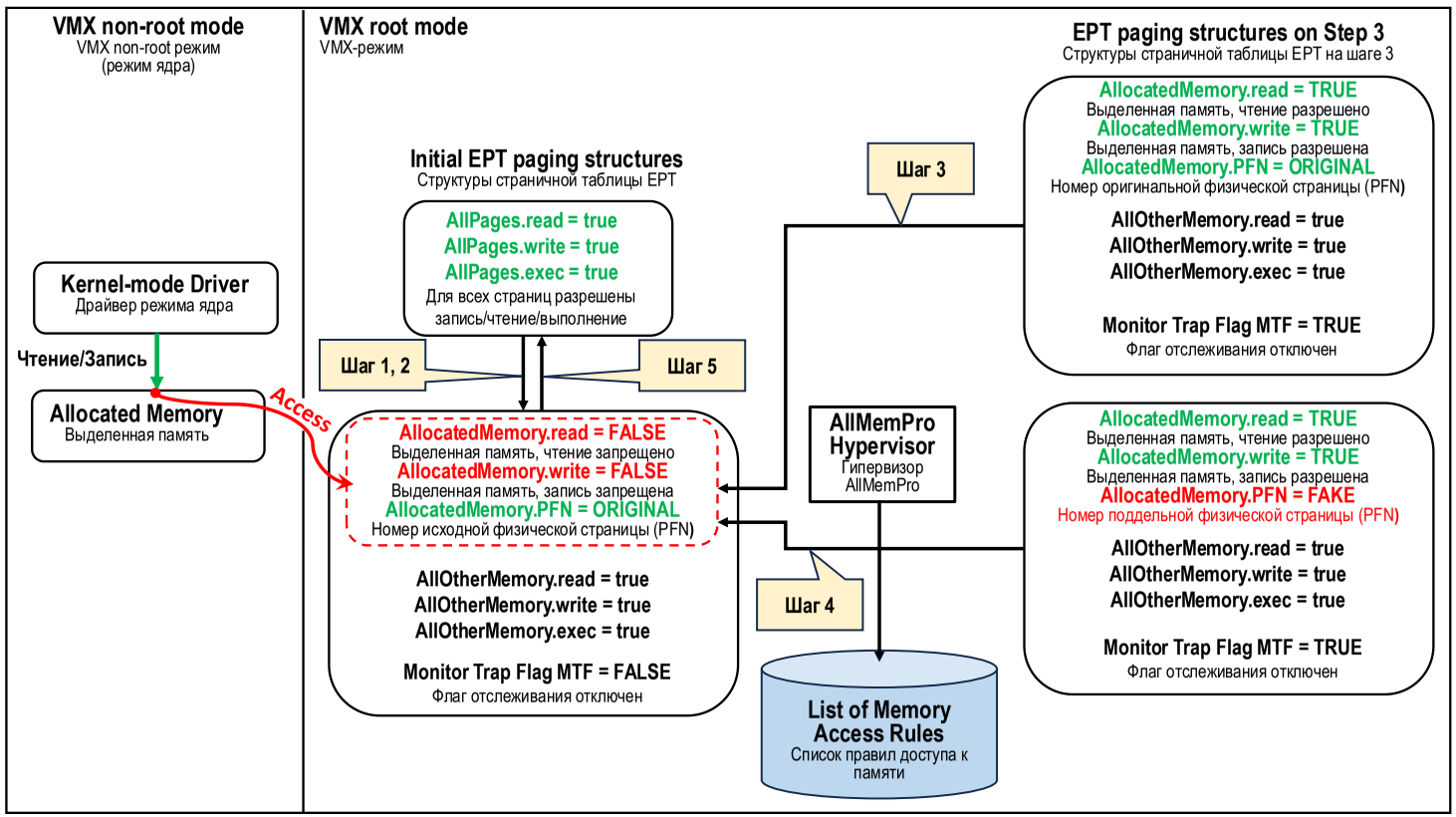


Рис. 49. Схема реализации предложенных мер защиты с использованием гипервизора AllMemPro

Контроллер выбирает, какой драйвер должен быть защищен, по его имени, но также возможно выбрать защищенный драйвер по наличию цифровой подписи. Контроллер перехватывает функцию выделения памяти (аллокации) ExAllocatePoolWithTag и функцию освобождения памяти (деаллокации) ExFreePoolWithTag с помощью DdiMon, описанного в работе [227]. DdiMon отслеживает и контролирует вызовы API ядра с помощью техники скрытых хуков, используя технологию EPT для обхода PatchGuard.

Контроллер получает сообщение о том, что защищенный драйвер выделяет память, и автоматически создает следующее правило доступа к памяти и отправляет его Коммутатору и Диспетчеру (рис. 50).

```
typedef struct _MEMORY_ACCESS_RULE {
        void*                   drvAddr;
        unsigned __int64        drvSize;
        void*                   allocAddr;
        unsigned __int64        allocSize;
        int                     is_readable;
        int                     is_overwritable;
}MEMORY_ACCESS_RULE, *PMEMORY_ACCESS_RULE;
```

Рис. 50. Структура для хранения правила доступа к памяти

Коммутатор контролирует доступ к данным защищенной памяти, используя средства гипервизора и EPT. Код коммутатора основан на гипервизоре MemoryMonRWX [245].

Сначала коммутатор создает структуру EPT для управления страницами памяти, используемыми в режиме ядра, и устанавливает для них начальные права доступа: чтение, запись и выполнение разрешены по умолчанию. Затем, получив от контроллера конкретные правила доступа к защищаемым данным, он обновляет настройки и запрещает чтение и запись для соответствующих страниц. В итоге любая попытка обращения к защищенной памяти приводит к возникновению EPT-исключения.

Коммутатор обрабатывает все исключения EPT и выбирает один из двух сценариев: разрешить или запретить доступ к исходным данным, опираясь на решение диспетчера, который обрабатывает правила доступа к памяти.

Сценарий разрешения доступа. Коммутатор временно разрешает доступ к защищенным данным и активирует флаг мониторинга (MTF), как показано на шаге 4 (см. рис. 50). После выполнения одной инструкции управление возвращается коммутатору, который восстанавливает ограничения: сбрасывает биты чтения и записи в структуре EPT и отключает MTF, ожидая повторного EPT-исключения.

Сценарий запрета доступа. Если доступ несанкционирован, то коммутатор перенаправляет запрос на поддельную страницу, изменяя номер физической страницы (PFN) в структуре EPT, как это показано на шаге 4 (см. рис. 50). При этом разрешается временный доступ и активируется MTF. После того как неизвестный код обращается к поддельным данным и выполняет одну инструкцию, управление снова перехватывается. Коммутатор восстанавливает исходные настройки EPT (см. рис. 49, шаги 1 и 2).

Выбор сценария осуществляется на основании решений, принимаемых диспетчером.

Диспетчер реализует механизм дискреционного контроля доступа (Discretionary Access Control), который предотвращает несанкционированный доступ даже к одному байту защищенных данных в соответствии со списком правил доступа к памяти.

Диспетчер обеспечивает полные привилегии владельцу выделенной памяти. Если незарегистрированный (или неизвестный) код пытается получить доступ к защищенным данным, то система применяет следующие правила:

- чтение данных: если параметр 'is_readable==0', то чтение данных запрещено. В противном случае код может прочитать данные;
- запись данных: если параметр 'is_overwritable==0', то запись в память запрещена. В противном случае запись разрешена.

Чтобы предоставить доступ другому драйверу или ядру ОС Windows к чтению или записи защищенных данных, необходимо добавить соответствующее правило доступа в список.

Система AllMemPro была разработана с использованием Microsoft Visual C++ и пакета Windows Driver Kit (WDK). Она тестировалась в виртуальной среде VMware Workstation на ОС

Windows 10 (версия 1709, сборка 16299.15, 64-разрядная) с использованием многоядерного процессора. Исходный код системы доступен в работе [271].

Преимущества предлагаемой гипервизорной системы AllMemPro:

- защита динамически выделяемой памяти: система может защищать память, выделенную в режиме реального времени, с помощью компонента Контроллер;
- предотвращение несанкционированного доступа: система блокирует чтение и запись даже одного байта защищаемых данных;
- работа без исходного кода: система может защищать память без исходного кода защищаемого драйвера.

8.3. ПРАКТИЧЕСКОЕ ПРИМЕНЕНИЕ ГИПЕРВИЗОРА ALLMEMPRO: ЗАЩИТА ВЫДЕЛЕННОЙ ПАМЯТИ ДРАЙВЕРА WINDOWS ОТ НЕСАНКЦИОНИРОВАННОГО ДОСТУПА

Продемонстрируем использование гипервизора AllMemPro для защиты памяти, выделенной драйверам в режиме ядра. Рассматривается три сценария, в которых AllMemPro изолирует динамически выделяемую память одного драйвера от несанкционированного доступа со стороны других модулей.

8.3.1. Первый сценарий — демонстрация несанкционированного доступа к выделенной памяти без защиты гипервизора

Сначала загружается драйвер режима ядра mem_allocator_driver.sys, который выделяет буфер в памяти. Этот драйвер считывает данные из выделенной памяти и обновляет ее содержимое после получения IOCTL-кода от консольного приложения (mem_allocator_console.exe). Это дает возможность пользователю с помощью консольного приложения получать актуальное содержимое выделенной памяти в ядре и изменять его.

Затем загружается второй драйвер — mem_attacker_driver.sys, играющий роль вредоносного модуля. Этот драйвер читает и записывает данные, выделенные первым драйвером. Счита-

ется, что вредоносному драйверу известен адрес памяти, которая была выделена легитимным драйвером mem_allocator_driver.sys. Управление вторым драйвером осуществляется через отдельное консольное приложение.

На рис. 51 представлена схема взаимодействия загруженных драйверов, а в табл. 19 приведены адреса драйверов и размеры области памяти, выделенной драйвером mem_allocator_driver.sys.

Этот пример подтверждает, что выделенная память, может быть, несанкционированно изменена без срабатывания штатных средств защиты ядра типа KPP (PatchGuard). Для предотвращения таких манипуляций необходимы сторонние средства защиты.

Таблица 19

Сведения об объектах в памяти для рис. 51

Объект в памяти	Начальный адрес	Размер
mem_allocator_driver.sys	FFFFF8016F630000	0xb000
Allocated data by mem_allocator_driver.sys	FFFFA400AC479FD0	0x10
mem_attacker_driver.sys	FFFFF8016F650000	0x9000

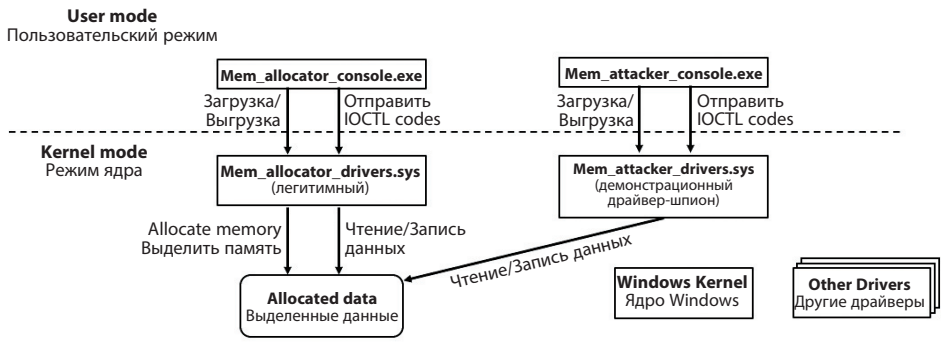


Рис. 51. Схема взаимодействия драйверов без защиты

Исходные коды драйверов mem_allocator_driver.sys и mem_attacker_driver.sys, а также консольных приложений доступны в репозитории [271].

8.3.2. Второй сценарий — гипервизор блокирует несанкционированный доступ к выделенной памяти в ядре

В начале работы выполняется загрузка гипервизора AllMemPro (рис. 52), далее загружается легитимный драйвер mem_allocator_driver.sys, который выделяет буфер в памяти. Гипервизор регистрирует загрузку легитимного драйвера, факт выделения памяти и автоматически добавляет следующее правило доступа к памяти (табл. 20). Затем осуществляется попытка чтения и записи этих данных с помощью нелегитимного драйвера mem_attacker_driver.sys.

Таблица 20

Правила доступа к выделенной памяти для mem_allocator_driver.sys и ntkrnlmp.exe

rule FFFFF8016F630000 B000 FFFFA400AC479FD0 10

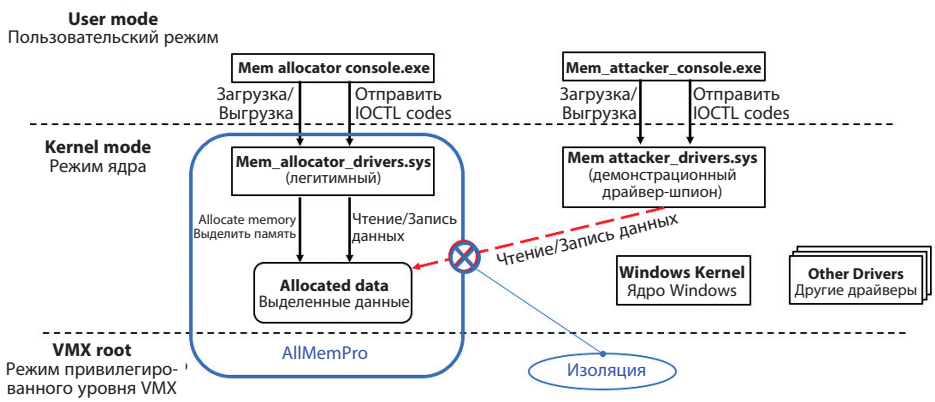


Рис. 52. AllMemPro предоставляет доступ к выделенной памяти только первому драйверу режима ядра в соответствии с правилом доступа к памяти

Информацию о расположении и размере объектов в памяти, представленных на рис. 52, содержит табл. 21. В таблице указаны начальные адреса и размеры различных модулей, включая драйверы mem_allocator_driver.sys и mem_attacker_driver.sys, выделенные ими данные, а также ядро ОС Windows (ntkrnlmp.exe).

Эти данные демонстрируют, какие области памяти используются каждым объектом и как AllMemPro изолирует доступ к ним.

Таблица 21

Сведения об объектах в памяти для рис. 52

Объект в памяти	Начальный адрес	Размер
mem_allocator_driver.sys	FFFFF8016F630000	0xb000
Allocated data by mem_allocator_driver.sys	FFFFA400AC479FD0	0x10
mem_attacker_driver.sys	FFFFF8016F650000	0x9000
nt (ntkrnlmp.exe)	FFFFF80170201000	0x8D2000

Эксперименты продемонстрировали, что все попытки доступа со стороны второго драйвера (mem_attacker_driver.sys) заканчиваются неудачей: при чтении возвращаются нулевые значения, а попытки записи не изменяют содержимое памяти. Соответствующие фрагменты отладочного вывода AllMemPro показаны на рис. 53.


```
Debug Output Fragment for legal read- access:
22:34:47.513        INF        #0          4          7732       System
S= FFFFF8016F6317C8 (FFFFF8016F630000), D= FFFFA400AC479FD8 (0000000000000000), T= R

Debug Output Fragment for legal write- access:
22:34:50.357        INF        #0          8020       8144       mem_allocator_
S= FFFFF8016F6314EA (FFFFF8016F630000), D= FFFFA400AC479FD8 (0000000000000000), T= W,
00 00 00 00 00 00 00 00 01 0a 00 00 00 00 00 00  => 00 00 00 00 00 00 00 00 ba 0a 00 00 00 00 00
00

Debug Output Fragment for illegal read-access:
illegal access FFFFF8016F651228 =>> FFFFA400AC479FD8
 ** RweHandleMonitorTrapFlag FFFFF8016F651228 FFFFA400AC479FD8 **

22:35:05.952        INF        #0          76         8060       mem_allocator_
[Protected via ActiveMemPolice] Memory is being READ. Returning fake contents.
22:35:05.952        INF        #0          76         8060       mem_allocator_
S= FFFFF8016F651228 (FFFFF8016F650000), D= FFFFA400AC479FD8 (0000000000000000), T= R

Debug Output Fragment: for illegal write-access:
illegal access FFFFF8016F651257 =>> FFFFA400AC479FD8
 ** RweHandleMonitorTrapFlag FFFFF8016F651257 FFFFA400AC479FD8 **

22:35:20.405        INF        #0          76         8060       mem_allocator_
S= FFFFF8016F651257 (FFFFF8016F650000), D= FFFFA400AC479FD8 (0000000000000000), T= W,
00 00 00 00 00 00 00 00 ba 0a 00 00 00 00 00 00  => 00 00 00 00 00 00 00 00 ba 0a 00 00 00 00 00
00

22:35:20.405        INF        #0          76         8060       mem_allocator_
[Protected via ActiveMemPolice] Memory is being WRITTEN. Returning fake contents.
```

Рис. 53. Фрагменты отладочного вывода для рис. 52

Результаты наглядно продемонстрировали, что во втором сценарии гипервизор блокирует все попытки доступа со стороны второго драйвера, а именно: при чтении возвращаются нулевые значения и все нелегитимные попытки записи не изменяют содержимое памяти. При этом гипервизор не блокирует легитимные попытки чтения и изменения выделенной памяти.

8.3.3. Третий сценарий — гипервизор обеспечивает безопасный совместный доступ к выделенной памяти со стороны драйвера и функций ядра операционной системы

Первый драйвер использует выделенную память для получения системной информации, обращаясь к ядру ОС Windows. На рис. 54 и в табл. 21 приведены примеры таких операций.

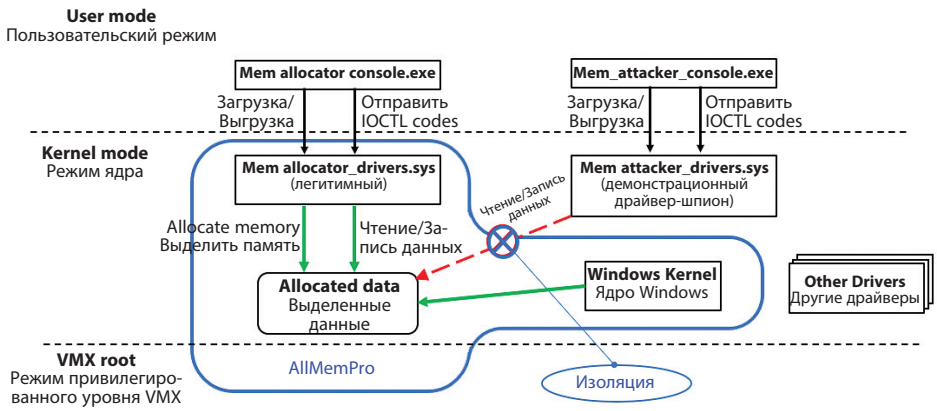


Рис. 54. Изоляция выделенной памяти с использованием AllMemPro (AllMemPro предоставляет доступ к выделенной памяти для mem_allocator_driver.sys и ядра ОС Windows. AllMemPro предотвращает доступ к mem_attacker_driver.sys, которого нет в списке правил)

Для организации совместного доступа к выделенной памяти для первого драйвера и ядра Windows ОС используются следующие два правила доступа (табл. 22):

- первое правило предоставляет доступ к памяти первому драйверу;
- второе — дает ядру Windows доступ на чтение и запись к выделенной памяти.

В результате ядро Windows успешно записывает данные в выделенный буфер, а гипервизор продолжает защищать эту память от второго драйвера. Сообщения в отладочном выводе подтверждают, что все попытки нелегального доступа блокируются гипервизором, а легитимный модуль и функции ядра ОС продолжают работать с защищенной памятью без ограничений (рис. 55).

Таблица 22

Сведения об объектах в памяти для рис. 54

Объект в памяти	Начальный адрес	Размер
mem_allocator_driver.sys	FFFFF8016F630000	0xb000
Shared Allocated data	FFFFA400AC479F80	0x40
mem_attacker_driver.sys	FFFFF8016F650000	0x9000
nt (ntkrnlmp.exe)	FFFFF80170201000	0x8D2000

Таблица 23

Правила доступа к выделенной памяти для mem_allocator_driver.sys и ntkrnlmp.exe

- // for mem_allocator_driver.sys
  rule FFFFF8016F630000 B000 FFFFA400AC479F80 40
- // for ntkrnlmp.exe
  rule FFFFF80170201000 8D2000 FFFFA400AC479F80 40

Кроме того, AllMemPro позволяет предотвращать нелегальное повышение привилегий путем модификации структуры EPROCESS. Таким образом, гипервизор обеспечивает защиту не только памяти, выделенной сторонними драйверами, но и системных структур ОС.

Следует отметить, что гипервизор AllMemPro значительно влияет на быстродействие системы в целом при частых обращениях к защищенным областям памяти.

Оценка потери производительности проводилась экспериментально путем измерения длительности 10 попыток доступа к выделенной памяти в трех случаях: без гипервизора с включенным кэшем памяти; без гипервизора и отключенным кэшем; с гипервизором AllMemPro и подменой времени. Результаты представлены в табл. 24.

Debug Output Fragment for illegal write- access

```
** RweHandleMonitorTrapFlag FFFFF8016F651228 FFFFA400AC479F80 **

22:58:18.560        INF        #0        76        8060        mem_allocator_
[Protected via ActiveMemPolice] Memory is being READ. Returning fake contents.

22:58:18.560        INF        #0        76        8060        mem_allocator_
S= FFFFF8016F651228 (FFFFF8016F650000), D= FFFFA400AC479F80 (0000000000000000), T= R
```

Debug Output Fragment for legal write-access (mem_allocator_driver.sys, memset function fragment):

```
22:51:03.306        INF        #0        4        7732        System
S= FFFFF8016F631743 (FFFFF8016F630000), D= FFFFA400AC479F8B (0000000000000000), T= W,
00 00 00 00 00 00 00 00 00 00 00 00 7d ff 0f 00  => 00 00 00 00 00 00 00 00 00 00 00 00 7d ff 0f 00

22:51:03.306        INF        #0        4        7732        System
S= FFFFF8016F631743 (FFFFF8016F630000), D= FFFFA400AC479F8C (0000000000000000), T= W,
00 00 00 00 00 00 00 00 00 00 00 00 7d ff 0f 00  => 00 00 00 00 00 00 00 00 00 00 00 00 00 ff 0f 00

22:51:03.306        INF        #0        4        7732        System
S= FFFFF8016F631743 (FFFFF8016F630000), D= FFFFA400AC479F8D (0000000000000000), T= W,
00 00 00 00 00 00 00 00 00 00 00 00 00 ff 0f 00  => 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0f 00

22:51:03.306        INF        #0        4        7732        System
S= FFFFF8016F631743 (FFFFF8016F630000), D= FFFFA400AC479F8E (0000000000000000), T= W,
00 00 00 00 00 00 00 00 00 00 00 00 00 00 0f 00  => 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

22:51:03.306        INF        #0        4        7732        System
S= FFFFF8016F631743 (FFFFF8016F630000), D= FFFFA400AC479F8F (0000000000000000), T= W,
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  => 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00
```

Debug Output Fragment for legal write-access (ntkrnlmp.exe, ZwQuerySystemInformation):

```
22:51:03.306        INF        #0        4        7732        System
S= FFFFF801702FB65B (FFFFF80170201000), D= FFFFA400AC479F84 (0000000000000000), T= W,
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  => 00 00 00 00 5a 62 02 00 00 00 00 00 00 00 00 00

22:51:03.306        INF        #0        4        7732        System
S= FFFFF801702FB65F (FFFFF80170201000), D= FFFFA400AC479F88 (0000000000000000), T= W,
00 00 00 00 5a 62 02 00 00 00 00 00 00 00 00 00  => 00 00 00 00 5a 62 02 00 00 10 00 00 00 00 00 00

22:51:03.306        INF        #0        4        7732        System
S= FFFFF801702FB737 (FFFFF80170201000), D= FFFFA400AC479F8C (0000000000000000), T= W,
00 00 00 00 5a 62 02 00 00 10 00 00 00 00 00 00  => 00 00 00 00 5a 62 02 00 00 10 00 00 7d ff 0f 00
```

Рис. 55. Фрагменты отладочного вывода для рис. 54

Проведенные измерения показывают, что гипервизор AllMemPro успешно защищает области памяти с низкой частотой доступа, такие как, например, поле EPROCESS.Token. Однако защита высокочастотных операций доступа к памяти приводит к значительным временны́м затратам из-за необходимости много-

кратного вызова гипервизора для каждого обращения и очистки буфера TLB и кэш-памяти.

Таблица 24

Оценка требуемых ресурсов для гипервизора AllMemPro

№ п/п	Эксперимент	Задержка доступа к памяти (такты TSC)
1	Без гипервизора AllMemPro с включенным кэшем памяти	70±2
2	Без гипервизора AllMemPro и отключенным кэшем	100,000±4,000
3	С гипервизором AllMemPro	500,000±10,000

Для снижения накладных расходов рекомендуется использовать несколько структур EPT, каждая из которых отвечает за отдельные диапазоны памяти. Этот подход позволяет изолировать память различных драйверов и уменьшить задержки. Подобная техника уже успешно реализована в гипервизоре MemoryMonRWX, что демонстрирует перспективность применения набора структур EPT в задачах защиты часто используемой памяти.

Глава 9
ПРОЕКТ MEMORYRANGER: ПРИМЕНЕНИЕ ГИПЕРВИЗОРА ДЛЯ ЗАПУСКА НЕСКОЛЬКИХ ДРАЙВЕРОВ В ИЗОЛИРОВАННЫХ АНКЛАВАХ ПАМЯТИ ЯДРА

Одной из задач обеспечения безопасности ОС является защита доверенного кода и обрабатываемых данных в недоверенной среде. Драйверы совместно используют единую память ядра и имеют полный доступ к структурам данных и исполнимому коду, которые могут быть скомпрометированы. Несмотря на внедрение новых средств защиты в Windows ОС, угрозы сохраняются: данные в памяти могут быть украдены или изменены без реакции со стороны средств защиты.

На протяжении более чем 20 лет ОС Windows функционирует в защищенном режиме, обеспечиваемом архитектурами процессоров x86–x64. Этот режим включает ряд механизмов безопасности, среди которых выделяется четыре уровня привилегий (кольца защиты), предназначенных для изоляции кода и данных ядра от несанкционированного доступа со стороны менее привилегированного кода. Как отмечают в 7-м издании книги Windows Internals, которое охватывает внутренние механизмы Windows и Windows Server [1], ОС Windows использует только два уровня привилегий: один для ядра ОС и драйверов (режим ядра), другой — для пользовательских приложений (пользовательский режим).

В настоящее время драйверы режима ядра используют общее адресное пространство с остальной частью ядра ОС. Это позволяет драйверам читать и записывать любые области памяти в режиме ядра без аппаратных ограничений, что, с одной стороны, способствует высокой производительности системы, а с другой — порождает новые угрозы информационной безопасности, поскольку нарушители, получив доступ к памяти ядра, могут вмешиваться в работу системы и средств защиты информации (рис. 56).

К наиболее распространенным техникам атак относятся [272]:

- установка вредоносных драйверов, подписанных цифровыми сертификатами;
- эксплуатация уязвимостей легитимных драйверов с актуальной цифровой подписью.

Используя возможности кода режима ядра, нарушители могут достичь следующих целей [273]:

- установить скрытый и долгосрочный контроль над целевой системой;
- повысить привилегии для получения несанкционированного доступа к пользовательским данным.

Эти атаки осуществляются путем несанкционированного доступа к коду драйверов ядра и манипуляции их выделенными данными. Примеры таких атак даны в работе [253].

Существующие механизмы защиты. В последние годы *Microsoft* внедрила ряд новых средств для защиты ядра ОС от подобных угроз. Наиболее известным из них является PatchGuard, который завершает работу ОС в аварийном режиме после обнаружения некоторых изменений во внутренних структурах, например при разрыве связей в EPROCESS (атака типа DKOM). Более современное средство — Device Guard — обеспечивает целостность всех загруженных драйверов путем блокировки записи в исполняемые страницы памяти. Любые изменения кода ядра приводят к остановке работы системы. Новое штатное средство защиты — HyperGuard — расширяет возможности PatchGuard и Device Guard.

Однако эти встроенные средства не обеспечивают защиту памяти, выделенной сторонними драйверами. Внутренние структуры ОС также остаются защищенными лишь частично, а код драйверов, потенциально содержащих ноу-хау-алгоритмы, может быть скопирован из памяти и подвергнут обратной разработке (реверс-инжиниринг).

Перспективные подходы к решению проблемы защиты памяти ядра. Исследователи в области кибербезопасности продолжают разрабатывать новые подходы для решения задач обеспечения безопасности. Например, концепция изоляции памяти появилась еще в 1965 г., когда была предложена система Multics

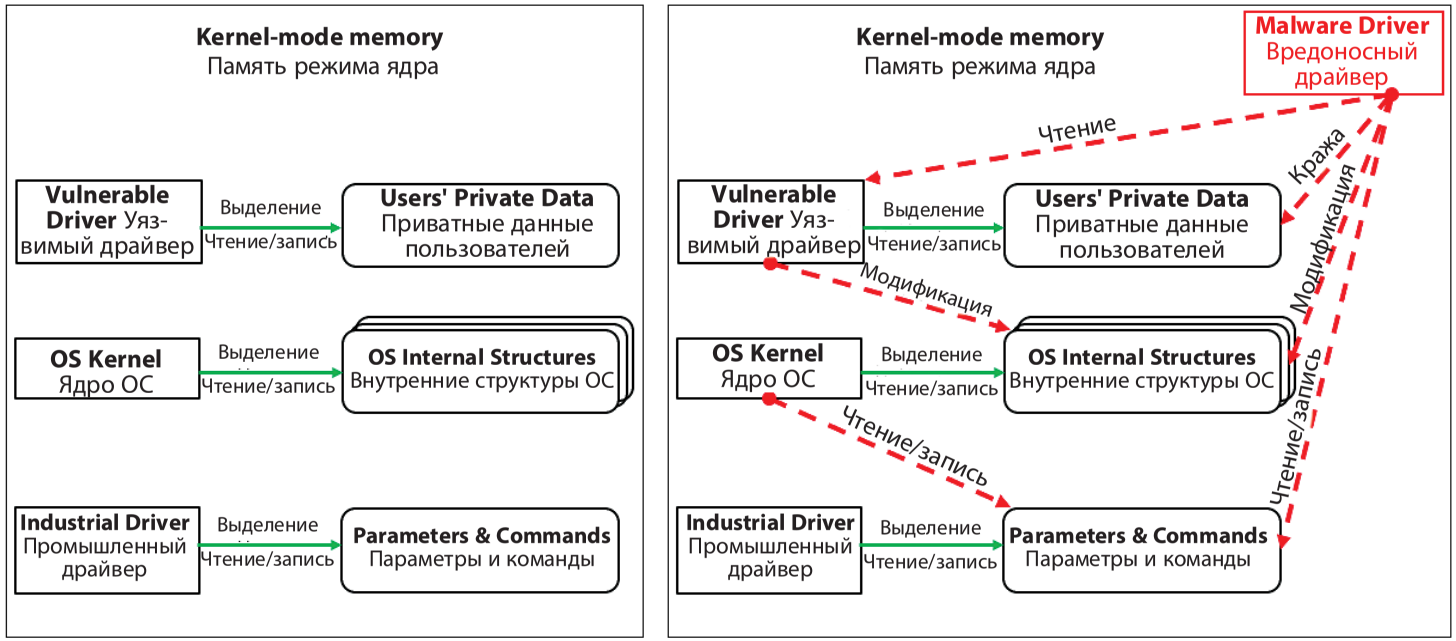


Рис. 56. Примеры легитимного и несанкционированного доступа к памяти режима ядра [274]: зеленые сплошные стрелки — легитимный; красные пунктирные стрелки — нелегитимный

(Multiplexed Information and Computing Service) [275], основанная на создании областей памяти с различными уровнями доступа. Она была реализована для мэйнфрейма GE 645. Мэйнфрейм GE 645 — крупномасштабный компьютер, разработанный компанией *General Electric* в 1960-х гг. специально для проекта Multics, оснащенный аппаратной поддержкой виртуальной памяти и сегментированной архитектурой.

За последние десятилетия было представлено множество проектов, направленных на совершенствование защиты ОС с использованием возможностей современных технологий, добавленных в процессоры *Intel*, *AMD* и *ARM*. Один из таких проектов — система AllMemPro (см. главу 8), которая защищает данные, выделенные сторонними драйверами, и предотвращает изменения внутренних структур ядра Windows. Однако AllMemPro не обеспечивает конфиденциальность кода драйверов.

Новое решение: проект MemoryRanger предлагается рассмотреть в этой и последующих главах раздела 2. Это новый подход к защите памяти режима ядра. Идея заключается в изолированном выполнении каждого драйвера в отдельной области памяти (isolated memory enclave). Каждая такая область имеет уникальную конфигурацию:

- драйвер может обращаться только к памяти, выделенной им ранее;
- драйвер не может обращаться к памяти, выделенной другими драйверами;
- в каждой изолированной области памяти (анклав) может быть загружен только один драйвер; выполнение кода других драйверов в этой области блокируется.

Во время работы ОС управление переключается между этими изолированными областями, что позволяет выполнять код драйверов и предотвращать несанкционированный доступ к их коду и данным.

Предложенный подход открывает принципиально новые возможности для защиты ОС, обеспечивая изолированное выполнение драйверов ядра и минимизируя последствия несанкционированного доступа к памяти ядра.

9.1. АНАЛИЗ СУЩЕСТВУЮЩИХ ПОДХОДОВ К ЗАЩИТЕ ПАМЯТИ ЯДРА ДЛЯ ОС WINDOWS И LINUX

Проведем сравнительный анализ встроенных инструментов Windows, научных публикаций и прототипов систем, предназначенных для защиты памяти режима ядра. Основное внимание уделим способности этих решений обеспечивать целостность и конфиденциальность кода ОС, сторонних драйверов и выделяемых ими буферов памяти.

9.1.1. Эволюция встроенных средств безопасности Windows

С момента выхода Windows XP *Microsoft* внедрила целый ряд компонентов безопасности в Windows 10, таких как SmartScreen, Application Guard, Device Guard, Exploit Guard и др. [276, 277].

Одним из компонентов Device Guard является KMCI, который предотвращает модификацию кода ядра ОС и сторонних драйверов. KMCI автоматически сбрасывает права на запись для страниц памяти с исполняемым кодом и вызывает BSOD с кодом ошибки 0xBE при попытке внести изменения.

Еще один компонент — PatchGuard — защищает внутренние структуры ОС, проверяя их целостность и вызывая BSOD с кодом 0x109 при обнаружении несанкционированных изменений. Например, PatchGuard предотвращает атаки типа DKOM unlinking (разрыв связей EPROCESS), но не обнаруживает отдельные атаки, связанные, скажем, с повышением привилегий [253].

Несмотря на внедренные меры, встроенные системы Windows не защищают память, выделенную сторонними драйверами, от чтения и записи, а также обеспечивают лишь частичную целостность внутренних структур ОС.

9.1.2. Гипервизорные системы защиты памяти

Исследователи по всему миру разрабатывают системы, которые используют аппаратную виртуализацию для защиты памяти режима ядра. Эти системы позволяют противостоять атакам со стороны драйверов ядра.

Системы Sentry, DADE, HACS, LKMG, LKRG, HUKO уже были описаны в параграфе 8.1.

DIKernel представляет новую архитектуру ядра, которая изолирует недоверенные расширения ядра (Loadable Kernel Modules, LKM) или драйверы ядра. Реализованное изолированное ядро (Domain Isolated Kernel, DIKernel) [278] предотвращает три широко используемые техники руткитов: перехват управления путем перезаписи пролога функций (inline hooking), перехват управления путем перезаписи указателей функций (function pointer hooking) и различные атаки типа DKOM. DIKernel контролирует доступ к памяти, отделяя расширения ядра от остальной части ядра ОС. Эта система использует возможности регистра контроля доступа к доменам (Domain Access Control Register, DACR) для архитектуры *ARM*. Ядро DIKernel реализовано в ОС Linux и протестировано на Raspberry Pi 2 model B.

Крайне важно предотвращать несанкционированное повышение привилегий пользовательских процессов. Система PrivGuard [279] отслеживает изменения соответствующих структур ядра путем перехвата управления в точках входа и выхода системного вызова. Система основана исключительно на внутренних механизмах ОС, без использования гипервизора, и ее прототип реализован на Ubuntu с архитектурой x86.

Система ExOShim защищает код ядра от чтения и записи, применяя гипервизор, который делает страницы памяти с кодом исполняемыми, но не читаемыми и не записываемыми [280]. Используя возможности EPT, система ExOShim помечает страницы памяти с кодом ядра как исполняемые без возможности чтения и записи. Еще одной особенностью ExOShim является самозащита.

Система OSsk, основанная на гипервизоре и предназначенная для защиты данных ядра ОС от атак руткитов, была предложена в работе [281]. Система предотвращает следующие атаки на ядро: сокрытие процессов и подмену указателей функций в системных структурах. OSsk проверяет целостность данных ядра в отдельном потоке, работающем параллельно с гостевой ОС, и защищает таблицу системных вызовов с помощью аппаратной защиты страниц и вызовов гипервизора. Прототип OSsk

реализован на основе гипервизора KVM с использованием Linux на Intel Core i7 860.

Гипервизор InkTag, который позволяет запускать доверенные приложения в недоверенной ОС, описан в работе [282]. Доверенные приложения выполняются в изолированном процессе (HAP), а их безопасность обеспечивается с помощью шифрования и хэширования. InkTag проверяет поведение ОС во время выполнения через специальные гипервызовы. Гипервизор создан на базе KVM и протестирован на Linux с процессором Intel Core i7 860. Несмотря на то, что InkTag не защищает память ядра, его подход к изоляции и защите приложений является перспективным.

Система BehaviorKI проверяет целостность ядра ОС, основываясь на выявлении вредоносного поведения [283]. Система использует аппаратную виртуализацию и возможности EPT для контроля отдельных компонентов ОС, включая статические и динамические данные. BehaviorKI анализирует последовательности событий, характерные для атак ВПО, и запускает проверку целостности только при совпадении с набором заданных шаблонов. В целях минимизации потерь производительности BehaviorKI контролирует события доступа к памяти только для фиксированных областей памяти. Система BehaviorKI обеспечивает целостность следующих компонентов ОС: кода ядра, а также статических и динамических данных ядра. Прототип реализован на основе гипервизора Xen и протестирован с использованием Linux и процессора Intel Core i7-4710MQ.

Система Hypernel для обеспечения целостности кода и данных ядра ОС описана в работе [284]. Она сочетает специальное аппаратное обеспечение и программный модуль Hypersec. Аппаратный компонент — монитор шины памяти (МШП, Memory Bus Monitor) — контролирует операции записи между процессором и оперативной памятью, вызывая прерывание при попытке изменить заданные фрагменты памяти. Однако МШП ограничен мониторингом статических объектов ядра, так как не поддерживает адресацию динамически выделяемых данных. Прототип Hypernel был реализован на платформе Versatile Express Juno r1 с ОС Linux.

Проект EPTI (Extended Page Table Isolation), направленный на предотвращение утечек памяти в режиме ядра, в частности для защиты облачных систем от атаки Meltdown [285], был предложен в работе [286]. Атака Meltdown позволяет считывать данные ядра из пользовательских приложений, однако она не может исполнять произвольный код. EPTI использует две структуры EPT для изоляции пространства пользователя и пространства ядра, что снижает накладные расходы за счет сохранения данных в отдельных буферах ассоциативной трансляции (TLB). Это предотвращает очистку TLB при переключении между пространствами. Прототип EPTI реализован для Linux-систем с процессором Intel Core i7-7700.

Система Sprobes для защиты ядра ОС смартфонов на процессорах *ARM* была предложена в работе [287]. Предполагается, что ядро содержит хотя бы одну уязвимость, которую нарушители могут использовать для перехвата управления или ROP-атак. Sprobes обеспечивает целостность кода ядра и предотвращает модификации памяти, используя аппаратную технологию ARM TrustZone. Это позволяет изолировать системные ресурсы. Прототип Sprobes реализован для Linux на ARM Cortex M15.

Система LAKEED защищает области памяти, используя несколько структур EPT [288]. Система предотвращает доступ кейлоггеров на уровне ядра к буферу пользовательских нажатий клавиш. Для этого LAKEED выделяет три EPT: два для драйверов клавиатуры и один для заданного драйвера ядра (расширения ядра), обеспечивая изоляцию памяти. Каждый драйвер может обращаться только к своей области, а взаимодействие между ними контролируется. LAKEED протестирована на Windows с процессором Intel Xeon E5606 и защищает данные, связанные с нажатиями клавиш. Система продемонстрировала результативность в работе с драйверами-фильтрами при минимальных накладных расходах, подтверждая возможность использования нескольких EPT для изоляции драйверов.

Система TZ-SSAP предназначена для защиты приложений от атак, связанных с уязвимостями ядра [289]. Эта проблема особенно актуальна для современных ОС с монолитным ядром, которое имеет полный доступ ко всем системным ресурсам. TZ-

SSAP использует технологию ARM TrustZone для обеспечения целостности кода и данных ядра. Система устанавливает для статических данных режим «только чтение» и защищает динамические данные от записи, перехватывая любые попытки их изменения. Прототип TZ-SSAP был протестирован на Linux с ARM CoreTile Express A9x4 и показал возможность предотвращения атак, включая модификации кода и данных вредоносными модулями ядра.

Система kR^X предназначена для защиты ядра современных ОС от атак типа ROP (Return-Oriented Programming, программирование на основе возвратов) и JIT-ROP (Just-In-Time Return-Oriented Programming, построение цепочек возврата «на лету» с использованием JIT-компиляции). Эти атаки позволяют запускать вредоносный код, используя уязвимости ядра. kR^X применяет следующие компоненты: политику R^X, которая ограничивает память либо чтением, либо исполнением, и KASLR, обеспечивающий загрузку кода ядра по различным адресам в целях затруднения эксплуатации уязвимостей. Система предотвращает перезапись указателей кода ядра и обеспечивает самозащиту памяти ядра. Прототип kR^X был реализован для ОС Linux x86-64 и протестирован на процессоре Intel Core i7-6700K, продемонстрировав работоспособность в защите от эксплойтов ядра [290].

Система DigTool предназначена для выявления и предотвращения уязвимостей в режиме ядра ОС Windows 7 и 10 [291]. Система способна обнаруживать такие попытки эксплуатации уязвимостей, как выход за границы памяти, использование памяти после ее освобождения (Use-After-Free, UAF) и нарушения типа «время проверки до времени использования» (Time-Of-Check to Time-Of-Use, TOCTOU). DigTool использует возможности гипервизора для мониторинга памяти, временно снимая флаг присутствия (P-флаг) с контролируемых страниц. При любом обращении к этим страницам система обрабатывает соответствующие исключения и отслеживает доступ с помощью механизма «одношаговой» операции, активируя флаг MTF или TF (Trap Flag). Основными целями защиты DigTool являются предотвращение доступа к памяти, находящейся за пределами выделенных буферов памяти, и блокировка обращений к уже освобожденной

памяти. Для этого система перехватывает функции управления памятью Windows, такие как ExAllocatePoolWithTag, ExFreePoolWithTag, RtlAllocateHeap и RtlFreeHeap, а также функции InterlockedPushEntrySList и InterlockedPopEntrySList, что позволяет отслеживать освобожденные блоки памяти и ассоциативные списки. DigTool предотвращает любые попытки чтения или записи в освобожденную память и выход за границы выделенных пулов.

Современные ядра операционных систем слишком сложны для полноценной защиты из-за значительного объема их кода, насчитывающего десятки миллионов строк. Это приводит к ежегодному обнаружению множества новых уязвимостей во многих популярных ОС. Для решения данной проблемы авторы работы [292] предложили систему SecPod, которая создает выделенное безопасное адресное пространство, функционирующее параллельно с существующим адресным пространством ядра. Главной целью безопасного пространства является обеспечение изоляции памяти посредством очистки обновлений гостевой таблицы страниц и предотвращение нелегальных модификаций. SecPod был протестирован на двух типах атак: исполнение неавторизованного кода и модификация данных. Для предотвращения первого сценария система регистрирует функцию обратного вызова для отслеживания обновлений страниц ядра. При создании новой исполняемой страницы в режиме ядра SecPod проверяет хэш-значения страниц кода. Если хэш совпадает с ожидаемым, то страница помечается как исполняемая в таблице теневых страниц. В противном случае фиксируется попытка выполнения несанкционированного кода. Для защиты данных SecPod применяет инварианты структур данных (Data Structure Invariants), идея которых была впервые использована в системе Gibraltar [293]. Система была разработана с использованием гипервизора KVM и технологии EPT. Тестирование SecPod проводилось на ОС Ubuntu с процессором Intel Core i5, где она продемонстрировала возможность предотвращения атак на код и данные ядра.

Гипервизорные решения могут защищать от физических атак на оперативную память, включая атаки методом холодной

перезагрузки (cold boot attacks) [294]. Одной из современных технологий для ограничения доступа к памяти является Intel SGX, использующая изолированные области памяти (анклавы) для защиты данных. Однако SGX защищает только память пользовательского режима, без поддержки защиты памяти ядра. Для устранения этого недостатка авторы предложили систему HyperCrypt, которая предотвращает физические атаки на память ядра. HyperCrypt использует технологии VT-x с EPT (SLAT) и симметричное шифрование AES для шифрования физических страниц памяти хоста. Система автоматически расшифровывает страницы, которые в данный момент используются гостевой ОС, без сохранения криптографических ключей в оперативной памяти. Прототип HyperCrypt был реализован на базе гипервизора BitVisor и протестирован в среде Linux. Хотя HyperCrypt не предотвращает доступ к памяти ядра со стороны кода ядра, он представляет собой перспективное программное решение для защиты от физических атак, таких как холодная перезагрузка.

В работе [295] выделяют ошибки в драйверах устройств сторонних производителей как основную причину сбоев операционных систем. Уязвимые легитимные драйверы могут нарушить целостность системы, а также быть использованы нарушителями для достижения вредоносных целей. Для решения этой проблемы была предложена система iKernel, обеспечивающая защиту ядра от ошибок и вредоносных действий драйверов. iKernel использует гипервизорную технологию виртуальных машин для изоляции драйверов устройств. Каждый драйвер запускается в отдельной виртуальной машине, что позволяет ограничить влияние его сбоев только на соответствующую виртуальную среду, не затрагивая ядро хоста и другие виртуальные машины. Система контролирует доступ драйверов к портам ввода-вывода и регистрам, отображаемым в память. Однако iKernel не обеспечивает целостность и конфиденциальность данных, выделенных сторонними драйверами. Система была реализована на базе виртуальной машины KVM и протестирована в среде Linux.

В работе [296] особое внимание было уделено вопросам защиты от успешной эксплуатации уязвимостей, которые при-

водят к повышению привилегий процессов. Для защиты от этих атак авторы разработали систему PrivWatcher, которая обеспечивает целостность соответствующих структур с помощью монитора ссылок. Этот механизм гарантирует согласованность защищенных полей данных от времени проверки до времени использования (TOCTOU). PrivWatcher применяет политику дискреционного управления доступом (DAC), предотвращая несанкционированное повышение привилегий процессов. Однако система не защищает код ядра и данные, выделенные сторонними драйверами, от чтения или модификации. Прототип PrivWatcher был реализован на основе ядра Linux с использованием KSM.

Система TrustZone-based Real-time Kernel Protection (TZ-RKP) использует аппаратные возможности *ARM* для защиты ядра ОС [297]. TZ-RKP предотвращает внедрение вредоносного кода в ядро и модификацию привилегированного кода, уже загруженного в память. Система обеспечивает изоляцию ядра с использованием технологии ARM TrustZone, что позволяет полностью защитить кодовую базу ядра. Однако TZ-RKP не предотвращает атаки, в которых ядро само модифицирует собственные данные под влиянием нарушителя. Прототип системы был реализован на ядре Linux для Android.

Прототип HyperCheck (монитор целостности) предназначен для обнаружения руткитов и атак, нарушающих целостность кода [298]. HyperCheck защищает ядра ОС Linux и Windows, а также гипервизоры XEN. HyperCheck использует возможности режима системного управления (SMM). Эта система гарантирует только целостность кода ядра ОС, не обеспечивая никакой защиты от выделенных данных.

Фреймворк HyperForce [299], основанный на гипервизоре, защищает код ядра от модификации со стороны нарушителя. HyperForce использует аппаратную виртуализацию для защиты от записи страниц памяти, содержащих инструкции и данные защищаемого кода. Система позволяет заданным фрагментам кода изменять свое содержимое, временно снимая блокировку страниц памяти перед вызовом прерывания и блокируя их обратно сразу после выполнения. Кроме того, HyperForce защищает таблицу дескрипторов прерываний (IDT) и ее регистр

(IDTR), предотвращая их модификацию. Однако HyperForce не обеспечивает защиту сторонних драйверов и выделенной ими памяти. Фреймворк реализован на базе гипервизора KVM и ядра Linux.

Гипервизорный фреймворк Shadow-box обеспечивает мониторинг и защиту целостности объектов ядра. Shadow-box включает две компоненты: легковесный гипервизор Light-box, изолирующий ядро и динамические объекты, и монитор безопасности Shadow-watcher, который проверяет статические и динамические элементы ядра. Shadow-box гарантирует целостность статических объектов ядра, устанавливая права на чтение и выполнение для кода, а для данных — только права на чтение. Однако фреймворк не защищает конфиденциальность кода и данных сторонних драйверов [300].

Изолированная система выполнения драйвера устройства, называемая SIDE (Streamlined Isolated Driver Execution), — это еще одна система, изолирующая ядро от проблемных драйверов устройств [301]. SIDE стремится максимально избегать изменения существующего кода ядра, использует возможности виртуальной памяти для настройки среды выполнения драйвера устройства и, например, может запустить немодифицированный драйвер устройства для сетевой карты Gigabit Ethernet в изолированной области памяти.

Анализ существующих исследовательских проектов показал, что для ОС Linux и Windows разработано множество гипервизорных прототипов, предназначенных для предотвращения несанкционированного доступа со стороны драйверов к коду и данным в памяти ядра, что подчеркивает высокую актуальность и значимость решаемой задачи. Однако ни одно из рассмотренных решений не обеспечивает комплексной защиты как для данных ядра ОС, так и для сторонних драйверов (табл. 25).

Некоторые проекты используют технологию Intel EPT для создания изолированных окружений в целях защиты памяти драйверов. Например, один набор структур EPT применяется в HACS и AllMemPro; два EPT используются в LKMG и EPTI; три EPT реализованы в системе LAKEED.

Таблица 25

Сводная таблица проектов по защите памяти

Название системы защиты, год	Защищаемые области памяти ядра						ОС
	Код ядра ОС и сторонних драйверов		Данные внутренних структур ОС		Данные сторонних драйверов		
	чтение	запись	чтение	запись	чтение	запись	
Windows built-in Protection, 2018	–	+	–	+	–	–	Windows
Multics, 1965	–	+	–	+	–	–	GE 645
Sentry, 2009	–	+	–	+	–	+	Linux
iKernel, 2007	–	+	–	+	–	–	Linux
HUKO, 2011	–	+	–	+	–	–	Linux, Windows
OSsk, 2011	–	+	–	+	–	–	Linux
HyperForce, 2012	–	+	–	–	–	–	Linux
HyperCheck, 2014	–	+	–	–	–	–	Linux, Windows
HyperCrypt, 2016	–	–	–	–	–	–	Linux
Hypernel, 2018	–	+	–	+	–	–	Linux
PrivWatcher, 2017	–	–	–	+	–	–	Linux
PrivGuard, 2018	–	–	–	+	–	–	Linux
TZ-RKP, 2014	–	+	–	–	–	–	Linux
TZ-SSAP, 2017	+	–	–	–	–	–	Linux
SIDE, 2013	–	+	–	+	–	+	Linux
Sprobes, 2014	–	+	–	–	–	–	Linux
SecPod, 2015	–	–	–	+	–	–	Linux
ExOShim, 2016	+	+	–	–	–	–	Windows
LAKEED, 2016	+	+	+	+	–	–	Windows
Shadow-box, 2017	–	+	–	+	–	–	Linux
kR^X, 2017	+	+	–	–	–	–	Linux
Digtool, 2017	–	–	+	+	–	–	Windows
DADE, 2017	–	+	–	+	–	–	Linux

Окончание табл. 25

Название системы защиты, год	Защищаемые области памяти ядра						ОС
	Код ядра ОС и сторонних драйверов		Данные внутренних структур ОС		Данные сторонних драйверов		
	чтение	запись	чтение	запись	чтение	запись	
HACS, 2017	–	–	–	+	–	–	Linux
DIKernel, 2018	–	+	–	+	–	–	Linux
BehaviourKI, 2018	–	+	–	+	–	–	Linux
LKRG, 2018	–	+	–	+	–	–	Linux
LKMG, 2018	–	+	+	+	+	+	Linux
EPTI, 2018	+	–	+	–	+	–	Linux
AllMemPro, 2018	–	–	+	+	+	+	Windows
MemoryRanger, 2018	+	+	+	+	+	+	Windows

9.2. РАЗРАБОТКА И РЕАЛИЗАЦИЯ ГИПЕРВИЗОРА MEMORYRANGER ДЛЯ ИЗОЛИРОВАННОГО ЗАПУСКА ДРАЙВЕРОВ

9.2.1. EPT: идея изоляции памяти

Предлагаемый прототип MemoryRanger основан на технологии Intel Virtualization Technology (Intel VT-x) и механизме расширенных таблиц страниц (EPT).

Подробный анализ применения EPT для мониторинга выполнения кода и контроля доступа к памяти был представлен в подпараграфах 7.2.1 и 8.2.1, а также в работе [245]. Детали использования EPT для защиты выделенных данных приведены в работе [253].

Предыдущие исследовательские проекты показывают, что можно изначально выделить фиксированные структуры EPT с различными атрибутами доступа и предотвратить доступ к памяти со стороны драйверов режима ядра путем переключения между ними; например, три набора структур EPT используются в LAKEED.

Основная идея MemoryRanger заключается в динамическом выделении структур EPT и обновлении атрибутов доступа страниц EPT в реальном времени.

MemoryRanger реализует механизм изоляции памяти драйверов, направленный на предотвращение несанкционированного доступа к данным и структурам ОС. Известно, что ядро ОС и драйверы свободно взаимодействуют с собственными данными и могут обращаться к памяти других компонентов, что включает легитимные операции выделения и чтения/записи данных (см. левую часть рис. 57). Однако без внешнего контроля такие действия, как модификация структур ОС, кража данных других драйверов или их выгрузка, несут угрозы информационной безопасности.

MemoryRanger решает эту проблему, создавая отдельные структуры EPT (Extended Page Table) для каждого загруженного драйвера, которые реализуют ограничение доступа к памяти соседних драйверов. На этапе загрузки драйверов A и B система выделяет отдельные EPT, назначая разные права доступа. Например, драйвер A получает доступ к своей памяти, но не может взаимодействовать с памятью драйвера B или модифицировать структуры ОС. Аналогично драйвер B ограничен в доступе к памяти A и структурам ОС (см. центральную и правую части рис. 57).

Этот подход обеспечивает соблюдение принципа минимальных привилегий. Каждый драйвер имеет доступ только к своей памяти, а любые несанкционированные попытки модификации данных или структур ОС блокируются.

Процесс работы MemoryRanger включает следующие шаги.

1. *Инициализация EPT по умолчанию*. При загрузке MemoryRanger создает структуру EPT по умолчанию (default EPT), где ядро ОС и загруженные драйверы получают полные права доступа к своей памяти. В рамках данной EPT структуры функционируют ядро ОС и его компоненты, а также все ранее загруженные драйверы. Однако все новые EPT структуры изначально устанавливаются как неисполняемые и недоступные для чтения/записи. MemoryRanger получает уведомления о загрузке драйверов, создании процесса и выделении/освобождении памяти сторонним драйвером.

2. *Создание EPT для драйвера A*. После загрузки драйвера A MemoryRanger создает отдельную EPT с настройками, позволяющими драйверу A выполнять операции только в собственной памяти. В структуре EPT по умолчанию доступ к памяти драйвера A блокируется (см. рис. 57, вторая колонка).

3. *Обновление EPT при выделении памяти*. При выделении драйвером A новой памяти она становится доступной в его EPT для операций чтения и записи. В остальных EPT, включая EPT по умолчанию, этот доступ блокируется.

4. *Создание и настройка EPT для драйвера B*. Аналогичный процесс повторяется при загрузке драйвера B. MemoryRanger создает новую EPT, где память B доступна только для драйвера B. Все остальные EPT обновляются, чтобы исключить доступ к памяти B.

5. *Переключение EPT*. MemoryRanger взаимодействует с планировщиком потоков Windows, переключая указатель EPT при выполнении защищаемых драйверов. Например, если ядро ОС передает управление драйверу A, то система переключается на EPT драйвера A. При этом попытки других драйверов обратиться к памяти A вызовут нарушение EPT, которое обрабатывается MemoryRanger.

6. *Обработка попыток несанкционированного доступа*. MemoryRanger фиксирует EPT-исключения при попытках чтения или записи, решая, предоставлять ли доступ. MemoryRanger обеспечивает принцип минимальных привилегий: доступ на чтение и запись к данным предоставляется только тем драйверам, которые ранее выделили эту память. Например, если драйвер A пытается получить доступ к памяти B, то MemoryRanger перенаправляет запрос на поддельную страницу. После чтения или записи фиктивных данных управление возвращается гипервизору, который восстанавливает оригинальные настройки страниц памяти.

После получения EPT violation MemoryRanger принимает решение о предоставлении или запрете доступа и реализует его. Решение принимается в соответствии со следующими параметрами:

- текущее значение указателя EPT;
- адрес источника (адрес инструкции кода, который пытается получить доступ к памяти);

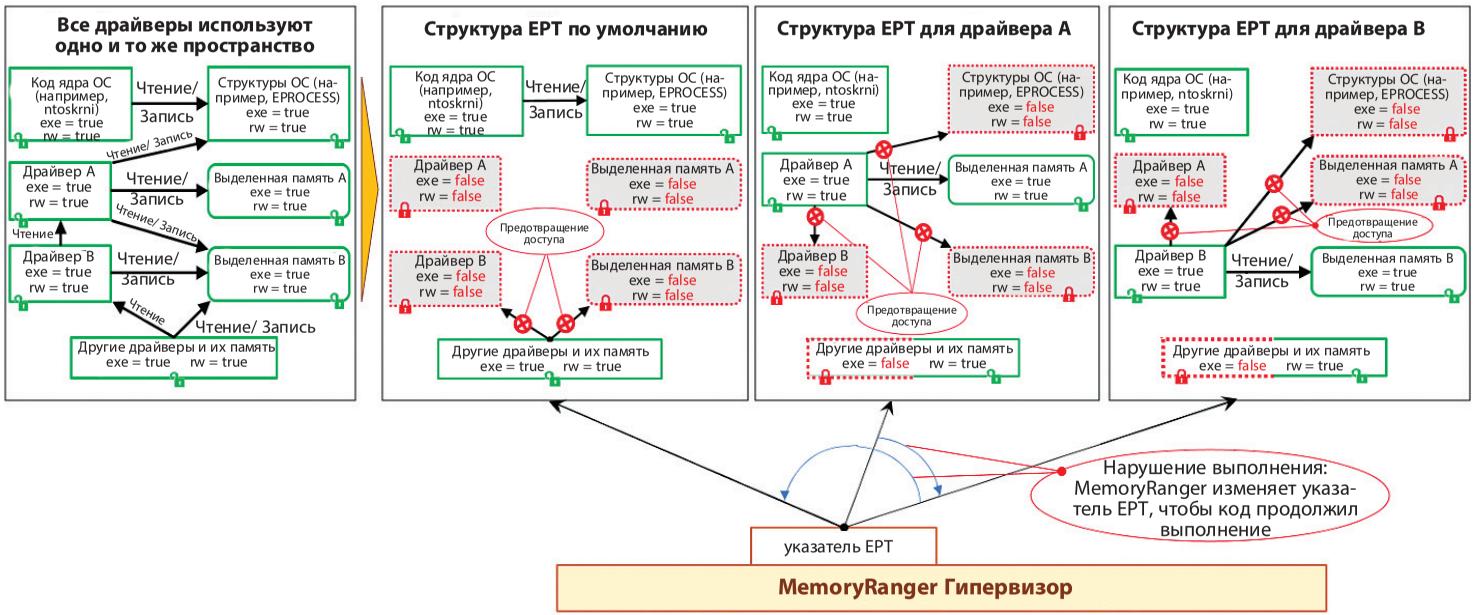


Рис. 57. Организация структур EPT для изоляции памяти двух драйверов режима ядра и выделенной им памяти: exe = true — выполнение разрешено; exe = false — выполнение запрещено; rw = true — чтение/запись разрешены; rw = false — чтение/запись запрещены

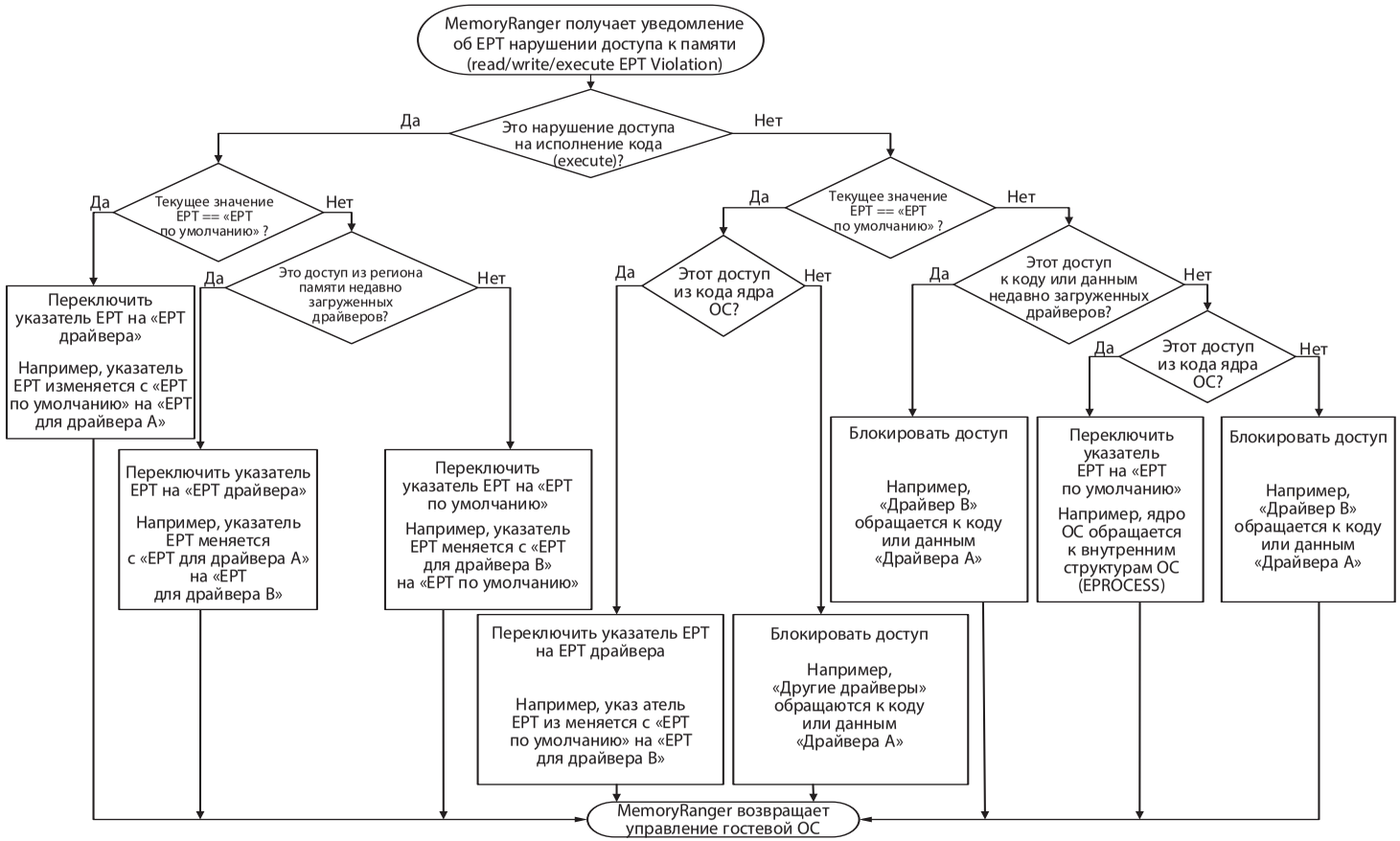


Рис. 58. Алгоритм диспетчеризации EPT-исключений в MemoryRanger

- адрес назначения или целевой адрес (адрес данных в памяти, к которым осуществляется доступ);
- тип доступа (чтение или запись).

На рис. 58 приведен алгоритм диспетчеризации EPT-исключений в MemoryRanger.

Исключение доступа. Легальный доступ к памяти на чтение/запись также может привести к EPT violation. Это исключение основано на страничной обработке памяти. Система управления памятью ОС Windows может выделять память для двух различных драйверов в одной и той же странице памяти размером 4 Кб. В результате MemoryRanger блокирует любой доступ к этой памяти во всех структурах EPT. Чтобы избежать такой блокировки, MemoryRanger выделяет буфер памяти размером, кратным размеру страницы (4 Кб).

Подробное описание работы системы MemoryRanger приведено в работах [302–305].

9.2.2. Архитектура MemoryRanger

MemoryRanger — это гипервизор типа bare-metal (см. параграф 1.1), который основан на технологии аппаратной виртуализации Intel VT-x и возможностях технологии EPT. Гипервизор MemoryRanger загружается с помощью консольного управляющего приложения, которое загружает драйвер ядра, запускающий гипервизор [302].

На рис. 59 представлена архитектура системы MemoryRanger, которая состоит из четырех компонентов:

1) драйвер режима ядра с обратными вызовами для получения уведомлений о загрузке (выгрузке) новых драйверов и запуске (завершении) процессов;

2) компонент DdiMon, который подключает подпрограммы управления памятью (перехват функций выделения и освобождения памяти);

3) компонент MemoryMonRWX, который обрабатывает EPT-исключения (EPT-нарушения) и управляет структурами EPT;

4) модуль управления политиками доступа к памяти (Memory Access Policy, MAP) выполняет роль центра принятия решений и обработки событий.

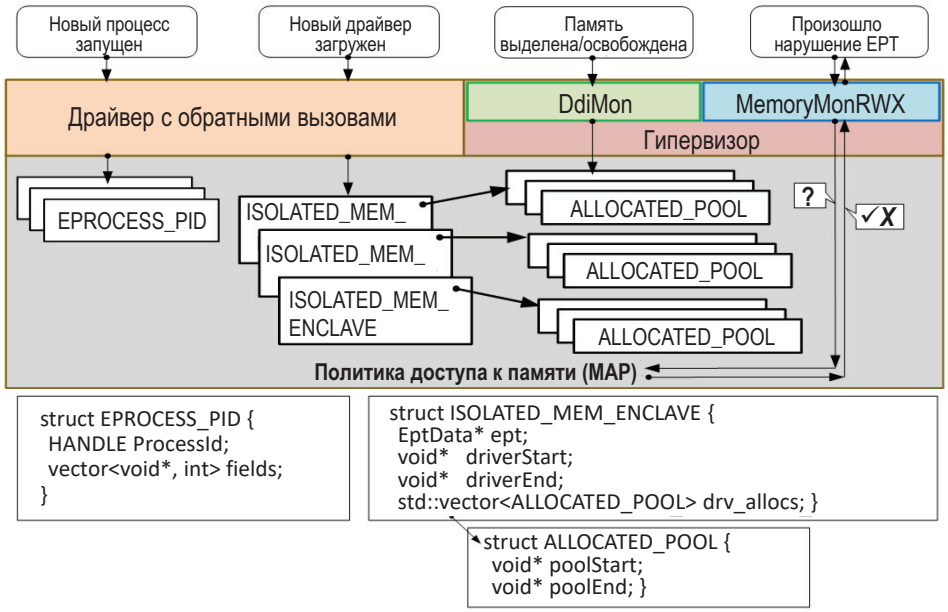


Рис. 59. Архитектура MemoryRanger

Драйвер режима ядра (драйвер с обратными вызовами) взаимодействует с ядром ОС и регистрирует обратные вызовы для обработки событий создания новых процессов и загрузки драйверов. Драйвер регистрирует две функции обратного вызова с помощью PsSetCreateProcessNotifyRoutineEx и PsSetLoadImageNotifyRoutine для получения уведомлений о создании процессов и загрузке драйверов. При создании процесса драйвер формирует структуру EPROCESS_PID, содержащую идентификатор процесса (Process ID), вектор адресов и размеров регионов памяти, которые необходимо защищать. Эта структура передается в компонент MAP, где она добавляется в список и используется для обновления структуры EPT с помощью гипервизора MemoryMonRWX. Аналогично при загрузке нового драйвера создается структура ISOLATED_MEM_ENCLAVE, содержащая адрес вновь выделенной структуры EPT, базовый адрес образа драйвера, конечный адрес (определяемый как сумма базового адреса и размера образа) и вектор выделенных пулов памяти. MAP добавляет эту структуру в соответствующий список, создает новую структуру EPT и обновляет все остальные EPT структуры.

Гипервизор DdiMon перехватывает вызовы API ядра, связанные с управлением памятью, такие как ExAllocatePoolWithTag и ExFreePoolWithTag. Эти вызовы используются сторонними драйверами для выделения и освобождения памяти. При перехвате DdiMon передает информацию о выделенной памяти в компонент MAP, включая адрес кода, выполняющего выделение памяти, адрес и размер выделенного буфера памяти. MAP создает структуру ALLOCATED_POOL и добавляет ее в вектор *drv_allocs* в соответствующей структуре ISOLATED_MEM_ENCLAVE. После этого обновляется структура EPT для учета нового региона памяти. Аналогичным образом DdiMon обрабатывает освобождение памяти, удаляя структуру ALLOCATED_POOL из списка. Такой подход позволяет отслеживать, какие пулы памяти были выделены и освобождены каждым драйвером, и учитывать это в политиках защиты памяти.

Гипервизор MemoryMonRWX отвечает за обработку EPT-исключений и контролирует доступ к памяти в реальном времени [245]. Этот компонент обрабатывает EPT-исключения, возникающие при попытках чтения, записи или выполнения кода. При каждом таком событии MemoryMonRWX отправляет в MAP данные, включая текущий указатель EPT, адрес источника (код, пытающийся получить доступ), адрес назначения (данные, к которым осуществляется доступ) и тип операции (чтение, запись или исполнение кода).

Модуль управления политиками доступа к памяти **(MAP)** принимает решения о разрешении или запрете доступа к памяти. MAP использует данные из структур EPROCESS_PID, ISOLATED_MEM_ENCLAVE и ALLOCATED_POOL для обновления структур EPT. На основе алгоритма, представленного на рис. 58, MAP принимает решение (разрешить или запретить доступ к памяти либо изменить указатель EPT) для обеспечения выполнения драйвера из другого EPT-контекста памяти.

Модуль MemoryRanger был разработан с использованием Microsoft Visual C++ с пакетом Windows Driver Kit (WDK) и протестирован в среде VMware Workstation на 64-разрядной версии Windows 10 (сборка 1709). Исходный код комплекса консольной программы, драйвера и гипервизора MemoryRanger представлен в открытом доступе [306].

Архитектура MemoryRanger была успешно протестирована в двух демонстрационных сценариях. В первом сценарии гипервизор защищал код и данные, выделенные сторонними драйверами, от несанкционированного доступа. Во втором сценарии система предотвращала атаку на повышение привилегий [307]. Таким образом, MemoryRanger реализует механизм изоляции выполнения драйверов, обеспечивая защиту памяти ядра ОС Windows.

9.3. ПРАКТИЧЕСКОЕ ПРИМЕНЕНИЕ MEMORYRANGER: ПРЕДОТВРАЩЕНИЕ НЕСАНКЦИОНИРОВАННОГО ДОСТУПА К ПАМЯТИ ЯДРА ДЛЯ СЛУЧАЯ НЕСКОЛЬКИХ НЕЗАВИСИМЫХ ДРАЙВЕРОВ

9.3.1. Результаты тестирования производительности MemoryRanger

Данный подпараграф посвящен результатам тестирования производительности MemoryRanger в сравнении с его ближайшим конкурентом — гипервизором AllMemPro [253].

Тестирование проводилось следующим образом: драйвер выделял буфер памяти, а затем измерял длительность доступа к данным с использованием счетчика Time Stamp Counter (TSC). Производительность оценивалась в четырех случаях:

1) без гипервизора с включенным кэшированием памяти;
2) без гипервизора с отключенным кэшированием памяти;
3) с гипервизором AllMemPro [253];
4) с гипервизором MemoryRanger [302].

Полученные значения средней задержки доступа к памяти и их отклонения представлены в табл. 26. Детали, связанные с количеством измерений и расчетов, опубликованы в исследовании [253].

Ближайшим конкурентом MemoryRanger является гипервизор AllMemPro, использующий одну структуру EPT для защиты данных. Такой подход приводит к значительным временны́м задержкам при каждом обращении к памяти, так как AllMemPro перехватывает как легитимные, так и нелегальные попытки доступа. MemoryRanger устраняет этот недостаток,

используя отдельные структуры EPT для каждого драйвера. Такой подход позволяет перехватывать только нелегальные обращения, тогда как легитимные запросы обрабатываются без перехвата.

Таблица 26

Оценка времени средней задержки доступа к памяти

№ п/п	Случай	Задержка доступа к памяти (значение счетчика TSC)
1	Без гипервизора с включенным кэшированием памяти	70 ± 2
2	Без гипервизора с отключенным кэшированием памяти	100,000 ± 4,000
3	С гипервизором AllMemPro	500,000 ± 10,000
4	С гипервизором MemoryRanger	170,000 ± 7,000

Результаты тестирования показывают, что MemoryRanger примерно в 3 раза быстрее своего ближайшего конкурента и лишь в 2 раза медленнее ОС без гипервизора с отключенным кэшированием памяти (см. табл. 26).

Временны́е задержки в работе MemoryRanger вызваны двумя основными причинами:

1) во время измерений ОС переключает указатель EPT между структурой EPT для драйвера X и основной структурой EPT, что увеличивает задержки;

2) смена указателя EPT вызывает очистку буфера TLB, что приводит к необходимости повторного заполнения буфера. Это занимает некоторое время, в течение которого адресация памяти использует длительный проход по страницам памяти (page-walk).

Первая причина обусловлена особенностями Windows как операционной системы с вытесняющей многозадачностью и неизбежна. Вторая проблема, связанная с очисткой TLB, может быть частично устранена. Авторы гипервизора EPTI показали, что можно ускорить работу гипервизора, избежав очистки TLB при смене указателя EPT [286].

Таким образом, MemoryRanger демонстрирует хорошие результаты тестирования, которые могут быть улучшены в дальнейшем.

9.3.2. Ограничения MemoryRanger

MemoryRanger является прототипом концептуального решения для изоляции драйверов и защиты памяти в режиме ядра. Несмотря на свои достижения, он имеет ряд ограничений, сходных с гипервизором AllMemPro. Основные ограничения включают:

- отсутствие функций по самозащите;
- отсутствие функций для защиты памяти, доступной для совместного использования;
- ограничения, связанные с механизмами файла подкачки и принудительной выгрузкой страниц;
- отсутствие защиты от прямого доступа к физической памяти (cold boot attacks);
- сложности обеспечения совместной работы с версиями Windows ОС на основе UEFI и технологиями SGX и Virtual Secure Mode.

Одним из важных недостатков является невозможность защиты буферов данных, которые должны использоваться как в режиме пользователя, так и в режиме ядра. Примером таких данных являются структуры IRP и MDL. MemoryRanger в текущей версии не защищает их.

9.3.3. Направления дальнейшего развития

MemoryRanger — это перспективный проект, который может быть существенно улучшен в нескольких направлениях, перечисленных ниже.

1. Защита от атак Spectre и Meltdown. MemoryRanger способен предотвращать утечки данных из режима ядра через сторонние каналы, такие как атаки Meltdown [285]. Для этого можно использовать изоляцию пользовательского режима и режима ядра с помощью дополнительных структур EPT (по аналогии с EPTI) и шифровать страницы памяти с конфиденциальными данными, как в проекте [294].

2. Ограничение доступа к ядру ОС. В текущей версии MemoryRanger драйверы изолированы, но ядро ОС остается доступным для них. Это создает риск того, что ВПО, используя уязвимости ядра, сможет получить доступ к конфиденциальным данным. Ограничение ядра ОС может быть реализовано путем

исключения его структур из EPT-драйверов после завершения их загрузки.

3. Изоляция файловой системы, реестра, сети и устройств. Системы безопасности Windows не предотвращают нелегальный доступ драйверов к файловой системе, реестру и сети. Это позволяет вредоносным драйверам читать, изменять, создавать или удалять файлы, а также незаконно взаимодействовать с устройствами, например камерой или микрофоном. MemoryRanger может внедрить правила доступа, которые изолируют эти элементы от нелегального доступа со стороны драйверов.

4. Генерация правил доступа из исходного кода драйверов. Текущая версия MemoryRanger не поддерживает совместный доступ к памяти между драйверами. Решением может стать генерация правил доступа к памяти на этапе компиляции драйверов. Обычно драйверы, выделяющие память для совместного использования, разрабатываются в одном окружении, например в одном файле решения (solution file) среды Visual Studio.

5. Интеграция в ядро Windows. Интеграция алгоритмов MemoryRanger в ядро Windows значительно повысит защиту данных от атак драйверов, что крайне важно для безопасности более миллиарда пользователей Windows. Решение защищает код режима ядра и данные от нелегального доступа, изолируя драйверы в специальных анклавах памяти, обеспечивая конфиденциальность и целостность данных. MemoryRanger демонстрирует высокую производительность благодаря созданию отдельных анклавов для каждого драйвера.

Глава 10
ПРОЕКТ MEMORYRANGER: ПРЕДОТВРАЩЕНИЕ НЕСАНКЦИОНИРОВАННОГО ДОСТУПА К СТРУКТУРАМ FILE_OBJECT В ЯДРЕ WINDOWS

Современные операционные системы с вытесняющей многозадачностью, такие как Windows и UNIX-подобные системы, функционируют в двух режимах: пользовательском и режиме ядра.

Память режима ядра, помимо самого ядра ОС и драйверов, содержит множество важных структур данных, которые могут быть использованы нарушителями. Современные процессоры не предоставляют встроенных механизмов безопасности для предотвращения несанкционированного доступа к памяти ядра. Это создает возможность для атак, при которых нарушители получают права на чтение и запись в память режима ядра, устанавливая вредоносные драйверы или используя уязвимости в легитимных драйверах.

Для противодействия этим угрозам в Windows внедрены механизмы защиты, такие как PatchGuard и Device Guard, однако они защищают лишь фиксированные области памяти. Например, PatchGuard обнаруживает несанкционированное изменение отдельных системных структур и полей, но не обеспечивает защиту динамически выделяемых структур данных.

Примером уязвимости является CVE-2018-8120, которая позволяет нарушителям выполнять произвольный код в режиме ядра [308]. Другие уязвимости, такие как CVE-2018-8611 и CVE-2018-8170, дают возможность повысить привилегии процессов даже на актуальных версиях Windows 10 [309, 310]. Современные целевые атаки (APT), такие как FruityArmor и SandCat, используют уязвимости нулевого дня в компонентах ядра Windows, например в модуле Win32k.sys. Примеры таких уязвимостей включают CVE-2018-8453, CVE-2018-8589 и CVE-2019-0797. Во время подобных атак нарушители изменяют соответствующие поля структуры EPROCESS, ассоциированной с конкретным процессом, что позволяет им повышать привилегии процессов без реакции со стороны средств защиты информации.

Рассмотрим новую атаку на структуры FILE_OBJECT в памяти режима ядра. Эта атака позволяет нарушителям несанкционированно получать права на чтение и запись содержимого файлов, открытых драйверами в режиме монопольного использования (эксклюзивном режиме), без возможности совместного доступа к файлу. В результате нарушители могут незаконно получить доступ к локальным и сетевым файлам, доступ должен быть ограничен механизмами ОС.

10.1. АНАЛИЗ АРХИТЕКТУРНЫХ УЯЗВИМОСТЕЙ ФАЙЛОВОЙ ПОДСИСТЕМЫ WINDOWS: ИССЛЕДОВАНИЕ ВОЗМОЖНОСТЕЙ НЕСАНКЦИОНИРОВАННОГО ДОСТУПА К ФАЙЛАМ, ОТКРЫТЫМ В РЕЖИМЕ МОНОПОЛЬНОГО ИСПОЛЬЗОВАНИЯ

Опишем внутреннее устройство функций файловой системы в режиме ядра с общим доступом и без него. Также приведем подробности о том, как получить полный доступ к файлу, открытому в эксклюзивном режиме.

10.1.1. Обзор файловой системы в режиме ядра

Во время выполнения операций с файлами драйверы Windows используют следующие функции:

- ZwCreateFile — для создания или открытия файла;
- ZwReadFile и ZwWriteFile — для чтения и записи содержимого файла;
- ZwClose — для закрытия дескриптора файла и освобождения системных ресурсов.

Подробное описание всех параметров этих функций представлено в MSDN [311]. Этапы создания файла и получения дескриптора описаны в работе [312].

Первая операция создания файла (ZwCreateFile) принимает полное имя файла, флаги и другие параметры, возвращая дескриптор открытого файла или код ошибки. Во время выполнения этой операции менеджер ввода-вывода (I/O Manager) вызывает менеджер объектов (Object Manager) для поиска указанного файла и разрешения символических ссылок на объект файла [313].

Менеджер объектов, в свою очередь, обращается к монитору безопасности (SRM) для выполнения проверки доступа (рис. 60). Согласно информации в главе 7 книги [1] монитор безопасности (SRM) определяет, разрешает ли список управления доступом (Access Control List, ACL) выполнять запрашиваемую операцию с файлом. Если доступ разрешен, то менеджер объектов предоставляет необходимые права доступа и связывает их с дескриптором файла, который возвращается драйверу.

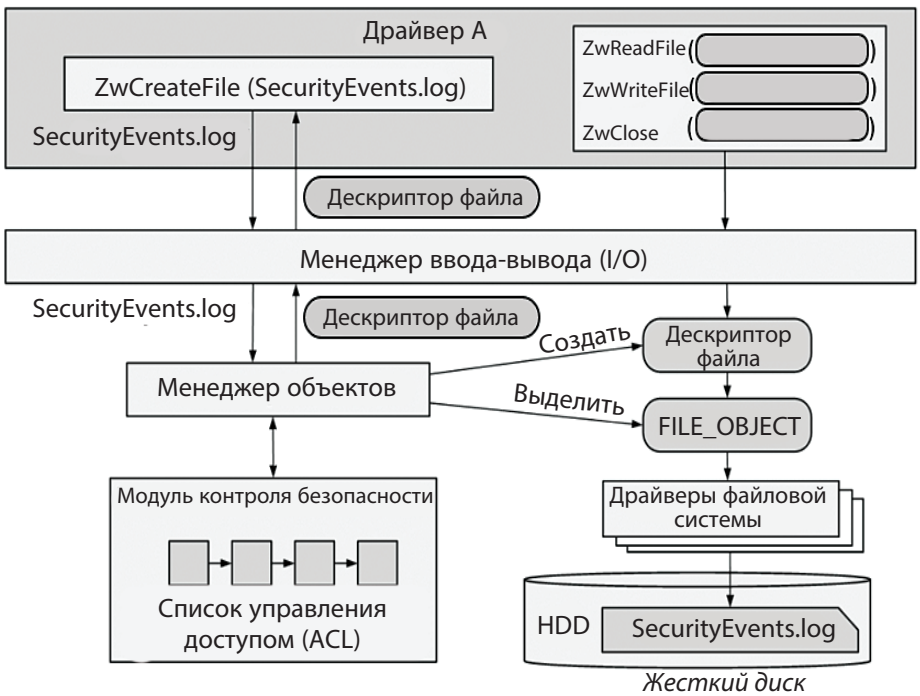


Рис. 60. Внутренние компоненты процедур файловой системы в ядре Windows

Далее менеджер ввода-вывода (I/O Manager) создает структуру FILE_OBJECT с помощью менеджера объектов (Object Manager) [314, 315]. Эта структура является внутренним элементом ОС и хранит информацию об открытом файле на уровне ядра. Таким образом, для каждого файла, открытого драйвером, существуют две сущности: хендл файла (файловый дескриптор), используемый для доступа, и FILE_OBJECT, представляющий файловый объект на уровне ядра (см. рис. 60).

Функции ZwReadFile и ZwWriteFile используют дескриптор, полученный на предыдущем этапе, для чтения и записи содержимого файла. Согласно [316] во время выполнения этих операций модуль SRM не задействован (см. рис. 60), что создает потенциальную уязвимость, которую могут использовать нарушители.

Функция закрытия файла ZwClose завершает все операции записи и закрывает открытый файл, используя переданный дескриптор. Аналогично операциям чтения и записи модуль SRM не участвует в данной процедуре. Эти нюансы работы с файловыми объектами могут быть использованы для атак на ОС.

10.1.2. Встроенные в Windows механизмы безопасности

Функция создания файла ZwCreateFile содержит параметр ShareAccess, который определяет режим совместного доступа к файлу. Этот флаг устанавливает, каким образом файл может быть открыт другими процессами или драйверами, а также типы доступа, которые могут быть разрешены или запрещены. В случае нарушения правил совместного доступа система возвращает код ошибки STATUS_SHARING_VIOLATION [315].

Рассмотрим следующий сценарий (рис. 61): драйвер А вызывает ZwCreateFile без указания разрешений на совместное использование файла или с флагом ShareAccess, равным NULL. В результате Object Manager успешно создает объект FILE_OBJECT для этого файла. Затем нарушитель, используя свой драйвер, пытается получить доступ к файлу, который уже открыт в эксклюзивном режиме драйвером А, вызывая ZwCreateFile. Однако Object Manager возвращает статус ошибки STATUS_SHARING_VIOLATION (нарушение совместного доступа), запрещая доступ к файлу. Таким образом, нарушитель не может открыть файл, доступ к которому уже ограничен, благодаря работе монитора безопасности (SRM).

В ходе исследования была выявлена уязвимость в работе системных функций файловой системы. Проблема заключается в том, что Object Manager выполняет проверку безопасности через SRM только при вызове функции ZwCreateFile. Во время выполнения операций ZwReadFile и ZwWriteFile никаких дополнительных проверок безопасности не проводится. Эта особен-

ность работы механизмов безопасности создает потенциальную уязвимость, которую могут использовать нарушители для обхода ограничений доступа после успешного открытия файла.

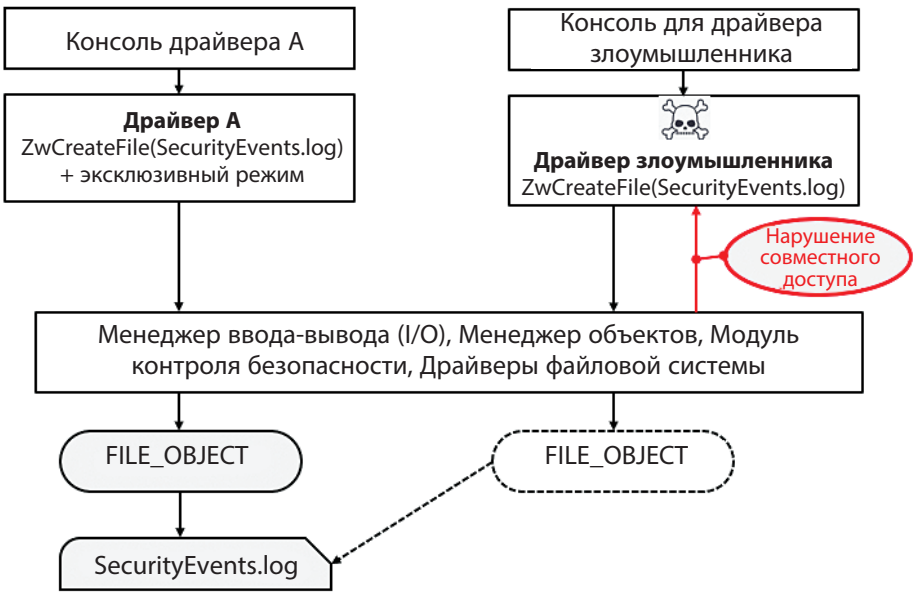


Рис. 61. ОС Windows предотвращает попытку незаконного доступа через вызов ZwCreateFile к файлу, открытому без прав общего доступа

10.1.3. Анализ структуры FILE_OBJECT

Структура FILE_OBJECT создается менеджером ввода-вывода (I/O Manager) при открытии файла драйвером. Эта структура содержит около 30 полей, детальное описание которых представлено в работе [317], а также в главе 4 работы [315].

Некоторые поля структуры FILE_OBJECT дублируют флаги, используемые при вызове функции ZwCreateFile. Например, если файл был открыт с флагом ShareAccess, равным NULL, то соответствующие поля SharedRead и SharedWrite в структуре FILE_OBJECT также будут равны нулю. Исследование показывает, что ручное изменение значений полей SharedRead и SharedWrite структуры FILE_OBJECT недостаточно для обеспечения общего доступа к уже открытому файлу.

Чтение и запись файлов осуществляются с использованием следующих полей структуры FILE_OBJECT:

1) Volume Parameter Block (Vpb);
2) FsContext;
3) FsContext2;
4) SectionObjectPointer.

Поле Vpb инициализируется менеджером ввода-вывода до отправки запроса на создание или открытие файла драйверу файловой системы. Это поле указывает на смонтированный блок параметров тома (Volume Parameter Block, VPB), связанный с целевым объектом устройства.

Согласно [315] поля FsContext и FsContext2 инициализируются и поддерживаются драйверами файловой системы и менеджером кэша NT.

Поле FsContext указывает на структуру FSRTL_COMMON_FCB_HEADER, которая должна быть выделена файловой системой или сетевым драйвером.

Поле FsContext2 ссылается на блок управления контекстом (Context Control Block, CCB) и связано с объектом файла.

Поле SectionObjectPointer ссылается на структуру типа SECTION_OBJECT_POINTERS и хранит информацию, связанную с отображением файла в память и его кэшированием.

Эти четыре поля используются при выполнении операций чтения и записи, которые осуществляются без привлечения модуля SRM и проверки разрешений на совместный доступ.

Таким образом, уязвимость структуры FILE_OBJECT состоит в том, что нарушители могут считывать эти поля и использовать их для получения доступа к открытому файлу. Эта уязвимость позволяет реализовать атаку на основе захвата управления (hijacking), которая будет рассмотрена далее.

10.1.4. Доступ к содержимому файла, открытого в монопольном режиме, путем перехвата его FILE_OBJECT

Рассмотрим сценарий атаки (рис. 62) при которой нарушитель получает доступ к файлу, открытому в эксклюзивном режиме. Как и в ранее описанных случаях, драйвер A (DriverA) открыл

целевой файл в монопольном режиме, и Object Manager выделил для него структуру FILE_OBJECT.

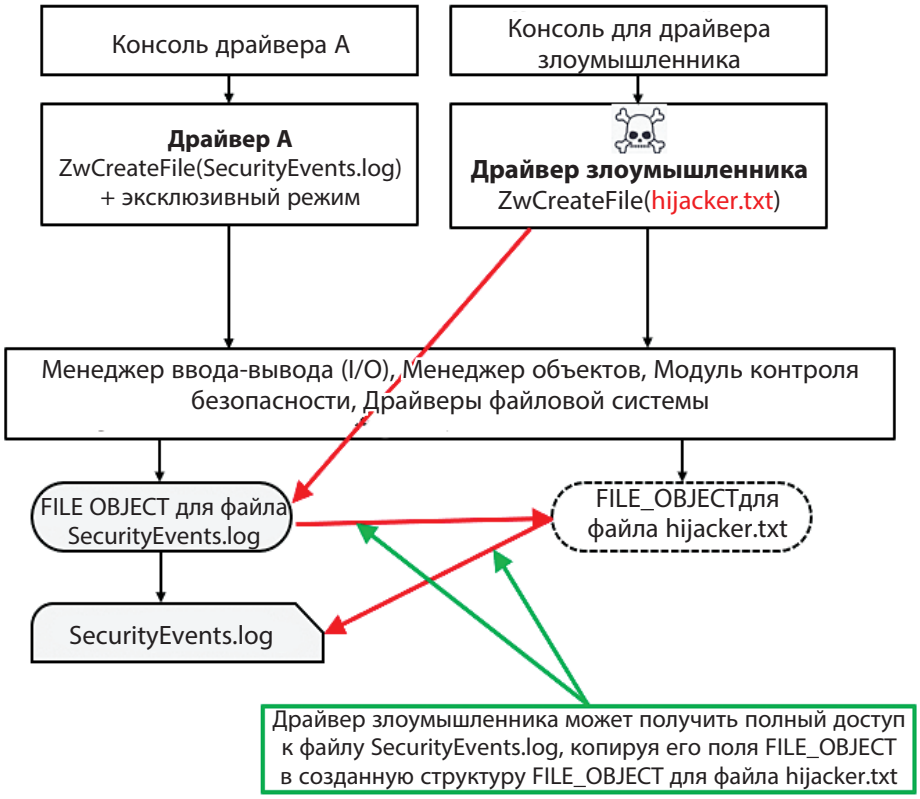


Рис. 62. Атакующий драйвер перехватывает целевую структуру FILE_OBJECT для получения незаконного доступа к файлу, открытому без разрешения на совместный доступ

Легальный доступ к этому файлу заблокирован, однако вредоносный драйвер использует следующие шаги для получения несанкционированного доступа.

1. Создание нового файла: нарушитель вызывает функцию ZwCreateFile, чтобы создать новый файл, например, с именем hijacker.txt.

2. Получение указателя на FILE_OBJECT: функция ObReferenceObjectByHandle используется для получения указателя на структуру FILE_OBJECT, связанную с файлом hijacker.txt.

3. Поиск целевой структуры FILE_OBJECT: используя имя целевого файла, нарушитель находит соответствующую структуру FILE_OBJECT, обходя список объектов в Object Directory [115, 268, 318–321].

4. Копирование полей: из структуры FILE_OBJECT целевого файла в структуру FILE_OBJECT для файла hijacker.txt копируются следующие четыре поля:

- Vpb — указывает на блок параметров тома (Volume Parameter Block), связанный с устройством;
- FsContext — содержит указатель на структуру заголовка общих данных FCB;
- FsContext2 — ссылается на блок управления контекстом (CCB), связанный с объектом файла;
- SectionObjectPointer — содержит информацию об отображении файла в память и кэшировании.

5. Чтение и запись данных: используя открытую дескрипторную структуру для файла hijacker.txt, вызываются функции ZwReadFile и ZwWriteFile для чтения и записи содержимого целевого файла.

После выполнения этих пяти шагов вредоносный драйвер получает полный доступ к чтению и записи содержимого целевого файла. Эксперимент подтвердил выполнимость этих действий на Windows 10 (1809×64). При этом механизм защиты PatchGuard, предназначенный для предотвращения нелегальных изменений в памяти, не вызвал ошибок типа BSOD даже через 10 ч после атаки.

Представленный сценарий атаки показал, что вредоносный драйвер может получить доступ к файлам, открытым в монопольном режиме, путем копирования содержимого их структур FILE_OBJECT. При этом PatchGuard не предотвращает данное вторжение. Эта уязвимость присутствует во всех версиях Windows начиная с Windows NT 4.0. Компоненты Windows, такие как Object Manager и Security Reference Monitor, участвующие в атаке, впервые упоминались в работе [322].

Для защиты содержимого файлов от кражи и обеспечения безопасной работы пользователя может быть использован гипервизор MemoryRanger. Подробное описание шагов по адаптации MemoryRanger для предотвращения данной атаки представлено в параграфе 10.2.

10.2. МОДИФИКАЦИЯ ГИПЕРВИЗОРА MEMORYRANGER ДЛЯ ЗАЩИТЫ СТРУКТУР FILE_OBJECT

10.2.1. Архитектура MemoryRanger

Архитектура MemoryRanger подробно описана в подпараграфе 9.2.2. Кратко повторим основные моменты.

MemoryRanger — это решение с открытым исходным кодом, направленное на защиту памяти режима ядра через изоляцию драйверов в специальных анклавах памяти с использованием гипервизорной технологии VT-x и механизма EPT [302]. Архитектура (см. рис. 59) включает драйвер режима ядра, модули DdiMon и MemoryMonRWX, а также механизм управления доступом к памяти (MAP). Драйвер режима ядра отслеживает события, такие как загрузка новых драйверов. Модуль DdiMon перехватывает вызовы API режима ядра, а MemoryMonRWX обеспечивает контроль всех операций доступа к памяти: чтения, записи и выполнения. MAP решает, разрешить или запретить доступ к защищенным областям памяти.

При загрузке нового драйвера MemoryRanger выделяет для него отдельную структуру EPT, в которой разрешено выполнение кода только этого драйвера и отдельных функций ОС, таких как диспетчер системных потоков. Выполнение кода других драйверов в данной структуре EPT запрещено. Кроме того, MemoryRanger перехватывает функцию ExAllocatePoolWithTag, что позволяет защищать выделяемую драйверами память: выделенные буферы становятся доступны только тому драйверу, который их создал, благодаря дополнительным ограничениям на уровне EPT.

10.2.2. Настройка MemoryRanger для защиты структур FILE_OBJECT

Для защиты структур FILE_OBJECT в MemoryRanger были модифицированы следующие компоненты:

- гипервизор DdiMon;
- компонент управления доступом к памяти (MAP).

Обновленный модуль DdiMon перехватывает вызовы функций файловой системы ZwCreateFile() и ZwClose() и обрабатывает их следующим образом.

Обработка вызова ZwCreateFile():

1) вызывает оригинальную функцию ZwCreateFile и проверяет успешность выполнения;

2) определяет, был ли файл создан с исключительным правом доступа (без совместного доступа);

3) проверяет, принадлежит ли адрес возврата защищенным драйверам;

4) получает указатель на выделенную структуру FILE_OBJECT с помощью функции ObReferenceObjectByHandle;

5) добавляет указатель на FILE_OBJECT и его размер (0xB байт) в список защищаемых областей памяти.

Обработка вызова ZwClose():

1) проверяет, принадлежит ли адрес возврата защищенным драйверам;

2) получает указатель на структуру FILE_OBJECT с помощью функции ObReferenceObjectByHandle;

3) удаляет указатель на FILE_OBJECT и его размер (0xB байт) из списка защищенных областей памяти.

Компонент MAP обрабатывает нарушения доступа к структурам FILE_OBJECT аналогично существующему алгоритму для работы с выделенными пулами памяти.

Результаты экспериментов показали, что обновленная версия MemoryRanger успешно защитила структуры FILE_OBJECT от модификации, не блокируя легитимный доступ к ним со стороны ОС (рис. 63). Эксперименты, проведенные на Windows 10x64, показали, что обновленный MemoryRanger приводит к умеренному снижению производительности.

На рис. 63 показано, как MemoryRanger предотвращает попытки нарушителя получить несанкционированный доступ к файлу, открытому в эксклюзивном режиме. Драйвер A создает файл SecurityEvents.log в режиме без совместного доступа. Нарушитель с помощью своего драйвера пытается получить доступ к этому файлу, вызывая ZwCreateFile для своего файла hijack.txt. Без защиты структура FILE_OBJECT файла SecurityEvents.log могла быть перехвачена нарушителем, что позволило бы ему получить нелегальный доступ к содержимому файла. MemoryRanger предотвращает такую атаку, изолируя

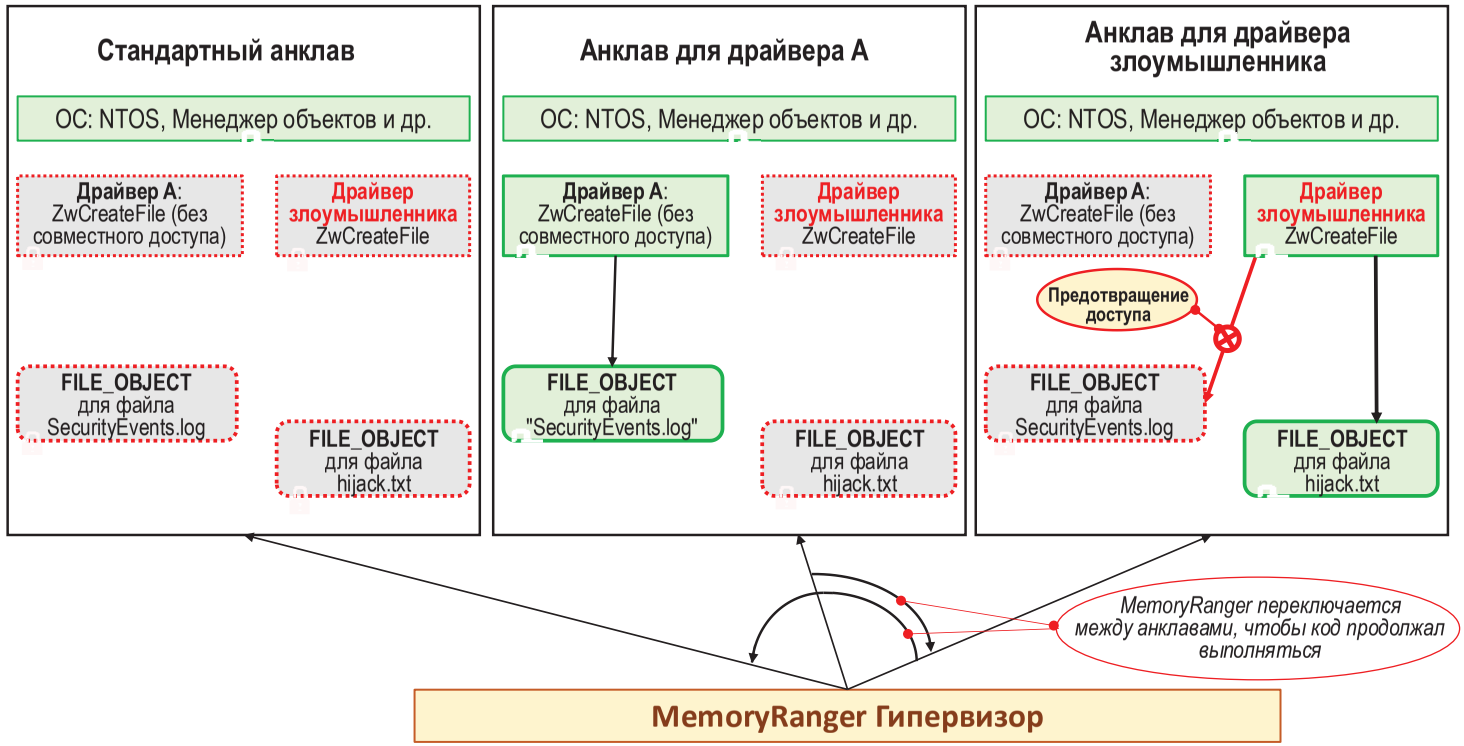


Рис. 63. MemoryRanger предотвращает перехват FILE_OBJECT путем запуска драйверов в изолированных анклавах памяти режима ядра

доступ к структуре FILE_OBJECT и блокируя несанкционированные операции доступа к памяти.

Исходный код обновленной версии MemoryRanger и демонстрационные видеоролики доступны в источнике [306].

Таким образом, предложенная модификация MemoryRanger доказала свою работоспособность в защите структур FILE_OBJECT, предотвращая их захват и сохраняя легальный доступ.

Глава 11
ПРОЕКТ MEMORYRANGER: ПРЕДОТВРАЩЕНИЕ АТАК ТИПА HIJACKING НА СТРУКТУРЫ ЯДРА WINDOWS

Рассмотрим три новых атаки типа hijacking, которые основаны на обходе защитных механизмов ОС. Первые две атаки приводят к получению нелегального доступа к содержимому файлов, открытых в режиме монопольного доступа. Третья атака повышает привилегии процесса, не применяя подмену токенов. Хотя специалисты по безопасности ОС Windows выпустили новые средства защиты, однако попытки доступа к динамически выделяемым данным в ядре не полностью контролируются. Гипервизор MemoryRanger восполняет этот пробел в системе безопасности. Обновленный MemoryRanger предотвращает эти новые атаки, а также работает в ОС Windows 10 1903 x64.

11.1. АНАЛИЗ НОВЫХ АТАК НА СТРУКТУРЫ ЯДРА WINDOWS: ИССЛЕДОВАНИЕ МЕТОДОВ НЕСАНКЦИОНИРОВАННОГО ДОСТУПА К ФАЙЛАМ И ПОВЫШЕНИЯ ПРИВИЛЕГИЙ ПРОЦЕССОВ

11.1.1. Анализ новых атак на структуры ядра Windows

В современных ОС, включая Windows, защита ядра и связанных с ним данных остается одной из важнейших задач информационной безопасности. Однако множество атак, основанных на уязвимостях в ядре и сторонних драйверах, демонстрируют возможность обхода существующих защитных механизмов, таких как PatchGuard, Device Guard и Security Reference Monitor. Эти атаки используют разнообразные техники, включая манипуляцию со структурами ядра (например, _TOKEN) и захват файловых дескрипторов (Handle Hijacking).

Дополнительной сложностью в защите ядра является наличие драйверов сторонних разработчиков, которые функционируют с тем же уровнем привилегий, что и ядро ОС, но содержат многочисленные уязвимости. Исследователи отмечают, что драйверы

ядра создают дополнительную поверхность атаки, так как имеют полный доступ к памяти ядра [323].

Ниже представлена таблица некоторых известных атак на ядро Windows, описывающая их принцип действия, годы обнаружения и использованные техники (табл. 27). В таблице также указаны источники информации, позволяющие получить более детальное описание каждой атаки.

Таблица 27

Известные атаки на ядро ОС Windows и их особенности

Название атаки и год обнаружения	Принцип действия атаки
GhostHook (2017) [324, 325]	Использует технологию Intel Processor Trace (PT) для обхода защиты PatchGuard, что позволяет перехватывать функции ядра
InfinityHook и ByePg (2018) [326]	Позволяет перехватывать системные вызовы и события ядра, обходя PatchGuard и Hyperguard
Banload (2019) [327, 328]	Банковский троян Banload использует драйвер ядра для удаления защитных программ, таких как AVG, Avast и IBM Trusteer Rapport, в целях кражи данных
RobbinHood Ransomware (2020) [329]	Хакеры с помощью вируса-вымогателя RobbinHood используют уязвимость драйвера для временного отключения проверки подписей драйверов Windows
Руткиты криптомайнеров (2019) [330, 331]	Криптомайнеры применяют руткиты ядра для предотвращения завершения их работы; было заражено более 50 000 систем в 90 странах
Уязвимости в драйверах (2019) [332]	Использование уязвимостей в более чем 40 драйверах от сертифицированных *Microsoft* производителей для повышения привилегий
Watchdog уязвимость (2019) [333]	Уязвимость в механизме контроля драйверов через инъекцию APC (Asynchronous Procedure Call) позволяет обходить проверку подписи драйверов

Анализ этой таблицы показывает, что штатные средства Windows ОС обеспечивают ограниченную защиту памяти ядра.

11.1.2. Модель угроз

Предположим, что нарушители могут выполнить вредоносный код в режиме ядра. Проанализируем два типа атак на данные ядра, которые приводят к следующему (рис. 64):

- получение доступа к файлам, открытым в эксклюзивном режиме, за счет перехвата уникальных идентификаторов (дескрипторов, Handle) и модификации структур NTFS;
- повышение привилегий процесса без использования техники подмены токенов (перехват токенов).

В случае атак на файлы легальный драйвер создает файл с помощью функции ZwCreateFile с нулевым флагом ShareAccess, что соответствует монопольному или эксклюзивному доступу.

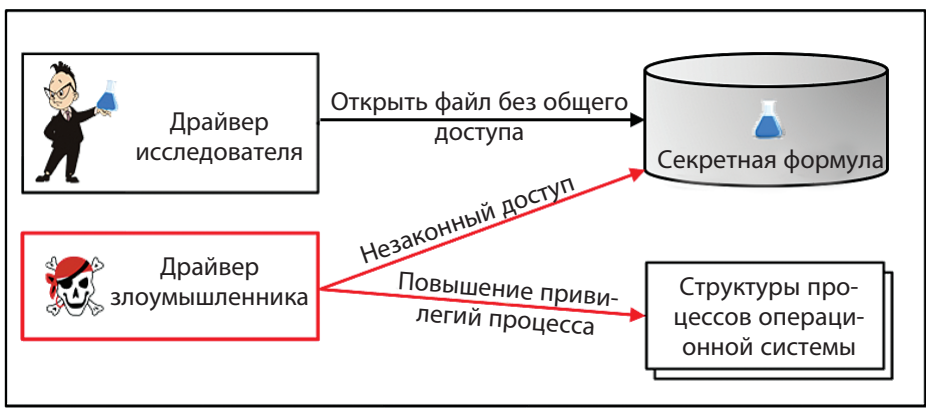


Рис. 64. Виды атак на файлы и атака с повышением привилегий

Проведенные исследования позволили выявить два вида атак, которые обходят защиту Windows, получая несанкционированный доступ к файлам, открытым без совместного доступа, через изменение внутренних структур ядра ОС, связанных с Object Manager и драйвером NTFS. Третья атака повышает привилегии процессов через модификацию структуры _TOKEN. Эта атака была описана в базе MITRE ATT&CK под названием Access Token Manipulation [334]. Техника повышения привилегий на основе SeImpersonatePrivilege находится за рамками данной работы [335].

Все представленные новые атаки работают на Windows 10 1903 x64, обходя защитные механизмы, такие как PatchGuard, Device Guard и Security Reference Monitor.

Для предотвращения этих атак предлагается модифицированный гипервизор MemoryRanger, который изолирует вновь загружаемые драйверы в отдельных адресных пространствах ядра и ограничивает доступ к заданным областям памяти.

Новые возможности MemoryRanger позволяют создавать специальные анклавы данных для защиты структур ядра, таких как _TOKEN. Эти анклавы включают защищенные структуры, ядро ОС и встроенные драйверы, ограничивая доступ как для новых, так и для ранее загруженных драйверов.

11.1.3. Две атаки на файловые объекты для случая файлов, открытых в режиме без совместного доступа

Опишем внутренние механизмы работы с файлами в ядре ОС Windows, структуры данных и их взаимосвязи. Рассмотрим две атаки перехвата, которые позволяют незаконно читать и перезаписывать содержимое файла, открытого в эксклюзивном режиме. Эти атаки основаны на модификации внутренних структур данных ОС, участвующих в операциях с файлами.

11.1.3.1. Поток управления и внутренние структуры данных, задействованные в операциях чтения и записи файлов

Операционная система Windows предоставляет четыре основные API-функции ядра для работы с файлами: ZwCreateFile, ZwReadFile, ZwWriteFile и ZwClose.

Во время операций с файлами задействуется несколько подсистем ядра ОС, таких как I/O Manager, Object Manager, SRM, драйвер NTFS и низкоуровневые драйверы, включая Disk Filter Driver и Disk Class Driver, которые обеспечивают доступ к физическим дискам [1, 312].

Уязвимость, выявленная в работе [303], заключается в том, что Security Reference Monitor проверяет права доступа к файлу только при вызове функции ZwCreateFile, тогда как функции ZwReadFile и ZwWriteFile не контролируются этой подсистемой.

При создании файла через вызов ZwCreateFile ОС создает дескриптор файла (handle), добавляет запись в таблицу дескрипторов (Handle Table), выделяет объект файла (FILE_OBJECT), данные NTFS и другие связанные структуры. Созданный де-

скриптор файла возвращается вызывающему драйверу и используется в функциях ZwReadFile и ZwWriteFile для операций чтения и записи в открытый файл.

На рис. 65 изображена последовательность обработки операций чтения и записи файлов в ядре ОС Windows, а также уязвимости, связанные с перехватом ряда структур данных.

Операции чтения/записи инициируются через API-функции ядра (ZwReadFile, ZwWriteFile). Управление передается через несколько компонентов ядра: менеджер ввода-вывода (I/O Manager), менеджер объектов (Object Manager), NTFS-драйвер и низкоуровневые драйверы диска.

Используя дескриптор файла, ОС просматривает таблицу дескрипторов, чтобы получить объект файла. Читая поля объекта файла, ОС находит структуры управляющих блоков (структуры данных NTFS) и переходит к ним. Дисковые драйверы получают доступ к файлу с помощью структур NTFS.

После успешного открытия файлового дескриптора все операции чтения и записи в файл обрабатываются ядром ОС без каких-либо проверок со стороны монитора безопасности (SRM).

Нарушители могут использовать эту уязвимость, подменяя следующие структуры:

- перехват таблицы дескрипторов (Handle Table Hijacking);
- перехват файлового объекта (Hijacking FILE_OBJECT Structure);
- перехват структур NTFS (Hijacking NTFS Data Structures).

На рис. 66 показано, что все попытки нарушителей получить доступ к файлу с помощью дескриптора перехватчика будут перенаправлены ОС на секретный файл. В результате нарушители получат доступ на чтение и запись в файл в обход средств защиты ОС.

Как следует из анализа рис. 66, нарушители могут изменить следующие данные, чтобы изменить поток управления (соответствующие атаки указаны в скобках):

- записи таблицы дескрипторов — handles (перехват таблицы дескрипторов);
- файловый объект (перехват структуры FILE_OBJECT);
- структуры данных NTFS (перехват структур NTFS).

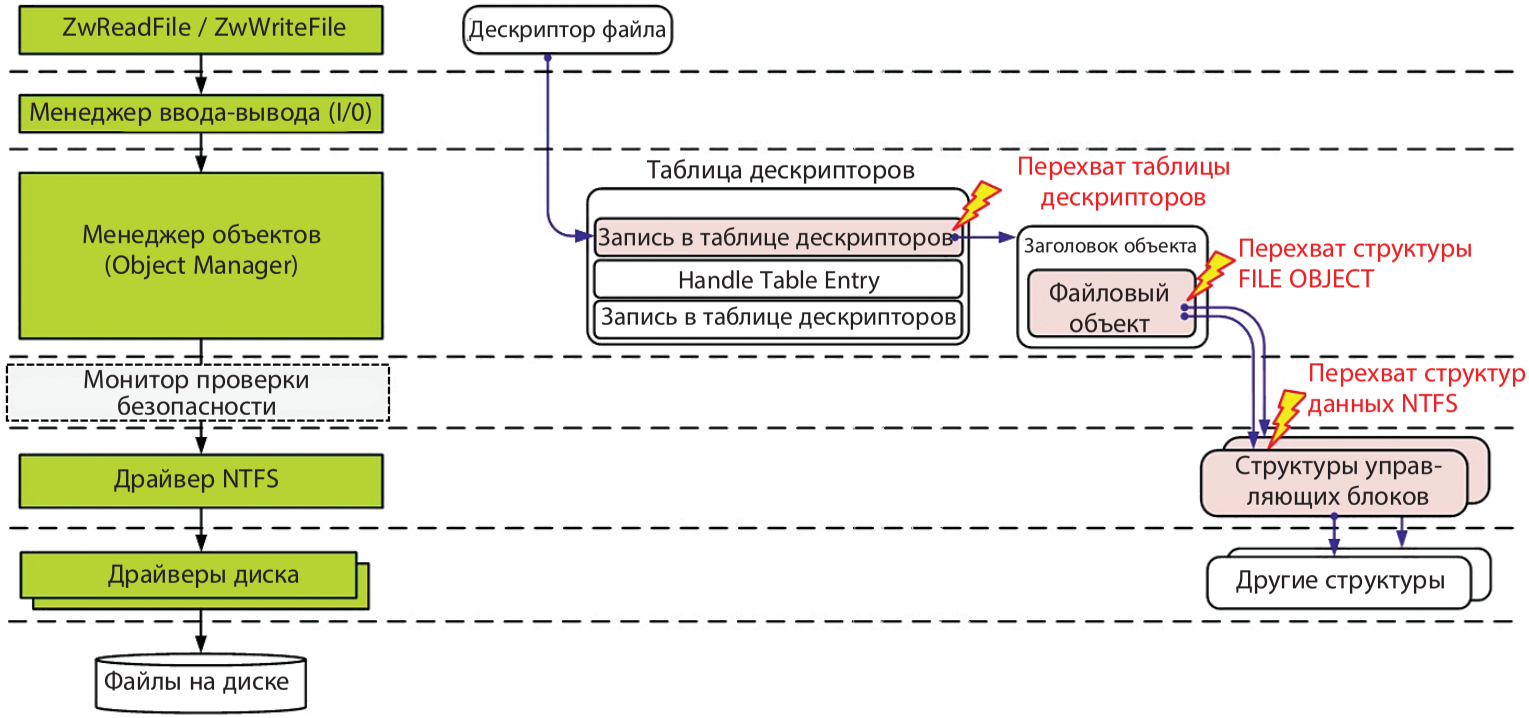


Рис. 65. Подсистемы ОС и соответствующие структуры данных, задействованные в операциях чтения и записи файлов

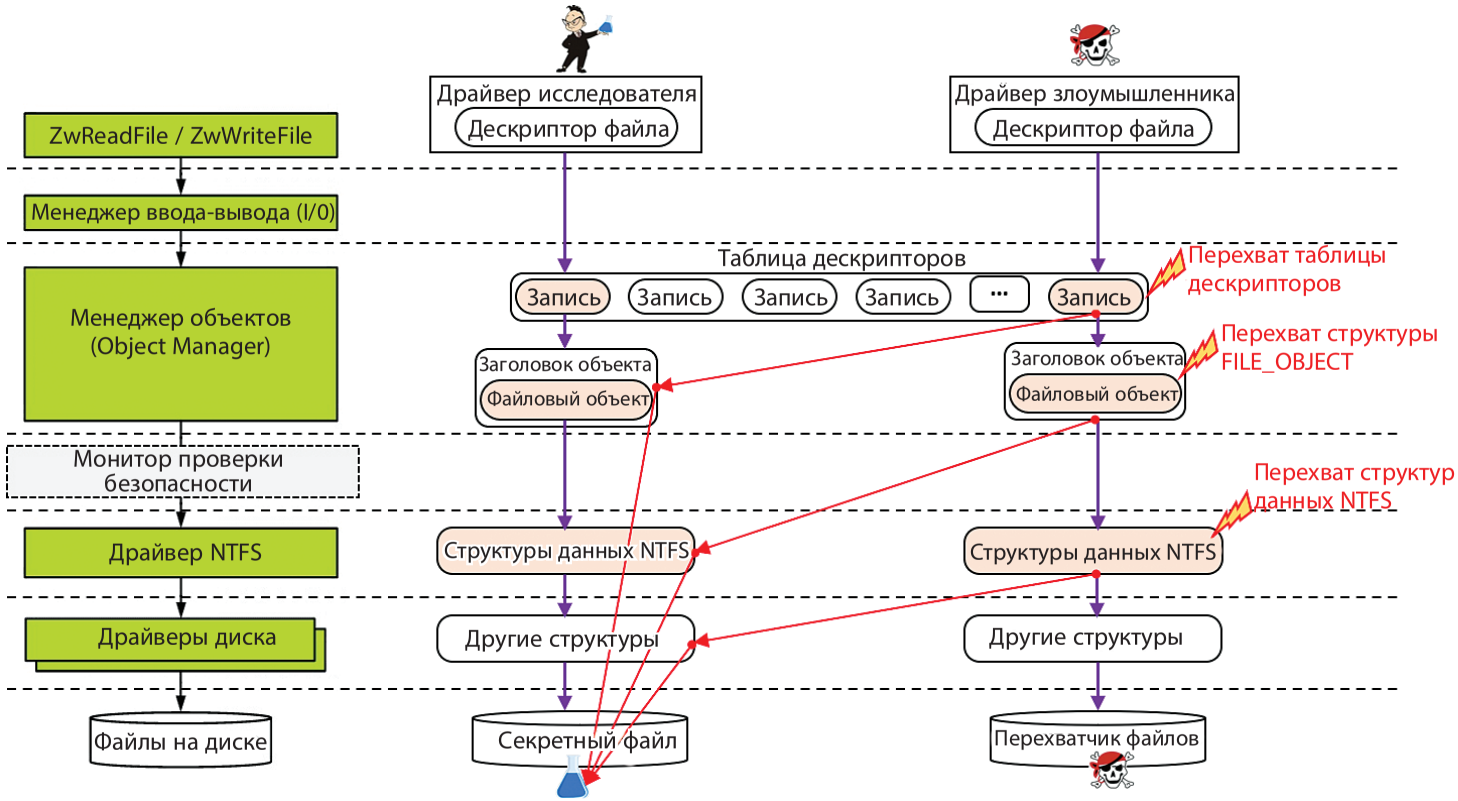


Рис. 66. Нарушитель создает перехватчик файлов и применяет три различные атаки, которые основаны на исправлении записей таблицы дескрипторов, файлового объекта и структур данных NTFS: 1) перехват таблицы дескрипторов (handles), 2) перехват структуры FILE_OBJECT, 3) перехват структур данных NTFS

Этот вид атаки File Object Hijacking был представлен в работе [303].

Далее будут описаны детали атак перехвата дескрипторов и перехвата NTFS-структур.

11.1.3.2. Перехват таблицы дескрипторов (Handle Table Hijacking)

Подробно рассмотрим структуру таблицы дескрипторов ядра (Kernel Handle Table) и механизм, посредством которого нарушители могут изменять ее значения для получения нелегального доступа к файлам.

Основы работы с таблицей дескрипторов. В ОС Windows часто возникает проблема безопасности, связанная с доступом к файлам, заблокированным одним приложением для другого. В работе [336] описан метод получения нелегального доступа к заблокированному файлу путем модификации таблицы дескрипторов, принадлежащей приложению. Подход автора был основан на работе с дескрипторами уровня пользователя (user-mode) в системах Windows XP и 2003. Опишем адаптацию этой идеи для модификации таблицы дескрипторов в ядре современных версий Windows.

Механизм изменения записей в таблице дескрипторов уровня пользователя для изменения прав доступа к дескриптору был реализован в проекте Blackbone [337].

Таблица дескрипторов ядра (Kernel Handle Table) используется Windows OS для отображения дескрипторов (handles) в соответствующие объектные структуры [312, 338, 339]. При обращении через дескриптор ОС использует Kernel Handle Table для получения доступа к объекту. Экспортируемый символ nt!ObpKernelHandleTable указывает на данную таблицу. Адрес этой таблицы можно также получить, прочитав поле EPROCESS.ObjectTable для процесса SYSTEM:4. Таблица дескрипторов ядра задействуется каждый раз при чтении или записи файла драйвером. Механизм дескрипторов управляет различными объектами, такими как файлы, процессы, потоки или ключи реестра.

Исследование, представленное ниже, сосредоточено на дескрипторах, связанных с файловой системой, но полученные результаты могут быть применены к различным объектам ядра.

Рассмотрим структуру таблицы дескрипторов ядра. Для каждого вновь созданного файла таблица дескрипторов включает запись, индекс которой соответствует возвращенному значению дескриптора [112]. Каждая запись определяется структурой HANDLE_TABLE_ENTRY, которая содержит:

- GrantedAccessBits — права доступа, предоставленные к объекту;
- ObjectPointerBits — ссылка на созданный объект.

Поле ObjectPointerBits включает 44 младших бита адреса OBJECT_HEADER, что позволяет получить адрес структуры FILE_OBJECT [340, 341].

В работе [312] отмечается, что системные вызовы, такие как ZwReadFile и ZwWriteFile, используют таблицу дескрипторов ядра, созданную менеджером объектов (Object Manager), чтобы преобразовать дескриптор в ссылку на базовый объект, например FILE_OBJECT, содержащий данные, необходимые для дальнейшего выполнения системного вызова.

Таблица дескрипторов может иметь несколько уровней. Количество уровней и записей в каждом уровне зависит от версии Windows [338, 339, 342]. ОС Windows предоставляет функцию ExEnumHandleTable для перечисления всех дескрипторов в таблице [343]. Эта функция использует callback-функцию для обработки каждого дескриптора.

Нарушители могут применять ExEnumHandleTable для доступа к записям таблицы дескрипторов. Процесс перечисления также требует использования блокировок для каждого дескриптора через вызов ExUnlockHandleTableEntry [344, 345]. Такой подход позволяет несанкционированно вносить изменения в содержимое записей таблицы дескрипторов без ответной реакции со стороны штатных средств защиты ОС.

Алгоритм перехвата записей таблицы дескрипторов. Исследование показывает, что во время операций чтения и записи ОС проходит через таблицу дескрипторов ядра и получает доступ к объекту файла без выполнения каких-либо проверок. Нару-

шители могут использовать эту уязвимость для получения нелегального доступа к файлу, открытому в эксклюзивном режиме, следующим образом:

1) определить адрес структуры OBJECT_HEADER, связанной с целевым файлом (секретным файлом);

2) создать файл-перехватчик (file hijacker) и найти соответствующую запись в таблице дескрипторов;

3) перезаписать поле ObjectPointerBits в данной записи, используя адрес OBJECT_HEADER секретного файла.

После выполнения данной атаки все операции чтения и записи, осуществляемые через подмененный дескриптор файла, будут перенаправляться на секретный файл. Этот процесс будет осуществляться автоматически операционной системой Windows, как показано на рис. 67.

Для выполнения атаки на таблицу дескрипторов необходимо перезаписать всего лишь 44 бита динамически выделяемых данных, что достаточно для получения нелегального доступа к файлу, открытому в эксклюзивном режиме. Перенаправление осуществляется прозрачно для всех встроенных механизмов защиты Windows.

С одной стороны, нарушители могут модифицировать записи таблицы дескрипторов ядра, чтобы воспользоваться механизмом трансляции дескрипторов, а с другой — встроенные функции безопасности Windows не проверяют целостность данной таблицы и, следовательно, не способны обнаружить эту атаку.

11.1.4. Перехват структур данных NTFS

Принцип атаки на структуру NTFS. Опишем атаку, известную как «Перехват структур данных NTFS». Эта атака является усовершенствованием метода, предложенного в работе [303], основанного на перехвате структуры FILE_OBJECT. Предположим, что нарушители не могут использовать ранее представленную атаку на перехват структуры FILE_OBJECT и вынуждены разрабатывать новую атаку. Тогда они переходят на более низкий уровень и атакуют управляющие блоки или структуры данных NTFS, на которые указывают поля в структуре FILE_OBJECT (см. рис. 67, *б*; рис. 68).

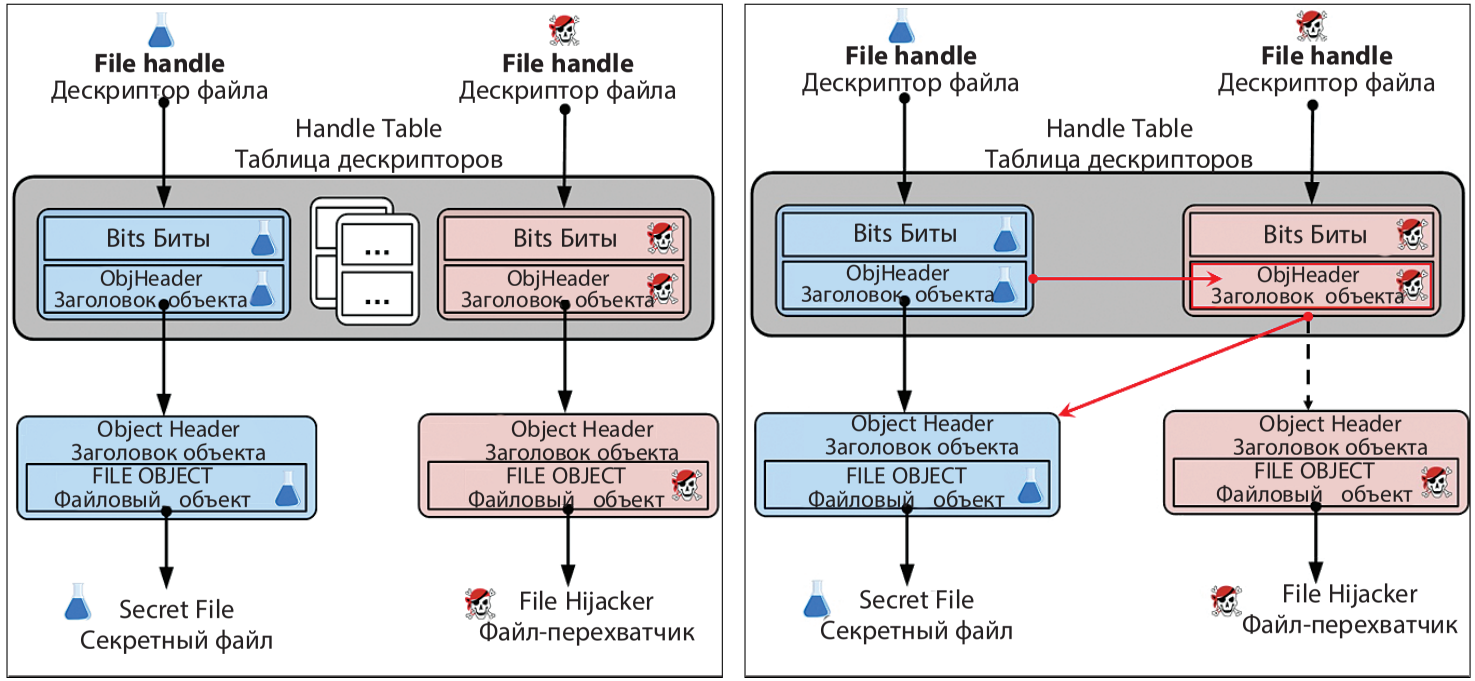


Рис. 67. Поток управления между файловыми структурами до перехвата записей таблицы дескрипторов (handles) (*а*) и после него (*б*)

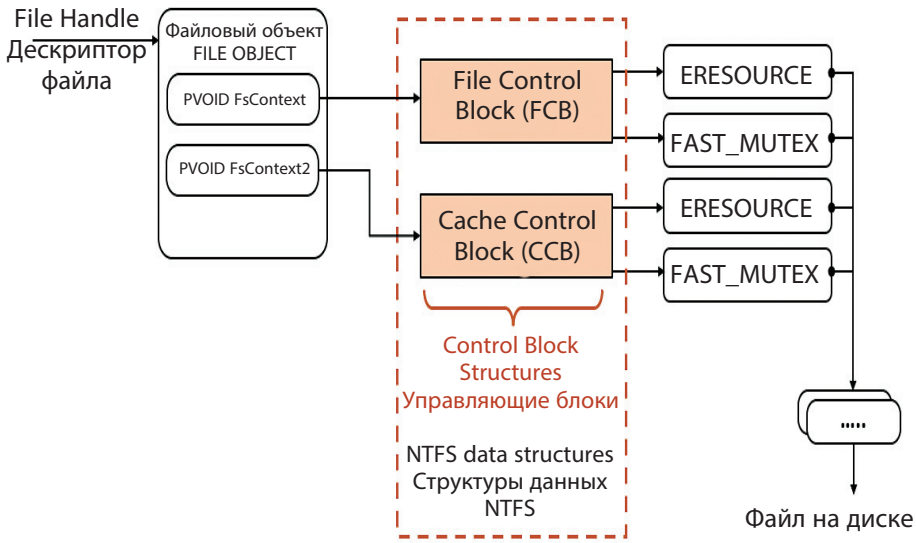


Рис. 68. Структура управления доступом к файлам в NTFS

Структура файлового объекта (FILE_OBJECT) включает ряд полей, два из которых FsContext и FsContext2 указывают на управляющие блоки:

1) FsContext указывает на блок управления файлом (File Control Block, FCB) или блок управления потоком (Stream Control Block, SCB);

2) FsContext2 указывает на блок управления контекстом (Context Control Block, CCB).

Каждый файловый поток в памяти представлен структурой FCB, а структура CCB создается файловыми системными драйверами для представления открытого экземпляра потока файла [315]. Эти механизмы были интегрированы в ядро Windows начиная с Windows NT.

Поле FsContext используется для представления физического контекста потока, а поле FsContext2 отвечает за пользовательский контекст потока. Поле FsContext2 часто указывает на структуру управления каналом (Channel Control Block, CCB) [318]. Механизмы синхронизации FAST_MUTEX и ERESOURCE обеспечивают безопасный совместный доступ нескольких потоков.

Структуры FCB и CCB, на которые указывают поля FsContext и FsContext2, частично документированы. Однако проведенное

исследование выявило следующие детали: определения SCB, FCB и CCB для Windows NT 4.0 находятся в файле ntfsstru.h [346], определения этих структур также можно найти в файле cdstruc.h из проекта [344].

Эти определения полезны для понимания базовых принципов работы файловой системы, хотя структуры были частично обновлены в недавних версиях Windows. Согласно документации MSDN поле FsContext объекта файла содержит указатель на структуру FSRTL_ADVANCED_FCB_HEADER, которая уникально идентифицирует файловый поток в файловой системе [347–349]. Исследование показало, что структура FSRTL_ADVANCED_FCB_HEADER, на которую указывает поле FsContext, является подходящей целью для атаки. Поля FsContext и FsContext2 обычно указывают на смежные блоки памяти, что позволяет нарушителям одновременно копировать и перезаписывать их содержимое в ходе атаки.

Эти структуры не защищены встроенными механизмами защиты ОС, а их модификация не вызывает срабатывания проверки безопасности ядра с последующим «синим экраном смерти» (BSOD).

Атака на структуры данных NTFS основана на поиске внутренних структур файловых объектов по полям FsContext и FsContext2, копировании их содержимого в соответствующие области памяти, на которые указывают поля перехватчика файловых объектов, и дополнительной модификации структур перед обращением. Без последней модификации во время операций чтения или записи ОС обнаруживает несанкционированный доступ и вызывает BSOD с кодом ошибки RESOURCE_NOT_OWNED (0x000000E3).

В результате этой атаки нарушители получают возможность незаконного доступа к файлам, используя уязвимость в защите управляющих блоков NTFS, что остается незамеченным встроенными средствами безопасности ядра.

Алгоритм перехвата данных NTFS. Для осуществления атаки «Перехват структур данных NTFS» нарушители должны найти соответствующие структуры NTFS для секретного файла и файла-перехватчика, а затем выполнить следующие три шага.

Шаг 1. Перезапись структуры FSRTL_ADVANCED_FCB_HEADER нарушителей. Сначала содержимое структуры FSRTL_ADVANCED_FCB_HEADER, принадлежащей нарушителям, заменяется данными из структуры FSRTL_ADVANCED_FCB_HEADER, связанной с секретным файлом. Однако этого недостаточно, так как ядро Windows обнаруживает, что поток драйвера пытается освободить ресурс, которому он не принадлежит. В этом случае система вызывает BSOD с кодом ошибки RESOURCE_NOT_OWNED. Чтобы обойти эту проблему, нарушители переходят ко второму шагу.

Шаг 2. Модификация идентификатора потока. Нарушители устанавливают идентификатор потока, полученный через функцию PsGetCurrentThread, в следующие поля структуры FSRTL_ADVANCED_FCB_HEADER:

- Resource → OwnerEntry. OwnerThread,
- PagingIoResource → OwnerEntry. OwnerThread.

Такая модификация позволяет драйверу ВПО избежать BSOD с кодом RESOURCE_NOT_OWNED.

Шаг 3. Повторение шагов 1 и 2 каждый раз перед получением доступа к секретному файлу с помощью дескриптора файла-перехватчика. Поскольку ядро Windows изменяет содержимое структуры FSRTL_ADVANCED_FCB_HEADER при каждом возврате результата чтения или записи драйверу, ранее модифицированная структура снова становится недействительной. Если нарушители хотят многократно обращаться к секретному файлу, то они должны повторять шаги 1 и 2 перед каждым несанкционированным запросом на чтение или запись. В результате каждый раз, когда драйвер ВПО обращается к файлу через перехваченный файловый дескриптор, ядро Windows проходит через измененные структуры и предоставляет несанкционированный доступ к секретному файлу без ошибок или сбоев системы.

Этот метод демонстрирует, как нарушители могут использовать уязвимости ядра Windows для обхода встроенных механизмов безопасности, манипулируя внутренними структурами NTFS. На рис. 69 представлены этапы перехвата структур блоков управления NTFS. В состоянии «до перехвата» (см. рис. 69, *а*) доступ к секретному файлу защищен, в то время как в состоянии

«после перехвата» (см. рис. 69, *б*) нарушитель копирует и модифицирует связанные структуры, перенаправляя операции ввода/вывода для несанкционированного чтения секретного файла.

До перехвата структур NTFS (см. рис. 69, *а*) FILE_OBJECT для секретного файла содержит ссылки на связанные структуры блоков управления (Control Block Structures), включая FAST_MUTEX и ERESOURCE. Эти структуры обеспечивают управление доступом и синхронизацией операций с файлом. FILE_OBJECT для перехватчика файла изолирован и не имеет доступа к структурам, связанным с секретным файлом. Таким образом, каждая структура функционирует независимо, а секретный файл защищен от несанкционированного доступа.

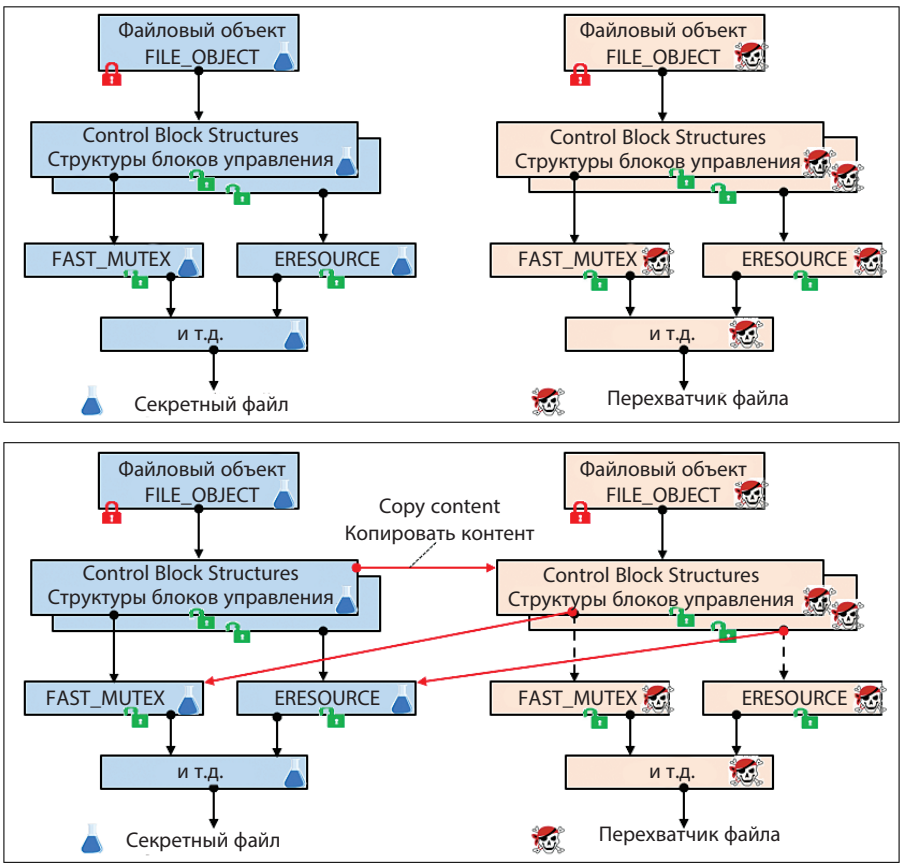


Рис. 69. Поток управления между структурами файлов при перехвате структур блоков управления NTFS, например, FSRTL_ADVANCED_FCB_HEADER до перехвата (левая часть) и после него (правая часть)

После перехвата структур NTFS (см. рис. 69, *б*) содержимое структур блоков управления секретного файла (Control Block Structures) копируется в структуры, связанные с перехватчиком файла. Поля ссылок, такие как FsContext и FsContext2, в FILE_OBJECT перехватчика файла модифицируются для перенаправления к ранее скопированным структурам. Поэтому после модификации обращения к перехватчику файла дают доступ к содержимому секретного файла. Этот процесс позволяет обойти штатные механизмы защиты Windows и получить несанкционированный доступ к защищенному файлу.

11.1.5. Перехват токена доступа

Модель управления привилегиями процессов играет важную роль в обеспечении безопасности ОС. Опишем механизм управления привилегиями процессов и техники, с помощью которых нарушители могут использовать динамически выделяемую память для реализации атак.

Токен (token) — это специальный объект или структура данных в операционных системах, которая содержит информацию о правах доступа пользователя или процесса. Токены используются для контроля доступа и обеспечения безопасности при взаимодействии процессов и объектов в системе.

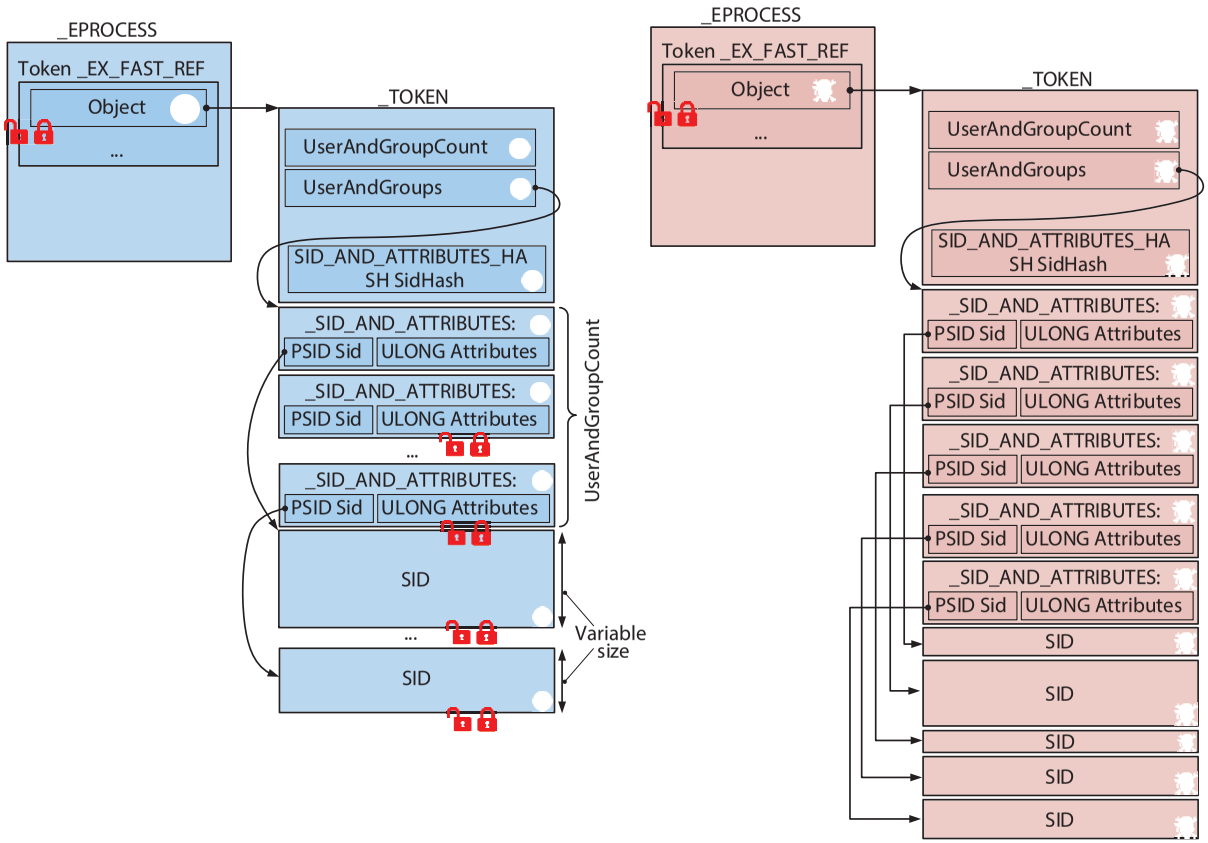
Каждый процесс в Windows OS имеет структуру _EPROCESS, в которой хранятся внутренние данные о процессе: его имя, идентификатор, информация о потоках и дескрипторах [312, 340]. Структура _TOKEN, связанная с процессом, содержит следующую информацию: идентификаторы пользователей и групп (SID), привилегии процесса и другие данные (рис. 70) [111].

11.1.5.1. Дискреционный механизм контроля доступа

Операционная система Windows реализует дискреционный механизм контроля доступа, который включает два основных элемента [350, 351]:

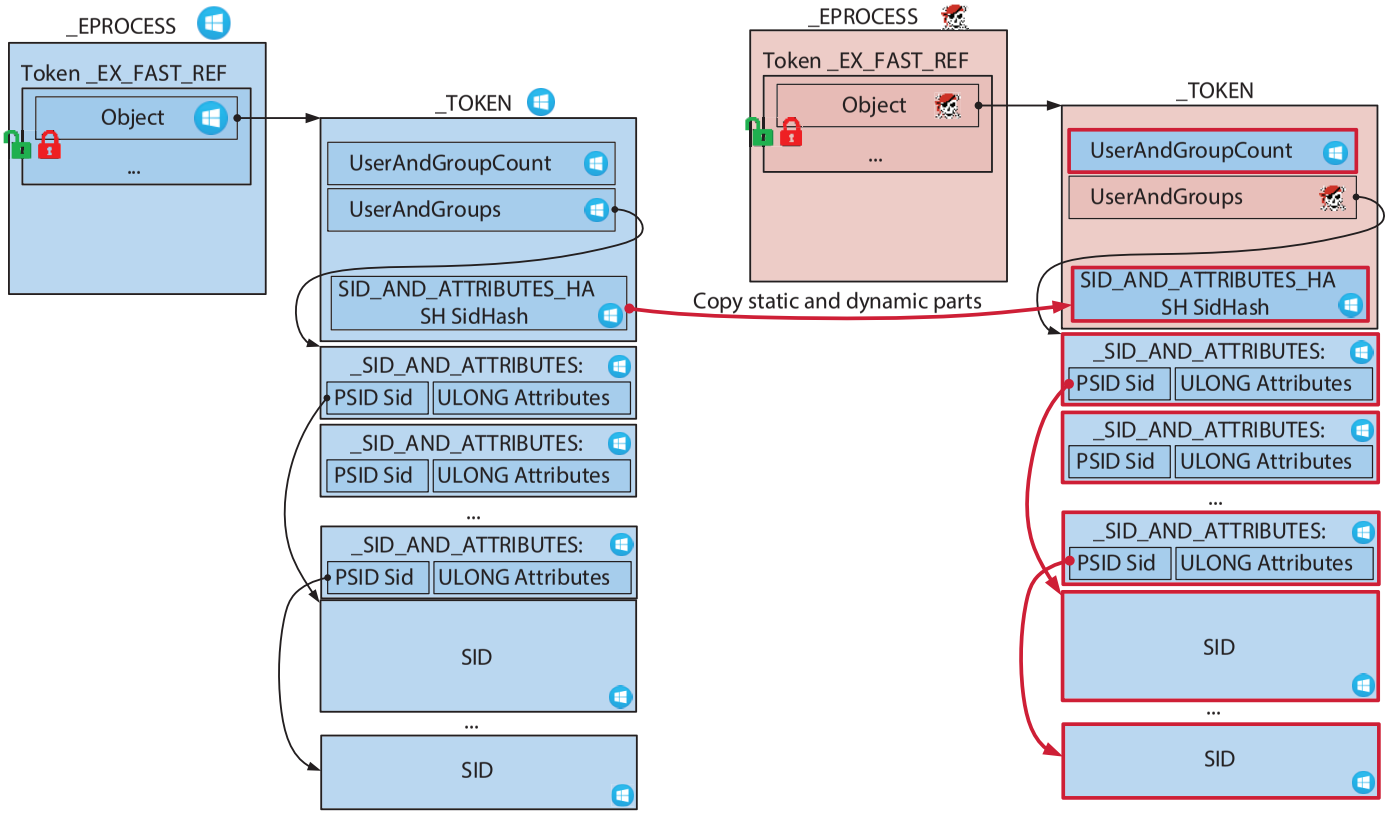
1) токен доступа, связанный с каждым процессом;

2) дескриптор безопасности, связанный с каждым объектом (например, файлом).



а

Рис. 70. Содержимое структур _EPROCESS и _TOKEN для SYSTEM:4 и процессов ВПО до перехвата токенов (*а*) и после него (*б*) (начало)



б

Рис. 70. Содержимое структур _EPROCESS и _TOKEN для SYSTEM:4 и процессов ВПО до перехвата токенов (*а*) и после него (*б*) (окончание)

Каждый раз, когда процесс пытается получить доступ к объекту, монитор безопасности проверяет SIDs и группы SIDs из структуры _TOKEN, чтобы определить, разрешен ли данный доступ [352, 353].

Нарушители используют различные техники для повышения привилегий вредоносного процесса. Метод на основе API предложен в работе [354] и использует несколько API-функций ядра.

Прямые манипуляции объектами ядра (DKOM) описаны в работе [111]. Авторы предложили изменить структуру Token путем модификации массива UserAndGroups, чтобы активировать требуемые привилегии.

С развитием средств защиты в ОС Windows были добавлены такие поля, как SidHash и RestrictedSidHash, для обеспечения целостности структуры _TOKEN. Эти поля проверяются системой, чтобы убедиться, что список SIDs не был изменен в обход механизмов защиты.

В работе [246] предложили три альтернативных метода для обхода защиты на основе хэш-значений. Одним из таких методов является кража или подмена токена, которая осуществляется путем изменения значения поля Object в структуре _EPROCESS процесса, выполняющего ВПО. В этом случае используется значение из структуры _EPROCESS, принадлежащей более привилегированному процессу, например SYSTEM:4. Однако современные версии антивируса Microsoft Windows Defender успешно обнаруживают такие атаки благодаря мониторингу попыток подмены токенов [355, 356]. Кроме того, в работе [357] показано, что манипуляции с токенами могут быть выявлены с использованием инструмента auditpol, основанного на механизме ETW. Однако данный метод обнаружения может быть также уязвим к атакам, направленным на сам механизм ETW.

11.1.5.2. Новая атака перехвата токенов

Рассмотрим новую атаку на структуры токенов, основанную на идеях, предложенных в книге [111]. Основная цель — модифицировать структуру _TOKEN так, чтобы пересчитанное значение SidHash оставалось корректным, а проверка целостности не выявляла внесенных изменений.

Нарушители могут достичь этого, перезаписывая три поля структуры _TOKEN, используя данные из процесса с более высокими привилегиями:

1) UserAndGroupCount;
2) массив UserAndGroups: поля Attributes и Sid;
3) структура SidHash.

Ключевая особенность заключается в полном копировании массива UserAndGroups с обновленной внутренней организацией структуры, в то время как ранее в работе [111] предлагалось модифицировать только отдельные поля, что недостаточно для успешного прохождения проверки целостности в современных версиях Windows ОС.

Поля UserAndGroupCount и SidHash имеют одинаковый размер, поэтому их изменение тривиально. Однако переменная часть массива UserAndGroups имеет различное количество записей и размеры структур SID для разных процессов (см. рис. 70).

Атака становится возможной благодаря тому, что структура _TOKEN для большинства процессов обычно содержит достаточно места для хранения переменной части. Одновременно с этим соответствующая структура процесса System:4 по размеру оказывается значительно меньше, однако имеет достаточно высокие привилегии для выполнения задач нарушителей.

Эта атака была успешно протестирована на Windows 10 версии 1903 x64. Исходный код и демонстрация доступны в исследовании [306].

Описанная атака демонстрирует, как нарушители могут использовать недостатки в обработке структуры _TOKEN для повышения привилегий вредоносного процесса. Новая атака является эволюцией предыдущих техник и обходит современные механизмы защиты Windows.

11.2. МОДИФИКАЦИЯ ГИПЕРВИЗОРА MEMORYRANGER ДЛЯ ЗАЩИТЫ СТРУКТУР ЯДРА

11.2.1. Краткий обзор MemoryRanger

MemoryRanger — это гипервизор первого типа (bare-metal hypervisor), предназначеный для обеспечения целостности и кон-

фиденциальности кода и данных в режиме ядра. MemoryRanger использует технологии Intel VT-x и расширенные таблицы страниц (EPT) для изоляции памяти. Основной механизм работы заключается в создании изолированных анклавов ядра, которые имеют ограниченный доступ к памяти. Это обеспечивает защиту кода и данных ядра от модификации со стороны вновь загруженных драйверов и защищает сами драйверы друг от друга [302, 303].

После загрузки гипервизор MemoryRanger создает основную область памяти, где продолжают работать ядро ОС и все драйверы, загруженные до его запуска. Все последующие драйверы, загружаемые в систему, MemoryRanger перехватывает и помещает в отдельные изолированные области памяти (анклавы), динамически выделяя для каждого драйвера собственный защищенный контекст памяти. Для этого MemoryRanger использует аппаратную виртуализацию, обеспечивая изоляцию между драйверами. В реальном времени гипервизор обновляет ограничения доступа к памяти для каждого анклава, что позволяет гибко настраивать правила доступа. При попытке несанкционированного доступа к защищенной области памяти MemoryRanger перенаправляет такие обращения на специально созданную подмененную страницу памяти, предотвращая утечки и повреждение данных.

11.2.2. Основные компоненты MemoryRanger

MemoryRanger состоит из следующих компонентов [303]:

- драйвер режима ядра — основной модуль для запуска гипервизора и работы с функциями ОС;
- DdiMon — компонент для перехвата и мониторинга системных вызовов ядра;
- MemoryMonRWX — модуль для реализации управления доступом к памяти с возможностью контроля до одного байта (бита) памяти;
- Memory Access Policy (MAP) — механизм динамического управления политиками доступа к памяти.

MemoryRanger выполняет следующие функции:

- регистрация функции обратного вызова (callback) для получения уведомления о создании (завершении) процессов и загрузке (выгрузке) драйверов;

- перехват API ядра через компонент DdiMon, что позволяет контролировать вызовы системных функций;
- динамическое обновление правил доступа к памяти через компонент MAP.

Основные изменения для предотвращения атак на структуры ядра. Для предотвращения атак на структуры ядра MemoryRanger был дополнен новыми возможностями, направленными на защиту важных объектов ядра [303].

1. *Предотвращение атак на файловые объекты и NTFS-структуры*. MemoryRanger перехватывает системные вызовы ZwCreateFile() и ZwClose() для отслеживания данных о создаваемых файлах и ограничения доступа к ним.

2. *Блокировка подмены объектов HANDLE*. MemoryRanger использует функцию ExEnumHandleTable, чтобы найти и защитить структуру HANDLE_TABLE_ENTRY, связанную с открываемым файлом.

3. *Защита структур NTFS*. Для защиты файловой системы MemoryRanger ограничивает доступ к структурам FSRTL_ADVANCED_FCB_HEADER, которые участвуют в управлении файловыми объектами.

4. *Предотвращение перехвата токенов*. MemoryRanger реализует новую технику, основанную на выделении специального изолированного анклава, который включает только защищаемые структуры ядра (рис. 71) [322].

11.3. ПРАКТИЧЕСКОЕ ПРИМЕНЕНИЕ МОДИФИЦИРОВАННОГО MEMORYRANGER: ПРЕДОТВРАЩЕНИЕ АТАК НА ФАЙЛОВЫЕ ОБЪЕКТЫ И ТОКЕНЫ ПРОЦЕССОВ

11.3.1. Предотвращение атак перехвата дескрипторов

Подмена дескрипторов (Handle Hijacking) позволяет нарушителям манипулировать файловыми дескрипторами для получения несанкционированного доступа к файловым объектам. MemoryRanger предотвращает эту атаку, блокируя только доступ на запись к полю ObjHeader, которое занимает 6 байт и соответствует заголовку объекта файла (см. рис. 71) [303]. MemoryRanger не ограничивает доступ на чтение к полю ObjHeader и не пре-

пятствует доступу к другим полям этой записи, поскольку они используются операционной системой.

MemoryRanger получает уведомления о создании нового файла путем перехвата вызова функции ZwCreateFile. После этого MemoryRanger определяет запись в таблице дескрипторов, используя полученный дескриптор файла и функцию ExEnumHandleTable.

11.3.2. Предотвращение атак на структуры NTFS

Нарушители могут быть нацелены на структуры управления файловой системой NTFS, такие как FSRTL_ADVANCED_FCB_HEADER. Эти структуры используются файловой системой для управления файлами и каталогами. Атака на них позволяет перехватывать управление над файлами [306].

Для предотвращения этой атаки MemoryRanger реализует аналогичный подход, основанный на обнаружении структур управляющих блоков и ограничении доступа к ним.

11.3.3. Предотвращение атак на токены доступа

Подмена токенов (Token Hijacking) представляет собой атаку, при которой нарушители модифицируют статические и динамические части структур _TOKEN, что приводит к эскалации привилегий. MemoryRanger предотвращает эту атаку, выделяя специальный изолированный анклав, в котором размещаются только защищаемые данные ядра, включая структуры _TOKEN, элементы ядра ОС ntoskrnl.exe, необходимые для взаимодействия с защищаемыми структурами.

Драйверы, загруженные после старта MemoryRanger, не имеют доступа к содержимому этого изолированного анклава. MemoryRanger регистрирует функцию обратного вызова с использованием PsSetCreateProcessNotifyRoutineEx, чтобы отслеживать создание новых процессов и обновлять политику доступа в реальном времени. Схема изоляции токенов изображена на рис. 71.

11.3.4. Эмпирическая проверка и тестирование

Предложенные механизмы защиты, реализованные в Memory Ranger, прошли тестирование на Windows 10 x64 (версия 1903). Все исследованные атаки, включая Handle Hijacking, NTFS

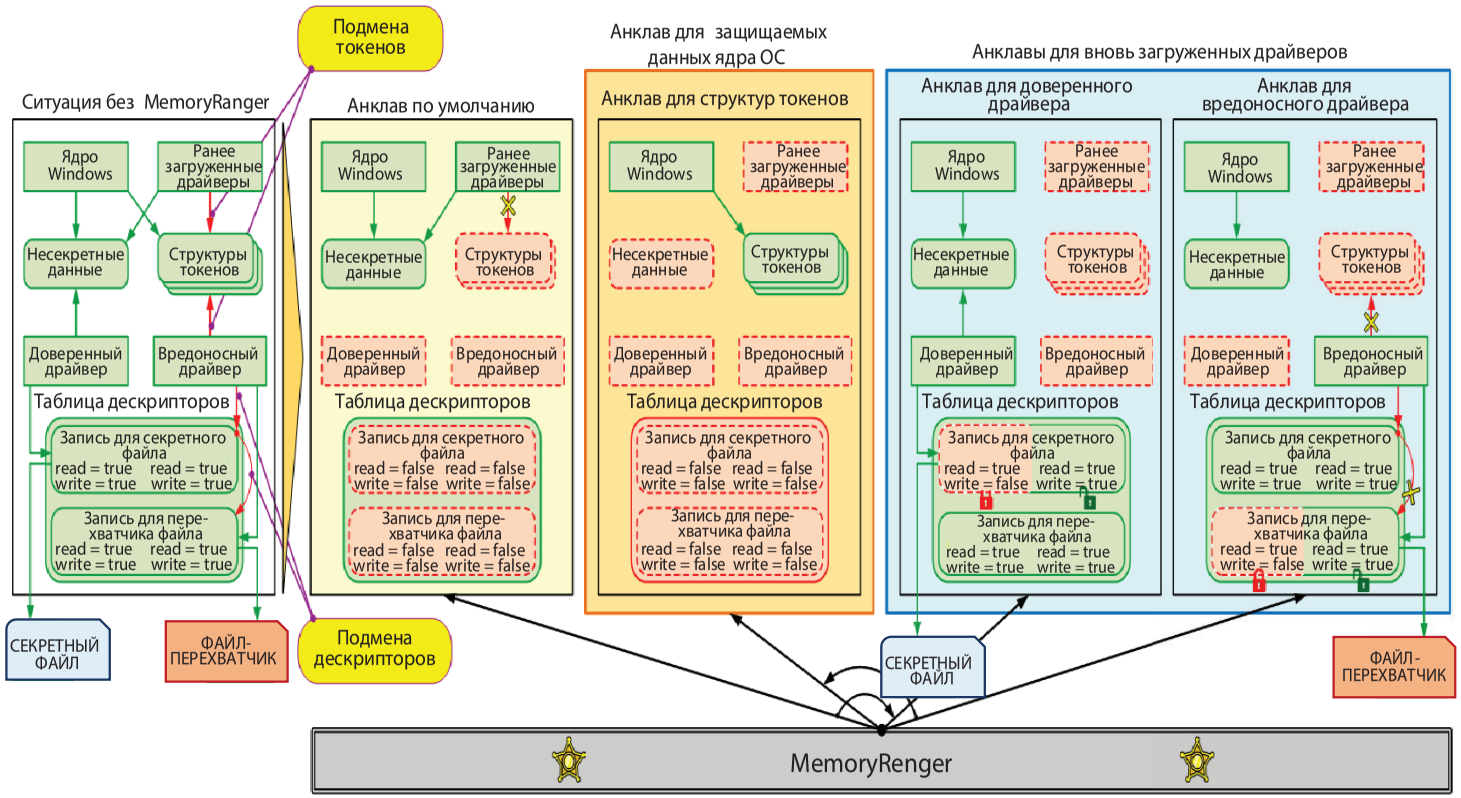


Рис. 71. MemoryRanger предотвращает атаки перехвата дескрипторов (handles) и токенов

Hijacking и Token Hijacking, были успешно предотвращены. Подробности экспериментов описаны в работе [306].

Для моделирования атак на структуры памяти ядра был создан экспериментальный драйвер. Он был подписан легитимной цифровой подписью и запущен в системе с активным PatchGuard без режима драйверной отладки. В течение 10 ч после внесения изменений в память ядра с помощью драйвера не наблюдалось каких-либо реакций штатной защиты (например, BSOD со стороны PatchGuard). Это подтверждает ключевую роль MemoryRanger в успешном предотвращении атак на уровне гипервизора и демонстрирует, что штатные механизмы самозащиты ядра ОС контролируют лишь ограниченный круг системных объектов.

11.3.5. Производительность MemoryRanger

MemoryRanger вызывает незначительное снижение производительности из-за переключения между анклавами ядра, что приводит к очистке и повторному заполнению TLB, однако новая версия поддерживает VPID (Virtual Processor Identifier), позволяя избежать этих задержек [302].

11.3.6. Сравнение MemoryRanger с Virtual Secure Mode

Операционная система Windows включает механизм VSM для защиты памяти. VSM является частным случаем защиты на основе анклавов и предоставляет защиту с использованием двух изолированных областей памяти VTL0 и VTL1, тогда как MemoryRanger поддерживает произвольное количество анклавов памяти. В тестах MemoryRanger успешно продемонстрировал защиту с тремя, четырьмя и пятью анклавами [302, 303]. MemoryRanger может быть расширен и адаптирован для различных сценариев использования. Алгоритмы гипервизора MemoryRanger могут применяться для защиты систем на базе Unix, работающих на процессорах *AMD* и *ARM*.

Глава 12
ПРОЕКТ MEMORYRANGER: ПРЕДОТВРАЩЕНИЕ АТАК НА ЗАЩИТНЫЙ МЕХАНИЗМ PROTECTED PROCESS LIGHT СО СТОРОНЫ ДРАЙВЕРОВ WINDOWS

С выходом Windows 8.1 в 2013 г. был добавлен новый механизм Protected Process Light (PPL), обеспечивающий защиту системных процессов и поддержку механизмов управления цифровыми правами (Digital Rights Management, DRM). PPL стал развитием концепции Protected Process, которая впервые появилась в Windows Vista для защиты аудиопотоков DRM. В отличие от Protected Process PPL допускает создание защищенных процессов не только для мультимедийных приложений, но и для различных системных служб, а также процессов и сервисов антивирусных систем защиты.

Однако нарушители могут использовать возможности драйвера ядра для несанкционированного вмешательства в работу механизма PPL путем модификации значений в динамически выделенной памяти (содержимое структур EPROCESS), например, в целях отключения PPL для защищенных процессов для беспрепятственного доступа к содержимому их памяти, а также для обеспечения самозащиты самих вредоносных процессов, препятствуя антивирусным решениям в их анализе. Система PatchGuard не контролирует целостность соответствующих полей PPL, что создает серьезную уязвимость в безопасности ОС.

Проанализируем внутренние механизмы работы PPL при создании и доступе к защищенным процессам, а также рассмотрим модифицированное гипервизорное решение MemoryRanger для предотвращения атак на PPL, которое было протестировано на Windows 10 версии 20H2 Build 19042.631 x64.

12.1. АНАЛИЗ АРХИТЕКТУРНЫХ УЯЗВИМОСТЕЙ МЕХАНИЗМА PROTECTED PROCESS LIGHT (PPL): ИССЛЕДОВАНИЕ ВОЗМОЖНОСТЕЙ НЕСАНКЦИОНИРОВАННОГО ОТКЛЮЧЕНИЯ И ПОВЫШЕНИЯ УРОВНЯ ЗАЩИТЫ ПРОЦЕССОВ

12.1.1. Защита памяти процессов: модель PPL, методы атаки и защиты

Защита памяти процессов имеет большое значение для ряда областей, включая управление цифровыми правами (DRM) [358], игровую индустрию [359], антивирусные технологии [360] и защиту учетных записей [361].

Для усиления защиты памяти системных процессов в Windows был добавлен механизм Protected Process (PP), обеспечивающий ограничение доступа к памяти защищенных процессов [1]. Protected Process Light является расширением этой модели, предоставляя дополнительные уровни защиты за счет новых двух параметров — Protection Type и Signer [312].

Механизм PPL использует содержимое поля PS_PROTECTION, добавленного в структуру EPROCESS. Содержимое этого поля устанавливается для процессов PPL и используется монитором безопасности при проверке прав доступа. Драйвер ВПО может несанкционированно изменить это поле, воздействуя на уровень защиты данного процесса.

Обеспечение целостности PPL в течение всего времени работы системы крайне важно, поскольку нарушители могут как отключить PPL для системных процессов, так и, наоборот, повысить PPL с целью защиты вредоносных процессов [360, 362]. Для выполнения таких манипуляций нарушители могут использовать различные техники: включая загрузку собственных драйверов ядра [363], эксплуатацию уязвимостей подписанных драйверов (например, драйвера MSI [364], уязвимость в CPU-Z [365], эксплойты для драйверов Gigabyte [366]), загрузку драйверов вредоносных программ [367] или эксплуатацию уязвимостей на уровне пользовательского режима [362]. Однако в новых версиях Windows эта уязвимость исправлена.

Примером инструмента для атаки на PPL является Mimikatz. Этот инструмент может отключать PPL путем изменения байта

PS_PROTECTION в структуре EPROCESS [360]. Это позволяет Mimikatz получить доступ к памяти защищенных процессов, таких как Local Security Authority Subsystem Service (LSASS) [361]. После отключения PPL нарушители могут извлекать хэши паролей из памяти LSASS и использовать их для атак методом перебора хэшей с помощью утилиты hashcat [368].

Microsoft ввела Microsoft Defender Credential Guard (WDCG) для предотвращения извлечения учетных данных из LSASS [369]. Однако исследовательские работы показывают, что при некоторых обстоятельствах Credential Guard не может быть активирован и в этих случаях нарушители все еще могут извлекать пароли.

Для защиты от атак на PPL Microsoft реализует Protected Environment Authentication and Authorization Export Driver (PEAuth) [370]. Однако эти меры оказались недостаточными [371]. Для защиты от подобных атак предложено гипервизорное решение MemoryRanger [305, 372], которое предотвращает атаки на PS_PROTECTION путем изоляции загружаемых драйверов в отдельные защищенные области памяти (анклавы), используя технологии виртуализации VT-x и EPT.

12.1.2. Внутреннее устройство PPL

Опишем механизмы PPL, обновления в структуре EPROCESS и работу функции OpenProcess. Также обсудим уязвимость PPL и причины, по которым данный механизм не обеспечивает ожидаемой защиты.

12.1.2.1. Введение в механизм PPL

Protected Process был разработан для защиты DRM-контента от несанкционированного чтения и записи со стороны других процессов. В механизме PPL можно выделить три элемента: тип защиты, подпись и аудит [1, 373], подробно описанные в работе [393]. PPL применим к ОС-приложениям и антивирусным решениям [375–378]. Антивирусный механизм раннего запуска (ELAM, Early Launch Anti-Malware) использует возможности механизма PPL в современных версиях Windows [379]. Защищенные PPL-процессы недоступны для управления, внедрения

кода или остановки со стороны обычных пользователей и даже большинства процессов с административными правами. PPL ограничивает незащищенные процессы в возможностях внедрения потоков, записи в память и отладки защищенных процессов (исключение — отладка на уровне ядра). Загружаемые DLL в PPL-процессы должны быть подписаны соответствующими сертификатами. Secure ETW предоставляет события, доступные только защищенным процессам [380].

12.1.2.2. Активация и проверка PPL

PPL активируется автоматически при запуске процесса, но, например, для защиты процесса LSASS (Local Security Authority Subsystem Service) требуются дополнительные настройки [381]. Проверка текущего уровня PPL защиты доступна через API-функции [382–384]:

- ZwQueryInformationProcess (ProcessProtectionInformation);
- PsGetProcessProtection;
- PsIsProtectedProcess;
- PsIsProtectedProcessLight.

12.1.2.3. Внутреннее устройство PPL (PPL Internals)

В EPROCESS добавлено поле Protection (PS_PROTECTION), указывающее на статус PPL [383]. На рис. 72 представлены две таблицы, отображающие содержимое структуры _PS_PROTECTION для процесса LSASS. Эта структура используется для идентификации статуса защиты процесса. На рис. 72, *а* PPL отключен (Level: 0x00 — уровень защиты отсутствует; Type: 0x00 — тип защиты не определен; Audit: 0x00 — режим аудита отсутствует; Signer: 0x00 — подпись отсутствует), т.е. процесс не защищен (все поля структуры _PS_PROTECTION равны 0x00). На рис. 72, *б* PPL включен (флаг Level: 0x41 — указывает на включение защиты уровня PPL; Type: 0x01 — PsProtectedTypeProtectedLight (защищенный процесс с облегченным уровнем защиты); Audit: 0x00 — режим аудита отключен; Signer: 0x04 — PsProtectedSignerLsa (подпись Local Security Authority)).

Если для PPL-процессов защита включена (см. рис. 72, *б*), то системные API учитывают это при создании и открытии ими других процессов [385–387].

Protection	struct _PS_PROTECTION
Level	0x00 ''
Type	0x00 ''
Audit	0x00 ''
Signer	0x00 ''

а

Protection	struct _PS_PROTECTION
Level	0x41 'A'
Type	0x01 ''
Audit	0x00 ''
Signer	0x04 ''

б

Рис. 72. Содержимое структуры EPROCESS.Protection (_PS_PROTECTION) для процесса LSASS:
а — при отключенной защите PPL; *б* — при включенной защите PPL

На рис. 73 представлена внутренняя структура функции CreateProcess на разных уровнях ОС Windows. Взаимодействие функций представлено на трех уровнях: ядро NT, диспетчер процессов (Process Manager) и монитор безопасности (SRM).

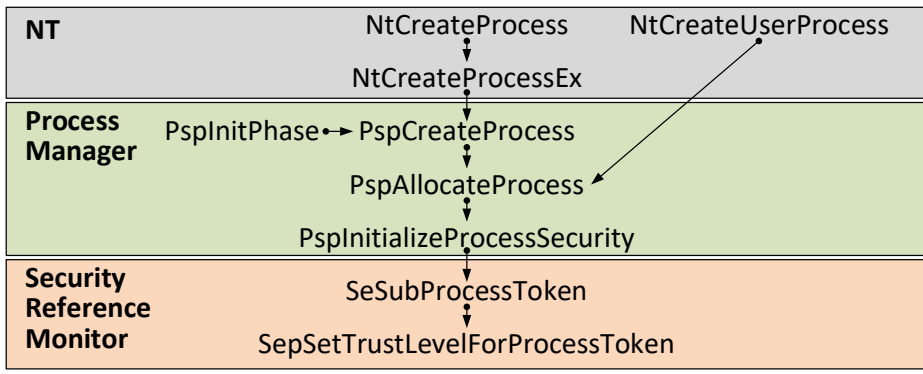


Рис. 73. Внутренняя структура функции CreateProcess

В ядре NT Windows располагаются основные функции для создания процессов: NtCreateProcess и NtCreateProcessEx. Эти функции являются точкой входа для создания процессов в ядре Windows.

Диспетчер процессов (Process Manager) управляет жизненным циклом процесса и его ресурсами. Следующие функции являются частью внутренней логики диспетчера процессов: PspInitPhase (инициализация фазы процесса); PspCreateProcess (создание процесса); PspAllocateProcess (выделение ресурсов для процесса); PspInitializeProcessSecurity (инициализация параметров безопасности процесса). Диспетчер процессов отвечает за инициализацию структуры процесса, выделение памяти и установку параметров безопасности.

На уровне монитора безопасности (Security Reference Monitor) обеспечиваются контроль безопасности процессов и управление их уровнями доверия с использованием функций, таких как SeSubProcessToken и SepSetTrustLevelForProcessToken [388]. На этом этапе выполняется проверка значений защиты процесса, включая PPL: определяются права доступа и уровень доверия, присваиваемые процессу.

Для настройки службы на запуск в режиме защищенного процесса (PPL) используется функция ChangeServiceConfig2 с кодом конфигурации SERVICE_CONFIG_LAUNCH_PROTECTED. Эта функция не «активирует» PPL напрямую, а лишь указывает системе, что служба должна быть запущена с уровнем защиты PPL, при условии, что исполняемый файл соответствует требованиям: он должен быть подписан соответствующим цифровым сертификатом для PPL и содержать корректный манифест [373, 389, 390].

12.1.2.4. Механизм защиты процессов и проверка PPL в ядре Windows

Механизм PPL играет важную роль в предотвращении вмешательства со стороны незащищенных процессов, в том числе вредоносных. Процесс защиты реализуется в ядре Windows.

Функция PsOpenProcess. Функция PsOpenProcess выполняет проверку привилегий при попытке открыть процесс. Если привилегия SeDebugPrivilege включена для вызывающего процесса, то ему предоставляется доступ к целевому процессу независимо от других ограничений. Алгоритм работы PsOpenProcess представлен на рис. 74, *а* и включает следующие операции:

- вызов функции SeCaptureSubjectContextEx;
- вызов функции SepCreateAccessStateFromSubjectContext;
- проверка наличия SeDebugPrivilege (привилегии отладки);
- если привилегия активна, то процесс успешно открывается с использованием функции ObOpenObjectByPointer, которая возвращает указатель на объект процесса.

Фрагмент стека вызовов функции PsOpenProcess представлен на рис. 74, *б*. Таким образом, SeDebugPrivilege позволяет обойти базовые ограничения доступа и является первым уровнем проверки в механизме защиты процессов.

Функция PspProcessOpen. Для процессов со статусом PPL реализована дополнительная проверка на уровне функции PspProcessOpen. Эта функция анализирует поле Protection.Level в структуре EPROCESS, которое хранит информацию о статусе защиты процесса. Алгоритм работы PspProcessOpen представлен на рис. 75, *а*:

- считывается значение Protection.Level из структуры EPROCESS;
- выполняется проверка уровней PPL с помощью PsTestProtectedProcessIncompatibility;
- если Protection.Level равно нулю (защита отсутствует), то функция возвращает STATUS_ACCESS_DENIED, блокируя доступ к PPL-защищенному процессу.

Фрагмент стека вызовов функции PspProcessOpen представлен на рис. 75, *б*. При включенном PPL проводится дополнительная проверка через nt!RtlTestProtectedAccess и nt!PspCheckForInvalidAccessByProtection (см. рис. 75, *б*). При недостаточных правах возвращается STATUS_ACCESS_DENIED.

Таким образом, функция PspProcessOpen добавляет дополнительный слой защиты для процессов с активированным PPL и предотвращает доступ даже при наличии привилегии SeDebugPrivilege.

Доступ к памяти процессов. Чтобы получить доступ к памяти процесса из любых других процессов, необходимо вызвать следующую последовательность функций:

- OpenProcess;
- ReadProcessMemory/WriteProcessMemory;
- CloseHandle.

Функция OpenProcess использует идентификатор процесса (Process Identifier, PID) в качестве одного из входных параметров и возвращает открытый дескриптор указанного процесса. Этот дескриптор используется в процедурах ReadProcessMemory (для чтения) и WriteProcessMemory (для записи) содержимого памяти целевого процесса. Процедура CloseHandle предназначена для завершения закрытия дескриптора и освобождения ресурсов.

При активированном PPL доступ к памяти ограничивается. Даже если вызывающий процесс имеет привилегию SeDebug

Privilege, проверка на уровне PspProcessOpen анализирует значение Protection. Level и может заблокировать запрос, возвращая STATUS_ACCESS_DENIED.

```
NTSTATUS PsOpenProcess()
{
   NTSTATUS Status;
   SeCaptureSubjectContextEx()
   SepCreateAccessStateFromSubjectContext()
   if (SePrivilegeCheck(SeDebugPrivilege,Prev
   iousMode))
      {...}
   PsLookupProcessByProcessId()
   status = ObOpenObjectByPointer()
   return status;
}
```

а

```
Stack fragment:

   nt!PsOpenProcess
   nt!NtOpenProcess
   nt!KiSystemServiceCopyEnd
   ntdll!NtOpenProcess
        KERNELBASE!OpenProcess
```

б

Рис. 74. Внутреннее устройство функции PsOpenProcess:
а — псевдокод с проверкой привилегий SeDebugPrivilege;
б — фрагмент стека вызовов

```
NTSTATUS PsOpenProcess()
{
   NTSTATUS Status;
   SeCaptureSubjectContextEx()
   SepCreateAccessStateFromSubjectContext()
   if (SePrivilegeCheck(SeDebugPrivilege,
   PreviousMode))
      {...}
   PsLookupProcessByProcessId()
   status = ObOpenObjectByPointer()
   return status;
}
```

а

```
Stack fragment:

   nt!PsOpenProcess
   nt!NtOpenProcess
   nt!KiSystemServiceCopyEnd
   ntdll!NtOpenProcess
        KERNELBASE!OpenProcess
```

б

Рис. 75. Внутреннее устройство функции PspProcessOpen
с проверкой PPL:
а — псевдокод с анализом уровня защиты; *б* — фрагмент стека вызовов

12.1.2.5. Возможное отключение механизма PPL нарушителем

Механизм PPL обеспечивает защиту процессов; его реализация основана на проверке значения поля Protection в структуре EPROCESS во время вызова функции OpenProcess. Данное поле указывает статус защиты процесса, однако нарушитель может его

несанкционированно изменить без реакции со стороны штатных средства защиты.

Нарушители могут использовать эту особенность:

- для отключения защиты PPL для защищенных процессов.
- повышения привилегий незащищенных процессов до уровня защищенных, что позволяет нарушителям использовать штатные возможности ОС для противодействия антивирусным системам защиты.

Эти действия открывают возможности для выполнения атак, включая:

- завершение работы антивирусных решений, что нарушает защиту системы;
- кражу учетных данных пользователей из памяти процесса LSASS.

Исследования и эксперименты [359, 360, 364–367, 371, 391–393] подтверждают, что уязвимость механизма PPL действительно существует. Эксперты доказали, что при очистке значения флага Protection в структуре EPROCESS процесс теряет свой статус защищенного (PPL). Это изменение не вызывает срабатывания механизмов защиты ядра, таких как PatchGuard, и не приводит к аварийному завершению работы системы (BSOD). Несанкционированная модификация значения PS_PROTECTION в структуре EPROCESS является важной проблемой для безопасности ОС и требует контроля и блокировки.

Далее рассмотрим инструмент Mimikatz, который способен отключать защиту PPL, а также существующие методы противодействия этой угрозе.

12.1.3. Пример отключения защиты PPL и извлечение паролей с использованием Mimikatz

Изучим механизм работы инструмента Mimikatz, который позволяет извлекать пароли пользователей из памяти процесса LSASS как при отключенном, так и при включенном статусе PPL. Также проанализируем существующие подходы к обнаружению и предотвращению атак Mimikatz.

12.1.3.1. Доступ к памяти LSASS с помощью Mimikatz: сравнительный анализ при активном и отключенном PPL

Инструмент Mimikatz способен извлекать различные типы конфиденциальных данных из памяти [360]. Обзор возможностей Mimikatz и его внутреннего устройства представлен в работах [394–397].

Доступ к памяти LSASS при отключенном и включенном PPL. Для получения содержимого хэшей паролей пользователей из процесса LSASS при отключенной защите PPL нарушитель может использовать две команды Mimikatz:

1) privilege::debug — предоставляет процессу Mimikatz привилегию отладки (SeDebugPrivilege);

2) lsadump::lsa /inject — извлекает хэши паролей из памяти процесса LSASS;

Эти команды позволяют получить доступ к памяти LSASS и извлечь конфиденциальные данные. Подробности работы команды lsadump::lsa /inject описаны в исследовании [398].

Если для процесса LSASS включена защита PPL, то выполнение двух упомянутых выше команд Mimikatz завершается неудачей, так как доступ к памяти защищенного процесса блокируется операционной системой.

Отключение PPL через Mimikatz. Для обхода защиты PPL разработчики Mimikatz добавили возможность отключать PPL путем изменения значения поля Protection в структуре EPROCESS. Процесс состоит из двух основных шагов:

1) запись нулей в содержимое поля Protection в структуре EPROCESS с помощью драйвера Mimikatz. Это действие отключает защиту PPL для процесса LSASS;

2) вызов функции OpenProcess консольным приложением Mimikatz, чтобы получить доступ к памяти процесса LSASS и извлечь из нее конфиденциальные данные (например, хэши паролей).

Для этого используются четыре команды (рис. 76):

1) **!+** — загружает драйвер Mimikatz;

2) **!processprotect /process:lsass.exe /remove** — обнуляет значение поля Protection в структуре EPROCESS для процесса LSASS (см. рис. 76);

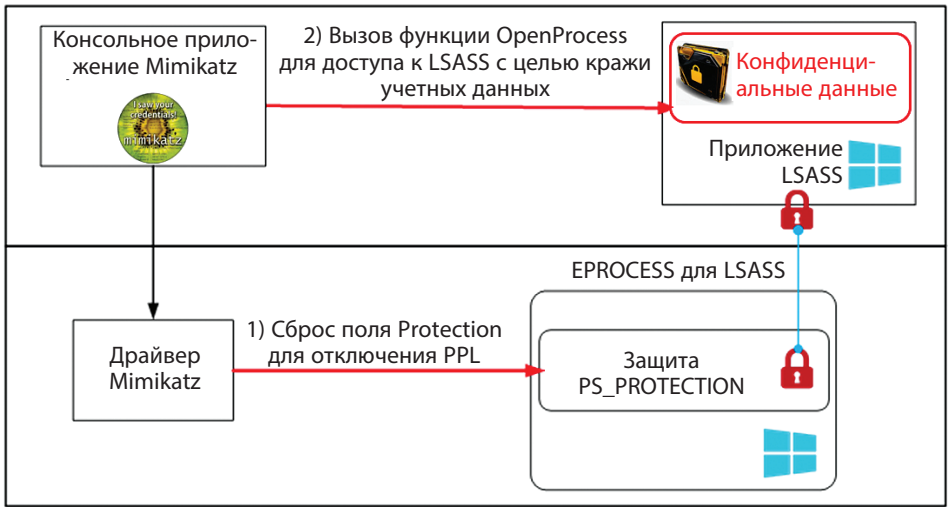


Рис. 76. Механизм отключения защиты PPL (Protected Process Light) для процесса LSASS с помощью инструмента Mimikatz

3) **privilege::debug** — предоставляет процессу Mimikatz привилегию отладки (SeDebugPrivilege);

4) **lsadump::lsa /inject** — извлекает хэши паролей пользователей из памяти LSASS.

После выполнения этих команд процесс LSASS теряет статус PPL, что позволяет извлечь из него данные с помощью Mimikatz.

Повышение уровня PPL для атакующего процесса. Другой подход, используемый нарушителями, заключается не в отключении PPL для LSASS, а в повышении уровня PPL для атакующего процесса (например, самого Mimikatz). Экспериментальные результаты показывают, что если уровень защиты процесса Mimikatz будет выше или равен уровню защиты LSASS (например, 0x41, рис. 77), то процесс Mimikatz может получить доступ к памяти LSASS без отключения PPL.

12.1.3.2. Анализ существующих подходов к борьбе с Mimikatz

Проанализируем методы обнаружения и предотвращения атак с использованием Mimikatz. Данные подходы предложены в исследованиях [399, 400].

```
kd> dx -r1 ((ntkrnlmp!_PS_PROTECTION *)0xffffa684cc8e577a)
    [+0x000] Level          : 0x41 [Type: unsigned char]
    [+0x000 ( 2: 0)] Type   : 0x1  [Type: unsigned char]
    [+0x000 ( 3: 3)] Audit  : 0x0  [Type: unsigned char]
    [+0x000 ( 7: 4)] Signer : 0x4  [Type: unsigned char]
```

```
kd> dx -r1 ((ntkrnlmp!_PS_PROTECTION *)0xffffa684cc8e577a)
    [+0x000] Level          : 0x0 [Type: unsigned char]
    [+0x000 ( 2: 0)] Type   : 0x0 [Type: unsigned char]
    [+0x000 ( 3: 3)] Audit  : 0x0 [Type: unsigned char]
    [+0x000 ( 7: 4)] Signer : 0x0 [Type: unsigned char]
```

Рис. 77. Содержимое структуры EPROCESS.Protection _PS_PROTECTION для процесса LSASS.exe с включенной защитой PPL до модификации драйвером Mimikatz (*а*) и после нее (*б*). В результате защита PPL для LSASS была отключена

1. Объявление Mimikatz вредоносным ПО. Самый простой из опубликованных подходов заключается в добавлении Mimikatz и его драйвера в черный список антивирусных решений. Примерно 70% антивирусных продуктов определяют Mimikatz как ВПО, однако существует мнение, что Mimikatz не является вирусом [401].

2. Использование Windows PowerShell. Для выявления последствий атаки Mimikatz можно использовать инструменты PowerShell [402]. После отключения PPL для LSASS можно запросить его атрибуты с помощью PowerShell и обнаружить их несоответствие ожидаемым значениям. Однако этот метод не позволяет предотвратить попытку несанкционированного изменения PPL.

3. Отзыв привилегии SeDebugPrivilege. Отзыв привилегии SeDebugPrivilege для всех администраторов с помощью групповой политики может предотвратить открытие защищенных процессов нарушителями. Однако ВПО, обладающее локальными правами администратора, может восстановить эту привилегию [400, 401].

4. Отключение протокола WDigest. Хранение учетных данных пользователей в памяти можно предотвратить путем отключения протокола WDigest. Для этого необходимо внести изменения в реестр Windows [246, 403]. Эта мера предотвращает извлечение паролей, но нарушители с правами администратора могут восстановить исходные настройки в реестре [404, 405].

5. Включение режима Restricted Admin. *Microsoft* предлагает режим Restricted Admin, который отключен по умолчанию. Для его активации требуется добавить специальный ключ в реестр Windows [400]. Однако нарушители с правами администратора могут отключить этот режим, изменив настройки реестра [406].

6. Использование Windows Defender Credential Guard (WDCG). Windows Defender Credential Guard (WDCG) — это механизм, основанный на VSM, который разделяет память ОС на два уровня доверия: VTL0 и VTL1. LSASS запускается на уровне VTL1, что защищает его память от процессов, рабо-

тающих на уровне VTL0. Однако WDCG доступен только в версиях Windows Enterprise и Windows Server [407, 408].

Рассмотренные методы не могут обеспечить защиту учетных данных пользователей от атак Mimikatz. Нарушители могут реализовать одну из следующих атак:

- отключение PPL путем обнуления значения PS_PROTECTION в структуре EPROCESS, что открывает доступ к памяти LSASS;
- повышение уровня PPL для вредоносного процесса.

Инструмент Mimikatz реализовал только первую атаку, но экспериментальные данные показывают, что процесс с повышенным уровнем PPL также может получить доступ к памяти LSASS, не отключая его защиту.

Чтобы защитить память процесса, необходимо гарантировать целостность поля Protection структур EPROCESS для всех процессов. Любые попытки изменить эти поля могут интерпретироваться как индикатор компрометации (IoC), поскольку Windows не предоставляет никаких документированных API для изменения уровня PPL во время выполнения.

12.2. МОДИФИКАЦИЯ ГИПЕРВИЗОРА MEMORYRANGER ДЛЯ ЗАЩИТЫ СТРУКТУР PPL

Основные механизмы MemoryRanger, включая использование гипервизора типа 1, технологий Intel VT-x и EPT, были рассмотрены в главах 9–11. Подробный обзор возможностей MemoryRanger представлен в работах [302–304]. Теперь сосредоточим внимание на новых возможностях MemoryRanger, направленных на защиту механизма PPL от рассмотренных атак.

12.2.1. Расширенные возможности MemoryRanger для защиты механизма PPL

MemoryRanger был обновлен для предотвращения двух типов атак на PPL:

1) отключение защиты PPL для системных процессов ОС (например, LSASS);

2) повышение уровня защиты PPL для произвольных тестовых процессов.

MemoryRanger был выбран в качестве базовой платформы для защиты PPL от упомянутых атак DKOM из-за его возможностей мониторинга обращений к памяти ядра и различных событий ОС.

12.2.2. Защита PPL: обновления MemoryRanger

В связи с тем, что отсутствует API Windows для изменения уровня защиты процесса PPL во время выполнения, MemoryRanger должен запретить любой доступ на запись в поле Protection структур EPROCESS, не ограничивая доступ на чтение. Для реализации этого были обновлены драйвер и гипервизор MemoryRanger.

Функции обновленного драйвера MemoryRanger:

- находит структуры EPROCESS для всех активных процессов и создает свой внутренний список активных процессов (Active Processes List, APL);
- отслеживает создание и завершение процессов, обновляя свой список активных процессов (APL);
- для каждого элемента списка определяет адрес структуры PS_PROTECTION, содержащей уровень защиты процесса и подписи, и передает его гипервизору.

Функции обновленного гипервизора MemoryRanger:

- разрешает или ограничивает доступ к соответствующей области памяти для каждого анклава после обработки обновления от драйвера MemoryRanger;
- перехватывает нарушения доступа к памяти и проверяет, выполняется ли запись в структуру PS_PROTECTION.

Из-за страничной организации памяти гипервизор Memory Ranger перехватывает любой доступ на запись к соответствующей странице памяти и проверяет, является ли это доступом на запись в структуру PS_PROTECTION.

Обновленный MemoryRanger ограничивает доступ к полю Protection в структуре EPROCESS с помощью механизма EPT. При попытке доступа к полю Protection гипервизор MemoryRanger получает уведомление за счет перехвата исключения EPT violation. Реализация механизмов защиты может быть

выполнена двумя способами: с использованием отдельного анклава для защищаемых данных или без него. Далее рассмотрим оба случая.

12.2.2.1. MemoryRanger с отдельным анклавом для защищаемых данных

В данном случае MemoryRanger после загрузки выделяет два анклава: анклав ядра по умолчанию и специальный анклав для защищаемых данных (рис. 78). В этом специальном анклаве размещаются структуры ядра ОС, включая поле PS_PROTECTION для процесса LSASS, что обеспечивает защиту данных процесса от модификации, например, драйвером Mimikatz.

Для изоляции драйвера Mimikatz он помещается в отдельный анклав, что позволяет MemoryRanger контролировать его выполнение и предотвращать несанкционированный доступ.

MemoryRanger динамически переключается между анклавами, предоставляя доступ к данным только авторизованным процессам. Попытки записи в защищенные области памяти перехватываются гипервизором, что предотвращает атаки на поле PS_PROTECTION. Однако экспериментальные результаты показали, что такой подход вызывает значительное снижение производительности и может привести даже к блокировке ОС из-за частого переключения между анклавом по умолчанию и анклавом для защищаемых данных. Следует отметить, что система MemoryRanger результативна только для защиты данных, к которым редко осуществляется доступ.

12.2.2.2. MemoryRanger без специального анклава для конфиденциальных данных

Во втором случае MemoryRanger изолирует содержимое памяти с помощью поля Protection только для вновь загруженных драйверов (рис. 79). Результаты эмпирического теста показывают, что в этом случае MR производит приемлемое снижение производительности. Подробности оценки производительности будут приведены ниже.

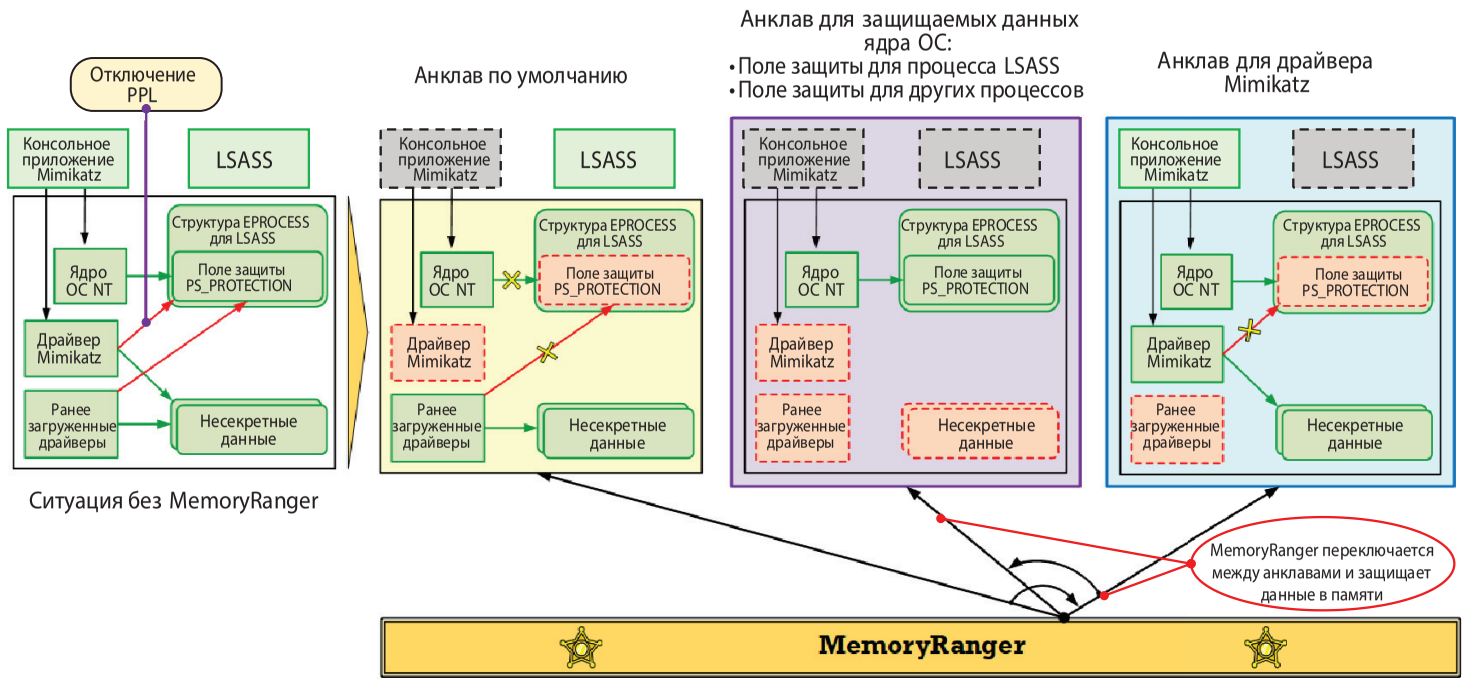


Рис. 78. Сценарий 1: MemoryRanger защищает структуру EPROCESS процесса LSASS от модификации драйвером Mimikatz, используя три анклава: базовый анклав для ранее загруженных драйверов, анклав для защищаемых данных ядра ОС и отдельный анклав для нового драйвера Mimikatz

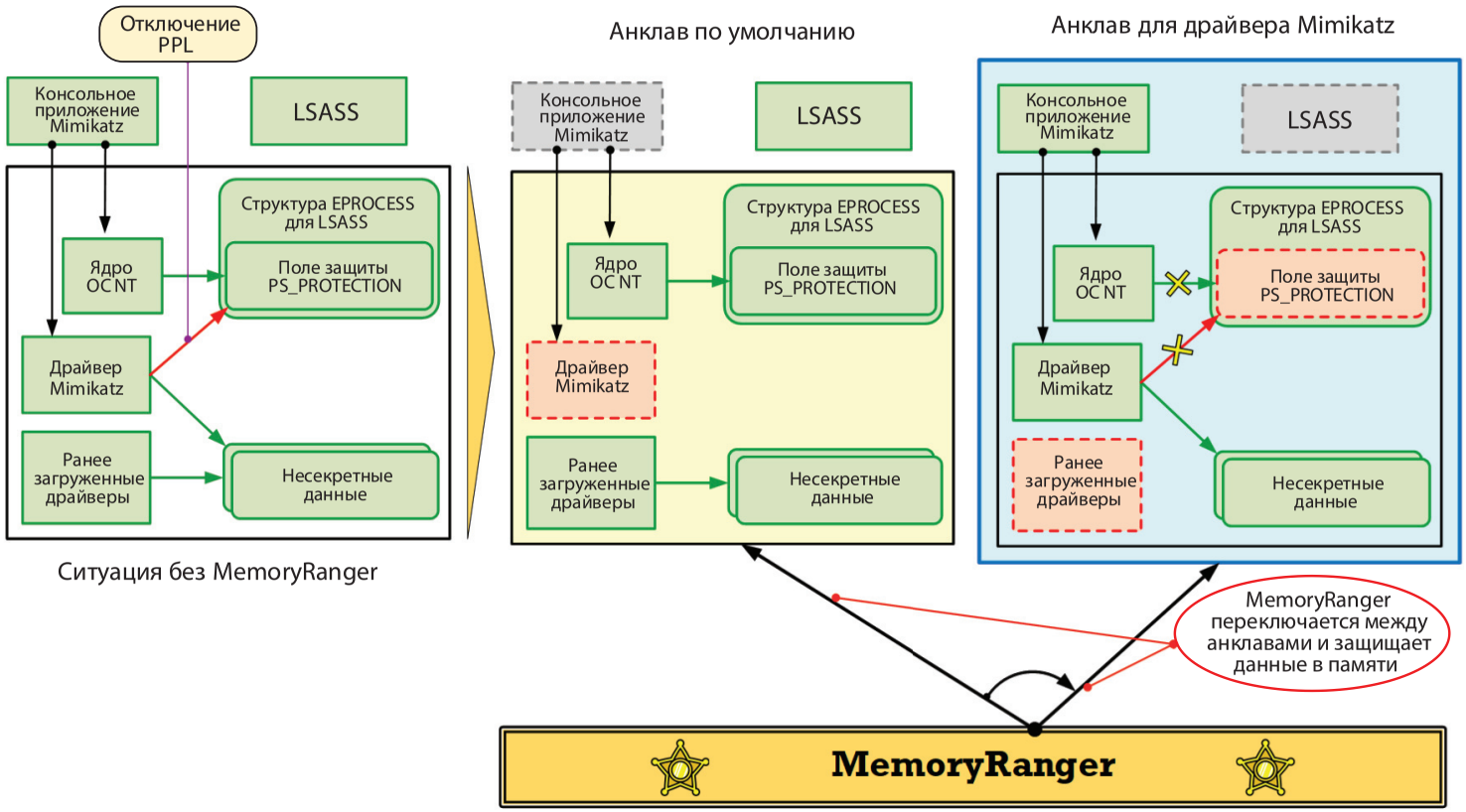


Рис. 79. Сценарий 2: MemoryRanger защищает структуру EPROCESS процесса LSASS от модификации драйвером Mimikatz, используя два анклава: базовый анклав для ранее загруженных драйверов и отдельный анклав для нового драйвера Mimikatz

12.3. ПРАКТИЧЕСКОЕ ПРИМЕНЕНИЕ МОДИФИЦИРОВАННОГО MEMORYRANGER: ПРЕДОТВРАЩЕНИЕ АТАК НА PPL И СРАВНЕНИЕ С WINDOWS DEFENDER CREDENTIAL GUARD

12.3.1. Оценка влияния MemoryRanger на производительность процессора

Для оценки влияния MemoryRanger на производительность системы было проведено тестирование с использованием инструмента Super PI, который измеряет время вычисления числа π до определенного количества знаков после запятой. Сравнительный анализ проводился для системы с включенным и выключенным MemoryRanger.

Цель эксперимента — оценить влияние MemoryRanger на производительность системы путем выполнения CPU-интенсивной задачи (расчет числа π).

Тестирование производительности проводилось на следующей вычислительной платформе:

- аппаратное обеспечение: ноутбук Dell 7579 с процессором Intel i7-7500U (4 логических ядра) и 12 Гб оперативной памяти;
- программное обеспечение: хостовая ОС Windows 10 версии 20H2 (Build 19042.631) x64;
- виртуализация: виртуальная машина (VM) запущена на VMware Workstation с двумя выделенными логическими ядрами и 4 Гб оперативной памяти.

MemoryRanger работал внутри виртуальной машины, и производительность измерялась при включенной и выключенной защите.

Результаты тестов показали, что при включенном MemoryRanger производительность процессора снизилась примерно на 50% (рис. 80).

Такая значительная потеря производительности объясняется тем, что текущая реализация MemoryRanger является прототипом, нацеленным в первую очередь на демонстрацию концепции предотвращения атак. Алгоритмы, отвечающие за распределение ресурсов, фильтрацию и контроль операций с памятью, могут быть неоптимальными, что создает дополнительную нагрузку. Кроме того, в рамках экспериментов используется среда виртуализации

VMware Workstation, которая сама по себе вносит существенные задержки, особенно при эмуляции поддержки VMX-технологий.

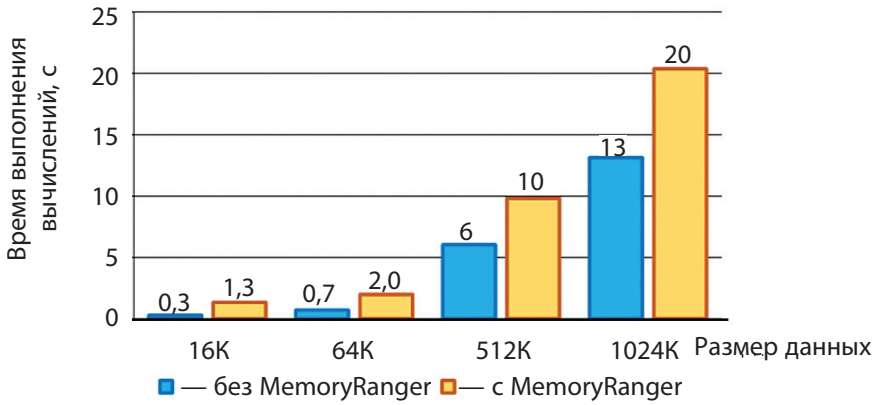


Рис. 80. Оценка производительности MemoryRanger: при включенном MemoryRanger производительность процессора снижается примерно на 50%

Использование более производительной среды тестирования и оптимизация внутренних алгоритмов MemoryRanger могли бы улучшить показатели производительности.

Для проверки корректности работы MemoryRanger по защите системных процессов от атак был использован известный инструмент Mimikatz версии 2.2.0 (сборка от 2020-09-18), позволяющий проводить атаки на LSASS. Кроме того, был использован разработанный опытный прототип драйвера для атак на память, предназначенный для демонстрации возможностей изменения уровня PPL для заданных процессов. Результаты тестов были записаны и опубликованы на канале автора [409].

12.3.2. Сравнение MemoryRanger и Windows Defender Credential Guard (WDCG)

MemoryRanger и WDCG являются решениями на основе виртуализации, использующими изолированные анклавы памяти, созданные с помощью технологии EPT. Сравнительный анализ этих инструментов приведен в табл. 28.

Windows Defender Credential Guard (WDCG) — встроенная технология защиты, работающая на основе VSM (Virtual Secure

Mode). WDCG создает изолированный защищенный контекст (анклав памяти), в котором хранятся конфиденциальные и учетные данные (например, хэши паролей), что исключает существующие атаки на LSASS, поскольку соответствующий новый процесс LSAISO (LSA Isolated) выполняется как процесс изолированного пользовательского режима (Isolated User Mode, IUM) в среде, память которого защищена от прямого доступа со стороны драйверов и процессов, работающих в режиме Normal mode. WDCG фактически защищает лишь ряд встроенных системных процессов, включая LSASS, без поддержки защиты сторонних приложений.

MemoryRanger лишен этого недостатка и защищает любые процессы от изменения значений PPL после загрузки процесса.

Архитектуры анклавов для WDCG и MemoryRanger различаются. WDCG использует фиксированное количество анклавов (два раздела памяти для обычных и защищенных операций). MemoryRanger предоставляет более гибкую и универсальную архитектуру с возможностью создания множества анклавов. MemoryRanger был протестирован с тремя [303], четырьмя [302] и пятью анклавами [304].

Таблица 28

Сравнение проектов Windows Defender Credential Guard (WDCG) и MemoryRanger

Функция	WDCG	Memory Ranger
Предотвращает ли доступ к памяти процесса LSASS?	Да	Да
Предотвращает ли доступ к памяти других защищенных процессов после отключения их PPL?	Нет	Да
Предотвращает ли несанкционированное повышение уровня PPL для обычных процессов для самозащиты?	Нет	Да
Запускается ли при старте системы?	Да	Нет
Поддерживает ли различные версии Windows?	Только Enterprise	Все версии

MemoryRanger позволяет предотвратить как отключение, так и незаконное включение PPL, обеспечивая более широкий спектр защитных мер по сравнению с WDCG. Однако WDCG встроен в Windows и оптимизирован для защиты учетных данных LSASS. MemoryRanger, в свою очередь, выступает как общий защитный механизм, предлагая более широкую функциональность в противодействии разнообразным атакам.

Глава 13
ПРОЕКТ MEMORYRANGER: ЗАЩИТА MICROSOFT DEFENDER ОТ АТАК НА УРОВНЕ ЯДРА WINDOWS

Современные вредоносные программы часто нацелены на отключение встроенного антивируса Windows — Microsoft Defender (MD), либо на несанкционированное изменение его настроек, что приводит к потере антивирусом способности реагировать на угрозы. Одна из опубликованных атак использует системные вызовы для понижения уровня привилегий процесса MD. Это блокирует доступ антивируса к памяти приложений и содержимому файловой системы. В результате нарушители могут использовать штатные механизмы защиты ОС, такие как мандатный контроль целостности (Mandatory Integrity Control, MIC) и монитор безопасности (SRM) для блокировки работы MD.

Хотя в последних обновлениях ОС были реализованы меры защиты по блокировке данной атаки и исправлению уязвимости, проведенные исследования показали, что нарушители могут добиться аналогичного результата с использованием драйверов ядра, изменяя содержимое структур системной области памяти. Такая модификация позволяет отключить функции безопасности антивируса MD без завершения его работы, что могло бы являться индикатором вредоносного вмешательства. Более того, такая манипуляция не приводит к срабатыванию защитного механизма PatchGuard.

Для защиты от этих угроз гипервизор MemoryRanger был адаптирован. Экспериментальные испытания подтвердили, что обновленный MemoryRanger успешно предотвращает снижение привилегий процесса MD, гарантируя его безопасную работу и устойчивость к подобным атакам.

13.1. АНАЛИЗ ПОПУЛЯРНЫХ АТАК НА MICROSOFT DEFENDER: ИССЛЕДОВАНИЕ ТЕХНИК ОТКЛЮЧЕНИЯ И ОБХОДА АНТИВИРУСНОЙ ЗАЩИТЫ

13.1.1. Проблемы безопасности ядра Windows и современные технологии защиты

Одной из острых проблем информационной безопасности Windows является отсутствие изоляции памяти ядра ОС и загруженных драйверов, которые совместно используют единое адресное пространство. Такая архитектура обеспечивает высокую производительность, но создает угрозы безопасности: нарушители могут использовать драйверы для выполнения вредоносных действий, включая обход антивирусных средств защиты.

Для противодействия этим угрозам разработаны следующие штатные механизмы защиты.

Kernel Patch Protection: осуществляет регулярную проверку целостности системных структур памяти ядра. При обнаружении изменений система в аварийном режиме завершает работу (BSOD). Недостатками PatchGuard являются ограничение проверки лишь некоторыми системными структурами и уязвимость к атакам драйверов. Существуют проекты, предназначенные для обхода PatchGuard, такие как NoPatchGuardCallback [410], UPGDSED [411] и EfiGuard [412].

Driver Signature Enforcement: реализует проверку цифровой подписи драйверов на основе политики KMCS. Однако нарушители могут обойти этот механизм, используя легитимные драйверы с уязвимостями или украденные сертификаты.

Virtualization-Based Security: использует гипервизор Hyper-V для защиты компонентов системы. Основные компоненты VBS:

- Secure Kernel Patch Guard (SKPG), также известный как HyperGuard [1], выполняющий функции PatchGuard, но работающий в более привилегированном режиме для защиты от атак со стороны драйверов;
- Kernel Data Protection (KDP), обеспечивающий защиту от несанкционированного изменения отдельных системных структур в памяти [413];

- Device Guard (DG) и Credential Guard (CG), которые изолируют важные компоненты системы.

Virtualization-Based Security создает виртуальный безопасный режим (Virtual Secure Mode) с двумя уровнями доверия (VTL): VTL0 — для ядра и драйверов; VTL1 — для изолированных процессов (trustlets). Однако VBS-технология имеет ограничения: она защищает только встроенные компоненты Windows, не предоставляя интерфейсов для защиты сторонних приложений и драйверов, а также VBS поддерживает всего лишь два уровня изоляции.

Одним из существенных недостатков защитных механизмов Windows является отсутствие защиты ряда важных процессов, например Microsoft Malware Protection Engine (MsMpEng.exe), отвечающего за работу MD. Это делает его уязвимым к атакам, направленным на снижение привилегий с последующей блокировкой доступа к ресурсам системы.

13.1.2. Анализ атак на Microsoft Defender

Встроенный антивирус MD — это комплексное решение для защиты Windows от современных угроз. MD включает компоненты пользовательского режима (MsMpEng.exe, NisSrv.exe, MsMpEngCP.exe, MpEngine.dll) и драйверы ядра (WdBoot.sys, WdFilter.sys и др.). Подробный анализ различных компонентов антивируса MD и существующих техник по противодействию можно найти в работах [414–417]. Несмотря на наличие встроенных механизмов защиты, нарушители активно используют уязвимости этих компонентов для выполнения атак, примеры которых приведены в табл. 29.

Наибольшую опасность представляет ВПО, которое использует возможности драйверов ядра для атак на различные средства защиты, в том числе антивирус MD.

Эксперты *Sophos X-Ops* обнаружили, что недавно выявленная программа-вымогатель использует уязвимый драйвер Process Explorer из пакета Sysinternals компании *Microsoft* [421]. Этот подход, известный как Bring Your Own Vulnerable Driver (BYOVD), позволяет нарушителям использовать возможности уязвимых легитимных драйверов для атак на антивирусные

системы. Примеры того, как нарушители могут загрузить драйверы, показаны на рис. 81 [34].

Таблица 29

Примеры вредоносных атак на компоненты MD

Название ВПО	Цель атаки	Принцип действия	Техника обнаружения
TrickBot (2020) — троян [418]	Отключение антивируса MD, снижение его работоспособности	Использует команды PowerShell для отключения компонент реального времени, системных уведомлений и удаления сервисов MD. Применяет штатные команды Windows	Поведенческий анализ, мониторинг команд PowerShell
Zloader (2021) — троян [419]	Сокрытие вредоносной активности, отключение защиты в реальном времени	Добавляет вредоносные процессы в антивирусный список исключений и отключает защиту от нежелательных приложений с помощью PowerShell. Использует встроенные функции добавления исключений	Мониторинг списка исключений
DeroHE (2022) — программа-вымогатель [420]	Сокрытие от MD и предотвращение обнаружения	Скрывает себя от MD, добавляя в антивирусный список исключений с использованием команды WMIC. Использует возможности управления исключениями через WMI	Мониторинг списка исключений, мониторинг команд WMI

Все виды атак можно разделить на два типа: атаки режима ядра и атаки пользовательского режима. Классификация атак на Microsoft Defender представлена на рис. 82.

1. *Атаки режима ядра* используют возможности драйверов для обхода защитных механизмов. Основные техники атак в режиме ядра включают:

- отключение ETW-сессий: антивирус MD использует систему трассировки событий ETW (Event Tracing for Windows) для сбора данных из защищенных сессий DefenderApiLogger и DefenderAuditLogger. Эти сессии защищены механизмом

PPL, предотвращающим их остановку стандартными средствами ОС. Нарушители могут отключить защиту PPL с помощью драйверов ядра, что позволяет остановить ETW-сессии с пользовательскими привилегиями. Такая атака обходит защитные механизмы ОС и позволяет компрометировать события ETW, снижая результативность работы антивируса;

- отключение PPL-защиты: драйверы ядра могут отключать защиту PPL защищенных процессов, что позволяет им завершать процесс MD или внедрять в него вредоносный код. Такие действия не всегда могут быть своевременно обнаружены [422];
- отключение callback-вызовов ядра: антивирус MD регистрирует callback-функции (функции обратного вызова) для мониторинга событий ОС. Можно выделить несколько техник отключения обратных вызовов ядра: обнуление адреса функции обратного вызова; удаление зарегистрированной функции обратного вызова; модификация кода функции обратного вызова. Техника удаления обратных вызовов реализована в ПО CheekyBlinder [423], которое загружает подписанный легитимный драйвер и эксплуатирует его уязвимости для модификации памяти ядра;

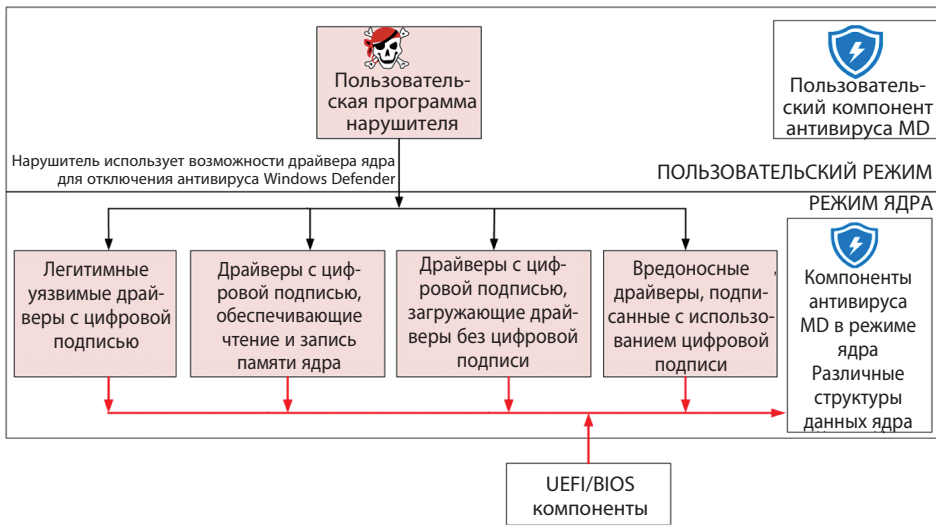


Рис. 81. Нарушители используют возможности драйверов режима ядра для отключения антивируса MD

- закрытие дескрипторов процесса антивируса MD: некоторые драйверы позволяют завершить процесс MD, несмотря на активную PPL-защиту. Это реализовано в ПО Backstab [424], использующем драйвер Process Explorer.

Рис. 82. Классификация атак на антивирус MD

2. *Атаки в пользовательском режиме* эксплуатируют возможности программ, работающих вне привилегированного контекста. Примеры таких атак:

- внедрение в «белые» списки: нарушители, помимо изменения самих списков исключения антивирусных программ, могут внедрять код в пользовательские незащищенные процессы, которые обычно уже добавлены в список доверенных процессов (например, explorer.exe, smartscreen.exe, werfault.exe и др.), чтобы избежать обнаружения. Техника Process Herpaderping

маскирует изменения, модифицируя содержимое файлов процесса на диске, затрудняя обнаружение вредоносной активности [425];

- полиморфные модификации кода: нарушители применяют упаковку, шифрование или обфускацию (запутывание) кода ВПО, чтобы затруднить его обнаружение антивирусом. В результате файловая сигнатура такого ВПО будет отсутствовать в списках известных вирусов, которые обычно используют антивирусы при сканировании файловой системы и памяти процессов;
- манипуляции с правами доступа к памяти: использование методов изменения прав доступа к страницам памяти процессов ВПО для препятствования сканированию и анализу содержимого памяти антивирусом;
- использование символьных ссылок для перенаправления системных путей. Например, нарушитель может временно перенаправить \Device\BootDevice на другой диск, выгрузить драйвер Windows Defender (WdFilter.sys) и заменить его поддельным драйвером. Эта атака использует встроенные функции Windows для обхода защиты [426].

13.2. АНАЛИЗ РАБОТЫ АНТИВИРУСА MICROSOFT DEFENDER В УСЛОВИЯХ АТАК ДРАЙВЕРОВ: МАНИПУЛЯЦИИ С УРОВНЕМ ЦЕЛОСТНОСТИ И ПРИВИЛЕГИЯМИ АНТИВИРУСНОГО ПРОЦЕССА

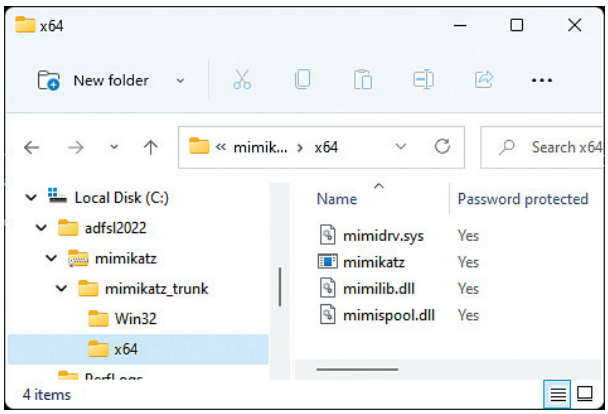
Представим результаты экспериментов, посвященных изучению работы антивируса MD в условиях, когда нарушители модифицируют привилегии процесса MD, чтобы лишить антивирус возможности реагировать на угрозы.

13.2.1. Обзор внутреннего устройства антивируса MD

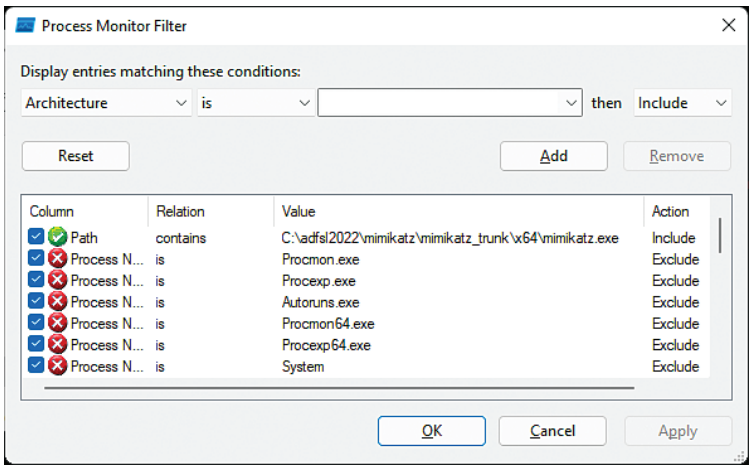
Для проведения экспериментального исследования работы антивируса MD был создан тестовый стенд с использованием следующих программных инструментов:

- виртуальная машина на платформе VMware с установленной Windows 11 x64 в целях обеспечения изолированной среды для безопасного запуска вредоносных программ;

- архив, защищенный паролем и содержащий файл ВПО (рис. 83, *а*). Использование архива с паролем препятствует обнаружению ВПО и обеспечивает воспроизводимость начальных условий при каждом запуске эксперимента;
- приложение Mimikatz, предназначенное для тестирования реакции антивируса MD на обнаружение вредоносных программ;
- утилита Process Monitor (рис. 83, *б*) с активированным фильтром для мониторинга изменений в поведении антивируса MD, связанных с модификацией привилегий его процесса.



а



б

Рис. 83. Компоненты тестового стенда:
а — защищенный паролем архив с ВПО (приложение Mimikatz);
б — утилита Process Monitor с включенным фильтром для мониторинга событий только для указанного файлового пути

13.2.2. Исходное поведение антивируса MD при обнаружении вредоносного программного обеспечения

Для анализа работы антивируса MD при обнаружении ВПО были выполнены следующие шаги:

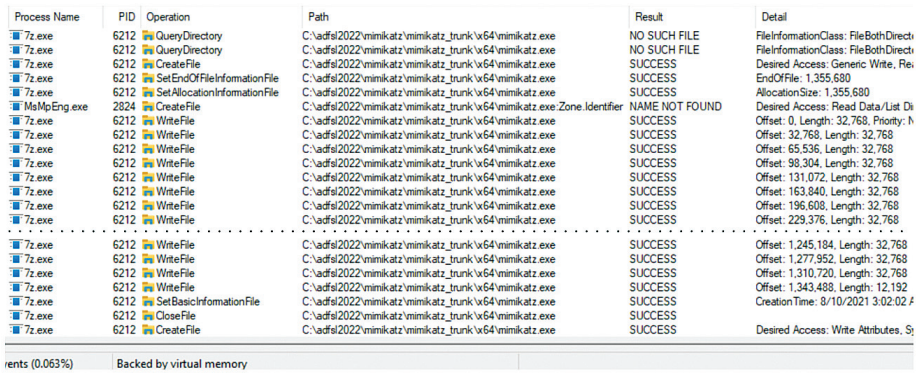
1) запуск утилиты Process Monitor для отслеживания файловой активности;

2) выполнение пакетного скрипта для распаковки защищенного архива и запуска процесса Mimikatz;

3) логирование и анализ работы MD с использованием монитора файловых событий.

Анализ логов Process Monitor позволил выделить три этапа работы MD при обнаружении вредоносного файла.

Этап 1. Извлечение файла на диск. Утилита 7Z.EXE распаковывает и записывает файл образа ВПО в указанную директорию. Это зафиксировано на рис. 84, где видно, что антивирус начинает отслеживать активность, связанную с созданием нового файла.

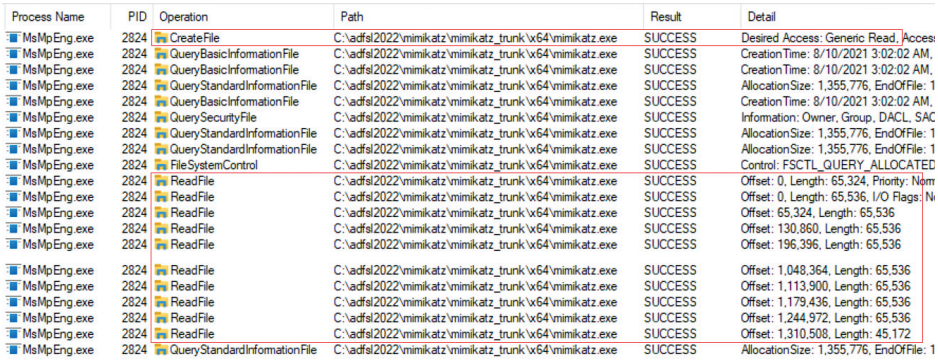


Process Name	PID	Operation	Path	Result	Detail
7z.exe	6212	QueryDirectory	C:\adfs\2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	NO SUCH FILE	FileInformationClass: FileBothDirect
7z.exe	6212	QueryDirectory	C:\adfs\2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	NO SUCH FILE	FileInformationClass: FileBothDirect
7z.exe	6212	CreateFile	C:\adfs\2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Desired Access: Generic Write, Re
7z.exe	6212	SetEndOfFileInformationFile	C:\adfs\2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	EndOfFile: 1,355,680
7z.exe	6212	SetAllocationInformationFile	C:\adfs\2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	AllocationSize: 1,355,680
MsMpEng.exe	2824	CreateFile	C:\adfs\2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe:Zone.Identifier	NAME NOT FOUND	Desired Access: Read Data/List Di
7z.exe	6212	WriteFile	C:\adfs\2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Offset: 0, Length: 32,768, Priority: N
7z.exe	6212	WriteFile	C:\adfs\2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Offset: 32,768, Length: 32,768
7z.exe	6212	WriteFile	C:\adfs\2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Offset: 65,536, Length: 32,768
7z.exe	6212	WriteFile	C:\adfs\2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Offset: 98,304, Length: 32,768
7z.exe	6212	WriteFile	C:\adfs\2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Offset: 131,072, Length: 32,768
7z.exe	6212	WriteFile	C:\adfs\2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Offset: 163,840, Length: 32,768
7z.exe	6212	WriteFile	C:\adfs\2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Offset: 196,608, Length: 32,768
7z.exe	6212	WriteFile	C:\adfs\2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Offset: 229,376, Length: 32,768
7z.exe	6212	WriteFile	C:\adfs\2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Offset: 1,245,184, Length: 32,768
7z.exe	6212	WriteFile	C:\adfs\2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Offset: 1,277,952, Length: 32,768
7z.exe	6212	WriteFile	C:\adfs\2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Offset: 1,310,720, Length: 32,768
7z.exe	6212	WriteFile	C:\adfs\2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Offset: 1,343,488, Length: 12,192
7z.exe	6212	SetBasicInfomationFile	C:\adfs\2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	CreationTime: 8/10/2021 3:02:02 A
7z.exe	6212	CloseFile	C:\adfs\2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	
7z.exe	6212	CreateFile	C:\adfs\2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Desired Access: Write Attributes, Sy
ents (0.063%)		Backed by virtual memory			

Рис. 84. Этап 1. Вывод Process Monitor показывает, что 7Z.EXE извлекает и записывает файл на диск

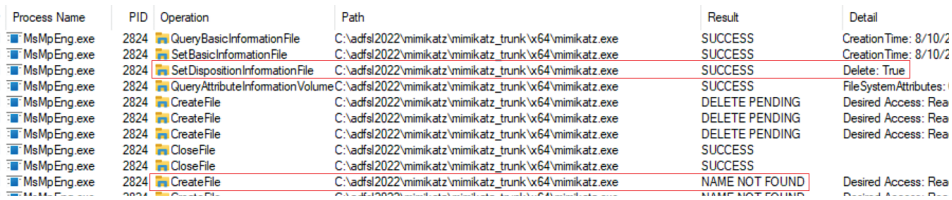
Этап 2. Сканирование извлеченного файла. Основной процесс антивируса MD (MsMpEng.EXE) выполняет чтение извлеченного файла Mimikatz из директории на диске. Это свидетельствует о начале анализа содержимого файла (рис. 85).

Этап 3. Удаление обнаруженного файла. Антивирусный процесс MsMpEng.EXE удаляет вредоносный файл и проверяет его удаление путем повторного открытия файла (рис. 86).



Process Name	PID	Operation	Path	Result	Detail
MsMpEng.exe	2824	CreateFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Desired Access: Generic Read, Acces
MsMpEng.exe	2824	QueryBasicInformationFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	CreationTime: 8/10/2021 3:02:02 AM,
MsMpEng.exe	2824	QueryBasicInformationFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	CreationTime: 8/10/2021 3:02:02 AM,
MsMpEng.exe	2824	QueryStandardInformationFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	AllocationSize: 1,355,776, EndOfFile: 1
MsMpEng.exe	2824	QueryBasicInformationFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	CreationTime: 8/10/2021 3:02:02 AM,
MsMpEng.exe	2824	QuerySecurityFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Information: Owner, Group, DACL, SAC
MsMpEng.exe	2824	QueryStandardInformationFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	AllocationSize: 1,355,776, EndOfFile: 1
MsMpEng.exe	2824	QueryStandardInformationFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	AllocationSize: 1,355,776, EndOfFile: 1
MsMpEng.exe	2824	FileSystemControl	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Control: FSCTL_QUERY_ALLOCATED
MsMpEng.exe	2824	ReadFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Offset: 0, Length: 65,324, Priority: Nor
MsMpEng.exe	2824	ReadFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Offset: 0, Length: 65,536, I/O Flags: N
MsMpEng.exe	2824	ReadFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Offset: 65,324, Length: 65,536
MsMpEng.exe	2824	ReadFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Offset: 130,860, Length: 65,536
MsMpEng.exe	2824	ReadFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Offset: 196,396, Length: 65,536
MsMpEng.exe	2824	ReadFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Offset: 1,048,364, Length: 65,536
MsMpEng.exe	2824	ReadFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Offset: 1,113,900, Length: 65,536
MsMpEng.exe	2824	ReadFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Offset: 1,179,436, Length: 65,536
MsMpEng.exe	2824	ReadFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Offset: 1,244,972, Length: 65,536
MsMpEng.exe	2824	ReadFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Offset: 1,310,508, Length: 45,172
MsMpEng.exe	2824	QueryStandardInformationFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	AllocationSize: 1,355,776, EndOfFile: 1

Рис. 85. Этап 2. Вывод Process Monitor показывает, что Microsoft Defender читает только что извлеченный файл



Process Name	PID	Operation	Path	Result	Detail
MsMpEng.exe	2824	QueryBasicInformationFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	CreationTime: 8/10/2
MsMpEng.exe	2824	SetBasicInformationFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	CreationTime: 8/10/2
MsMpEng.exe	2824	SetDispositionInformationFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Delete: True
MsMpEng.exe	2824	QueryAttributeInformationVolume	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	FileSystemAttributes:
MsMpEng.exe	2824	CreateFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	DELETE PENDING	Desired Access: Rea
MsMpEng.exe	2824	CreateFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	DELETE PENDING	Desired Access: Rea
MsMpEng.exe	2824	CreateFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	DELETE PENDING	Desired Access: Rea
MsMpEng.exe	2824	CloseFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	
MsMpEng.exe	2824	CloseFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	
MsMpEng.exe	2824	CreateFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	NAME NOT FOUND	Desired Access: Rea

Рис. 86. Этап 3. Вывод Process Monitor показывает, что Microsoft Defender удаляет файл и проверяет его удаление

Для дальнейшего исследования рассмотрим два встроенных механизма защиты ОС — MIC и SRM. Далее будет показано, как нарушители могут использовать эти механизмы для блокировки работы процесса MD.

13.2.3. Механизм мандатного контроля целостности

13.2.3.1. Монитор безопасности и дискреционный контроль доступа

Монитор безопасности (SRM) реализует дискреционный контроль доступа (Discretionary Access Control, DAC) через токены и дескрипторы безопасности. Токен процесса содержит идентификаторы безопасности (Security Identifiers, SID), определяющие привилегии процесса. Дескрипторы безопасности объектов, таких как файлы и каталоги, включают дискреционный список управления доступом (Discretionary Access Control List, DACL), состоящий из записей управления доступом (Access Control Entry, ACE), которые указывают SID и права доступа.

Когда процесс запрашивает доступ к объекту, SRM проверяет запрос, сравнивая SID процесса с DACL объекта с использованием функции ядра nt!SeAccessCheck. Эта функция играет важную роль в модели безопасности Windows [427].

Microsoft Defender как системное приложение выполняется с повышенными привилегиями учетной записи SYSTEM. Однако начиная с Windows Vista механизм SRM был расширен за счет добавления нового механизма, который реализует мандатный контроль целостности (MIC), используя который нарушитель может отключить антивирус MD.

13.2.3.2. Мандатный контроль целостности

Механизм MIC расширяет возможности SRM, добавляя уровни целостности для объектов ОС, к которым относятся процессы, файлы и папки на диске, ветки и ключи реестра. Эти уровни ограничивают доступ процессов с низким уровнем целостности к объектам с более высоким уровнем. Windows поддерживает следующие уровни целостности:

- недоверенный (Untrusted);
- низкий (Low);
- средний (Medium);
- высокий (High);
- системный (System);
- защищенный (Protected).

Обычно системные приложения работают с уровнем целостности высокого и системного уровня, а пользовательские программы используют средние и высокие уровни целостности. Программы веб-браузеры работают с низким уровнем целостности, что обеспечивает защиту данных пользователя даже при успешных эксплуатациях процессов веб-браузеров. Когда какой-либо субъект (например, процесс) пытается взаимодействовать с целевым объектом (например, другим процессом или файлом), MIC проверяет уровень целостности инициатора и запрашиваемого (целевого) объекта.

Процесс MD запускается с системным уровнем целостности. MIC предотвращает запись или удаление объектов с более высоким уровнем целостности, даже если DACL позволяет такой доступ. Однако MIC ограничивает доступ к памяти, но не к файлам [428].

Антивирус MD осуществляет периодическое сканирование файлов на диске, памяти процессов и символьных ссылок; соответствующие функции WinAPI вызывают nt!SeAccessCheck(), как показано на рис. 87.

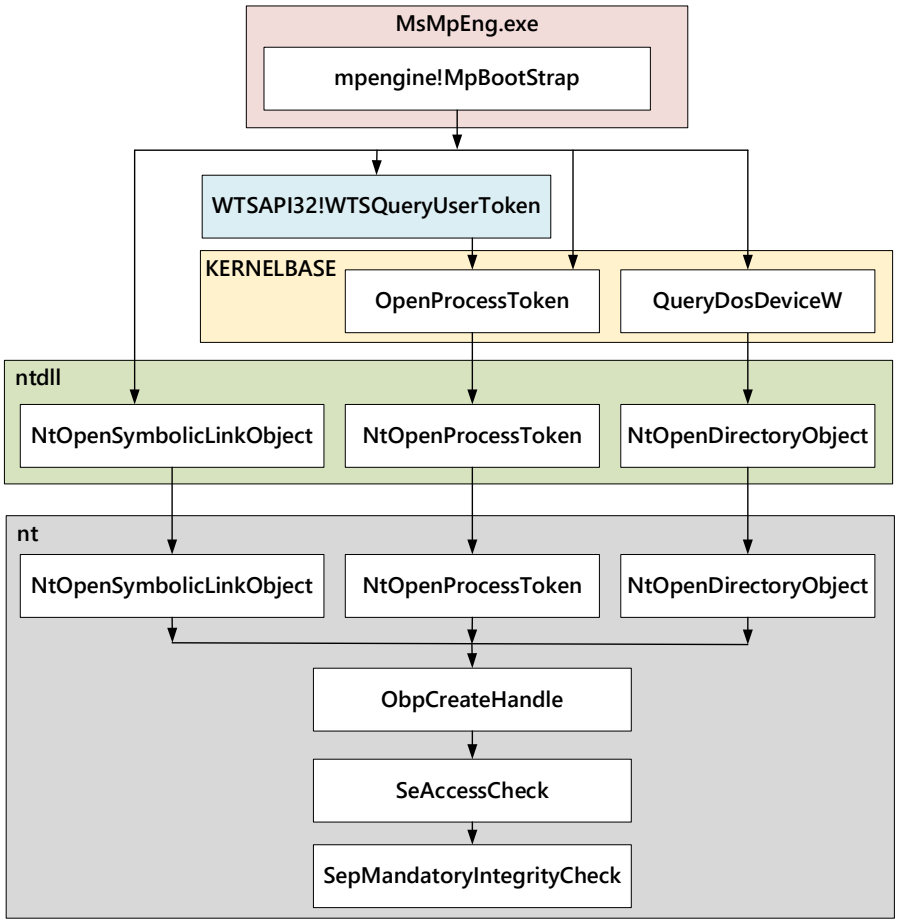


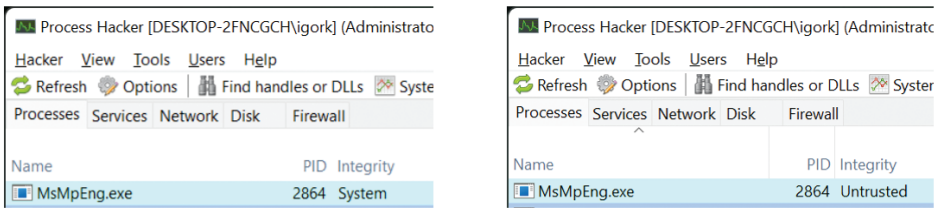
Рис. 87. Антивирус MD осуществляет сканирование памяти процессов и файлов на диске с помощью обращения к символьным ссылкам. Соответствующие функции WinAPI вызывают nt!SeAccessCheck()

13.2.3.3. Эксперимент: манипуляции с уровнем целостности

В ходе эксперимента была проведена симуляция атаки на MD путем изменения уровня целостности через модификацию поля IntegrityLevelIndex в токене процесса. Значение поля было изме-

нено на -1 или 0xFFFFFFFF, что понизило уровень целостности процесса MD до минимального значения:

- до модификации MD обладал системным уровнем целостности (рис. 88, *а*);
- после модификации уровень целостности был понижен (рис. 88, *б*).



а *б*

Рис. 88. Уровень целостности Procmon:
а — до исправления поля IntegrityLevelIndex; *б* — после исправления

После уменьшения уровня целостности были выполнены восстановления исходных настроек виртуальной среды для тестирования и повторная попытка извлечения Mimikatz из архива и его запуска:

- антивирус MD утратил возможность удалять вредоносные файлы;
- запуск извлеченного файла был заблокирован с кодом ошибки 0xC0000906. Этот код был инициирован драйвером WdFilter.sys, входящим в состав MD. Соответствующий лог файловой активности представлен на рис. 89.

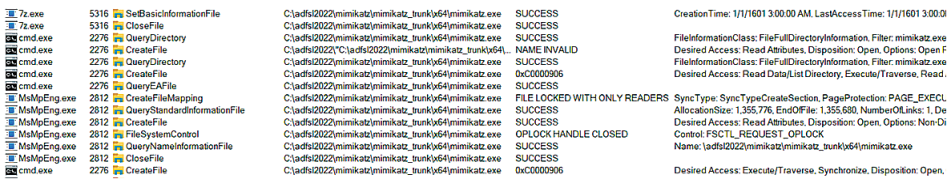


Рис. 89. Выходные данные Process Monitor показывают, что cmd не может открыть файл mimikatz со статусом ошибки 0xC0000906

Эксперимент показал, что в результате изменения уровня целостности через модификацию поля IntegrityLevelIndex антивирус MD утратил возможность удалять вредоносные файлы

с диска, однако запуск извлеченного файла остается заблокированным. ОС возвращает код ошибки 0xC0000906, генерируемый драйвером WdFilter.sys, который фильтрует файловые операции и предотвращает запуск ВПО.

13.2.3.4. Атака на токен: сброс привилегии отладки для процесса

Антивирус MD осуществляет проверку содержимого памяти всех запущенных процессов, используя функции WinAPI. Эти вызовы включают функции OpenProcess() и OpenThread(), как показано на рис. 90.

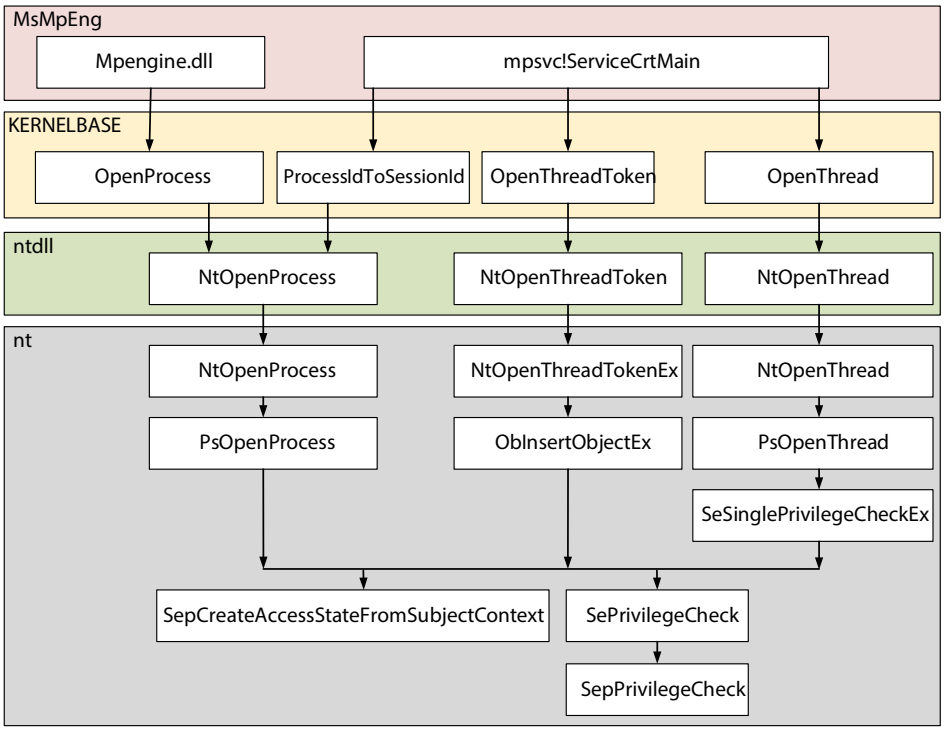


Рис. 90. Defender инициирует вызовы функций SepPrivilegeCheck() и SepCreateAccessStateFromSubjectContext() во время доступа к запущенным процессам

Нарушители могут заблокировать доступ антивируса MD к памяти процессов путем изменения привилегий процесса антивируса. Привилегии процесса хранятся в структуре _SEP_

TOKEN_PRIVILEGES, которая является частью структуры EPROCESS. Эта структура содержит поля, определяющие активные привилегии процесса.

Основные шаги атаки:

- сброс привилегии отладки: удаление бита SeDebugPrivilege из активных привилегий. Эта привилегия позволяет процессу считывать и изменять содержимое памяти других процессов;
- модификация поля Enabled: изменение значения в поле EPROCESS.Token. Privileges.Enabled, которое определяет текущие активные привилегии процесса.

В результате экспериментов установлено, что сброс бита SeDebugPrivilege и изменение поля Enabled блокируют антивирус MD, лишая его доступа к памяти процессов и файловой системе. Это позволяет нарушителям препятствовать работе MD без завершения его процесса.

Данная атака демонстрирует уязвимость антивируса MD к модификации системных структур, что подчеркивает необходимость усиления механизмов защиты привилегий процесса.

13.2.3.5. Экспериментальная симуляция атаки на антивирус Microsoft Defender

Антивирус MD предотвращает распространение ВПО, используя два основных этапа:

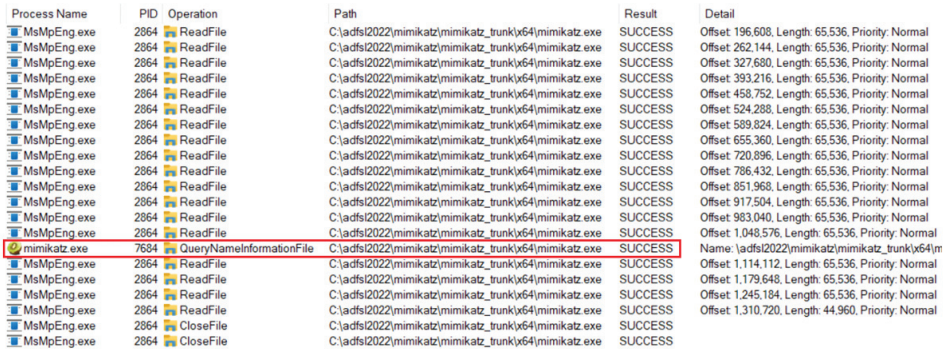
1) проверка содержимого файлов на диске,

2) проверка памяти запущенных процессов.

В ходе эксперимента была выполнена симуляция атаки на MD путем изменения полей в структуре EPROCESS.Token:

- поле IntegrityLevelIndex изменено на значение 0xFFFFFFFF или -1, что соответствует минимальному уровню привилегий;
- поле Privileges. Enabled изменено, сброшен бит SeDebugPrivilege.

Эти манипуляции оказались достаточны для успешного извлечения и запуска ВПО. Выходные данные Process Monitor показали, что процесс Mimikatz был успешно запущен без какой-либо реакции со стороны Microsoft Defender. Это продемонстрировано на рис. 91.



Process Name	PID	Operation	Path	Result	Detail
MsMpEng.exe	2864	ReadFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Offset: 196,608, Length: 65,536, Priority: Normal
MsMpEng.exe	2864	ReadFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Offset: 262,144, Length: 65,536, Priority: Normal
MsMpEng.exe	2864	ReadFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Offset: 327,680, Length: 65,536, Priority: Normal
MsMpEng.exe	2864	ReadFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Offset: 393,216, Length: 65,536, Priority: Normal
MsMpEng.exe	2864	ReadFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Offset: 458,752, Length: 65,536, Priority: Normal
MsMpEng.exe	2864	ReadFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Offset: 524,288, Length: 65,536, Priority: Normal
MsMpEng.exe	2864	ReadFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Offset: 589,824, Length: 65,536, Priority: Normal
MsMpEng.exe	2864	ReadFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Offset: 655,360, Length: 65,536, Priority: Normal
MsMpEng.exe	2864	ReadFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Offset: 720,896, Length: 65,536, Priority: Normal
MsMpEng.exe	2864	ReadFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Offset: 786,432, Length: 65,536, Priority: Normal
MsMpEng.exe	2864	ReadFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Offset: 851,968, Length: 65,536, Priority: Normal
MsMpEng.exe	2864	ReadFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Offset: 917,504, Length: 65,536, Priority: Normal
MsMpEng.exe	2864	ReadFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Offset: 983,040, Length: 65,536, Priority: Normal
MsMpEng.exe	2864	ReadFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Offset: 1,048,576, Length: 65,536, Priority: Normal
mimikatz.exe	7684	QueryNameInformationFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Name: \adfsl2022\mimikatz\mimikatz_trunk\x64\m
MsMpEng.exe	2864	ReadFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Offset: 1,114,112, Length: 65,536, Priority: Normal
MsMpEng.exe	2864	ReadFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Offset: 1,179,648, Length: 65,536, Priority: Normal
MsMpEng.exe	2864	ReadFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Offset: 1,245,184, Length: 65,536, Priority: Normal
MsMpEng.exe	2864	ReadFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	Offset: 1,310,720, Length: 44,960, Priority: Normal
MsMpEng.exe	2864	CloseFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	
MsMpEng.exe	2864	CloseFile	C:\adfsl2022\mimikatz\mimikatz_trunk\x64\mimikatz.exe	SUCCESS	

Рис. 91. Выходные данные Process Monitor подтверждают успешный запуск приложения Mimikatz без завершения процесса MD (MsMpEng.exe)

Эксперимент показал, что нарушители могут запускать даже известное ВПО, предварительно отключив MD без срабатывания механизмов самозащиты Windows. Атака была протестирована с использованием проекта Metasploit [34], а также на нескольких популярных антивирусных решениях.

Для оценки универсальности атаки была проведена проверка с использованием ряда популярных антивирусных программ. Результаты представлены в табл. 30. По соображениям профессиональной этики названия протестированных антивирусов были обезличены.

Таблица 30

Сводная таблица атакованных антивирусных решений

Название антивирусного ПО	Способность антивируса обнаруживать вредоносные файлы на диске	Способность антивируса обнаруживать вредоносные процессы в памяти	Результат атаки
Microsoft Defender	Не способно	Не способно	Атака успешна
Антивирус 1	Не способно	Не способно	
Антивирус 2	Не способно	Не способно	
Антивирус 3	Не способно	Не способно	
Антивирус 4	Не способно	Не способно	
Антивирус 5	Не способно	Способно	Атака неуспешна

Окончание табл. 30

Название антивирусного ПО	Способность антивируса обнаруживать вредоносные файлы на диске	Способность антивируса обнаруживать вредоносные процессы в памяти	Результат атаки
Антивирус 6	Способно обнаружить, но не может удалить вредоносные файлы	Не способно	

Эксперимент продемонстрировал, что большинство сторонних антивирусных решений уязвимы к модификации привилегий процесса, что позволяет запускать вредоносное ПО. Эти результаты подчеркивают необходимость улучшения механизмов защиты привилегий процесса в современных антивирусных решениях.

13.3. МОДИФИКАЦИЯ ГИПЕРВИЗОРА MEMORYRANGER ДЛЯ ЗАЩИТЫ СТРУКТУР MICROSOFT DEFENDER

Для предотвращения атак на антивирус MD, основанных на несанкционированном доступе к памяти ядра, была проведена модификация гипервизора MemoryRanger (MR) [304]. Гипервизор MemoryRanger благодаря своей архитектуре и использованию технологий аппаратной виртуализации способен защищать ядро ОС от атак драйверов.

Напомним, что MemoryRanger состоит из двух основных компонентов:

1) *драйвер режима ядра*, который регистрирует функции обратного вызова для отслеживания событий (загрузка/выгрузка драйверов, создание/завершение процессов) и передает информацию о событиях гипервизору;

2) *гипервизор*, использующий технологию EPT для создания изолированных областей памяти (анклавов) с индивидуальной конфигурацией доступа, предназначенных для выполнения драйверов ядра, предотвращая их взаимодействие с защищаемыми структурами ОС.

Для защиты структур ядра антивируса MD в системе MR были внесены следующие обновления:

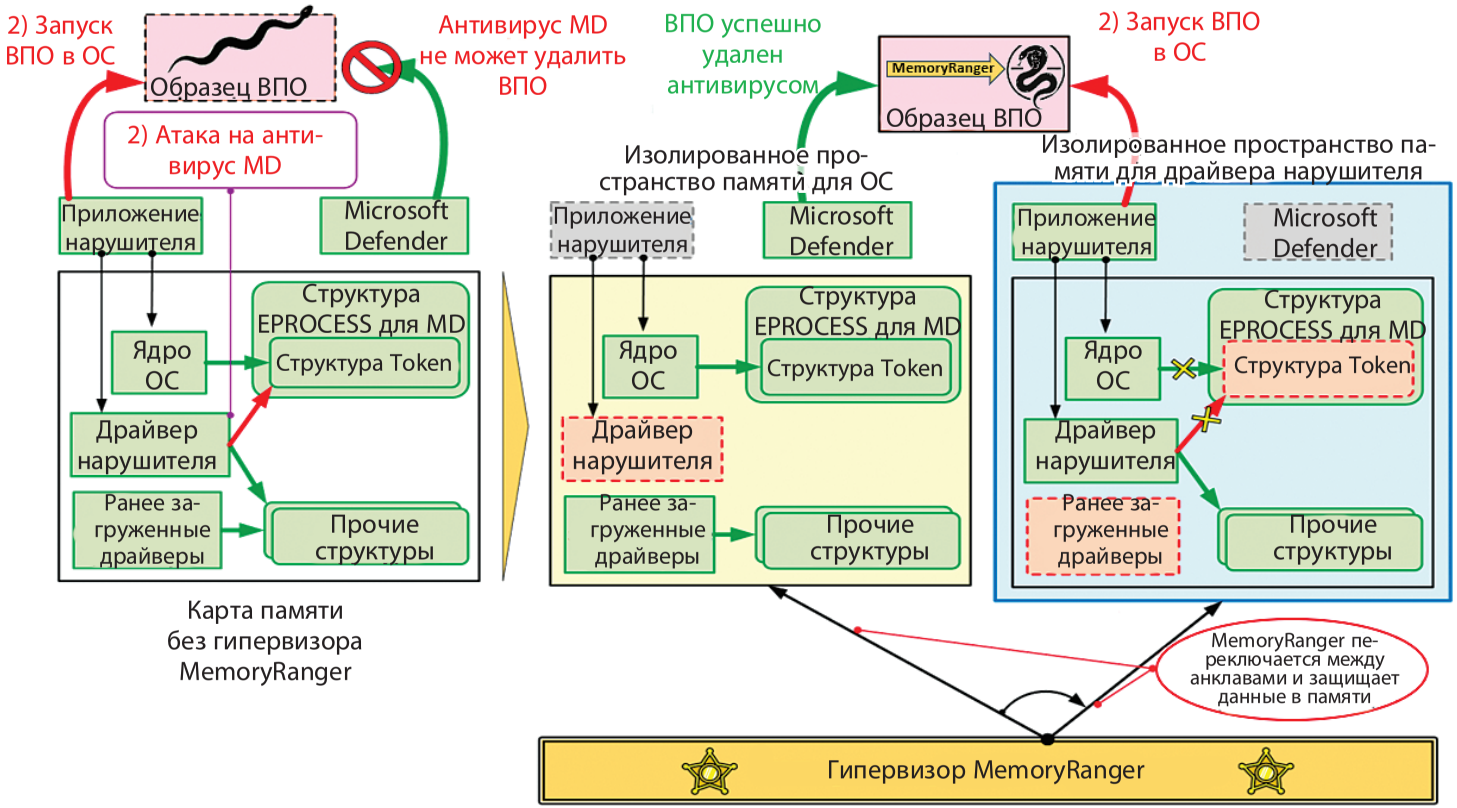


Рис. 92. MemoryRanger блокирует попытки изменить значения полей структуры Token для антивируса MD с использованием двух изолированных областей памяти: первой для загруженных ранее драйверов и второй для загруженного драйвера нарушителя

- *поиск структур EPROCESS*. Драйвер MR производит поиск адресов структур EPROCESS для процесса антивируса MD, чтобы идентифицировать и контролировать заданные области памяти;
- *определение защищаемых областей памяти*. Драйвер MR определяет, какие именно области подлежат защите, и указывает их адреса и размеры. К таким областям относятся: Token. IntegrityLevelIndex (4 байта), Token. Privileges.Enabled (8 байт);
- *создание изолированных областей памяти*. Гипервизор MR получает уведомления от драйвера MR о вновь загруженных драйверах. Для каждого нового драйвера гипервизор формирует отдельную изолированную область памяти (анклав), в которой ограничен доступ к указанным выше структурам ядра.

Данный механизм предотвращает несанкционированное вмешательство, он показан на рис. 92.

Эксперименты подтвердили, что обновленный гипервизор MR блокирует атаки на структуры ядра MD. Использование изолированных областей памяти позволило предотвратить модификацию полей Token, что сохраняет работоспособность антивируса даже при загрузке драйверов нарушителей.

13.4. ПРАКТИЧЕСКОЕ ПРИМЕНЕНИЕ МОДИФИЦИРОВАННОГО MEMORYRANGER: ПРЕДОТВРАЩЕНИЕ АТАК НА MICROSOFT DEFENDER

13.4.1. Экспериментальное исследование работоспособности MemoryRanger

В ходе исследования модифицированный MemoryRanger использовался для защиты ряда структур, задействованных в работе Microsoft Defender на уровне ядра. В качестве аппаратной платформы использовался компьютер с процессором Intel i9-11900 и 64 Гб ОЗУ; на нем была установлена Windows 10 (версия 20H2, сборка 10.0.19044) x64. Параллельно в среде VMware Workstation запускалась виртуальная машина с четырьмя логическими ядрами и 16 Гб ОЗУ. Виртуальная машина работала под управлением Windows 11 (версия 20H2, сборка 10.0.22000) x64.

Модифицированный MemoryRanger активировался в гостевой системе (VM) для защиты структур ядра Microsoft Defender. В рамках эксперимента имитировалась атака, при которой нарушитель пытается отключить антивирус путем модификации (патчинга) соответствующих полей в структурах ядра, ответственных за контроль привилегий и уровня целостности.

В результате эксперимента установлено, что модифицированный MemoryRanger успешно предотвращает данную атаку, ограничивая доступ драйверов к защищенным областям памяти. Эксперименты проводились под нагрузкой, чтобы оценить влияние механизма изоляции на производительность системы. Полученные данные свидетельствуют о том, что применение гипервизора MemoryRanger не приводит к существенному снижению производительности.

13.4.2. Преимущества обновленного MemoryRanger

В современных операционных системах (Windows, UNIX-подобных) драйверы ядра используют общее адресное пространство совместно с самим ядром, что ускоряет их взаимодействие, но создает риски компрометации системы со стороны уязвимых или вредоносных драйверов. Для предупреждения этих рисков *Microsoft* разработала технологию Hyper-V с разделением на режимы VTL0 (Normal Mode) и VTL1 (Secure Mode), где защищаемые компоненты (например, skci.dll и cng.sys) переносятся в изолированное пространство. Однако такая схема не охватывает сторонние драйверы, выделенную ими память, широкий спектр системных структур и поддерживает лишь два уровня изоляции.

Модифицированный MemoryRanger, напротив, динамически формирует отдельные, изолированные области памяти (анклавы) для каждого загружаемого драйвера, позволяя тонко настраивать политику доступа к памяти.

Исследования показали, что увеличение количества анклавов может привести к некоторому росту накладных расходов (см. главу 12). В ходе эксперимента с отключением Microsoft Defender MemoryRanger успешно блокировал попытки понижений привилегий антивируса, изменений полей Token, предотвращая блокировку работы защитного решения.

Выводы

В современных операционных системах, таких как Windows, драйверы ядра совместно используют единое адресное пространство с ядром ОС. Это обеспечивает высокую производительность, но создает угрозы, которые активно реализуются со стороны ВПО. Нарушители активно используют уязвимости в драйверах для проведения атак, направленных на компрометацию данных и механизмов безопасности ОС. Примеры атак, таких как Turla и Project Sauron, показывают, что современные угрозы не только сложны в обнаружении, но и способны оставаться активными в течение нескольких лет.

Существующие защитные механизмы Windows, такие как PatchGuard и цифровая подпись драйверов, часто оказываются недостаточными для предотвращения современных атак. Нарушители могут противодействовать этим механизмам защиты путем изменения данных ядра, внедрения вредоносного кода и манипуляций с привилегиями процессов, что позволяет им противодействовать антивирусам без завершения антивирусных процессов.

Рассмотренные в данном разделе решения MemoryMonRWX, AllMemPro и MemoryRanger представляют собой новые инструменты защиты, основанные на изоляции и мониторинге защищаемых областей памяти ядра.

Архитектура MemoryRanger предусматривает выделение отдельных изолированных анклавов для каждого драйвера, ограничивая доступ к заданным структурам ядра и препятствуя атакам на структуры Token Microsoft Defender, включая изменение привилегий и параметры Integrity Level. Кроме того, модифицированная версия MemoryRanger не только предотвращает атаки на файловые объекты (Handle Hijacking, NTFS structures hijacking), но и выявляет новый тип угроз — токен-хайджекинг (Token Hijacking), при котором нарушители копируют SID и их атрибуты из процессов с повышенными привилегиями. Для противодействия подобным приемам выделен дополнительный анклав с защищаемыми данными ядра, куда прочие драйверы не имеют доступа.

Проект MemoryRanger демонстрирует на практике, что изоляция драйверов в отдельных гипервизорных анклавах может превосходить возможности штатных механизмов Windows (например, Virtual Secure Mode). Тестирование проводилось на Windows 10 x64 (версия 1903). Кроме того, показана потенциальная применимость MemoryRanger и на Unix-системах с архитектурами *AMD* и *ARM*. Подобные решения представляют собой важное направление в развитии технологий информационной безопасности, повышая защищенность ядра ОС перед современными атаками и закладывая фундамент для дальнейшего совершенствования систем защиты информации.

Раздел 3
МАТЕМАТИЧЕСКИЕ ОСНОВЫ ОБНАРУЖЕНИЯ СКРЫТЫХ ГИПЕРВИЗОРОВ В УСЛОВИЯХ ПРОТИВОДЕЙСТВИЯ

If you think technology can solve your security problems, then you don 't understand the problems and you don 't understand the technology.

Bruce Schneier,
Secrets and Lies: Digital Security in a Networked World, 2000

Если вы думаете, что технологии решат Ваши проблемы информационной безопасности, значит, Вы не постигли ни сути проблем, ни принципов работы самих технологий.

Брюс Шнайер
«Секреты и ложь: безопасность данных в цифровом мире», 2000

Глава 14
ВВЕДЕНИЕ В ТЕХНОЛОГИЮ ВИРТУАЛИЗАЦИИ И ПРИМЕРЫ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ, ИСПОЛЬЗУЮЩЕГО ТЕХНОЛОГИЮ АППАРАТНОЙ ВИРТУАЛИЗАЦИИ

В последние десятилетия виртуализация стала важным элементом современных информационных технологий, позволяя эффективно распределять вычислительные ресурсы и обеспечивать надежную изоляцию между различными средами исполнения.

Появление технологии аппаратной виртуализации открыло возможность запускать программный код на более привилегированном уровне, чем сама ОС (см. параграф 1.2). Такой код обычно называют монитором виртуальных машин, или гипервизором. Первые процессоры с поддержкой аппаратной виртуализации (Intel VT-x, ранее известная как Vanderpool) появились в 2005 г. *AMD* представила свою технологию аппаратной виртуализации (AMD-V, ранее Pacifica) в 2006 г. с выпуском процессоров семейства Athlon 64. В последующие годы компании совершенствовали свои решения: в 2008 г. *Intel* начала внедрять расширенные функции, такие как Extended Page Tables, а *AMD* — Rapid Virtualization Indexing (Nested Page Tables). Этот этап стал временем массового внедрения и расширения возможностей аппаратной виртуализации в процессорах для серверных и настольных платформ.

Данный раздел посвящен анализу вопросов обеспечения безопасности гипервизоров, работающих на платформе *Intel*.

Гипервизор получил возможность контролировать различные события, происходящие в ОС, например: попытки обращения к памяти, выполнение определенных инструкций и изменение содержимого регистров процессора. Список перехватываемых событий состоит из двух частей: безусловно перехватываемые события и события, для перехвата которых необходимо задавать настройки при запуске гипервизора.

Применение гипервизоров существенно расширило возможности различных системных утилит и средств защиты информации, таких как средства отладки (например, проект HyperDbg), средства поиска уязвимостей с использованием фаззинга (fuzzing), антивирусные системы и комплексы систем защиты типа «песочница» (Sandbox) для проведения динамического анализа программного кода.

Однако технология аппаратной виртуализации открыла новые возможности и для нарушителей. Гипервизоры могут быть использованы для защиты вредоносных драйверов от обнаружения, оставаясь при этом невидимыми для большинства средств защиты, работающих на уровне ОС. Также гипервизоры могут потенциально использоваться как самостоятельные модули

для скрытного управления системой, внедрения кода и других несанкционированных действий. Эксперты по всему миру исследуют возможности по созданию прототипов гипервизоров, а также алгоритмов, методов и средств для их обнаружения, в том числе в условиях противодействия со стороны нарушителя. Среди прототипов гипервизоров можно выделить следующие проекты:

- проект Blue Pill, разработанный командой *Invisible Things Lab* и представляющий собой Windows-драйвер для систем *AMD*;
- проект Vitriol, разработанный командой *Matasano Security* и представляющий собой драйвер MAC OS для *Intel*-систем;
- прототип гипервизора, загружающийся из BIOS и исследованный в рамках доклада «Malicious Hypervisor Threat» [429];
- проекты с открытым исходным кодом: HyperPlatform, Simplevisor, hvpp, KSM.

В случае технологии аппаратной виртуализации процессоров *Intel* отсутствуют штатные средства для обнаружения гипервизоров. Задача обнаружения гипервизоров усложняется тем, что они могут противодействовать своему обнаружению, а также быть загружены в систему во вложенном виде, другими словами — «под прикрытием» легитимного гипервизора. На рис. 93 представлена схема работы ОС до загрузки гипервизоров и в случае наличия двух вложенных гипервизоров — вредоносного и легитимного.

Обнаружение скрытых гипервизоров — задача чрезвычайно сложная, особенно в условиях активного противодействия со стороны нарушителей. Экспертами по безопасности разработано множество методов для выявления гипервизоров. Одна из наиболее популярных техник обнаружения основана на измерении длительности выполнения инструкций, которые безусловно перехватываются гипервизором, таких как CPUID (CPU Identification) и VMCALL (Virtual Machine Call). В случае присутствия гипервизора длительность выполнения таких инструкций оказывается на порядок выше, чем в случае его отсутствия; в таком случае принять однозначное решение о наличии гипервизора в системе не составляет труда. Однако нарушители могут преднамеренно компрометировать счетчик тактов процес-

сора (TSC) и изменять его значения так, что обнаружить гипервизор по среднему значению длительности выполнения счетчика будет невозможно. Принятие решения о том, что найденный гипервизор является нелегитимным или вредоносным, может быть после проведения его реверс-инжинирингового анализа.

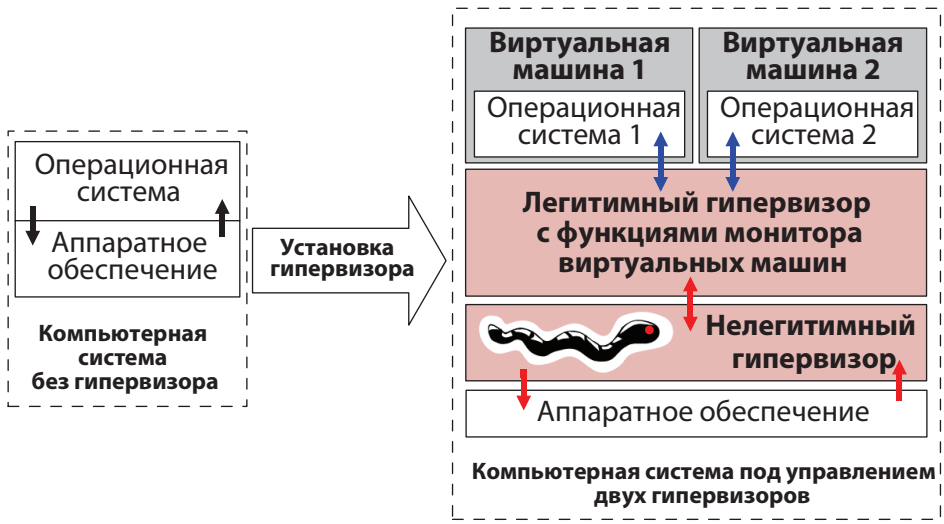


Рис. 93. Схема взаимодействия ОС с аппаратным обеспечением в случаях отсутствия (присутствия) двух гипервизоров: легитимного с функциями МВМ и нелегитимного, находящихся во вложенном виде

В этом разделе рассматриваются математические основы разработки и применения методов для обнаружения скрытых гипервизоров в условиях противодействия. Основное внимание уделяется моделям и алгоритмам, позволяющим анализировать системные метрики и выявлять признаки наличия гипервизора. Цель — предоставить системное понимание методов и моделей, способных выявлять скрытые гипервизоры, в условиях применения нарушителями активных мер по противодействию обнаружению.

14.1. МЕТОДЫ ОБНАРУЖЕНИЯ ГИПЕРВИЗОРА

В настоящее время встроенные механизмы обнаружения гипервизоров для процессоров *Intel* отсутствуют. Существующее встроенное средство обнаружения для процессоров *AMD* имеет

уязвимости, которые могут быть использованы гипервизорами для сокрытия от обнаружения. Поэтому исследователи активно занимаются разработкой новых методов для решения данной проблемы.

История разработки этих методов берет свое начало в 2007 г., вскоре после того как J. Rutkowska в 2006 г. представила первый гипервизорный руткит под названием Blue Pill [25]. Этот гипервизор поддерживал процессоры *AMD* и запускался с помощью драйвера Windows. Почти одновременно исследователь Dino Dai Zovi разработал аналогичный гипервизор для процессоров *Intel* и операционной системы Mac OS, названный Vitriol [430]. Благодаря Blue Pill и Vitriol появилось множество исследований и практических разработок в области информационной безопасности. В исследованиях [431, 432] проведен сравнительный анализ руткитов. В работе [432] исследованы современные руткиты-гипервизоры, которые представляют собой наиболее сложный и опасный тип угроз. Они используют возможности гипервизоров для скрытия своего присутствия и воздействия на ОС и приложения. Статья изучает существующие подходы к обнаружению гипервизоров и анализирует их результативность в различных сценариях, включая методы, основанные на анализе длительности выполнения инструкций (таймингов), особенностях взаимодействия с регистрами процессора и выявлении других изменений, вызванных присутствием гипервизоров.

На рис. 94 приведена разработанная автором данной монографии классификация известных методов и подходов к обнаружению гипервизоров. Все методы разделены на четыре большие группы: 1) методы, основанные на сигнатурах; 2) методы, основанные на поведении; 3) методы, использующие доверенный гипервизор; 4) методы, применяющие временной анализ. Обнаружение на основе сигнатур включает сканирование памяти на наличие характерных шаблонов гипервизоров. Три других подхода основываются на анализе взаимодействия с гипервизором.

Рис. 94. Классификация методов обнаружения гипервизоров

14.2. СИГНАТУРНЫЕ МЕТОДЫ ОБНАРУЖЕНИЯ

Сигнатура представляет собой последовательность байтов в памяти, указывающую на присутствие гипервизора. Такие сигнатуры могут включать фрагменты кода обработчиков гипервизора, а также связанные с ними структуры данных. Методы обнаружения на основе сигнатур заключаются в поиске этих последовательностей в дампах памяти [21, 25, 26, 87, 433–438].

Когда гипервизор загружен, его обработчик и связанные структуры находятся в физической памяти [435]. По результатам анализа копии памяти определяют присутствие гипервизора. Например, руткит Blue Pill выделяет области памяти (пулы) с тегами (названиями) BLPB, BLUE и BLUP. Используя эти названия и соответствующие размеры областей памяти, можно обнаружить присутствие Blue Pill [434].

Группу сигнатурных методов можно разделить на две подгруппы: программные и аппаратные, руководствуясь принципом получения дампа памяти (см. рис. 94). Программные методы используют встроенное аппаратное обеспечение для получения дампа памяти, а аппаратные методы требуют подключения дополнительных устройств для получения дампа памяти.

В статье [439] представлен обзор методов динамического анализа для выявления ВПО. Описаны как программные, так и аппаратные средства защиты. В более свежем исследовании [440] приведены современные инструменты и методы для получения и анализа энергозависимой памяти для идентификации ВПО. Этот обзор охватывает не только методы получения и анализа дампов памяти, но и применение аппарата машинного обучения для улучшения процесса обнаружения.

14.2.1. Программные методы обнаружения

Рассмотрим программные методы, которые используют обход каталогов страниц и таблиц, отображение памяти (memory mapping), анализ таблиц преобразования графических адресов, а также перепрограммирование контроллера памяти.

Метод прохода по таблицам страниц и каталогам позволяет получить копию адресного пространства, включая бинарный код драйвера гипервизора [441]. Например, при реализации гипервизора в виде драйвера ОС данный метод позволяет извлечь дамп адресного пространства, в котором будет содержаться бинарный код драйвера гипервизора [25]. Аналогичный дамп можно получить с помощью преобразования записей таблиц страниц [433].

С использованием функции ZwMapViewOfSection можно реализовать получение дампа памяти путем отображения страниц физической памяти.

Метод использования периферийных устройств реализуется посредством считывания содержимого памяти через шины FireWire и PCI [82, 434, 442–444].

В работе [445] использованы таблицы преобразования графических адресов (Graphics Address Remapping Table, GART) для получения дампа памяти.

Метод DeepWatch для обнаружения гипервизора основывается на использовании встроенного контроллера северного моста процессора. Этот метод был предложен Ю. Булыгиным из компании *Intel* [85].

Для противодействия программным способам обнаружения гипервизора могут быть применены различные средства. Например, технология защиты внешнего доступа (External Access Protection) от компании *AMD* помогает маскировать сегменты памяти ПО при взаимодействии с периферийными устройствами [86], а технологии Shadow Page Tables (*AMD*) и Extended Page Tables (*Intel*), использующие механизмы скрытых страниц памяти, могут предотвратить обнаружение вредоносного ПО по сигнатурам [59].

14.2.2. Аппаратные методы обнаружения

Современные аппаратные методы для получения дампа памяти предоставляют исследователям важные инструменты для анализа систем безопасности и цифровой криминалистики. Эти методы позволяют обойти многие программные ограничения, используемые для защиты данных, и обеспечивают низкоуровневый доступ к памяти. При этом они часто являются устойчивыми к противодействию со стороны гипервизора, что делает их востребованными инструментами в арсенале специалистов по безопасности.

В период с 2005 по 2010 г. широко использовались аппаратные средства Tribble, Copilot и RAM Capture Tool [87] для получения дампа памяти. Специальная плата Tribble подключалась к компьютеру для проведения криминалистических исследований [446]. Другое аппаратное средство Copilot (разработка компании *Komoku*) также было основано на PCI-карте и использовалось для мониторинга состояния памяти компьютера. В марте 2008 г. компания *Microsoft* приобрела *Komoku*, и с тех пор информация о достижениях Copilot больше не публикуется.

В настоящее время широко используются следующие аппаратные средства.

PCILeech FPGA — расширение инструментария PCILeech для получения доступа к памяти (DMA), основанное на FPGA

(Field-Programmable Gate Array). FPGA-устройства предоставляют полный доступ к 64-битному пространству физической памяти без необходимости использования модулей ядра на целевой системе. Они также обеспечивают возможность отправки PCIe пакетов (TLPs) и поддерживают подключения через USB-C и Thunderbolt.

Проект **Inception** также использует DMA для извлечения дампов памяти через интерфейсы, такие как FireWire или Thunderbolt. Этот инструмент применяется для анализа памяти, когда необходимо обойти защитные механизмы ОС.

Хотя аппаратные методы обладают устойчивостью к противодействию со стороны гипервизоров, они остаются неудобными в массовом использовании и применимы исключительно в специализированных лабораториях. Кроме того, существуют программные методы, которые могут предотвратить считывание памяти даже с использованием аппаратных средств.

14.3. ПОВЕДЕНЧЕСКИЕ МЕТОДЫ ОБНАРУЖЕНИЯ

Поведенческие методы обнаружения гипервизоров основываются на анализе различий в поведении вычислительных систем в ситуациях отсутствия гипервизора и его наличия. Эти методы используют выявленные аномалии, вызванные внедрением и работой гипервизора, путем наблюдения за характерными изменениями в функционировании системы. Поведенческие методы обнаружения гипервизоров подразделяются на методы, использующие буфер ассоциативной трансляции, и методы, применяющие ошибки в процессорах.

14.3.1. Метод обнаружения на основе буфера ассоциативной трансляции (Translation Lookaside Buffer)

Буфер ассоциативной трансляции (БАТ, Translation Lookaside Buffer, TLB) — это кэш памяти, используемый процессором для ускорения трансляции виртуальных адресов в физические [25, 433, 447] благодаря хранению информации о парах «виртуальный адрес — физический адрес», к которым было недавно обращение. Когда гипервизор активен, он влияет на управление памятью и изменяет содержимое БАТ. Например, каждый раз

при передаче управления из виртуальной машины гипервизору происходит принудительная очистка БАТ, называемая TLB shootdown; после возврата управления в ОС требуется некоторое время для повторного заполнения этого кэш-буфера.

Поведенческие методы обнаружения гипервизоров с использованием буфера ассоциативной трансляции БАТ описаны в работах [59, 84, 434, 437, 448]. В основе этих методов лежит различие в адресации памяти. Процесс обнаружения гипервизора с использованием методов на основе БАТ начинается с подготовки памяти и ее заполнения определенными значениями. Эти данные затем заносятся в БАТ для оптимизации трансляции адресов. На первом этапе выделяется участок памяти, который заполняется заданными значениями байтов. Информация о виртуальных (ВА) и физических адресах (ФА) этой области сохраняется в БАТ, чтобы обеспечить ускоренную трансляцию адресов при последующих обращениях (рис. 95). Этот этап необходим для установления предсказуемого состояния системы перед началом тестирования.

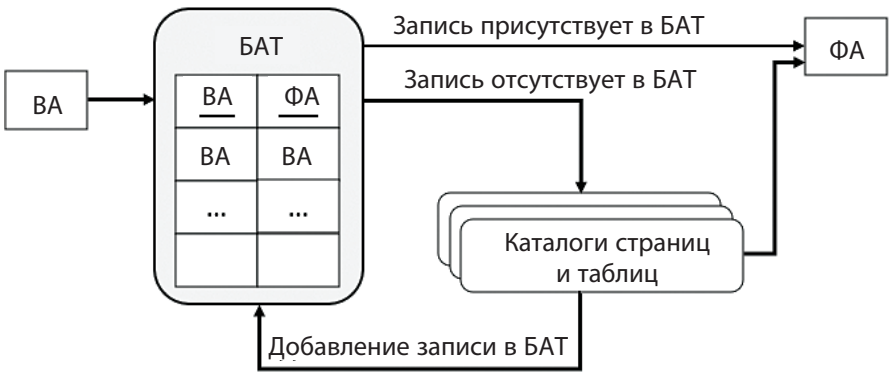


Рис. 95. Заполнение БАТ при обращении к памяти

На втором этапе происходит изменение записей в каталоге страниц и таблицах, которые используются для трансляции виртуальных адресов в физические. В результате этих изменений при обращении к виртуальным адресам будет использоваться новый набор физических адресов, отличающийся от первоначально выделенных (рис. 96).

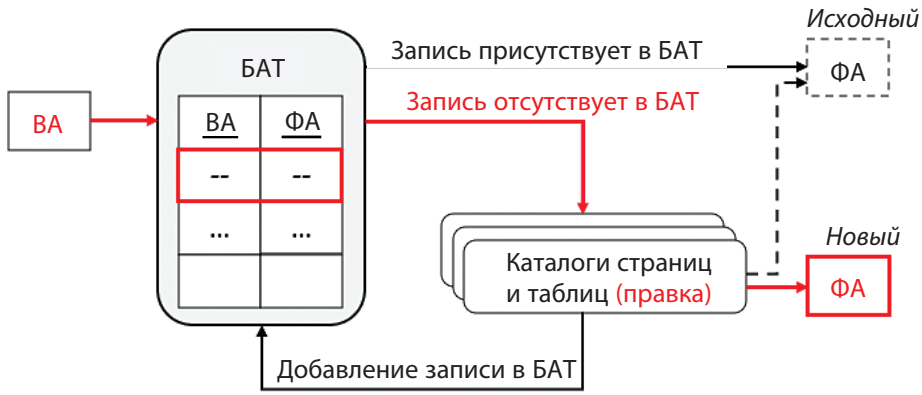


Рис. 96. Обращение к памяти в случае присутствия гипервизора

На третьем этапе инициируется демаскирующее событие, предназначенное для передачи управления гипервизору.

На четвертом этапе считываются данные из той же области памяти, которая была инициализирована на первом этапе. Если гипервизор отсутствует, то БАТ не очищается и трансляция адресов осуществляется с использованием данных в буфере БАТ, поэтому обращение происходит к ранее выделенной области памяти (рис. 97). Если же данные отличаются, то это указывает на присутствие гипервизора, поскольку его вмешательство изменило поведение системы.

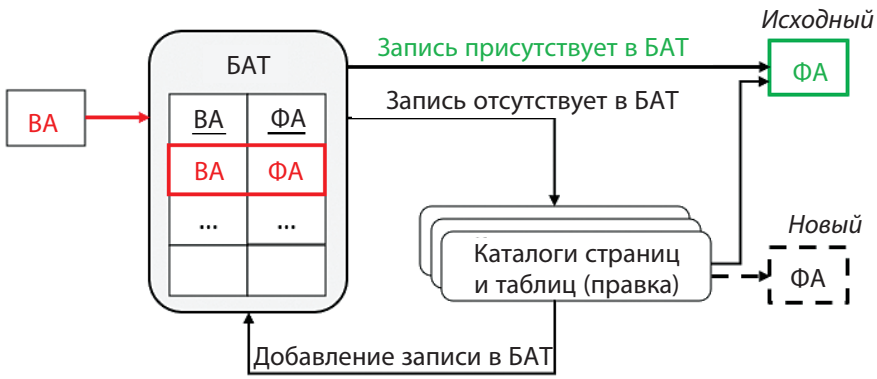


Рис. 97. Обращение к памяти в случае отсутствия гипервизора

Этот метод работоспособен на сравнительно старых моделях процессоров *Intel* и применяется для выявления гипервизоров,

которые не используют техники сокрытия. Современные процессоры поддерживают новые поля БАТ, такие как Address Space Identifier (ASID) и Process-Context Identifier (PCID), что делает данный метод непригодным для использования. В связи с этим применение метода ограничивается отдельными специфическими случаями старых платформ.

14.3.2. Методы обнаружения гипервизоров на основе ошибок в процессорах и реализациях гипервизоров

Методы выявления гипервизоров, основывающиеся на ошибках в процессорах, используют выявленные уязвимости и недочеты реализации технологии аппаратной виртуализации. Такие дефекты проявляются при выполнении специфических инструкций и служат индикаторами присутствия гипервизоров [25, 436].

Одним из известных примеров является аппаратная ошибка, обозначенная как Erratum 140 в официальном отчете компании *AMD* [433].

Данная ошибка связана с некорректной работой инструкции RDMSR 10h, которая используется для чтения значения счетчика тактов процессора (TSC). В случае наличия гипервизора инструкция RDTSC возвращает значение, скорректированное на основе поля TSC_OFFSET из структуры управления виртуальной машиной (VMCS — Virtual Machine Control Structure). Однако инструкция RDMSR 10h возвращает реальное значение счетчика тактов без учета этого поля. Поэтому можно определить присутствие гипервизора в системе, если значения, прочитанные с помощью этих двух инструкций, не совпадают. Метод применим к некоторым устаревшим моделям процессоров *AMD*.

Другим примером аппаратной ошибки является ошибка, связанная с инструкцией VMSAVE 0x67, описанная в докладе [449]. При наличии гипервизора выполнение инструкции VMSAVE с префиксом 0x67 вызывает системный сбой и остановку работы системы, в то время как при отсутствии гипервизора команда выполняется без прерывания работы системы. Этот метод, однако, нарушает нормальную работу компьютера и поэтому пригоден только для лабораторных исследований.

Кроме того, существуют методы, основанные на ошибках в обработке определенных инструкций самим гипервизором. Например, в работе [434] предложен метод обнаружения гипервизора Blue Pill на основе использования «гипервызовов», или Bpknock. Было обнаружено, что при записи в регистр EAX значения 0xbabecafe и выполнении инструкции CPUID в случае наличия гипервизора в EAX записывается значение 0x69696969. При отсутствии гипервизора в системе EAX будет содержать ноль. Сравнение значений регистра позволяет определить наличие гипервизора Blue Pill.

В некоторых случаях обнаружение гипервизоров возможно путем анализа отладочных сообщений, которые гипервизор может выводить, например, с помощью функции DbgPrint. Следует отметить, что данные ошибки могут быть результатом непреднамеренных упущений в процессе разработки и недостаточного контроля качества кода гипервизора.

Адаптация методов обнаружения гипервизоров на основе ошибок в аппаратном и программном обеспечении к современным системам требует углубленных исследований и поиска новых уязвимостей, что нетривиально.

14.4. ОБНАРУЖЕНИЕ УГРОЗ НА ОСНОВЕ ДОВЕРЕННОГО МОНИТОРА ВИРТУАЛЬНЫХ МАШИН

Методы обнаружения на основе доверенного монитора виртуальных машин (MBM), или доверенного гипервизора, — один из возможных подходов в области обеспечения безопасности и предотвращения несанкционированного доступа к системам. В данном случае предполагается, что доверенный гипервизор получает управление приоритетным образом при загрузке системы, например, используя возможности BIOS (UEFI).

Эти методы активно используются в ряде коммерческих продуктов и программных прототипов, таких как McAfee DeepSAFE, HyperSafe [207], Hypersight Rootkit Detector [450], Красная пилюля [451] и Symantec Endpoint Protection [452]. Они также активно обсуждаются в научных исследованиях [207, 453, 454]. Современные системы защиты типа sandbox (системы класса

«песочница») основаны на загрузке доверенного гипервизора перед запуском потенциально вредоносного образца.

Доверенный MBM перехватывает обращения к памяти и процессорные инструкции, а также регулирует запуск новых гипервизоров. Пользователь может быть уведомлен о подобных попытках и получает возможность принять решение о разрешении или запрете запуска гипервизора.

Технологии Trusted Execution Technology (TXT) компании *Intel* и Secure Extension Mode (SEM) компании *AMD* позволяют реализовывать доверенные MBM с помощью специального аппаратного модуля, называемого Trusted Platform Module (TPM) [60, 61, 455–457]. TPM предоставляет дополнительные средства для обеспечения доверенной загрузки и безопасного хранения криптографических ключей. Однако, как показывают исследования, эти технологии также могут быть уязвимы. Исследователи продемонстрировали многочисленные атаки на Intel Trusted Execution Technology (TXT), включая техники подмены и реализации виртуальных модулей TPM, которые позволяют обходить или компрометировать механизмы аппаратной доверенной загрузки системы [3, 458, 459].

Одним из основных недостатков методов, основанных на доверенном гипервизоре, является их уязвимость к атакам типа «человек посередине» (Man In The Middle). Вредоносный гипервизор может получить контроль на ранних этапах загрузки системы, что позволяет ему скомпрометировать легитимный гипервизор, загружаемый позднее [15, 462]. Аппаратные модули для доверенной загрузки [463] могут помочь предотвратить такие атаки, но их использование часто затруднено из-за технических ограничений или сложности в реализации [464].

Хотя методы на основе доверенного MBM могут оказаться результативными в лабораторных условиях, их практическое применение сталкивается с серьезными трудностями. Главная из них заключается в необходимости адаптации к разным типам материнских плат, что требует дополнительных исследований и экспериментов.

14.5. ВРЕМЕННЫЕ МЕТОДЫ ОБНАРУЖЕНИЯ ГИПЕРВИЗОРОВ

Измерение времени выполнения ряда операций в гостевой ОС лежит в основе всех временны́х методов обнаружения гипервизоров. Эксперты также предложили техники обнаружения гипервизора на основе измерения времени доступа к жесткому диску, оперативной памяти, а также продолжительности криптографических вычислений [465–467]. Однако такие события, как правило, могут быть выявлены только специализированными гипервизорами.

С учетом характера выполняемых операций данные методы классифицируются на основе использования буфера ассоциативной трансляции (BAT или TLB), буфера возвратных адресов стека, списка перехватываемых событий, а также операций по доступу к памяти.

При наличии гипервизора время выполнения этих операций в гостевой ОС увеличивается, что связано с передачей ему управления. Таким образом, сравнение наблюдаемого времени выполнения операций с эталонными значениями позволяет определить наличие гипервизора в исследуемой системе. Вариабельность времени выполнения при повторных операциях достаточно велика, что может быть использовано для целей обнаружения гипервизора.

Однако все временны́е методы обладают общим недостатком — их можно обойти путем искажения показаний счетчика времени или вре́менной выгрузкой гипервизора из памяти (прототип BlueChicken [487]).

14.5.1. Метод обнаружения на основе буфера ассоциативной трансляции (Translation Lookaside Buffer)

Методы использования БАТ для обнаружения присутствия гипервизоров описаны в работах [25, 59, 433, 434, 437, 448, 469–472]. Как было указано ранее (см. подпараграф 14.3.1), БАТ хранит записи, включающие виртуальные и физические адреса страниц памяти. При обращении операционной системы к ВА кэш-контроллер проверяет наличие соответствующей записи в БАТ (см. рис. 95). Если запись найдена, то происходит обращение к ФА, значение которого было извлечено из БАТ, что

существенно сокращает время доступа к памяти. В случае отсутствия записи в БАТ происходит более длительный процесс поиска физического адреса с использованием прохода по дереву системных таблиц страниц, такие как каталоги страниц. После нахождения адреса запись добавляется в БАТ, что ускоряет повторное обращение к этому виртуальному адресу.

Использование БАТ позволяет значительно сократить время доступа к памяти, так как исключает необходимость долговременного обращения к системным таблицам при повторных вызовах. Процесс добавления новых записей в БАТ управляется специальными алгоритмами. Например, в работе [473] описан алгоритм управления рабочим набором, который адаптирует содержимое кэша, включая БАТ, к изменяющемуся поведению программ. Это позволяет минимизировать промахи кэша и улучшить производительность системы. В главе 5 книги [474] приведены алгоритмы замещения наименее востребованной страницы (Least Recently Used, LRU) и алгоритм случайного замещения (Random Replacement). Поскольку БАТ имеет ограниченный размер, алгоритм LRU помогает использовать его пространство, заменяя записи, которые не использовались дольше всего. Это снижает вероятность промахов при обращении к часто используемым страницам. В главе 9 книги [475], кроме названных выше алгоритмов, приведен еще один алгоритм замещения с использованием счетчиков (Counting Algorithms), который замещает страницы с наименьшей частотой обращений.

Присутствие гипервизора можно выявить по изменению состояния БАТ. При передаче ему управления содержимое буфера ассоциативной трансляции изменяется, что можно использовать как индикатор присутствия гипервизора [437].

Для выявления гипервизора необходимо сначала прочитать значения по определенному адресу в памяти, затем выполнить повторное обращение к этому же адресу для того, чтобы измерить время доступа. После этого выполняется демаскирующее событие, а затем снова измеряется время доступа к тому же адресу. Исходя из анализа изменения длительности операции доступа к памяти до и после произведения демаскирующего события, делается заключение о наличии или отсутствии гипервизора.

При первом обращении к адресу памяти через системные таблицы буфер БАТ заполняется соответствующей записью. Это может быть как зарезервированный сегмент памяти, так и уже используемый участок памяти объекта. При повторных обращениях данные считываются из БАТ, что исключает необходимость повторного прохода по таблицам и сокращает время доступа к памяти.

При выполнении демаскирующего события, если в системе присутствует гипервизор, буфер БАТ очищается; если гипервизор отсутствует, то состояние буфера не изменяется. При повторном обращении к тому же адресу возможны два варианта: БАТ либо очищен, либо нет. В случае его очистки доступ к памяти занимает больше времени. Сравнивая время доступа до и после демаскирующей операции, можно определить наличие гипервизора: если время совпадает в пределах погрешности, то гипервизор отсутствует; если нет — присутствует.

Этот метод прост в реализации и обеспечивает быстрое принятие решения о наличии гипервизора. Однако он имеет ряд ограничений: на процессорах *AMD* и современных процессорах *Intel* данный метод может не работать из-за наличия дополнительных полей в БАТ, таких как ASID и PCID, которые поддерживают целостность буфера при передаче управления гипервизору. Различия в архитектуре процессоров, вложенные уровни кэш-памяти и их ассоциативность могут осложнить переносимость метода на разные платформы.

Таким образом, использование временны́х характеристик БАТ позволяет обнаруживать гипервизор лишь на старых моделях процессоров.

14.5.2. Метод обнаружения на основе буфера стека возврата (Return Stack Buffer)

Производительность средств вычислительной техники (СВТ), работающих на процессорах *Intel*, может быть существенно увеличена за счет использования сохраненных адресов возвратов из функций в буфере стека возврата (БВС) [476–478]. БВС функционирует как внутрипроцессорный стек, который, согласно исследованию [479], включает 16 записей. При вызове

функции ее адрес возврата сохраняется в этом буфере, что позволяет ускорить процесс возврата управления.

Метод использования БСВ для выявления гипервизора в процессорах *Intel* был предложен в работах [433, 434, 480]. Этот метод основан на измерении времени возврата управления из 16 вложенных функций. Пример кода на рис. 98 показывает, как основная функция main() вызывает первую из 16 вложенных функций, которые последовательно вызывая друг друга, заканчивают цикл вызовом 16-й функции, исполняющей демаскирующее событие, фиксацию TSC, после чего управление возвращается в первую функцию, которая регистрирует второе показание счетчика времени.

```
main()
{
 cli;
 call func_0();
 RDTSC; // снять вторые показания счетчика
 sti;
}

func_0 ()
{
 call func_1();
 ret;
}

func_1 ()
{
 call func_2();
 ret;
}

.
.
.
 func_15()
{
 CPUID; // вызов ВМ-выхода
 RDTSC; // снять первые показания счетчика
 ret;    // запустить 16 возвратов
}
```

RDTSC;

ret; func_15
ret; func_14
ret; func_13
ret; func_12
ret; func_11
ret; func_10
ret; func_9
ret; func_8
ret; func_7
ret; func_6
ret; func_5
ret; func_4
ret; func_3
ret; func_2
ret; func_1
ret; func_0

Перезапись БСВ из-за вызова обработчика гипервизора

RDTSC;

Рис. 98. Пример кода для обнаружения гипервизора на основе БСВ

В условиях без гипервизора БСВ будет содержать адреса возврата всех вызванных функций начиная с func_15 и заканчивая func_0. Если гипервизор загружен, то при выполнении демаскирующего события он получит управление, что приведет к перезаписи БСВ адресами возвратов функций обработчика гипервизора. Это представлено красным текстом, где показано, как адреса func_2, func_1 и func_0 могут быть перезаписаны.

Этот метод достаточно прост и малозатратен по времени принятия решения. Однако он не защищен от противодействия. В работе [433] показано, что если обработчик не содержит подфункций или содержит минимально возможное их число, то обнаружить гипервизор затруднительно.

14.5.3. Метод обнаружения на основе списка демаскирующих событий

Этот метод обнаружения гипервизора был подробно рассмотрен в целом ряде исследований: [25, 59, 84, 433, 434, 437, 481, 445, 471, 472, 482–486]. Его суть заключается в измерении длительности выполнения в ОС набора специфических действий, называемых демаскирующими событиями. Выделяют два типа демаскирующих событий: безусловно, перехватываемые гипервизором и события, перехват которых требует предварительной настройки гипервизора (условно перехватываемые). В случае присутствия гипервизора выполнение демаскирующего события приводит к передаче управления гипервизору. Путем сравнения значений длительности выполнения демаскирующих событий с заранее установленными эталонными значениями можно принять решение о наличии гипервизора: в случае присутствия гипервизора длительность оказывается почти на порядок больше, чем в случае его отсутствия. Однако данная техника применима только для случаев, когда гипервизор не оказывает противодействия своему обнаружению в виде компрометации счетчика времени или временной выгрузки своего обработчика из памяти.

Процесс выявления гипервизора обычно включает следующие этапы.

1. *Фиксация исходного времени*: перед выполнением демаскирующих событий замеряется текущее значение таймера (например, счетчика тактов процессора).

2. *Активация демаскирующих действий*: запускается серия инструкций или операций, потенциально вызывающих перехват со стороны гипервизора:

- в процессорах *Intel* существует набор инструкций, которые, безусловно, перехватываются гипервизором (БП-инструкции). При наличии гипервизора выполнение таких инструкций, например CPUID, вызывает принудительную передачу управления гипервизору. После обработки гипервизор возвращает управление ОС для выполнения следующей инструкции. По данным [437], в среде без гипервизора это занимает порядка 200 тактов, а при наличии гипервизора длительность выполнения CPUID может увеличиваться на несколько порядков;
- в процессорах *AMD* аналогом могут служить операции RDMSR, поскольку гипервизор должен перехватывать их выполнение для того, чтобы скрыть свое присутствие [433]. Дополнительно в AMD-процессорах можно проверять бит SVME в регистре EFER, сигнализирующий о включенном режиме аппаратной виртуализации.

3. *Измерение времени выполнения*: считывается значение таймера после завершения действий и рассчитывается длительность выполнения.

4. *Сравнение с эталоном*: если полученная длительность выполнения значительно превышает эталонное значение, то это служит признаком наличия гипервизора. Эталонное значение длительности может быть заранее определено на доверенной машине без гипервизора или же путем измерения выполнения эквивалентных инструкций, которые гарантированно не перехватываются гипервизором [25].

Метод на основе демаскирующих событий легко реализуется и применяется для демонстрации возможностей обнаружения прототипов гипервизоров. Однако гипервизор может скрыться от обнаружения с использованием следующих техник противодействия:

- компрометация, или «скручивание» счетчиков времени [59];
- временная выгрузка гипервизора из памяти, техника Blue Chicken.

Для таких случаев требуется разработка специальных методов обнаружения гипервизора [13, 487].

Метод остается востребованным благодаря своей простоте, универсальности и переносимости на различные аппаратные платформы. Однако он основан на базовом принципе: гипервизор неизбежно тратит дополнительные ресурсы при управлении виртуальными машинами, что отражается в замерах времени.

14.5.4. Метод обнаружения на основе обращения к памяти

Метод обнаружения гипервизора на основе обращения к памяти предложен в работе [468] и основывается на различии во времени доступа к памяти в зависимости от его наличия или отсутствия. Когда гипервизор активен, в памяти присутствуют его обработчик и связанные структуры, что можно использовать для принятия решения о его наличии на основе анализа копии памяти. Способ обнаружения гипервизора с применением сигнатур описан в параграфе 14.2.

Гипервизор может препятствовать получению точной копии содержимого памяти, перехватывая соответствующие обращения различными методами [434, 445]. В результате время обработки таких обращений в случае присутствия гипервизора увеличивается по сравнению с ситуацией его отсутствия.

Для обнаружения гипервизора необходимо последовательно обращаться к заранее выбранным адресам памяти, измерять время доступа к каждому из них и формировать массив данных с результатами. Далее проводится анализ собранных данных для выявления наличия гипервизора.

Этот метод позволяет не только обнаружить присутствие гипервизора, но и определить диапазон адресов, доступ к которым он перехватывает. Это достигается путем выявления адресов с аномально увеличенным временем отклика. Однако стоит учитывать, что гипервизор может изменять показания таймера, затрудняя процесс его обнаружения.

В условиях, когда гипервизор преднамеренно искажает показания таймера или временно выгружается из памяти, временны́е методы обнаружения становятся непригодными.

14.6. ПРИМЕНЕНИЕ КОМПЬЮТЕРНЫХ СЧЕТЧИКОВ ДЛЯ ОЦЕНКИ ВРЕМЕНИ ВЫПОЛНЕНИЯ ПРОЦЕССОРНЫХ ИНСТРУКЦИЙ

В рамках исследования, целью которого является измерение времени выполнения процессорных инструкций, были проведены анализ и классификация компьютерных счетчиков. Все счетчики времени можно разделить на две основные группы: локальные (встроенные в систему) и сетевые (использующие удаленные вычислительные машины) [488, 489]. Эти группы представлены на рис. 99.

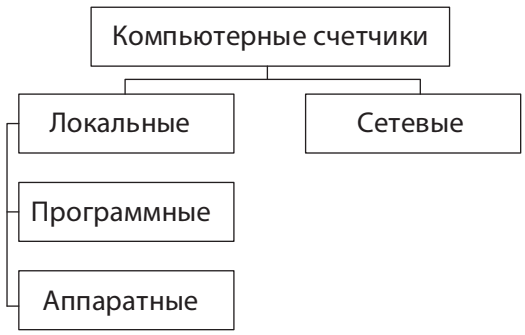


Рис. 99. Классификация компьютерных счетчиков времени

14.6.1. Локальные счетчики

Локальные счетчики делятся на программные и аппаратные. Программные счетчики измеряют длительность выполнения инструкций через программный код, выполняемый на целевой СВТ. Один из таких счетчиков был предложен экспертом E. Barbosa [449], а исследования выполнены в работах [25, 437]. Программные счетчики работают через инкрементацию (увеличение) переменной в одном ядре процессора, в то время как в другом ядре выполняются измеряемые события. Решение о присутствии гипервизора принимается путем сопоставления значений переменной цикла с заранее установленным эталонным значением. Несмотря на высокую точность, программные счетчики уязвимы перед атаками, использующими временную выгрузку гипервизора из памяти, и могут иметь сложности с синхронизацией потоков, как отмечено в работе [434].

Достоинства программного счетчика — высокая устойчивость к компрометации, высокая разрешающая способность и разрядность счетчика.

Аппаратные счетчики основываются на показаниях таймеров встроенных устройств. Выделяют несколько видов аппаратных счетчиков (см. рис. 99) [490–492]:

- TSC-счетчик тактов центрального процессора (Time Stamp Counter). Это 64-битный регистр в процессорах архитектуры x86, который отсчитывает количество процессорных тактов, прошедших с момента последнего сброса. Точность счетчика составляет один такт, что делает его пригодным для измерений, требующих высокой разрешающей способности. TSC часто используется для проведения высокоточных временных измерений при оценке производительности. В каждом ядре процессора имеется отдельный счетчик TSC. Согласно работе [488] для процессора с тактовой частотой 2 ГГц погрешность счетчика тактов составляет всего 0,5 нс. Учитывая эти особенности, а также то, что процессорные инструкции RDTSC и RDMSR 10h позволяют снимать показания счетчика TSC на повышенном 31-м уровне IRQL [493], можно считать его приемлемым инструментом для измерения времени выполнения процессорных операций;
- RTC-счетчик (Real Time Clock, или часы реального времени) отличается нестабильностью работы и зависимостью от заряда батареи, что делает его непригодным для точных измерений [494];
- ACPI-таймер (Advanced Configuration and Power Interface) обладает низкой точностью и не подходит для задач, требующих точных временны́х измерений [495]. В статье [491] отмечена недостаточная точность этого счетчика (порядка 0,3 мс), а обновление счетчика происходит «рывками». Это не позволяет рассматривать ACPI-таймер как подходящее средство для измерения длительности выполнения процессорных инструкций;
- APIC-таймер (Advanced Programmable Interrupt Controller Timer) является частью APIC — программируемого контроллера прерываний, встроенного в процессоры для обра-

ботки прерываний в многопроцессорных системах. APIC-таймер используется для генерации прерываний через определенные интервалы времени, что необходимо для координации работы процессоров и их взаимодействия с периферийными устройствами. Он имеет разрешающую способность 1 мс [494]. Поскольку длительность выполнения набора из 10 процессорных инструкций может составлять 1 мкс, этот счетчик не подходит для измерения длительности выполнения процессорных инструкций;

- HPET-таймер (High Precision Event Timer) мало распространен и его использование ограничено [491]. Значения HPET таймера можно получить путем обращения к определенным адресам памяти [496];
- PIT-таймер (Programmable Interval Timer) не подходит для высокоточных измерений, несмотря на длительное время использования [491]. Считывание его показаний осуществляется посредством взаимодействия с портами ввода/вывода, на что тратится дополнительное время.

14.6.2. Сетевые счетчики

Сетевые счетчики получают данные с удаленных узлов и используют различные протоколы: NTP (Network Time Protocol) [497], PTP (Precision Time Protocol) и другие [498]. Однако из-за сетевых задержек эти счетчики не обладают достаточной разрешающей способностью для измерения длительности процессорных инструкций.

Проведенный анализ показал, что счетчик TSC является пригодным инструментом для измерения длительности выполнения процессорных инструкций. Счетчики TSC доступны на всех компьютерах с процессорами *Intel* и *AMD*.

Глава 15
МЕТОДЫ МАТЕМАТИЧЕСКОЙ СТАТИСТИКИ ОБНАРУЖЕНИЯ ГИПЕРВИЗОРА В УСЛОВИЯХ ПРОТИВОДЕЙСТВИЯ СО СТОРОНЫ НАРУШИТЕЛЯ

В данной главе изложены теоретические предпосылки для обнаружения гипервизора, построена модель нарушителя, а также проведен анализ техник компрометации счетчиков тактов. Описывается процедура измерения времени выполнения инструкций программы (трассы), которая играет ключевую роль в выявлении индикаторов присутствия гипервизора.

Для анализа процесса выполнения программы используются модели, основанные на теории графов, которые учитывают как отсутствие, так и присутствие гипервизора. Причем рассматриваются как одиночные экземпляры гипервизоров, так и сложные случаи с вложенными гипервизорами. Экспериментальные данные используются для проверки адекватности предложенных моделей, что позволяет выявить специфические особенности выполнения процессорных инструкций в различных сценариях.

Кроме того, вводится и обосновывается понятие критерия различимости распределений, который позволяет статистически отличить нормальные условия выполнения инструкций от аномальных, вызванных присутствием гипервизора. Доказательство существования этого критерия представлено для теоретических распределений времени выполнения в обоих случаях: с гипервизором и без него.

15.1. МОДЕЛЬ НАРУШИТЕЛЯ

Опишем модель нарушителя, который может незаконно внедрить ВПО, используя разнообразные методы: установку драйвера ОС, модификацию главной загрузочной записи (MBR) на жестком диске или изменение микропрограммы BIOS (UEFI).

Предполагается, что внедренный гипервизор может скрываться от обнаружения, применяя такие техники, как манипуляция процессорным счетчиком тактов, временная выгрузка

из системы, а также блокирование создания корректной копии дампа памяти, содержащей гипервизор.

Согласно работе [499] нарушителями могут выступать производители программного и аппаратного обеспечения, относящиеся к категориям с уровнем возможностей Н3–Н4 (рис. 100). Эти категории охватывают участников цепочки поставок, включая разработчиков и поставщиков программного и аппаратного обеспечения, обладающих средними (Н3) и высокими (Н4) возможностями для реализации атак. В частности, такие нарушители способны внедрять гипервизор в BIOS/UEFI либо поставлять его в составе прикладного ПО, что соответствует атакам типа supply-chain.

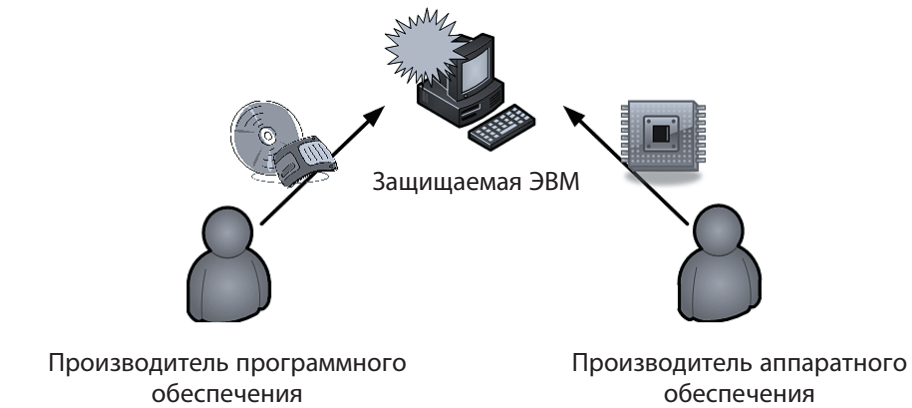


Рис. 100. Схема возможных нарушителей

Действия таких нарушителей представляют серьезную угрозу информационной безопасности, включая нарушение конфиденциальности, целостности и доступности обрабатываемых данных. Изоляция системы от сетей, таких как интернет, может снизить риск утечки конфиденциальной информации, однако остается угроза утечки через внешние носители [500].

Угрозы целостности и доступности заключаются в том, что гипервизор может нарушать нормальное функционирование системы, влияя как на данные, так и на оборудование, включая периферийные устройства. Это может привести к сбоям в работе информационно-телекоммуникационных систем и нарушению технологических процессов.

15.2. ВОЗМОЖНЫЕ СПОСОБЫ КОМПРОМЕТАЦИИ СЧЕТЧИКА ТАКТОВ

Пердставим методику обнаружения гипервизора на основе временных статистических данных с использованием списка демаскирующих событий в условиях преднамеренной компрометации счетчика тактов со стороны нарушителя. Детально рассмотрим возможные способы компрометации счетчика тактов:

- перехват БП-инструкций;
- перехват инструкций чтения счетчика тактов (например, RDTSC);
- использование поля TSC_OFFSET в структуре управления виртуальной машиной VMCS.

Дополнительно альтернативным способом сокрытия активности гипервизора может быть его временная выгрузка из оперативной памяти, что было рассмотрено в параграфе 14.5.

15.2.1. Компрометация при перехвате БП-инструкции

Как было рассмотрено в параграфе 14.5, любая реализация гипервизора должна обеспечивать обработку безусловно перехватываемых (БП) инструкций процессора. На рис. 101 представлена схема последовательности процедур измерения времени выполнения трассы с учетом перехвата инструкций гипервизором.

В данном примере рассматривается трасса, состоящая из одной инструкции CPUID. В общем случае трасса может состоять из набора различных инструкций, перехватываемых гипервизором. Слева на рис. 101 изображена ось времени, соответствующая показаниям процессорного счетчика тактов (TSC). Левый прямоугольник обозначает защищенный режим процессора (R-режим), в котором проводится измерение времени выполнения трассы. Правый прямоугольник соответствует режиму монитора виртуальной машины (V-режим), где находится обработчик гипервизора, который может искажать показания TSC для сокрытия своей активности.

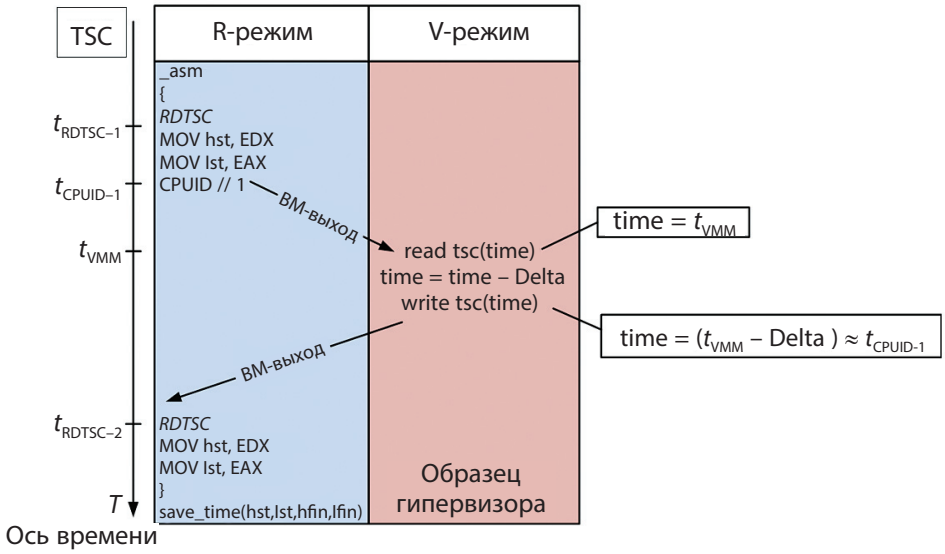


Рис. 101. Компрометация показаний счетчика тактов при перехвате БП-инструкций

Переключения между R- и V-режимами обозначены стрелками. Они отображают возможные переходы процессора между этими режимами в процессе выполнения инструкций.

Рассмотрим процесс компрометации показаний TSC в случае перехвата БП-инструкций.

При наличии гипервизора его обработчик получает управление, и в момент начала работы счетчик фиксирует значение *t* VMM. Гипервизор считывает текущее значение счетчика с помощью вызова функции read_tsc(time) и сохраняет его во временную переменную time. Затем из этого значения вычитается заранее заданная величина компрометации Δ, и результат записывается обратно в переменную time. Обновленное значение записывается в TSC с помощью функции write_tsc(time), после чего управление возвращается в R-режим. При повторном чтении счетчика TSC регистрируется значение $t_{RDTSC-2}$.

Длительность выполнения трассы рассчитывается как следующая разность: $t_{RDTSC-2} - t_{RDTSC-1}$. В случае присутствия гипервизора можно подобрать величину компрометации Δ таким образом, чтобы длительность выполнения трассы в случае при-

сутствия гипервизора совпадала с длительностью в случае отсутствия гипервизора.

Для достижения точной настройки TSC можно применять метод трассировки инструкций [59]. Этот метод предполагает не корректировку фиксированного значения, а учет времени выполнения каждой отдельной перехватываемой инструкции для того, чтобы обеспечить точное совпадение среднего времени выполнения анализируемой последовательности команд.

Длительность выполнения инструкции t определяется по формуле

$$t = t'' - t', \tag{1}$$

где t' и t'' — значения счетчика тактов, зафиксированные в начале и в конце выполнения набора инструкции соответственно.

При многократном измерении времени выполнения инструкции $t_{\text{ОТ}}$ в цикле (в отсутствие гипервизора) процесс можно представить, как показано на рис. 102, *а*. Показания счетчика имеют недетерминированный характер, из-за чего длительность выполнения инструкции является **случайной величиной**, значительно увеличивающейся при наличии гипервизора (рис. 102, *б*). Если не будут предприняты меры по искажению показаний, позволяющие уменьшить $t_{\text{ПР}}$ до значения $t_{\text{ОТ}}$, то обнаружить гипервизор не составит труда [501].

Для сокрытия наличия гипервизора показания счетчика должны искажаться так, чтобы регистрируемое время $t_{\text{ПР}}$ незначительно отличалось от времени $t_{\text{ОТ}}$ (рис. 102, *в*). Это достигается за счет уменьшения показаний $t''_{\text{ПР}}$ на величину $\Delta T_{\text{ИСК}}$, таким образом, чтобы отличие $t_{\text{ПР,ИСК}}$ от $t_{\text{ОТ}}$ было незначительным:

$$t_{\text{ПР,ИСК}} = \left(t''_{\text{ПР}} - \Delta T_{\text{ИСК}}\right) - t'_{\text{ПР}} \approx t_{\text{ОТ}}. \tag{2}$$

Метод не зависит от количества инструкций в трассе.

15.2.2. Компрометация при перехвате инструкций чтения счетчика

Помимо компрометации при перехвате БП-инструкций, возможен сценарий компрометации процессорного счетчика тактов при перехвате инструкций чтения этого счетчика. В данном

случае образец гипервизора должен обрабатывать не только БП-инструкции, но и инструкции, связанные с чтением счетчика.

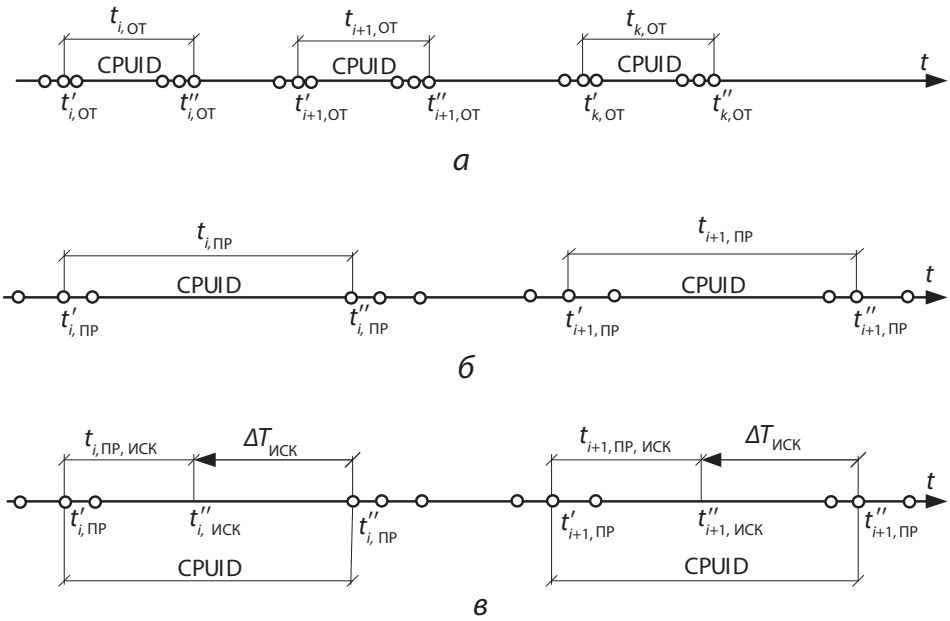


Рис. 102. Схема измерения времени выполнения трассы из одной инструкции CPUID:
а — без гипервизора; *б* — с гипервизором без искажения показаний; *в* — гипервизор с искажением показания TSC

Например, рассмотрим процессорную инструкцию RDTSC (Read Time-Stamp Counter) [502]. Схема измерения длительности выполнения трассы с учетом перехвата инструкций RDTSC гипервизором, представленная на рис. 103, аналогична схеме на рис. 101, но дополнена переключениями между R- и V-режимами процессора во время выполнения инструкции RDTSC.

В ходе выполнения первой инструкции RDTSC обработчик гипервизора обнуляет переменную — счетчик числа вызовов инструкций чтения счетчика и записывает показания счетчика тактов во временную переменную start_t. Затем при выполнении каждой инструкции трассы управление передается гипервизору, который инкрементирует переменную-счетчик, как показано через оператор i_cpuid++.

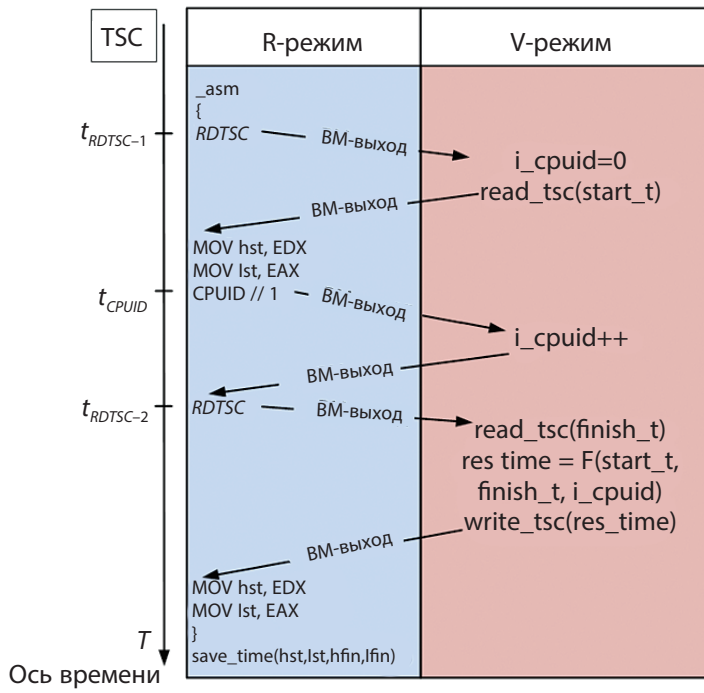


Рис. 103. Компрометация показаний счетчика тактов при перехвате инструкции чтения счетчика тактов RDTSC

Когда выполняется вторая инструкция RDTSC, обработчик производит ряд действий: сначала производится запись значений счетчика тактов процессора во временную переменную finish_t, затем производится расчет переменной res_time. Для этого необходимо учесть значения счетчика как до выполнения трассы, так и после него, а также общее количество перехваченных инструкций, содержащихся в трассе. Наконец, обработанный результат, содержащийся в переменной res_time, заносится в счетчик TSC.

Формула для вычисления времени выполнения инструкции:

$$t = t_{finish} - t_{start},$$

где t_{start} — значение счетчика тактов, зафиксированное в начале выполнения инструкции; t_{finish} — значение счетчика тактов, зафиксированное в конце выполнения инструкции; t — общее время выполнения инструкции.

Описанный способ позволяет задать точное значение счетчика, которое будет считано измеряющей программой при повторном чтении, но ограничен сложностью реализации. Применение его обеспечивает высокую точность подмены временных данных и затрудняет обнаружение гипервизора, скрывающего свое присутствие путем манипуляции с показаниями счетчика.

15.2.3. Компрометация с использованием поля TSC_OFFSET в структуре управления виртуальной машиной

Современные технологии аппаратной виртуализации позволяют изменять показания счетчика тактов (TSC) автоматически. Например, в технологии Intel VT-x это достигается с помощью установки флага use TSC offsetting и задания значения дельты в специальном MSR-регистре (Model-Specific Register) IA32_TIME_STAMP_COUNTER. В результате при выполнении инструкции RDTSC происходит чтение текущего значения из MSR-регистра, после чего к нему добавляется указанное значение дельты, и результат записывается обратно в счетчик TSC, уменьшая истинное значение.

Этот метод использован в реализации гипервизоров, например в программе Blue Pill [485]. Однако, как указывается в работе [59], применение данного метода сталкивается с определенными трудностями, связанными с точным подбором величины смещения, что может усложнить корректное выполнение компрометации.

Таким образом, хотя использование поля TSC_OFFSET предоставляет простой способ подмены показаний TSC, для точной настройки компрометации требуется учитывать множество факторов, таких как архитектурные особенности процессора и поведение виртуальных машин, что затрудняет реализацию этого метода на практике.

15.3. ПОСТРОЕНИЕ МОДЕЛЕЙ ВЫПОЛНЕНИЯ ТРАССЫ

Проанализируем схемы переключения режимов работы процессора при наличии или отсутствии гипервизора. Исследование опирается на теорию графов и экспериментальные данные, выявляя особенности времени выполнения трасс в разных сценариях.

Трасса (trace) — это последовательность инструкций процессора, используемая для анализа взаимодействия между гостевой ОС и гипервизором. В частности, трасса может использоваться для выявления изменений в производительности или аномалий, вызванных присутствием гипервизора.

Обычно в трассу добавляют инструкции CPUID как безусловно перехватываемые гипервизорами в случае технологии аппаратной виртуализации процессоров *Intel*.

Рассмотрим модель выполнения трассы в случае отсутствия гипервизора и определим критерий обнаружения гипервизора, основанный на изменении длительности выполнения трассы.

15.3.1. Анализ режимов работы процессора и влияния гипервизора на время выполнения инструкций

Для разработки подходов к обнаружению гипервизора проведем анализ длительности выполнения трассы в условиях отсутствия и присутствия одного или нескольких образцов гипервизоров.

Согласно исследованию [63], если гипервизор отсутствует, то процессор может функционировать в одном из двух режимов: защищенном режиме (R-режим), в котором выполняются пользовательские инструкции, или системном режиме (S-режим), предназначенном для выполнения системных задач. Режим системного управления (S-режим) частично документирован, и его работа описана в исследованиях [62, 503]. Если рассматривать R-режим как основной, то схема переключения между R- и S-режимами при отсутствии гипервизора показана на рис. 104.

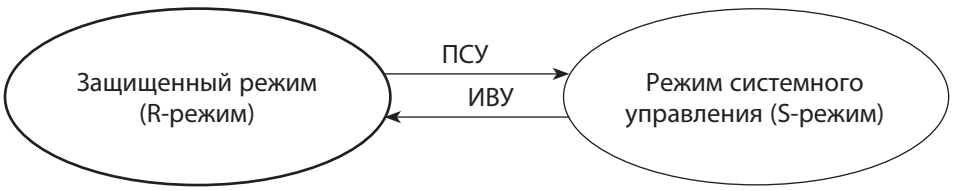


Рис. 104. Схема переключения между режимами работы процессора

При появлении прерывания системного управления (ПСУ) процессор переходит из R-режима в S-режим, а возвращение в R-режим происходит после выполнения инструкции возврата

управления (ИВУ). Поскольку ПСУ носит случайный характер, переход в S-режим происходит с определенной вероятностью. Если прерывание не происходит, то переключения не осуществляется и процессор остается в R-режиме.

В случае присутствия гипервизора процессор может находиться не в двух, а в трех режимах: сохраняются R- и S-режимы, добавляется режим монитора виртуальных машин (VMX root mode, V-режим). В этом случае R-режим становится режимом гостевой машины, а V-режим отвечает за режим работы гипервизора. Сравнение режимов системного управления и монитора виртуальных машин описано в работе [62]. На рис. 105 представлена схема переключения между тремя режимами при наличии гипервизора. Здесь R-режим принят за основной, а S-режим условно разделен на два — S и S′. В R-режиме выполнение БП-инструкции всегда приводит к переключению в V-режим, после чего управление возвращается обратно в R-режим. В случае прерывания ПСУ процессор переходит в S-режим, после чего управление возвращается в режим, в котором произошло прерывание (R- или V-режим). Это повышает вероятность случайных переключений между режимами, увеличивая вариабельность значений длительности выполнения трассы.

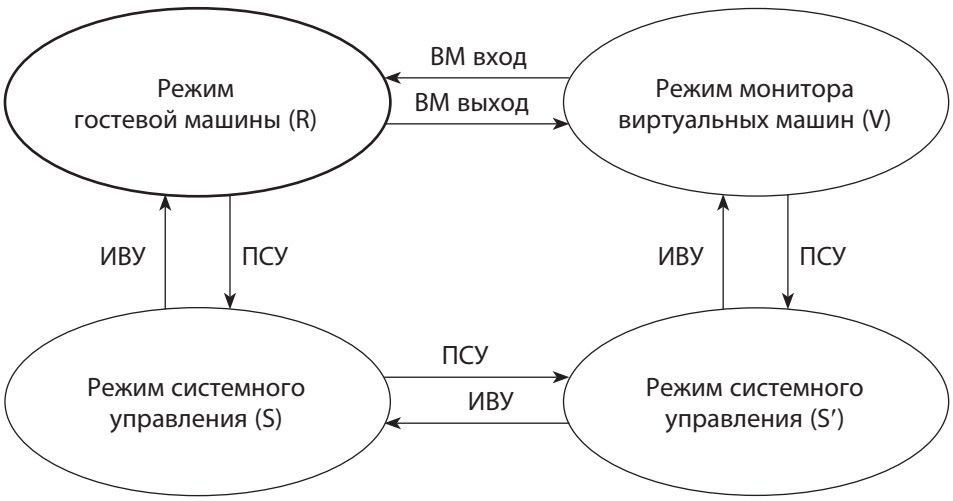


Рис. 105. Схема переключения между режимами процессора при выполнении набора инструкций в случае присутствия гипервизора

При выполнении БП-инструкции в R-режиме всегда происходит переключение в V-режим, из которого управление возвращается обратно в R-режим (на рис. 105 показаны стрелки «ВМ вход» и «ВМ выход»).

При возникновении прерывания ПСУ происходит переход в S-режим, после чего управление возвращается в тот режим, где произошло прерывание, — это может быть либо R-, либо V-режим. Появление прерывания также имеет случайный характер, что увеличивает вероятностную природу переключений. Добавление V-режима усиливает вариабельность длительности выполнения трассы, так как частота переключений между режимами увеличивается.

Из анализа можно сделать вывод, что присутствие гипервизора увеличивает время выполнения трассы из-за более частых переключений в S-режим, что, в свою очередь, приводит к росту как общей продолжительности выполнения, так и показателей ее вариабельности.

Данная схема может быть расширена для случая с несколькими вложенными образцами гипервизоров, как показано на рис. 106. В этом случае процессор также функционирует в трех режимах, но каждый новый обработчик гипервизора может быть прерван системным прерыванием. Обработчики гипервизоров работают в привилегированном V-режиме, который на схеме показан заштрихованным прямоугольником. При выполнении каждой инструкции в R-режиме происходит переключение в V-режим, после чего управление передается от одного обработчика к другому по цепочке, и только затем возвращается в R-режим (стрелки «ВМ вход» и «ВМ выход») [13, 504]. При возникновении прерывания ПСУ выполнение любого из обработчиков может быть прервано и процессор переключится в S-режим (стрелки «ИВУ» и «ПСУ»).

При наличии нескольких вложенных образцов гипервизоров наблюдается еще большее увеличение длительности выполнения перехватываемых инструкций и вариабельности режимов переключения. Этот анализ позволяет перейти к построению моделей выполнения трассы, которые будут рассмотрены далее.

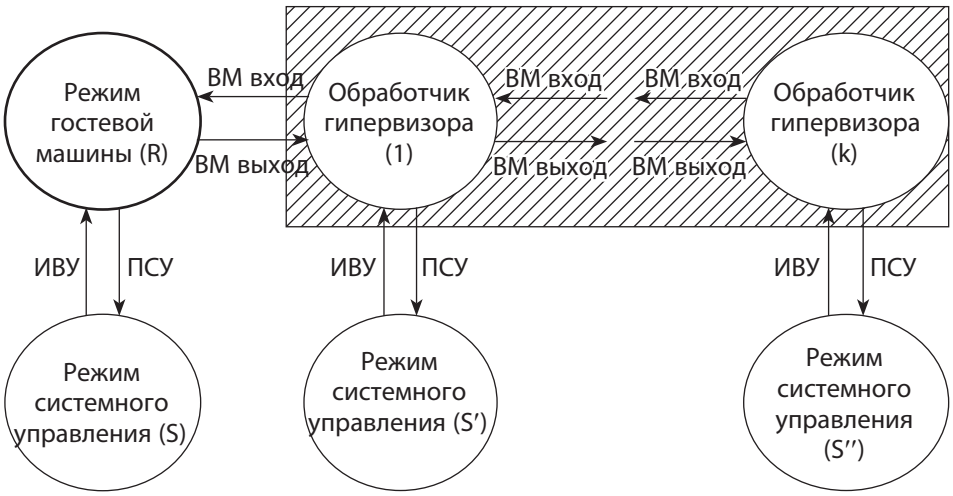


Рис. 106. Схема переключения между режимами работы процессора при выполнении набора инструкций в случае присутствия вложенных гипервизоров

15.3.2. Особенности выполнения набора процессорных инструкций

При выполнении инструкций процессор сталкивается с прерываниями и переходами между R- и S-режимами, которые приводят к тому, что время выполнения трассы становится случайным. Для анализа выполнения инструкций при переключении режимов работы процессора сначала нужно рассмотреть ситуацию, в которой программа выполняется без аппаратных вмешательств и прерываний. На рис. 107 представлен фрагмент программы для измерения длительности трассы с использованием инструкции RDTSC (Read Time-Stamp Counter) для чтения показаний счетчика тактов TSC. На рис. 107 показано, как происходит измерение длительности выполнения трассы (набора инструкций CPUID) путем считывания двух значений счетчика тактов до выполнения трассы и после него.

Считывание счетчика тактов осуществляется до и после выполнения трассы с использованием инструкции RDTSC. Моменты выполнения инструкций считывания счетчиков тактов ($t_{RDTSC-1}$ и $t_{RDTSC-2}$) и инструкций из трассы ($t_{CPUID-1}$ и $t_{CPUID-10}$) отме-

чены на оси времени, которая показана на рис. 107. Длительность выполнения трассы t определяется по показаниям счетчика TSC:

$$t = t_{RDTSC-2} - t_{RDTSC-1}. \quad (3)$$

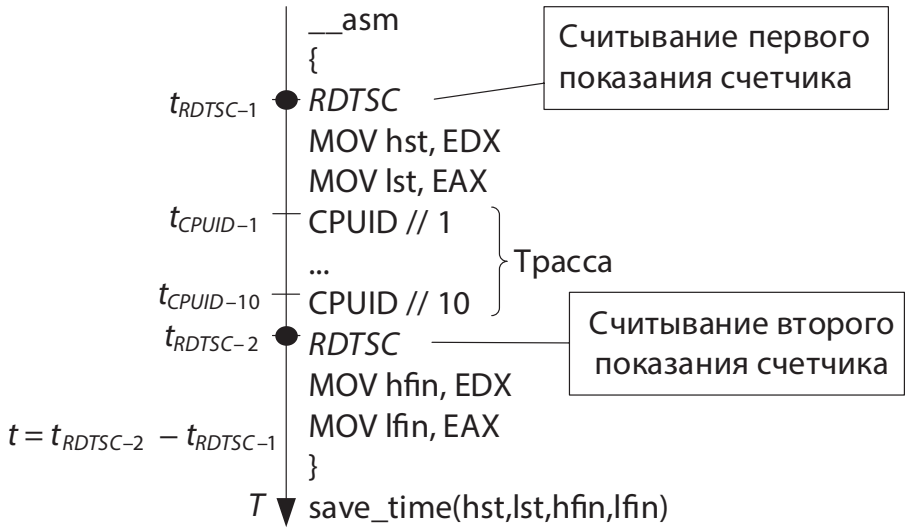


Рис. 107. Фрагмент программы измерения длительности трассы

Если бы процессор выполнял исключительно инструкции программы, без обработки прерываний и изменения режимов, то зарегистрированное время выполнения трассы t совпадало бы с ее фактической продолжительностью. Однако в реальных условиях работы процессора из-за прерываний фиксируемое время выполнения оказывается больше реального. На рис. 108 представлена схема смены режимов работы процессора при выполнении трассы в условиях отсутствия гипервизора. Здесь важно отметить, что переключения в S-режим носят случайный характер.

В левой части изображена ось времени, на которой указаны моменты считывания тактов процессорного счетчика (RDTSC) до и после выполнения инструкций трассы. Область R-режима (защищенный режим) находится слева, в то время как S-режим (системный режим) представлен справа и используется для выполнения команд обработчика прерываний.

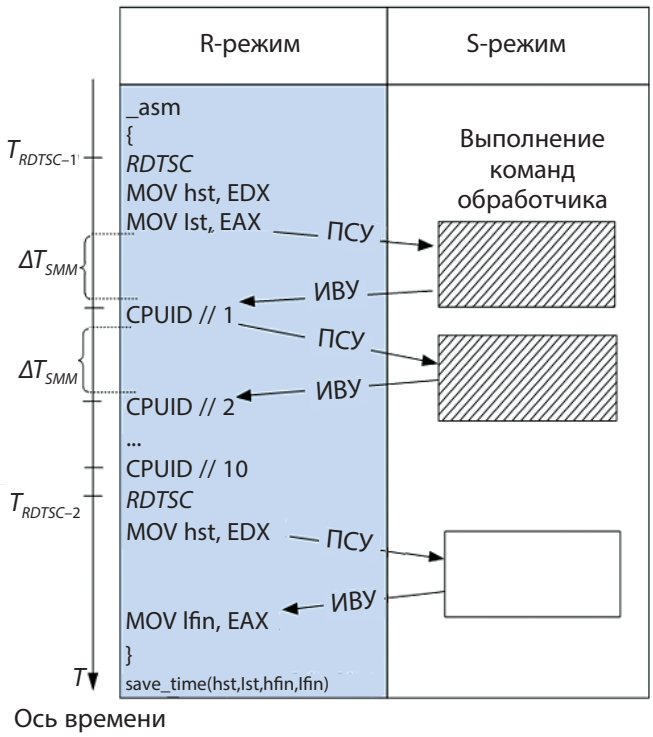


Рис. 108. Смена режимов процессора при выполнении трассы в случае отсутствия гипервизора

Согласно схеме процесс начинается с выполнения команды RDTSC для считывания значения процессорного счетчика и сохранения его в регистры EDX и EAX. Далее выполняется несколько инструкций трассы, например CPUID, после чего снова выполняется RDTSC для считывания значения счетчика тактов.

Во время выполнения трассы процессор может переключаться в S-режим при поступлении ПСУ, после чего управление возвращается в R-режим с помощью инструкции возврата (ИВУ). Каждое прерывание сопровождается временным выходом в S-режим, что отображено на оси времени через изменения между ΔT_SMM (длительность нахождения в системном режиме).

В итоге собранные данные о времени выполнения сохраняются для дальнейшего анализа с использованием функции save_time, которая фиксирует моменты считывания счетчиков в переменные (hst, lst, hfin, lfin).

При многократном выполнении одной и той же пользовательской инструкции или набора инструкций (трассы) измеряемое время выполнения будет распределяться по определенным фиксированным уровням. Эти уровни обусловлены дискретной природой квантов времени и ограниченным числом состояний процессора. При этом фактическое время выполнения может немного варьироваться из-за случайных факторов.

Если экспериментально установить, что эти фиксированные уровни времени выполнения трассы изменяются при наличии гипервизора (например, появляются новые уровни или изменяются существующие), то эту зависимость можно использовать для обнаружения такого воздействия. Таким образом, *анализ распределения измеряемого времени выполнения трассы может служить новым методом для выявления гипервизора.*

После запуска гипервизора в системе появляется дополнительный режим работы процессора, обозначаемый как V-режим. Схема определения длительности выполнения трассы и переключений процессора в случае присутствия одного гипервизора представлена в трехмерных координатах (рис. 109). Переключение между режимами показано линиями.

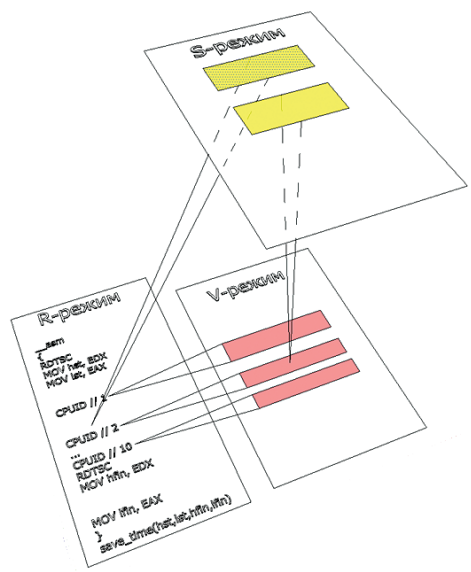


Рис. 109. Схема переключения между режимами процессора в случае присутствия одного гипервизора

Во время выполнения каждой инструкции трассы CPUID происходит переключение из R-режима в V-режим, где гипервизор обрабатывает перехваченную инструкцию. После завершения этой обработки управление возвращается в R-режим. Также возможны переходы в системный режим (S-режим) как из R-, так и из V-режима для обработки прерываний.

Наличие гипервизора вызывает увеличение общей длительности выполнения трассы и повышает вариабельность ее значений. Дополнительные переключения между режимами и обработка инструкций в V-режиме приводят к увеличению времени. Кроме того, в целях своего сокрытия гипервизор может изменять значения счетчиков тактов процессора, что уменьшает измеряемую длительность выполнения трассы. Однако гипервизор не способен устранить или контролировать статистический разброс в измерениях этой длительности, обусловленный аппаратными и системными факторами.

В ситуации, когда в системе присутствует сразу несколько образцов гипервизоров, наблюдается усложнение паттернов переключений и времени выполнения. Каждый последующий гипервизор может обрабатывать запросы предыдущего, что приводит к дальнейшему росту длительности и вариабельности выполнения трассы.

Для выявления присутствия гипервизора автором предложена методика, основанная на анализе временны́х характеристик выполнения трассы и переключений режимов процессора. Многократное выполнение идентичной трассы и измерение времени ее выполнения позволяют собрать статистические данные о поведении системы. В условиях отсутствия гипервизора значения времени выполнения распределяются по фиксированным уровням (слоям). Присутствие гипервизора вводит новые значения (дополнительные уровни) времени выполнения и увеличивает вариабельность измеренных значений. Сопоставление полученных данных с эталонными значениями, характерными для системы без гипервизора, позволяет выявить статистически значимые отклонения, указывающие на присутствие гипервизора.

Ключевым элементом методики является разработка математических моделей, описывающих зависимость времени вы-

полнения трассы от параметров работы процессора и наличия гипервизоров. Эти модели учитывают количество и типы переключений между режимами, влияние гипервизоров на счетчики тактов и другие системные характеристики. Применение таких моделей повышает возможность обнаружения гипервизора и позволяет прогнозировать их влияние на работу системы.

15.3.3. Допущения для построения моделей выполнения трассы

Для построения представленных далее моделей выполнения трассы принимаются следующие допущения [13]:

- *последовательное выполнение инструкций* — все инструкции процессора выполняются строго последовательно на одном и том же абстрактном вычислительном устройстве без параллельной обработки или конвейеризации инструкций: каждая следующая инструкция начинает выполняться только после того, как полностью завершилась предыдущая;
- *равная длительность выполнения инструкций* — длительность выполнения каждой инструкции считается одинаковой и составляет k тактов процессора. В общем случае время выполнения инструкций может варьироваться и зависеть от значений регистров, однако в рамках рассматриваемой модели это опускается;
- при получении управления *гипервизор изменяет показания счетчика тактов на фиксированную величину* для того, чтобы скрыть свое присутствие;
- *случайное число переключений процессора* из защищенного режима (R-режим) в системный режим (S-режим) *подчиняется биномиальному распределению*. Биномиальное распределение выбрано потому, что переключения являются дискретными событиями с двумя возможными исходами.

15.3.4. Модель выполнения трассы в случае отсутствия гипервизора

Модель выполнения трассы в случае отсутствия гипервизора может быть представлена как ориентированный мультиграф с помеченными дугами (рис. 116) [505].

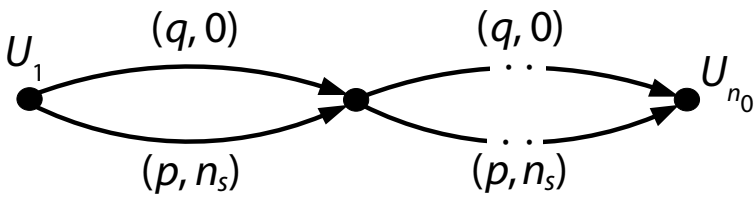


Рис. 110. Модель выполнения трассы в случае отсутствия гипервизора

Вершины графа $U_1, U_2, \ldots, U_{n_0}$ соответствуют инструкциям трассы, где n_0 — общее число инструкций трассы; n_S — число инструкций в обработчике S-режима. Дуги отражают возможные переключения между режимами процессора. При выполнении каждой инструкции трассы процессор может перейти из R- в S-режим с вероятностью p, что показано нижней дугой. Верхняя дуга иллюстрирует выполнение инструкции без перехода в S-режим с вероятностью $q = 1 - p$ [506].

Длительность выполнения трассы $t_{\text{ОТ}}$ из n_0 инструкций в случае отсутствия гипервизора можно выразить формулой (4), где случайная величина $m_{\text{ОТ}}$, соответствующая числу переключений в S-режим, подчиняется закону биномиального распределения с параметрами n_0 и p [507]:

$$t_{\text{ОТ}} = \left(n_0 + m_{\text{ОТ}} \cdot n_S\right) \cdot k. \qquad (4)$$

В случае присутствия одного образца гипервизора модель трассы также описывается ориентированным мультиграфом с дугами (рис. 111), однако при выполнении каждой инструкции трассы происходит передача управления гипервизору. Вершины $V_1, V_2, \cdots, V_n$ обозначают инструкции гипервизора, где n_V — число инструкций в этом образце. При выполнении любых инструкций как трассы, так и гипервизора возможен переход в S-режим, что обозначено нижними дугами на рис. 111. Ситуации, когда таких переходов нет, обозначены верхними дугами.

В случае присутствия одного гипервизора, когда он не изменяет и не искажает значения счетчика тактов процессора, длительность выполнения трассы $t'_{\text{ПР}}$ из n_0 инструкций можно представить следующей формулой:

$$t'_{\text{ПР}} = \left(n_0 + n_0 \cdot n_V + m_{\text{ПР}} \cdot n_S\right) \cdot k, \qquad (5)$$

где случайная величина $m_{\text{ПР}}$, соответствующая числу переключений в S-режим, подчиняется закону биномиального распределения с параметрами $(n_0 + n_0 \cdot n_V)$ и p; слагаемое $n_0 \cdot n_V$ соответствует тому, что при выполнении каждой из n_0 инструкций трассы происходит переключение в V-режим (передача управления гипервизору), на что тратится дополнительное время.

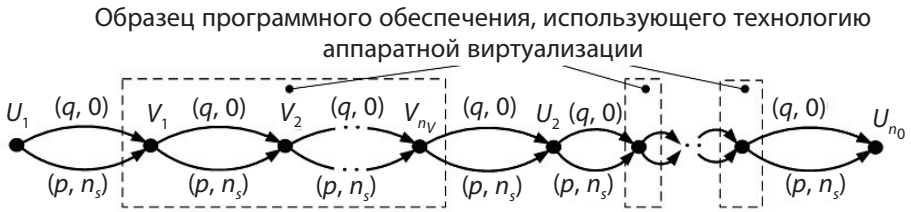


Рис. 111. Модель выполнения трассы в случае присутствия одного образца гипервизора

Если гипервизор производит компрометацию счетчика, прибавляя фиксированное значение $\delta < 0$, то длительность выполнения трассы с компрометацией можно представить следующим образом:

$$t_{\text{ПР}} = \left(n_0 + n_0 \cdot n_V + m_{\text{ПР}} \cdot n_S + n_0 \cdot \delta\right)k. \qquad (6)$$

В случае наличия нескольких вложенных образцов гипервизоров модель выполнения трассы также описывается ориентированным мультиграфом с помеченными дугами (рис. 112). Основное отличие от предыдущей модели заключается в том, что управление передается последовательно каждому из вложенных образцов гипервизоров.

В результате выполнения трассы длительность увеличивается за счет обработки инструкций каждым из вложенных образцов. Вершина $V_i^{(j)}$, где $i = 1, \ldots, n_j$, $j = 1, \ldots, e$ интерпретируется как i-я инструкция j-го образца гипервизора; n_j — число инструкций в обработчике j-го гипервизора; e — число вложенных образцов гипервизоров.

Тогда длительность выполнения трассы определяется как

$$t_{\text{ВЛ}} = \left(n_0 + n_0 \cdot \sum_{j=1}^{e} n_j + m_{\text{ВЛ}} \cdot n_S + n_0 \cdot \sum_{j=1}^{e} \delta_j \right) k, \qquad (7)$$

где $m_{\text{ВЛ}}$ — случайная величина, соответствующая числу переключений в S-режим и подчиняющаяся закону биномиального распределения с параметрами $\left(n_0 + n_0 \cdot \sum_{j=1}^{e} n_j \right)$ и p. При этом вероятность переключения в S-режим p осталась неизменной, а число инструкций, при выполнении которых возможен переход в S-режим (количество независимых испытаний в терминах схемы Бернулли), увеличилось по сравнению со случаем отсутствия гипервизора на величину $n_0 \cdot \sum_{j=1}^{e} n_j$, свидетельствующую о возрастании числа переключений в S-режим. Сумма $\sum_{j=1}^{e} n_j$ обусловлена тем, что выполнение очередной инструкции трассы прерывается и обрабатывается поочередно каждым вложенным гипервизором. Слагаемое $n_0 \cdot \sum_{j=1}^{e} \delta_j$ — величина длительности компрометации счетчика при обработке инструкций каждым гипервизором. При передаче управления j-му гипервизору он прибавляет к значению счетчика тактов величину δ_j, $j = 1, \ldots, e$.

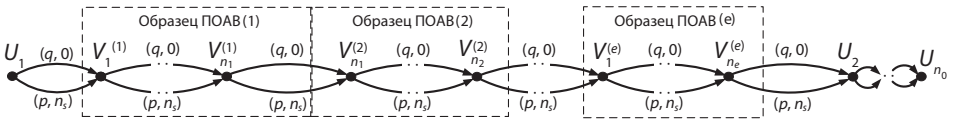


Рис. 112. Модель выполнения трассы в случае присутствия *e* вложенных образцов гипервизоров

Таким образом, с увеличением количества вложенных гипервизоров возрастает как общее время выполнения трассы, так и количество переключений в S-режим.

15.3.5. Валидация разработанных моделей на основе экспериментальных данных

Целью экспериментов было создание эмпирической базы, необходимой для последующего сравнения распределений времени

выполнения трассы, полученных экспериментально и сформированных на основе математических моделей.

Проверка заключалась в оценке возможности получения с помощью формул (4)–(7) значений длительности выполнения трассы, сопоставимых с реально наблюдаемыми в условиях отсутствия и присутствия гипервизора с искажением показаний счетчика тактов. Кроме того, осуществлялась оценка соответствия распределений расчетных и наблюдаемых значений длительности выполнения трассы с использованием критерия согласия.

В результате исследования установлено, что подбор приемлемых значений параметров, входящих в формулы (4)–(7), позволяет добиться совпадения расчетных величин длительности выполнения трассы с экспериментальными данными. Это свидетельствует о том, что предложенные теоретические модели не противоречат полученным опытным данным.

Для оценки согласования между теоретическими и экспериментальными распределениями было проведено 1000 последовательных измерений времени выполнения цикла из 10 инструкций CPUID на различных компьютерах. Эксперименты выполнялись в условиях преднамеренного искажения показаний счетчика тактов, при котором выборочные средние значения длительности выполнения трассы в случаях отсутствия и присутствия гипервизора совпадали.

Такой подход позволил оценить степень соответствия теоретических предположений реальным наблюдениям и проверить адекватность предложенной модели исследуемого технического процесса.

Характерной особенностью полученных гистограмм и полигонов распределения является наличие протяженного правого «хвоста», указывающего на относительно большое количество редких значений в вариационных рядах длительности выполнения трассы. В подобных случаях дисперсия и центральные моменты четвертого порядка оказываются очень большими [508]. Это приводит к тому, что различия между статистическими характеристиками в случаях отсутствия и присутствия гипервизора не всегда удается обнаружить во всех экспериментах.

Для преодоления данной проблемы экспериментальные массивы данных о длительности выполнения трассы подвергались низкочастотной фильтрации.

Эксперимент показал, что даже при небольшом уровне фильтрации (всего 0,02) длина вариационных рядов сокращалась с 10–15 до стабильных 2–4 значений (рис. 113 и 114). В результате фильтрации оставались только значимые данные, определяющие характерные особенности выборок. Это обеспечивало приемлемую различимость статистических показателей, используемых для обнаружения гипервизора. В параграфе 16.2 подробно описаны методы достижения данной цели.

Другой характерной особенностью экспериментальных данных является недостаточная сходимость и воспроизводимость результатов измерений. Согласно ГОСТ Р ИСО 5725-1–2002 «Точность (правильность и прецизионность) методов и результатов измерений» (Часть 1. Основные положения и определения) под условиями сходимости результатов понимаются такие условия, при которых независимые результаты измерений получают одним и тем же методом на идентичных объектах при неизменных факторах выполнения измерений в пределах одной лаборатории. Согласно ГОСТ Р ИСО 5725-1–2002 условия воспроизводимости — это условия, при которых результаты измерений получают одним и тем же методом на идентичных объектах в разных лабораториях, разными операторами, с использованием различного оборудования.

Как показано на рисунках, это проявляется в смещении одних полигонов относительно других при неизменных условиях повторных экспериментов.

Генерация массивов длительности выполнения трассы с использованием формул (4) и (6), основанных на биномиальном распределении, осуществлялась в среде Matlab с использованием функции binornd. Для полученных массивов были построены вариационные ряды, представленные модельными полигонами (см. рис. 113 и 114).

Установлено, что в условиях отсутствия гипервизора модельные полигоны полностью совпадают с экспериментальными (на рис. 113 для наглядности сравниваемые полигоны искусственно смещены по горизонтали на один такт).

В случае присутствия гипервизора согласие распределений оценивалось с помощью критерия согласия Колмогорова [509–513] с использованием функции kstest2 в Matlab. Проверка показала, что гипотеза о совпадении модельного и экспериментального распределений на уровне значимости 5% не отвергается. Правомерность применения критерия Колмогорова для дискретных распределений обоснована в работах [510, 512, 513].

Аналогичные результаты были получены и для других экспериментальных данных. На основании этого сделан вывод, что гипотеза об адекватности предложенных моделей выполнения трассы экспериментальным данным не отвергается и данные модели могут быть использованы для разработки критериев присутствия гипервизоров.

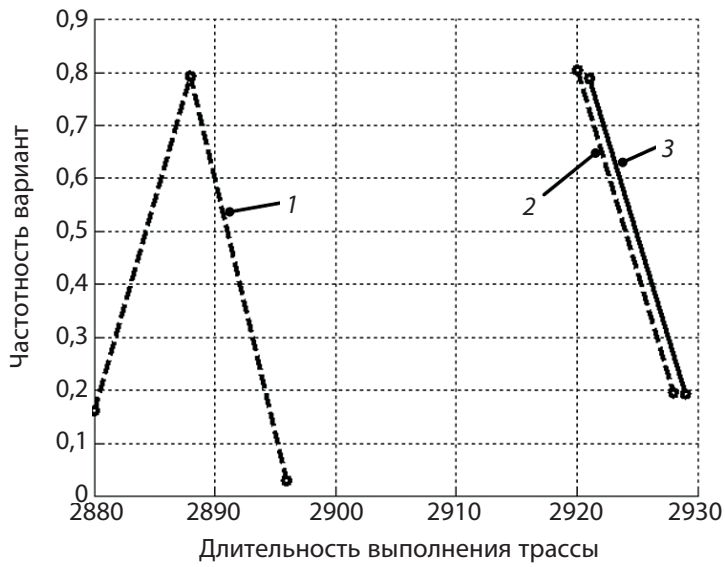


Рис. 113. Экспериментальные (*1* и *2*) для двух повторных опытов и модельный (*3*) полигоны распределений длительности выполнения трассы в случае отсутствия гипервизора для компьютера с процессором Intel Core 2 Duo E8200 под управлением ОС Windows 7

В результате анализа рис. 113 можно заметить, что графики, полученные из двух повторных испытаний при отсутствии гипервизора, показывают разные значения среднего времени выполнения трассы. В опыте № 1 интервал составил 2880–2890 тактов, а в опыте № 2 — 2920–2930 тактов.

Когда гипервизор присутствует (см. рис. 114), графики также демонстрируют различия в среднем времени выполнения, трассы с интервалами варьирования в опыте № 1 от 2875 до 2900 тактов и в опыте № 2 от 2908 до 2935 тактов соответственно. Подобные тенденции зафиксированы и на других компьютерных системах в условиях как отсутствия, так и присутствия гипервизоров.

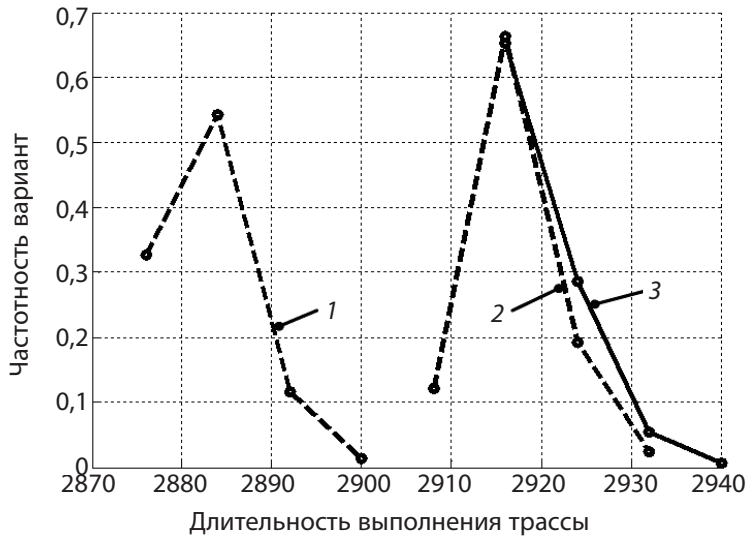


Рис. 114. Экспериментальные (*1* и *2*) и модельный (*3*) полигоны распределения длительности выполнения трассы в случае присутствия гипервизора с искажением счетчика тактов для процессора Intel Core 2 Duo E8200 и ОС Windows 7

Анализ экспериментальных результатов выявил низкую степень их согласованности, что существенно затрудняет оценку соответствия данных модельным предсказаниям. Несмотря на возможность подбора отдельного модельного распределения для каждого эксперимента — как было осуществлено в одном случае — сравнение модельных и эмпирических распределений, полученных на одной и той же вычислительной системе, показало, что выбранные модели не способны адекватно описать все экспериментальные данные.

Неоднородность экспериментальных данных осложняет процесс сопоставления модельных распределений с реальными результатами, что указывает на ограниченность применения ме-

тодов математической статистики для подтверждения гипотез о соответствии в данных условиях. В частности, стандартные статистические критерии могут быть неприменимы из-за несоблюдения необходимых предпосылок, таких как однородность и независимость наблюдений.

Невозможность соответствия модельного распределения всем экспериментальным данным из-за их недостаточной повторяемости (сходимости) указывает на неприменимость классических методов математической статистики для проверки гипотез согласия в этой ситуации.

Данное явление, заключающееся в различиях средних значений результатов повторных экспериментов, стало известно как «*дрейф данных*». В связи с этим было принято решение об использовании альтернативного критерия для обнаружения присутствия гипервизора, который будет описан в следующих параграфах.

Использование дополнительных характеристик длительности выполнения трассы представляется перспективным подходом для обнаружения гипервизора, поскольку такие параметры позволяют более точно фиксировать отклонения в поведении системы, вызванные присутствием гипервизора. Так, из уравнения (5) следует, что если количество переключений $m_{\text{ПР}}$ в S-режим оказывается достаточно малым,

$$m_{\text{ПР}} < \left(1 + n_V + \delta\right)\frac{n_0}{n_S}, \tag{8}$$

где $\delta < 0$, то измеряемая длительность выполнения трассы $t_{\text{ПР}}$ в СВТ с гипервизором приобретет отрицательное значение. Можно предположить, что среднее выборочное длительности выполнения трассы $t_{\text{ПР}}$ также может стать отрицательным при достаточно большом количестве таких событий.

Присутствие отрицательных значений в массивах длительности выполнения трассы не только указывает на наличие гипервизора, но и демонстрирует несостоятельность манипуляций со счетчиком тактов, используемых для маскировки гипервизора, что делает их выявление более надежным.

Поскольку величины $m_{\text{ОТ}}$, $m_{\text{ПР}}$ и $m_{\text{ВЛ}}$ являются случайными, длительности выполнения трассы $t_{\text{ОТ}}$, $t'_{\text{ПР}}$, $t_{\text{ПР}}$ и $t_{\text{ВЛ}}$ могут рассматриваться как линейные функции случайного аргумента. Согласно работе [506] законы распределения длительности выполнения трассы и числа переключений в S-режим совпадают, что позволяет вычислять их числовые характеристики через распределение аргументов.

Для случая биномиального распределения математическое ожидание длительности выполнения трассы $E[t_{\text{ОТ}}]$ в случае отсутствия гипервизора с учетом [514] определяется следующим образом:

$$E\left[t_{\text{ОТ}}\right] = n_0 \cdot k\left(p \cdot n_S + 1\right). \tag{9}$$

В случае наличия гипервизора и с учетом компрометации счетчика тактов на величину δ математическое ожидание длительности выполнения трассы $E[t_{\text{ПР}}]$ определится как

$$E\left[t_{\text{ПР}}\right] = n_0 \cdot k\left(\delta + p \cdot n_S\left(n_V + 1\right) + n_V + 1\right). \tag{10}$$

Приравняв правые части формул (9) и (10), после преобразований можно получить выражение для величины искажения счетчика тактов δ, при котором математические ожидания длительности выполнения трассы $\overline{t}_{\text{ОТ}}$ и $\overline{t}_{\text{ПР}}$ совпадут:

$$\left[\delta\right] = -\left(p \cdot n_S + 1\right)n_V. \tag{11}$$

Из уравнения (11) следует, что теоретически возможно такое искажение показаний счетчика тактов, при котором на основе выборочного среднего значения длительности выполнения трассы не удастся выявить наличие гипервизора в системе. Однако, поскольку параметры уравнения (11) неизвестны, определить точное значение [δ] сложно. Следовательно, нарушителю придется экспериментально подбирать величину искажения [δ], чтобы средние значения длительности выполнения трассы $\overline{t}_{\text{ОТ}}$ и $\overline{t}_{\text{ПР}}$ совпадали с определенной точностью.

Случайные изменения выборочных средних при повторных измерениях затрудняют этот процесс. Это означает, что даже при постоянном значении [δ] случайные вариации в количестве

переключений процессора в S-режим $m_{ОТ}$, $m_{ПР}$ и $m_{ВЛ}$ могут привести к значительным различиям между выборочными средними $\overline{t}_{ОТ}$ и $\overline{t}_{ПР}$. Таким образом, средние значения $\overline{t}_{ПР}$ могут быть как значительно выше, так и ниже, чем $\overline{t}_{ОТ}$.

Из этого следует, что нарушитель не может точно исказить показания счетчика для сокрытия гипервизора. Поэтому процесс тестирования на наличие гипервизора рекомендуется начинать с анализа тестовой выборки $T_{тест}$ длительности выполнения трассы. Необходимо проверить наличие отрицательных значений в выборке и сопоставить выборочную среднюю $\overline{t}_{тест}$ с нормативной $[\overline{t}_{ОТ}]$, которую можно определить до начала тестирования СВТ.

Дополнительный признак для обнаружения гипервизора можно выявить при анализе модельных формул. Поскольку переменные в уравнениях (4)–(7) принимают только положительные целые значения, длительности выполнения трассы, как в случае отсутствия, так и присутствия гипервизора, хотя и представляют собой случайные величины, могут группироваться на определенных *постоянных* уровнях. Это подтверждается полигонами распределения, полученными в экспериментах (см. рис. 113 и 114). По этой причине точечные графики длительности выполнения трассы должны демонстрировать слоистую структуру, и в случае присутствия гипервизора количество уровней будет больше по сравнению со случаем отсутствия гипервизора. Эти различия могут быть использованы для обнаружения гипервизора.

Анализ предложенных моделей функционирования процессора дал возможность выделить ряд параметров, с помощью которых можно обнаружить гипервизор. Другие идентификационные параметры представлены в следующих параграфах.

15.3.6. Критерии обнаружения гипервизора

Рассмотрим критерии, позволяющие выявлять присутствие гипервизора на основе различия выборок длительности выполнения трассы в случаях его отсутствия и присутствия [515].

Длительность выполнения трассы в случаях отсутствия и присутствия гипервизора $t_{\text{ОТ}}$ и $t_{\text{ПР}}$ были ранее представлены уравнениями (4) и (6):

$$t_{\text{ОТ}} = \left(n_0 + m_{\text{ОТ}} \cdot n_S\right)k;$$
$$t_{\text{ПР}} = \left(n_0 + n_0 \cdot n_V + m_{\text{ПР}} \cdot n_S + n_0 \cdot \delta\right)k. \quad (12)$$

Здесь n_0 — количество инструкций в трассе; n_S и n_V — количество инструкций в обработчике режима системного управления (S-режим) и в обработчике гипервизора соответственно; $m_{\text{ОТ}}$ и $m_{\text{ПР}}$ — случайные величины, соответствующие числу переключений процессора в S-режим и подчиняющиеся закону биномиального распределения с параметрами $[n_0,\ p]$ и $[n_0 + n_0 \cdot n_V,\ p]$ соответственно для случаев отсутствия и присутствия гипервизора; k — длительность выполнения одной инструкции; δ — величина компрометации счетчика тактов со стороны нарушителя при выполнении каждой инструкции в трассе ($\delta < 0$).

Ранее было показано, что нарушитель может исказить показания счетчика тактов таким образом, что выборочные средние значения длительности выполнения трассы в условиях отсутствия и присутствия гипервизора становятся неразличимыми. Это делает среднее значение длительности выполнения трассы непригодным для обнаружения гипервизора. Однако практический опыт свидетельствует о том, что такие статистические характеристики, как дисперсия δ^2, момент четвертого порядка υ_2, длина вариационного ряда l, при соблюдении определенных условий могут быть использованы для обнаружения гипервизора в условиях противодействия со стороны нарушителя [515, 516].

Пусть параметры $n_0, n_S, n_V, \delta, k \in \mathbb{Z};\ p \in \mathbb{R}$ — некоторые фиксированные значения. Введем обозначения:

- $X_{\text{ОТ}}$ — множество значений, полученных при реализации $t_{\text{ОТ}}$;
- $X_{\text{ПР}}$ — множество значений, полученных при реализации $t_{\text{ПР}}$;
- генеральная совокупность X определяется как $X = X_{\text{ОТ}} \cup X_{\text{ПР}}$;
- $\vec{x}_n = (x_1, ..., x_n)$ — выборка объема n из генеральной совокупности X.

Ниже сформулированы критерии для обнаружения присутствия гипервизора.

Критерий на *основе дисперсии*. *Критическое множество* (гипервизор присутствует):

$$W = \left\{ \vec{x}_n : \acute{\sigma}^2 \left(\vec{x}_n \right) \geq d \right\},$$

где σ^2 — выборочная дисперсия; $d \in \mathbb{Z}$ — пороговое значение, которое определяется экспериментально.

Правило принятия решения:

- если $\vec{x}_n \in W$, то гипервизор присутствует;
- если $\vec{x}_n \notin W$, то гипервизор отсутствует.

Критерий на основе момента четвертого порядка. *Критическое множество* (гипервизор присутствует):

$$W = \left\{ \vec{x}_n \ \acute{\upsilon} \ \left(\vec{x}_n \right) \geq \mu \right\},$$

где υ_2 — выборочный центральный момент четвертого порядка; $\mu \in \mathbb{Z}$ — пороговое значение, которое определяется экспериментально.

Правило принятия решения:

- если $\vec{x}_n \in W$, то гипервизор присутствует;
- если $\vec{x}_n \notin W$, то гипервизор отсутствует.

Критерий на основе длины вариационного ряда. *Критическое множество* (гипервизор присутствует):

$$W = \left\{ \vec{x}_n : \acute{l} \left(\vec{x}_n \right) \geq e \right\},$$

где l — длина вариационного ряда, построенного по выборке; $e \in \mathbb{Z}$ — определяется экспериментально.

Правило принятия решения:

- если $\vec{x}_n \in W$, то гипервизор присутствует;
- если $\vec{x}_n \notin W$, то гипервизор отсутствует.

Пороговые значения d, μ, e, значения вероятностей ошибок I и II рода α и β, а также объем выборки n определяются экспериментально.

Утверждение 1. Пусть t — выборка объема n, $f(t) = D_{\text{в}}[t]$ — выборочная дисперсия выборки t, тогда

$$D_{\text{в}}\left[t_{\text{ПР}}\right] = D_{\text{в}}\left[t_{\text{ОТ}}\right]\left(n_V + 1\right). \tag{13}$$

Доказательство

Приведем значения дисперсий для случайных величин $t_{\text{ОТ}}$ и $t_{\text{ПР}}$ [514].

Выполняются равенства:

$$D_{\text{в}}\left[t_{\text{ОТ}}\right] = n_0\left(1 - p\right)p \cdot k^2 \cdot n_S^2; \tag{14}$$

$$D_{\text{в}}\left[t_{\text{ПР}}\right] = n_0\left(1 - p\right)p \cdot k^2 \cdot n_S^2\left(n_V + 1\right). \tag{15}$$

Из равенств (14) и (15) получаем, что:

$$\frac{D_{\text{в}}\left[t_{\text{ПР}}\right]}{D_{\text{в}}\left[t_{\text{ОТ}}\right]} = \frac{n_0\left(1 - p\right)p \cdot k^2 \cdot n_S^2\left(n_V + 1\right)}{n_0\left(1 - p\right)p \cdot k^2 \cdot n_S^2} = n_V + 1. \tag{16}$$

Значит, справедливо выражение (13). Утверждение доказано.

Из формулы (13) следует, что выборочная дисперсия в случае присутствия гипервизора превышает выборочную дисперсию в случае его отсутствия в $(n_V + 1)$ раз.

Утверждение 2. Пусть t — выборка объема n, $f(t) = \mu_{\text{в}}[t]$ — выборочный центральный момент четвертого порядка выборки t, тогда $\mu_{\text{в}}[t_{\text{ПР}}] > \mu_{\text{в}}[t_{\text{ОТ}}]$.

Доказательство

Приведем значения центральных моментов четвертого порядка для случайных величин $t_{\text{ОТ}}$ и $t_{\text{ПР}}$ [514].

Выполняются равенства:

$$\mu_{\text{в}}\left[t_{\text{ОТ}}\right] = n_0 \cdot p \cdot q\left(3\left(n_0 - 2\right)p \cdot q + 1\right); \tag{17}$$

$$\mu_{\text{в}}\left[t_{\text{ПР}}\right] = \left(n_0 + n_0 \cdot n_V\right)p \cdot q\left(3\left(n_0 + n_0 \cdot n_V - 2\right)p \cdot q + 1\right). \tag{18}$$

Из равенств (17) и (18) получаем, что:

$$\frac{\mu_в[t_{ПР}]}{\mu_в[t_{ОТ}]} = \frac{(n_0 + n_0 \cdot n_V)p \cdot q(3(n_0 + n_0 \cdot n_V - 2)p \cdot q + 1)}{n_0 \cdot p \cdot q(3(n_0 - 2)p \cdot q + 1)}. \quad (19)$$

Приведем подобные члены и получим:

$$\frac{\mu_в[t_{ПР}]}{\mu_в[t_{ОТ}]} = 1 + 6\frac{n_0 \cdot n_V}{3(n_0 - 2) + \frac{1}{p \cdot q}} > 1. \quad (20)$$

Значит, при $n_0 > 2$

$$\mu_в[t_{ПР}] > \mu_в[t_{ОТ}]. \quad (21)$$

Из формулы (21) следует, что центральный момент четвертого порядка в случае присутствия гипервизора превышает соответствующее значение в случае его отсутствия. Утверждение доказано.

Гипотеза. Пусть t — выборка объема n, $f(t) = v_в[t]$ — значение длины вариационного ряда выборки t, тогда $v_в[t_{ПР}] > v_в[t_{ОТ}]$.

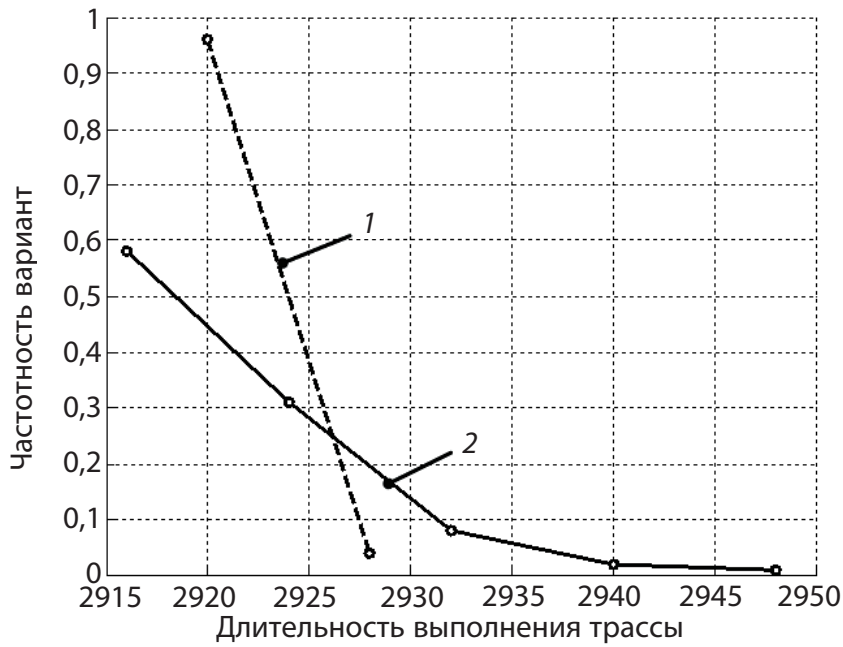


Рис. 115. Графики полигонов двух модельных распределений в случае отсутствия гипервизора (*1*) и присутствия гипервизора (*2*)

Другими словами, длина вариационного ряда выборки в присутствии гипервизора больше, чем длина вариационного ряда выборки при его отсутствии.

Графики полигонов модельных распределений в случаях отсутствия и присутствия гипервизора представлены на рис. 115. Из этих графиков следует, что число уровней (точек) в случае присутствия гипервизора значительно больше, чем в случае его отсутствия. Эта отличительная особенность подтверждается многочисленными экспериментами, проведенными на различных компьютерах.

Глава 16
СТАТИСТИЧЕСКИЕ ХАРАКТЕРИСТИКИ ДЛИТЕЛЬНОСТИ ВЫПОЛНЕНИЯ НАБОРА ПРОЦЕССОРНЫХ ИНСТРУКЦИЙ. МЕТОДИКА ОБНАРУЖЕНИЯ СКРЫТЫХ ГИПЕРВИЗОРОВ

По данной проблематике существует значительный дефицит публикаций. Информация, представленная в доступных источниках, обладает ограниченной детализацией и содержит противоречивые данные. В частности, в исследовании [434] представлены дискретные графики, иллюстрирующие отдельные реализации случайного процесса, связанного с измерениями времени выполнения заданного набора процессорных инструкций. Однако в работе отсутствует комплексный анализ полученных результатов, а также не сформулированы соответствующие выводы и рекомендации, касающиеся обнаружения гипервизоров в системах.

В другой работе [25] предлагается подход к проблеме обнаружения гипервизоров, основанный на предположении, что распределение длительности выполнения трассы подчиняется нормальному закону. Однако авторы не предоставили экспериментального подкрепления данного предположения. Кроме того, не учитываются возможные противодействия нарушителя попыткам обнаружить гипервизор. Важно отметить, что вопреки требованиям стандарта ГОСТ Р ИСО 5725–2002 в работе не приведено никаких сведений о прецизионности (сходимости и воспроизводимости) опытных данных.

В ходе исследования, проведенного автором данной монографии, включающего серию экспериментов на различных компьютерах, было выявлено, что распределение длительности выполнения трассы отклоняется от нормального закона во всех без исключения случаях. Этот вывод иллюстрирует рис. 116, на котором показаны типичные гистограммы с характерными одним или двумя высокими столбцами и незначительным правым «хвостом».

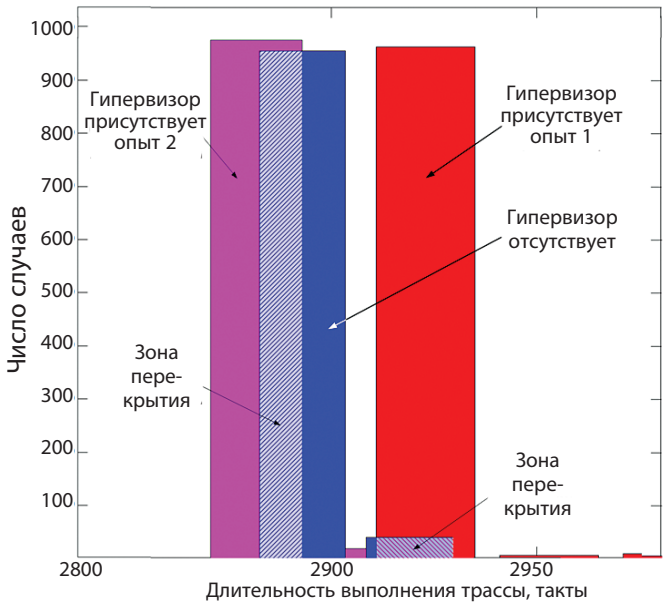


Рис. 116. Гистограммы длительности выполнения трассы по данным опытов без гипервизора и с искажением показаний счетчика тактов в случае присутствия гипервизора

Другая особенность — неудовлетворительная сходимость опытных данных. На рис. 116 типичные гистограммы двух повторных опытов для случая с гипервизором статистически неоднородны. Аналогичные расхождения между гистограммами повторных опытов наблюдаются и в случаях отсутствия гипервизора. Таким образом, можно сделать вывод, что данный эффект не зависит от наличия или отсутствия гипервизора. Этот эффект подтвердился в результате многочисленных опытов.

Поскольку неудовлетворительная прецизионность имеет постоянный и неустранимый характер, получаемые из опытов выборки длительности выполнения трассы нельзя считать статистически представительными (репрезентативными) в соответствии с действующим в настоящее время стандартом России ГОСТ Р ИСО 5725–2002 [517], который регулирует точность методов и результатов измерений. Однако в 2019–2023 гг. были выпущены новые редакции международных стандартов ISO 5725-2:2019, ISO 5725-4:2020, ISO 5725-1:2023 и ISO 5725-3:2023, в которых некоторые методы и подходы были обновлены [518–521].

Анализ экспериментальных данных (см. рис. 116) показывает, что компрометация счетчика тактов в целях сокрытия присутствия гипервизора путем уменьшения его показаний на некоторую постоянную величину не гарантирует точного совпадения выборочных средних значений длительности выполнения трассы в случаях отсутствия и присутствия гипервизора. Фактически при повторных экспериментах гистограммы длительностей выполнения трассы в присутствии гипервизора могут смещаться как влево, так и вправо относительно гистограмм для случая его отсутствия.

Из-за случайного характера длительностей выполнения трассы у нарушителя отсутствует гарантированная возможность точно исказить показания счетчика тактов, чтобы полностью скрыть присутствие гипервизора. Однако для сокрытия гипервизора по признаку совпадения выборочных средних значений этого и не требуется, поскольку интервалы варьирования этих статистик перекрываются. Даже при незначительных искажениях показаний счетчика тактов гистограммы длительностей выполнения трассы в случаях отсутствия и присутствия гипервизора могут существенно перекрываться. Это обстоятельство затрудняет применение стандартных статистических методов для обнаружения гипервизора в системе.

Из изложенного следует, что:

- распределение длительностей выполнения трассы не подчиняется нормальному закону, что ограничивает применение параметрических методов статистического анализа;
- гистограммы длительностей выполнения трассы при отсутствии и присутствии гипервизора с целенаправленным искажением показаний счетчика тактов могут значительно перекрываться, затрудняя дифференциацию этих случаев.

В таких условиях применение методов математической статистики, как параметрических, так и непараметрических, оказывается неправомерным для выявления гипервизоров.

Следовательно, при разработке алгоритмов обнаружения гипервизора необходимо учитывать, что выявление по выборочным средним значениям длительности выполнения трассы возможно только в тех случаях, когда нарушитель недостаточно тщательно искажает показания счетчика тактов.

В общем случае для обнаружения гипервизора требуются статистические показатели, способные выявлять его присутствие независимо от степени искажения счетчика тактов. Исследуем возможности использования альтернативных статистик, характеризующих вариабельность длительностей выполнения трассы, таких как дисперсия и ее диапазон вариации, вариационные ряды, полигоны распределения и их длины, а также зависимость распределения длительностей выполнения трассы по фиксированным уровням в случаях присутствия и отсутствия гипервизора.

Для повышения точности результатов эксперименты были тщательно организованы и применялись специально разработанные методы обработки статистических данных.

Экспериментальные исследования проводились в два этапа:

- первый этап включал экспериментальную проверку выводов, полученных из анализа моделей функционирования процессора (частично описанных в главе 15). Были выявлены особенности длительностей выполнения трассы в случаях отсутствия и присутствия гипервизора, на основании чего определялись подходы к обработке экспериментальных данных;
- второй этап был посвящен поиску статистических показателей длительностей выполнения трассы, пригодных в качестве индикативных признаков для обнаружения гипервизора.

Эксперименты проводились по схеме однофакторных испытаний с учетом разрабатываемой методики обнаружения нелегитимного гипервизора. Варьируемым фактором выступала операционная среда вычислительной системы с двумя качественными уровнями: отсутствие и присутствие гипервизора. Результирующими признаками являлись различные статистические показатели длительности выполнения трассы.

На основании проведенных исследований предложена методика обнаружения одного и нескольких образцов гипервизоров, основанная на анализе статистических характеристик вариабельности длительностей выполнения трассы, что позволяет выявлять присутствие гипервизоров даже при попытках нарушителя скрыть присутствие гипервизора посредством компрометации счетчика тактов.

16.1. ЭКСПЕРИМЕНТАЛЬНЫЕ ИССЛЕДОВАНИЯ ИЗМЕРЕНИЙ ДЛИТЕЛЬНОСТИ ВЫПОЛНЕНИЯ ТРАССЫ

На основе данных первоначальных экспериментов разработка методов обнаружения гипервизоров была сосредоточена на анализе статистических различий по времени выполнения трассы, состоящей из 10 инструкций CPUID, выполняемых в режиме ядра с высоким приоритетом в условиях присутствия и отсутствия гипервизора. Методика определения длительности выполнения трассы в случаях отсутствия $t_{\text{ОТ}}$ и присутствия $t_{\text{ПР}}$ гипервизора с компрометацией счетчика тактов изложена в главе 15.

Для минимизации влияния случайных факторов на распределение длительности выполнения трассы $t_{\text{ПР}}$ был разработан оригинальный образец гипервизора, содержащий минимальный набор инструкций в диспетчере и обеспечивающий компрометацию процессорных счетчиков тактов.

Компрометация счетчика тактов осуществлялась таким образом, чтобы средние значения многократных измерений длительности выполнения трассы при отсутствии гипервизора ($\overline{t}_{\text{ОТ}}$) и в его присутствии ($\overline{t}_{\text{ПР}}$) практически не различались. Однако результаты измерений длительности трассы оказались нестабильными. Эксперименты подтвердили теоретические выводы, сделанные при анализе моделей (4)–(7), указывая на нестабильность времени выполнения трассы: при одинаковых условиях экспериментов и фиксированной величине компрометации счетчика δ длительность выполнения трассы $t_{\text{ПР}}$ могла значительно варьироваться в разные дни, а в некоторых случаях принимать отрицательные значения. Это явление, известное как «дрейф данных», свидетельствует о существенном влиянии случайных факторов.

Таким образом, подтверждается рекомендация, изложенная в главе 15, о том, что тестирование на наличие гипервизора следует начинать с проверки тестовых массивов данных длительности выполнения трассы $T_{\text{тест}}$ на наличие отрицательных значений, а также с анализа степени отклонения средней длительности выполнения трассы $\overline{t}_{\text{тест}}$ от нормативного значения $[t_{\text{ОТ}}]$, характерного для системы без гипервизора.

Измерения длительности выполнения трассы проводились с использованием инструкции RDTSC, которая позволяет считывать показания счетчика тактов процессора (TSC) до и после выполнения трассы. Для предотвращения системных искажений регистрация данных выполнялась с учетом механизма сериализации [522–524], подробно рассмотренного в подпараграфе 14.2.2 и параграфе 14.6. В целях повышения точности и надежности результатов между вложенными циклами измерений устанавливалась пауза длительностью 2 с. В результате экспериментов были получены матрицы данных $T_{\text{ОТ}}$ и $T_{\text{ПР}}$ размером 1000×10, где каждый столбец представлял результаты измерений длительности выполнения трассы для конкретного вложенного цикла.

В соответствии с международными и российскими стандартами [517–521] для оценки сходимости данных эксперименты проводились сериями из пяти повторяющихся испытаний с небольшими интервалами между ними. Для оценки воспроизводимости результаты проверялись серией повторных экспериментов, выполненных в течение 10 дней. Эти испытания показали, что данный период был достаточен для стабилизации границ варьирования определяемых статистических показателей.

Общее тестирование проводилось на шести компьютерах с различными моделями процессоров. Характеристики процессоров, версии ОС представлены в табл. 31. На первых пяти компьютерах был установлен разработанный образец прототипа гипервизора, реализованный в виде драйвера ОС. На шестом использовался специализированный образец скрытого гипервизора, получающий управление при загрузке системы из BIOS. Исследования показали, что такой скрытый образец работал только на одном из ядер многоядерной системы, ожидая получения управляющей команды для запуска на всех ядрах.

Таблица 31

Модели процессоров и ОС на компьютерах, использованных в опытах

Номер СВТ	Модель процессора и установленная ОС
1	Intel Core 2 Duo E6300 с Windows 7
2	Intel Core 2 Duo E8200 с Windows 7

Окончание табл. 31

Номер СВТ	Модель процессора и установленная ОС
3	Intel Core 2 Duo E8600 с Windows Live CD XP
4	Intel Core i7 950 с Windows XP
5	Intel Xeon X5600 с Windows 7
6	AMD Phenom X4 945 с Windows Live CD XP

16.2. ХАРАКТЕРИСТИКИ ВЫПОЛНЕНИЯ ПРОЦЕССОРНЫХ ИНСТРУКЦИЙ И МЕТОДЫ ОБРАБОТКИ ЭКСПЕРИМЕНТАЛЬНЫХ ДАННЫХ

Основная цель экспериментов — обнаружить гипервизор в системе на основе анализа длительности выполнения набора процессорных инструкций (трасс).

Экспериментальные исследования подтвердили, что измеренная с помощью процессорного счетчика тактов (TSC) длительность выполнения процессорных инструкций является случайной величиной. На эту величину влияют различные факторы, включая модель процессора, версию ОС и текущее состояние вычислительной среды. В условиях присутствия гипервизора длительность выполнения трассы $t_{ПР}$ и ее вариабельность значительно выше по сравнению с аналогичными показателями при его отсутствии $t_{ОТ}$. Этот разрыв в статистических характеристиках особенно заметен при наличии нескольких вложенных экземпляров гипервизоров в системе.

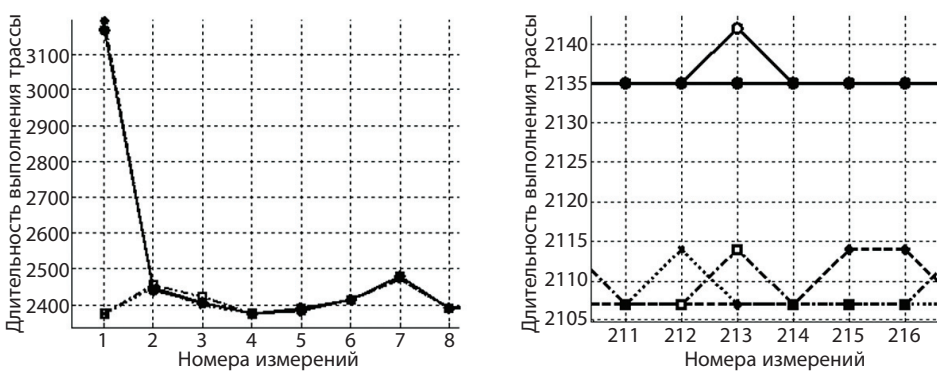
Если гипервизор не искажает показания счетчика тактов, то анализ средних арифметических значений массивов длительности выполнения трасс $\overline{t}_{ОТ}$ и $\overline{t}_{ПР}$ позволяет легко обнаружить присутствие гипервизора [501].

Однако современные технологии аппаратной виртуализации, такие как Intel VT-x и AMD-V, предоставляет встроенные средства для изменения показаний TSC и позволяют значительно уменьшить эту разницу, что усложняет выявление гипервизора, основанное только на средних значениях.

Для обнаружения гипервизора в таких условиях необходимо использовать другие статистические характеристики

длительности выполнения трасс. Эти характеристики выявляются посредством массовых наблюдений, требующих проведения экспериментов, в которых выполнение трасс и измерение их длительностей $t_{ОТ}$ и $t_{ПР}$ рассматриваются как случайные процессы. При этом длительности выполнения трасс записываются в виде функциональных последовательностей $T_{ОТ}(i)$ и $T_{ПР}(i)$, где i — номер измерения в последовательности измерений, выполняемых во вложенном цикле.

Типичные реализации этих процессов представлены на рис. 117. Одной из характерных особенностей является наличие выбросов в начале каждого цикла измерений (см. рис. 117, *а*). Эти выбросы в случае наличия гипервизора могут иметь значительно бóльшую величину по сравнению с выбросами при его отсутствии. Это явление наблюдается даже в случае компрометации показаний счетчика тактов. После возникновения выбросов процесс стабилизируется, однако переходный период может занимать до 10–12 измерений. В дальнейшем наблюдается распределение длительностей выполнения трассы $t_{ОТ}$ по нескольким фиксированным уровням (слоям) (см. рис. 117, *б*), что придает процессу *слоистую структуру*.



а *б*

Рис. 117. Графики семейства реализаций случайного процесса

В случае присутствия гипервизора с компрометацией процессорного счетчика тактов графики длительностей $t_{ПР}$ демонстрируют аналогичную уровневую структуру, однако количество уровней может быть больше и эти уровни могут находиться

близко к уровням $t_{ОТ}$, но не совпадать с ними. Эти особенности наблюдаются на всех исследованных СВТ и подтверждаются в зарубежных исследованиях [434]. Важно отметить, что выявленные особенности проявляются не только при выполнении инструкций CPUID, но и других БП-инструкций.

Эксперименты подтверждают важный вывод, сделанный на основе анализа предложенной модели работы процессора при измерении длительности выполнения трассы: *процесс измерения длительности выполнения трассы является случайным, но обладает определенной упорядоченностью как в условиях отсутствия, так и в присутствии гипервизоров. Эта упорядоченность проявляется в том, что измеренные значения длительности выполнения трассы распределяются по фиксированным уровням случайным образом в каждом цикле измерений.*

Сложности в выявлении гипервизора на основе средних значений можно сформулировать следующим образом: эксперименты подтверждают вывод о том, что процесс измерения длительности выполнения трассы является случайным, но с определенной упорядоченностью, выраженной в распределении значений по фиксированным уровням. Однако распределение измерений по этим уровням непредсказуемо, что затрудняет обнаружение гипервизора на основе среднего значения. Более того, из-за случайных изменений длительностей выполнения трасс средние значения $\overline{t}_{ОТ}$ и $\overline{t}_{ПР}$ могут изменяться от эксперимента к эксперименту («дрейфовать»), что делает *критерий среднего значения непригодным для выявления гипервизора*.

Дополнительно в наблюдаемых данных $t_{ОТ}$ и $t_{ПР}$ могут присутствовать выбросы и разрывы, изменяющие структуру процесса и усложняющие анализ. Эти неоднородности могут быть устранены с помощью методов сглаживания или селекции данных (рис. 118), а также путем применения фильтров.

Исходя из изложенного выше, можно заключить, что процесс выполнения трассы в цикле представляет собой *нестационарный случайный процесс, который часто обладает нестабильной структурой*. При этом его характеристики случайным образом варьируются от одной реализации к другой.

Перспективные методы обнаружения гипервизора должны учитывать уровневую структуру распределения длительностей выполнения трасс.

Один из простейших методов обнаружения гипервизоров основан на том, что уровни длительностей выполнения трассы $t_{ОТ}$ в случае отсутствия гипервизора, после применения частотной фильтрации с определенным уровнем ($f = 0{,}1$), образуют компактные множества, которые могут служить нормативными эталонами для его обнаружения. Эти эталонные множества $[U_{ОТ}]$ сравниваются с тестовыми множествами $t_{ПР}$, что позволяет делать выводы о наличии гипервизора.

В качестве альтернативного подхода был рассмотрен метод вариационных рядов, однако его применение оказалось затруднительным из-за нестабильности и значительного перекрытия интервалов значений $t_{ОТ}$ и $t_{ПР}$. Для преодоления этой проблемы автором предложена методика поэтапного обследования систем. В этой методике пороговые значения статистических показателей $t_{ОТ}$ и $t_{ПР}$ определяются сначала для системы без гипервизора, а затем для системы с установленным гипервизором, что позволяет выявлять вложенные экземпляры гипервизоров.

Также возможно использование выборочных методов математической статистики при условии, что сами опыты и обработка их результатов проводятся в соответствии с ГОСТ Р ИСО 5725–2002 с учетом недостаточной прецизионности получаемых данных [517–521].

Применение классических методов математической статистики было затруднено из-за значительных отклонений («дрейфа») экспериментальных данных. Одной из проблем является недостаточная сходимость (повторяемость) результатов измерений, выполненных сериями с небольшими интервалами между опытами, а также низкая воспроизводимость этих данных при проведении экспериментов в разные дни.

Применение упрощенного метода М.И. Корнфельда [525, 526] позволило решить проблему интервальной оценки статистик. В качестве доверительных границ варьирования экспериментальной статистики S использовались минимальные и максимальные значения S_{min} и S_{max}, полученные в ряду экспериментов,

при доверительной вероятности, которая вычисляется по следующей формуле:

$$\alpha = 1 - \left(\frac{1}{2}\right)^{l-1}, \qquad (22)$$

где l — длина ряда значений статистики S.

Поскольку во всех опытах $l = 10$, имеем $\alpha = 0{,}998 \approx 1$. Такое значение доверительной вероятности избыточно высоко, поэтому было предложено укорачивать обрабатываемый ряд путем отбрасывания его крайних значений и пересчитывать доверительную вероятность [9].

Выбросы и разрывы в наблюдаемых рядах данных $t_{ОТ}$ и $t_{ПР}$ (см. рис. 118) — это еще одна серьезная проблема, существенно влияющая на выборочные характеристики, делая различия между статистиками в условиях отсутствия и присутствия гипервизора неустойчивыми [508]. На рис. 118 можно видеть выбросы в исходных данных, которые указывают на аномальные значения длительности выполнения трассы, значительно отличающиеся от остальных измерений.

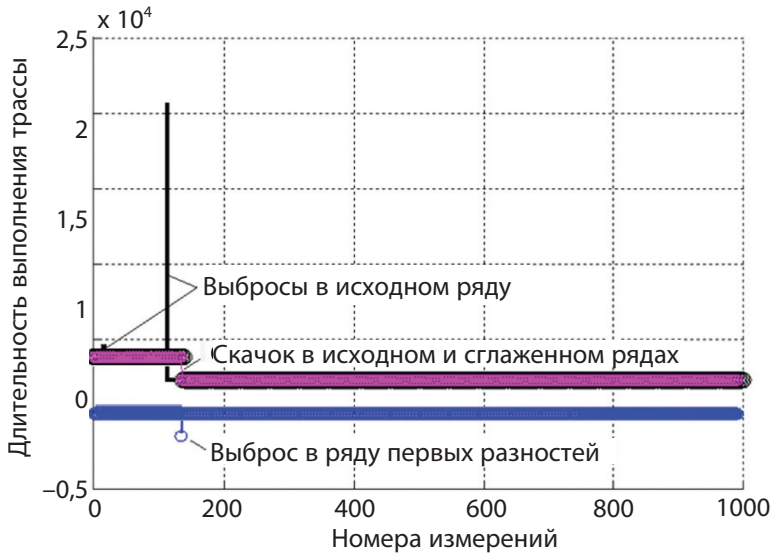


Рис. 118. Графики ряда длительности выполнения трассы $t_{ОТ}$: исходного, сглаженного с помощью медианного фильтра и первых разностей сглаженного ряда в опыте НИИ1 в случае отсутствия гипервизора

Для устранения этих неоднородностей была использована фильтрации наблюдений [508, 527]. Исследования показали, что это позволяет минимизировать влияние выбросов и разрывов при выполнении следующих условий:

- массивы данных длительности выполнения трассы $T_{ОТ}$ и $T_{ПР}$ должны быть представлены в виде матриц, столбцы которых рассматриваются как статистические выборки. Эти матрицы должны иметь достаточный объем (например, 1000×10), а количество серий повторных опытов должно быть не менее пяти;
- серии повторных экспериментов должны проводиться в течение длительного времени (порядка 10 дней);
- для устранения выбросов и разрывов данные должны быть предварительно обработаны с использованием фильтрации или сглаживания с весовым усреднением статистик;
- разработка пороговых значений для идентификации признаков присутствия гипервизора должна учитывать статистики $t_{ОТ}$ и $t_{ПР}$, интервалы варьирования которых не перекрываются, или частота перекрытий должна быть достаточно мала.

Выборочные методы обладают важным преимуществом — возможностью адаптации к особенностям получаемых данных. Так, использование низкочастотных фильтров позволяет удалять выбросы из массивов данных, а разрывы устраняются путем определения статистик на участках между разрывами и последующего усреднения данных по всему ряду наблюдений.

При этом выбор типа статистики требует особого внимания. Например, пересечение дисперсий длительности выполнения трассы $D_{ОТ}$ и $D_{ПР}$ в условиях отсутствия и присутствия гипервизора делает использование дисперсии непригодным для большинства экспериментальных данных.

Для улучшения точности анализа статистические показатели вычислялись только после предварительной обработки первичных данных. В результате были получены вторичные статистики, такие как производные от первичных статистик. Это значительно сузило области пересечения данных.

Исследования подтвердили, что применение синтеза комбинированных показателей для анализа случайных величин длительности выполнения трассы является перспективным. Однако

результаты экспериментов показали, что «дрейф данных» может существенно осложнить их воспроизводимость.

Определение пороговых значений для обнаружения гипервизора должно предшествовать началу экспериментов. Тестирование на наличие гипервизора следует начинать с сопоставления тестового среднего значения длительности выполнения трассы $\overline{t}_{\text{тест}}$ и порогового среднего значения для случая отсутствия гипервизора $[\overline{t}_{\text{ОТ}}]$. Существенное отклонение между ними, а также отрицательные значения в полученных измерениях $t_{\text{тест}}$ будут свидетельствовать о присутствии гипервизора.

Анализ показал, что для выявления пороговых значений статистик длительности выполнения трассы в случаях вложенных гипервизоров необходимо проводить поэтапное обследование системы. Сначала следует определить пороговые значения статистик для случая отсутствия гипервизоров S_1, а затем, установив в систему легитимный образец гипервизора, например Acronis True Image [528] или другой авторский образец, следует повторить измерения и определить пороговые значения статистик для нового состояния системы S_2. Этот процесс продолжается при установке других образцов гипервизоров.

Для наглядной демонстрации изменения пороговых значений при различном количестве гипервизоров представлена диаграмма на рис. 119. На ней показаны изменения статистик для случаев отсутствия гипервизоров, наличия одного гипервизора (Acronis) и двух вложенных гипервизоров (Acronis + авторский). Тестовая статистика $S_{\text{тест}}$ — интегральный показатель, который характеризует дополнительную нагрузку на систему, возникающую из-за работы гипервизоров. Он может быть рассчитан как усредненное время выполнения определенных операций или как другой параметр, фиксирующий влияние виртуализации.

Анализ рис. 119 демонстрирует, что прирост среднестатистических значений ($S_{\text{тест}}$) с увеличением числа гипервизоров позволяет уверенно классифицировать состояния системы. Этот процесс продолжается при установке других образцов гипервизоров.

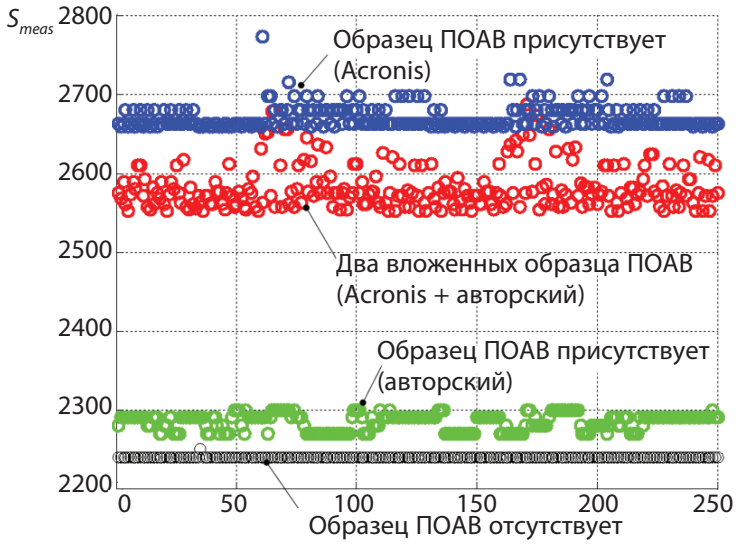


Рис. 119. Значения длительностей выполнения трассы для случаев отсутствия гипервизора, присутствия одного и двух вложенных образцов:

черные точки — отсутствие гипервизоров; зеленые точки — наличие одного гипервизора (авторский); синие точки — наличие одного гипервизора (Acronis); красные точки — два вложенных гипервизора (Acronis + авторский)

На рис. 120 представлена числовая ось с порогами идентификационных статистик $S_1, S_2, \ldots, S_n$, которые разделяют состояния отсутствия и присутствия одного или нескольких образцов гипервизоров.

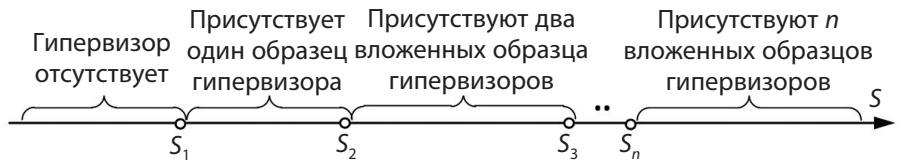


Рис. 120. Пороговые значения показателей для выявления одного, двух образцов гипервизоров и более

Тестовая статистика $S_{\text{тест}}$ интерпретируется следующим образом:

- если $S_{\text{тест}} \le S_1$, то гипервизор в системе отсутствует;
- если $S_{\text{тест}} \le S_2 \le S$, то в системе присутствует один легитимно установленный гипервизор;

- если $S_{\text{тест}} > S_2$, то можно сделать вывод о присутствии двух вложенных образцов гипервизоров, один из которых нелегитимный.

В табл. 32 представлены пороговые условия и принимаемые решения о наличии гипервизора в зависимости от статистических значений.

Таблица 32

Пороговые условия для обнаружения гипервизоров

Пороговые условия	Принимаемое решение о наличии гипервизоров
$S \le S_1$	Гипервизор отсутствует
$S \in (S_1, S_2]$	Присутствует один гипервизор из установленного ПО
$S \in (S_2, S_3]$	Два вложенных гипервизора из установленного ПО
...	...
$S \in (S_{n-1}, S_n]$	$(n-1)$ вложенных гипервизоров из установленного ПО
$S > S_n$	n вложенных гипервизоров, в том числе $(n-1)$ — из установленного ПО и один нелегальный гипервизор

Альтернативным подходом к выявлению вложенных гипервизоров может быть одновременное использование двух методов обнаружения одного гипервизора: на основе длины вариационных рядов и механизмов кэширования [529].

16.3. АНАЛИЗ ЭКСПЕРИМЕНТАЛЬНЫХ ДАННЫХ И ВЫДЕЛЕНИЕ ПРИЗНАКОВ ДЛЯ ОБНАРУЖЕНИЯ СКРЫТЫХ ГИПЕРВИЗОРОВ

Изложим методику предварительной обработки первичных данных длительности выполнения трассы с использованием метода Корнфельда, который основан на сравнении каждого измеренного значения с некоторым пороговым значением, основанном на среднем значении выборки и среднеквадратическом отклонении [525].

16.3.1. Фильтрация и обработка данных

Данные матриц длительности выполнения трассы $T_{ОТ}$ и $T_{ПР}$, полученные в ходе экспериментов, подвергались фильтрации по шести уровням частот ($f \in \{0; 0{,}02; 0{,}05; 0{,}1; 0{,}15; 0{,}2\}$) для удаления выбросов. Отфильтрованные массивы данных $T_{ОТ,f}$ и $T_{ПР,f}$ были проверены на наличие разрывов (скачкообразных выбросов), которые определялись как максимальные значения в ряду первых разностей [508]. В качестве порогового значения для разрывов было принято значение 300 тактов. Это значение может быть скорректировано в зависимости от характеристик данных.

Для дальнейшей обработки были выбраны отфильтрованные массивы $T_{ОТ,f}$ и $T_{ПР,f}$, у которых при фильтрации на уровне $[f] = 0{,}1$ статистики длительности выполнения трассы становились стабильными на всех шести обследованных компьютерах.

Методика обработки данных основывалась на двух характеристиках длительности выполнения трассы: детерминированности уровней распределения $t_{ОТ,f}$ и $t_{ПР,f}$, а также случайности попадания данных на эти уровни. Для выявления признаков наличия гипервизоров использовалось два метода: уровневый метод и метод выборочных статистик.

16.3.2. Уровневый метод выявления признаков гипервизоров

Для обнаружения гипервизора был использован *уровневый метод*. Статистические ряды уровней длительности выполнения трассы $U_{ОТ}$ и $U_{ПР}$ рассчитывались по сериям экспериментов. В ряде экспериментов был получен обобщенный ряд значений длительности для случая отсутствия гипервизора $U_{ОТ} = 2160$, 2168, 2184, 2192, 2200, 2478, 2480, 2880, 2888, 2904, 2920, 2936, а также в серии опытов получен ряд значений для случая присутствия гипервизора $U_{ПР} = 2876$, 2884, 2892, 2900, 2908, 2916, 2924. Оказалось, что эти ряды не пересекаются (нет совпадений).

В сериях последующих экспериментов, проведенных в различные дни и в условиях наличия гипервизоров, были зафиксированы похожие результаты. Однако в отличие от предыдущих

наблюдений в некоторых случаях были зафиксированы отрицательные значения в данных, касающихся длительности прохождения трассы.

Опыты показали, что ряды $U_{\text{ОТ}}$ и $U_{\text{ПР}}$ на различных компьютерах не пересекаются. Лишь в редких случаях в системах с процессором Intel Core 2 Duo E6300 и Windows 7 при наличии гипервизоров наблюдались совпадения значений с рядом $U_{\text{ОТ}}$, однако частота таких совпадений была крайне низкой. Следовательно, при проведении нескольких повторных тестов вероятность их повторения может стать практически незначительной, учитывая теорему умножения вероятностей для повторных событий.

Эксперименты показали, что обобщенные ряды уровней длительности выполнения трассы $U_{\text{ОТ}}$, полученные на основе достаточного объема статистических данных и в разные дни, можно считать нормативными, свидетельствующими об отсутствии гипервизоров, благодаря их детерминированному характеру и редким пересечениям. Уровневый метод обнаружения гипервизоров оказался простым и эффективным, однако в некоторых случаях он может быть недостаточно результативным, требуя применения альтернативных методов, основанных на анализе различий статистических характеристик длительности выполнения трассы $t_{\text{ОТ}}$ и $t_{\text{ПР}}$.

16.3.3. Метод выборочных статистик

Для разработки методов обнаружения гипервизоров были исследованы статистические свойства длительности выполнения трассы в условиях отсутствия ($t_{\text{ОТ}}$) и присутствия гипервизора ($t_{\text{ПР}}$). Анализировались выборочные статистики массивов $T_{\text{ОТ},f}$ и $T_{\text{ПР},f}$, включая средние значения, дисперсии, моменты четвертого порядка и длины вариационных рядов.

В случаях, когда в данных отсутствовали разрывы, статистики вычислялись непосредственно для каждого столбца массива. При наличии разрывов данные делились на участки между разрывами и статистики рассчитывались для каждого участка с последующим взвешенным усреднением.

Использование длин вариационных рядов для детекции гипервизоров. В качестве иллюстрации приведена табл. 33, в которой в столбцах 2–11 представлены значения длины $l_{\text{ПР},f}$ вариационных рядов, полученных по столбцам матрицы $T_{\text{ПР},f}$ в эксперименте на СВТ с процессором Intel Core 2 Duo E8200 под управлением ОС Windows 7 в случае присутствия гипервизора. 12-й столбец таблицы $l_{\text{в,ПР},f}$ по вектору $T_{\text{в,ПР},f}$ характеризует изменчивость данных в исходной матрице на каждом уровне фильтрации, показывая, сколько уникальных значений или какую длину $l_{\text{в,ПР},f}$ имеют ряды данных в этих векторах.

Анализ этой таблицы показал, что начиная с уровня фильтрации $f = 0{,}1$ значения статистик $\overline{l}_{\text{ПР},f}$ и $l_{\text{в,ПР},f}$ стабилизируются. Это свидетельствует о том, что данный уровень фильтрации является оптимальным для дальнейшего анализа.

Таблица 33

Статистические характеристики длины $l_{\text{ПР},f}$ вариационных рядов времени выполнения трасс по столбцам матрицы $T_{\text{ПР},f}$ и вектору $T_{\text{в,ПР},f}$ в эксперименте при наличии гипервизора

Уровень фильтрации f	Длины вариационных рядов $l_{\text{ПР},f}$ для каждого столбца матрицы $T_{\text{ПР},f}$				$\overline{l}_{\text{ПР},f}$ по столбцам матрицы $T_{\text{ПР},f}$	$l_{\text{в,ПР},f}$ по вектору $T_{\text{в,ПР},f}$
	1	2	...	10		
1	2	3	...	11	12	13
0	28	29	...	10	12	53
0,02	4	3	...	3	4	6
0,05	3	3	...	3	3	6
$[f] = 0{,}1$	2	2	...	3	3	3
0,15	1	2	...	2	2	3
0,20	1	2	...	2	2	2

Итак, для дальнейшего анализа были выбраны статистики при уровне фильтрации $f = 0{,}1$ или те, для которых сумма вероятностей ошибок I (α) и II (β) рода была минимальной (смысл этих ошибок будет рассмотрен далее). Их значения для всех 10 опытов на данном СВТ сведены в табл. 34.

Таблица 34

Статистики распределения длины вариационных рядов длительности выполнения трассы по данным опытов при уровне фильтрации $f = 0{,}1$ (фрагмент таблицы)

Код серии опытов	Гипервизор отсутствует		Гипервизор присутствует	
	$\overline{l}_{\text{ОТ},f}$ по столбцам матрицы $T_{\text{ОТ},f}$	$l_{\text{в,ОТ},f}$ по вектору $T_{\text{в,ОТ},f}$	$\overline{l}_{\text{пр},f}$ по столбцам матрицы $T_{\text{ПР},f}$	$l_{\text{в,ПР},f}$ по вектору $T_{\text{в,ПР},f}$
1	2	3	4	5
Иг10	5	23	11	47
	4	18	11	52
	4	15	10	34
	5	21	13	53
	4	15	14	68
...	...	...	...	...
Иг19	4	20	19	102
	6	32	15	77
	6	32	16	79
	6	32	20	88
	10	50	21	105
Интервал варьирования статистик	[4, 14]	[10, 110]	[8, 21]	[29, 105]
Пороговое условие $[P_{\text{ОТ}}]$ и $[P_{\text{ПР}}]$	≤ 7	≤ 32	≤ 8	≤ 33
Вероятность ошибки I рода, α	0,04	0,12	—	—
Вероятность ошибки II рода, β	0	0,16	—	—

Статистики распределения длины вариационных рядов длительности выполнения трассы по данным опытов при уровне фильтрации $f = 0{,}1$ содержит табл. 34. В таблице представлены статистики $\overline{l}_{\text{ОТ},f}$ и $l_{\text{в,ОТ},f}$ в отсутствие гипервизора (столбцы

2–3) и статистики $\overline{l}_{\text{ПР},f}$ и $l_{\text{в,ПР},f}$ в присутствии гипервизора (столбцы 4–5). Нижние строки: интервалы варьирования статистик, пороговые условия, вероятности ошибок I рода (α) и II рода (β).

Ошибка I рода (α) — это ситуация, когда тестовая статистика указывает на наличие гипервизора, хотя на самом деле его нет. Ошибка II рода (β), наоборот, возникает, когда гипервизор присутствует, но тестовая статистика его не обнаруживает.

Анализ таблицы показал, что интервалы варьирования статистик в случаях отсутствия и присутствия гипервизора могут перекрываться. Это означает, что существует вероятность того, что при тестировании полученные значения статистик попадут в область перекрытия, и однозначное обнаружение гипервизора будет затруднено.

Чтобы минимизировать вероятность неопределенности, были рассчитаны вероятности того, что статистики $\overline{l}_{\text{ОТ},f}$ и $\overline{l}_{\text{ПР},f}$, $l_{\text{в,ОТ},f}$ и $l_{\text{в,ПР},f}$ не попадут в зоны перекрытия. Для этого:

- определялись относительные частоты значений статистик, выходящих за пределы установленных пороговых значений;
- рассчитывались вероятности ошибок I рода (α) — вероятность ложного обнаружения гипервизора при его отсутствии;
- вычислялись вероятности ошибок II рода (β) — вероятность необнаружения гипервизора при его наличии.

Пороговые значения статистик $[\overline{l}_{\text{ОТ},f}]$ и $[\overline{l}_{\text{ПР},f}]$ устанавливались интерактивно в целях минимизации суммы вероятностей ошибок (α + β). В качестве тест-статистик для обнаружения гипервизора выбирались те, у которых сумма вероятностей ошибок не превышала 0,2.

Описанная методика была использована при обследовании всех компьютеров, указанных в табл. 31. Полученные идентификационные признаки и соответствующие пороговые условия представлены в табл. 35.

Согласно табл. 35, для большинства обследованных СВТ вероятность ошибок I и II рода составляет менее 5%. Однако если проводить несколько повторных тестов и определять частоту вы-

полнения тест-статистиками пороговых условий, то вероятность этих ошибок значительно уменьшится благодаря применению теоремы умножения вероятностей для повторных событий и станет пренебрежимо малой.

Таблица 35

Пороговые условия тест-статистик длительности выполнения трассы (для СВТ, указанных в табл. 31).

СВТ	Тест-статистика	Уровень фильтрации f	Пороговое значение		Вероятность ошибки	
			гипервизор отсутствует	гипервизор присутствует	I рода, α	II рода, β
1	$\bar{T}_{\text{от},f}$	0	≤ 2911	—	—	—
	$\bar{L}_f$	0	≤ 7	≥ 8	0,04	0
	$\bar{D}_f$	0	≤ 14	≥ 18	0,02	0
	$\bar{M}_f$	0,1	≤ 679	≥ 947	0,02	0
	μ_f	0,1	≤ 104 161	≥ 111 041	0,02	0,10
2	$\bar{T}_{\text{от},f}$	0	≤ 2492	—	—	—
	$\bar{L}_f$	0	≤ 11	≥ 12	0,1	0,06
	$\bar{D}_f$	0,2	≤ 100	≥ 101	0,08	0,1
	$\bar{M}_f$	0,2	≤ 168	≥ 13 030	0,14	0,02
3	$\bar{T}_{\text{от},f}$	0	≤ 2431	—	—	—
	$\bar{L}_f$	0	≤ 6	≥ 8	0	0
	$\bar{D}_f$	0,1	≤ 15	≥ 41	0	0
	μ_f	0,1	≤ 609	≥ 3410	0	0
4	$\bar{T}_{\text{от},f}$	0	≤ 5018	—	—	—
	$\bar{L}_f$	0	≤ 22	≥ 26	0,02	0,02
	$\bar{D}_f$	0,1	≤ 177	≥ 181	0,1	0,1
5	$\bar{T}_{\text{от},f}$	0	≤ 2852	—	—	—
	$\bar{L}_f$	0	≤ 67	≥ 71	0,04	0
	$\bar{D}_f$	0	≤ 16 416	≥ 48 920	0	0

Окончание табл. 35

СВТ	Тест-статистика	Уровень фильтрации f	Пороговое значение		Вероятность ошибки	
			гипервизор отсутствует	гипервизор присутствует	I рода, α	II рода, β
6	$\overline{T}_{\text{ОТ},f}$	0	≤ 2126	—	—	—
	$\overline{L}_f$	0	≤ 34	≥ 241	0	0
	$\overline{l}_f$	0	≤ 134	≥ 593	0	0
	$\overline{D}_f$	0	≤ 216	≥ 5478	0	0
	d_f	0	≤ 345	≥ 5422	0	0
	$\overline{M}_f$	0,02	≤ 54	≥ 956	0	0

Примечание: $\overline{T}_{\text{ОТ},f}$ — средняя длительность выполнения трассы для всей матрицы $T_{\text{ОТ},f}$; $\overline{L}_f$ — средняя длина вариационных рядов; $\overline{D}_f$ — средняя дисперсия и $\overline{M}_f$ — средний момент четвертого порядка столбцов матриц $T_{\text{ОТ},f}$ и $T_{\text{ПР},f}$; d_f и μ_f — дисперсия и момент четвертого порядка векторов $T_{\text{в,ОТ},f}$ и $T_{\text{в,ПР},f}$.

Использование координат центра тяжести полигона частот для обнаружения гипервизора. Исследования показали, что для выявления гипервизора могут быть использованы *координаты центра тяжести полигона частот* (ЦТП). Эти координаты рассчитываются на основе вариационных рядов, построенных по столбцам матриц длительности выполнения трасс $T_{\text{ОТ}}$ и $T_{\text{ПР}}$, полученных в экспериментах.

Для массивов размером 1000×10 формируется 10 вариационных рядов, длины которых составляют векторы $L_{\text{ОТ}}$ и $L_{\text{ПР}}$. На основе этих векторов строятся полигоны частот, для которых определяются полярные координаты центра тяжести (ρ — длина радиуса и φ — угол) [530]. Эти параметры позволяют оценить распределение длительностей выполнения трасс в зависимости от наличия или отсутствия гипервизора. Примеры таких полигонов и их центры тяжести для процессора Intel Core 2 Duo E8600 (Windows Live CD XP) представлены на рис. 121.

Координаты $\rho_{\text{ОТ}}$ и $\varphi_{\text{ОТ}}$, $\rho_{\text{ПР}}$ и $\varphi_{\text{ПР}}$ изменяются в зависимости от наличия гипервизоров, что позволяет использовать их для его обнаружения. Метод ЦТП показал высокую достоверность

на большинстве тестируемых компьютеров. Однако для системы с процессором Intel Core 2 Duo E6300 и ОС Windows 7 метод оказался менее достоверным из-за пересечения координат ЦТП для случаев с наличием и отсутствием гипервизора.

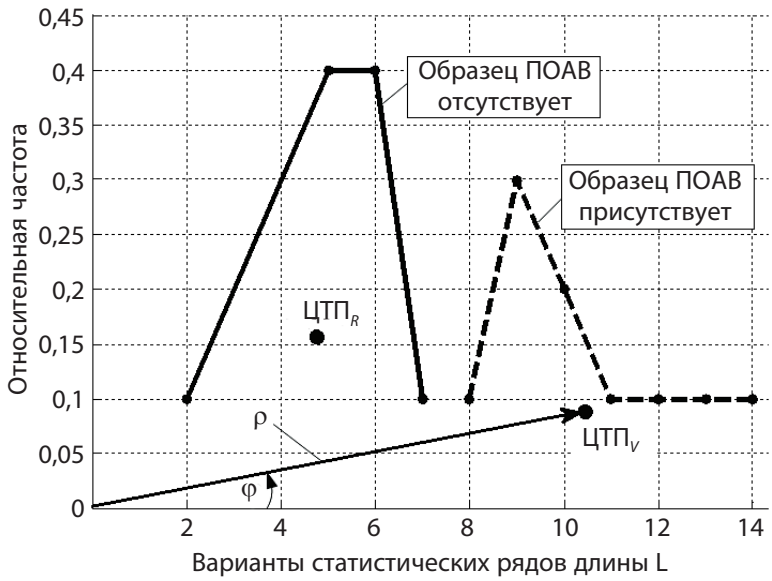


Рис. 121. Примеры полигонов частот длин $L_{ОТ}$ и $L_{ПР}$ статистических рядов матриц $T_{ОТ}$ и $T_{ПР}$ и их центры тяжести

Результаты экспериментов показали, что положение центров тяжести существенно различается в случаях присутствия и отсутствия гипервизора, что делает данный метод эффективным для его обнаружения. Однако в отдельных случаях возможно получение ложных отрицательных результатов из-за значительного перекрытия координат центров тяжести в ситуациях с наличием и отсутствием гипервизора. Для повышения достоверности предлагается организовывать многократные испытания и на основании созданных тестовых матриц оценивать, насколько часто срабатывают эти альтернативные пороговые критерии для ЦТП координат. Эти полученные данные следует затем сравнивать с заранее установленными порогами. Применение такой методики позволит обнаружить гипервизор даже в условиях противодействия.

16.3.4. Пороговые условия обнаружения гипервизоров: результаты экспериментальной проверки

Экспериментальная проверка предложенных методов обнаружения скрытых гипервизоров проводилась по методике, описанной в параграфе 16.2. В табл. 36 приведены пороговые значения идентификационного показателя — средней длины вариационных рядов длительности выполнения трассы $\overline{L}_f$ для одной из тестируемых СВТ.

Таблица 36

Пороговые значения средней длины вариационных рядов выполнения трассы $\overline{L}_f$ для системы с процессором Intel Core i7 950 и ОС Windows XP при фильтрации $f = 0$

№ п/п	Пороговое значение	Решение о наличии гипервизора
1	$\overline{L}_f \le 31$	Гипервизор отсутствует
2	$32 \le \overline{L}_f \le 67$	Присутствует один гипервизор Acronis
3	$\overline{L}_f \ge 86$	Присутствуют два вложенных гипервизора: Acronis и авторский

Экспериментальные данные показали следующее:

- если при тестировании статистика меньше или равна 31 ($\overline{L}_f \le 31$), то вероятность ошибочного определения наличия одного гипервизора составляет $\alpha = 0{,}14$. Вероятность ошибки при определении двух вложенных гипервизоров равна $\beta = 0$;
- если диапазон изменения $32 \le \overline{L}_f \le 67$, то вероятность ложного заключения об отсутствии гипервизора составляет $\alpha \le 0{,}06$, при этом вероятность ошибочного обнаружения двух вложенных гипервизоров остается равной нулю ($\beta = 0$);
- если $\overline{L}_f \ge 86$, то вероятность ошибочного заключения как о наличии, так и об отсутствии одного гипервизора равна нулю ($\alpha = \beta = 0$).

На основании полученных результатов можно заключить, что для повышения достоверности выявления гипервизоров рекомендуется комбинированное использование всех предложенных методов с проведением нескольких серий повторных опытов; это позволит минимизировать вероятность ошибок и повысить точность обнаружения.

16.4. МЕТОДИКА ОБНАРУЖЕНИЯ СКРЫТЫХ ГИПЕРВИЗОРОВ

Процесс обнаружения гипервизоров в исследуемой вычислительной системе включает два этапа: предварительный и оперативный.

1. Предварительный этап. На этом этапе проводится подготовка системы к тестированию. Для начала по матрицам длительности выполнения трасс $T_{\text{ОТ}}$ и $T_{\text{ПР}}$ определяются нормативные показатели для случая отсутствия гипервизора. Это включает:

- расчет выборочных средних значений $[\overline{T}_{\text{ОТ},f}]$ для уровня фильтрации $f = 0$;
- формирование рядов уровней длительности выполнения трасс $[U_{\text{ОТ}}]$;
- определение пороговых значений статистик для различных значений фильтрации.

Цель предварительного этапа — создание базы пороговых значений $[S]$, которая будет использоваться для сопоставления результатов, полученных на оперативном этапе. На основе этих данных можно точно установить пороговые значения для выявления аномалий.

2. Оперативный этап. На оперативном этапе осуществляется непосредственное тестирование вычислительной системы. Этот процесс включает следующие шаги:

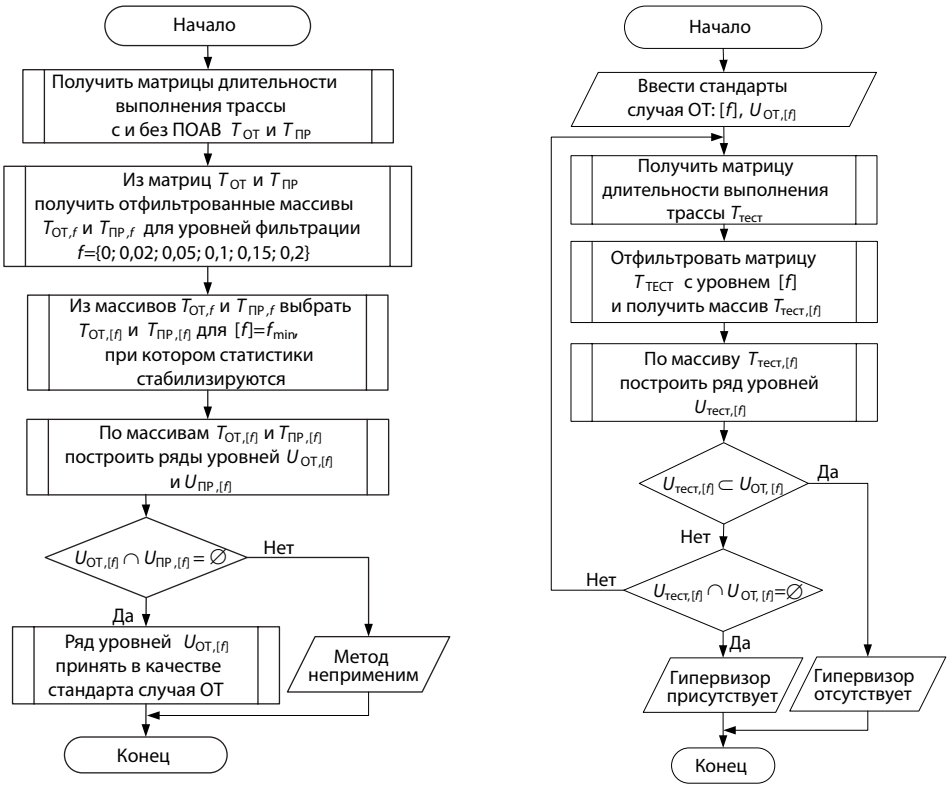
- формирование тестовых матриц $T_{\text{тест}}$, по которым вычисляются тестовые статистики $S_{\text{тест}}$;
- сравнение тестовых статистик с нормативными значениями $[S]$, рассчитанными на предварительном этапе.

Цель оперативного этапа — выявить наличие гипервизора в реальных условиях эксплуатации, основываясь на анализе данных, полученных во время тестирования, и их сопоставлении с нормативными показателями.

16.4.1. Алгоритмы проверки на наличие гипервизора

Алгоритмы для определения пороговых значений и обнаружения гипервизора основаны на двух методах: уровневом и комбинированном (см. параграф 9.3).

Уровневый метод (рис. 122) используется для выявления нормативных уровней длительности выполнения трассы.



а *б*

Рис. 122. Алгоритмы с использованием уровневого метода:
а — для выявления нормативных уровней длительности выполнения трассы;
б — для обнаружения гипервизора

Если отрицательные значения в тестовых матрицах $T_{тест}$ отсутствуют, то происходит сравнение средней длительности выполнения трассы $\overline{t}_{тест}$ с нормативной величиной $[\overline{t}_{ОТ}]$. Если $\overline{t}_{тест} \gg [\overline{t}_{ОТ}]$, то принимается решение о наличии гипервизора. Если выборочные средние значения $\overline{t}_{тест} \approx [\overline{t}_{ОТ}]$, то переходят к комбинированному методу для дальнейшего анализа.

Комбинированный метод (рис. 123) предполагает использование как уровневого анализа, так и вычисления статистических характеристик для определения аномалий в данных:

- если $S_{\text{тест}} \le S_{\text{ОТ}}$, то гипервизор отсутствует;
- если $S_{\text{тест}} \ge S_{\text{ПР}}$, то делается вывод о наличии гипервизора.

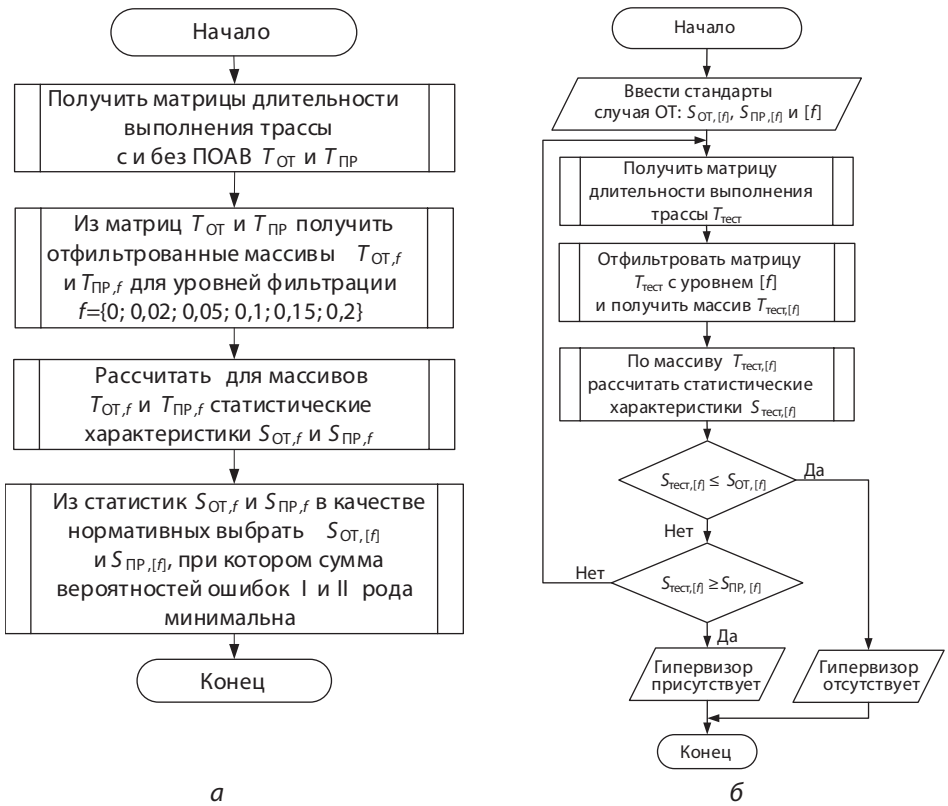


Рис. 123. Алгоритмы с использованием комбинированного метода:
а — для выявления нормативных статистик длительности выполнения трассы;
б — для обнаружения гипервизора

16.4.2. Порядок тестирования на оперативном этапе

Порядок тестирования на оперативном этапе включает несколько шагов:

- проверка тестовых матриц $T_{\text{тест}}$ на наличие отрицательных значений. Их обнаружение свидетельствует о наличии гипервизора;
- если отрицательных значений не обнаружено, то проводится сравнение средней длительности выполнения трассы $\overline{t}_{\text{тест}}$ с нормативной величиной $[\overline{t}_{\text{ОТ}}]$.

В случае если тестовые данные находятся в пределах нормы, выполняется комбинированный анализ статистических характеристик (см. рис. 123).

16.4.3. Пошаговая методика обнаружения нелегитимного гипервизора

Пошаговая методика обнаружения нелегитимного гипервизора приведена в табл. 37.

Таблица 37

Методика обнаружения нелегитимного гипервизора

Название этапа	Содержание шагов каждого этапа
Предварительный	1. Запись доверенной микропрограммы в BIOS. 2. Установка ОС. 3. Определение пороговых значений для выявления гипервизора
Оперативный	1. Начало проверки на отсутствие гипервизора. 2. Установка дополнительного ПО. 3. Отслеживание сообщений об обнаружении гипервизора. 4. Адаптация средства обнаружения к легитимному образцу гипервизора (повтор шага 3)

Предварительный этап включает сбор данных и создание пороговых значений для тестирования, а оперативный этап — это непосредственное тестирование и обнаружение гипервизора на основе сравнения рассчитанных статистических характеристик с пороговыми величинами. Ниже рассмотрим этапы методики более подробно.

При определении пороговых значений для обнаружения гипервизора важно, чтобы система не содержала вредоносного гипервизора ни в BIOS, ни в ОС. На первом этапе предварительного тестирования необходимо произвести запись проверенной версии BIOS с использованием аппаратного программатора, например ТРИТОН+ V5.8TU. Образ BIOS можно скачать с официального сайта производителя материнской платы и проверить его на отсутствие гипервизора с помощью реверс-инжиниринга. Использование аппаратного программатора предотвращает воз-

можные угрозы, которые могут возникнуть при обновлении BIOS через программные средства.

На втором этапе происходит установка ОС, в которой отсутствует нелегитимный гипервизор. Для этого рекомендуется использовать официальные дистрибутивы ОС либо применять Live CD, например, образ Windows Live CD XP (DDD).

На третьем этапе происходит получение пороговых значений по описанному алгоритму. Для этого в систему устанавливается легитимный образец гипервизора, который содержит минимальный набор инструкций и способен искажать показания счетчика тактов. Это один из наиболее сложных случаев для обнаружения.

На четвертом этапе проводятся анализ показателей, полученных из тестовых выборок, и их сравнение с ранее установленными пороговыми значениями. Для этого используются специализированные алгоритмы обнаружения гипервизора, а проверка на его наличие осуществляется непрерывно в ходе эксплуатации системы.

На пятом этапе устанавливается необходимое пользователю ПО. Если после установки обнаружено наличие гипервизора, то требуется проверить его легитимность. Это выходит за рамки данной работы, и в таких случаях рекомендуется обращаться в техническую поддержку. Если гипервизор легитимен, то систему обнаружения необходимо адаптировать под новые условия, повторив третий этап. Таким образом, устанавливаются пороговые значения для каждого вложенного гипервизора, что обеспечивает корректную работу системы обнаружения.

16.5. РЕЗУЛЬТАТЫ АПРОБАЦИИ ПРЕДЛАГАЕМОЙ МЕТОДИКИ ОБНАРУЖЕНИЯ ГИПЕРВИЗОРОВ В УСЛОВИЯХ ПРОТИВОДЕЙСТВИЯ СО СТОРОНЫ НАРУШИТЕЛЯ

Проведенные исследования подтвердили сложность задачи обнаружения гипервизоров при компрометации счетчиков тактов, направленной на сокрытие его присутствия. Нестабильность результатов измерений и явление «дрейф данных» подчеркивают необходимость тщательного анализа и использования дополни-

тельных статистических показателей при выявлении гипервизора. Разработанный минимальный образец гипервизора и методология проведения экспериментов обеспечили получение репрезентативных данных, необходимых для дальнейшего исследования и разработки эффективных методов обнаружения ПО аппаратной виртуализации.

Кратко представим результаты апробации предложенного метода обнаружения гипервизоров на основе анализа временны́х характеристик выполнения инструкций программы (трассы) в условиях противодействия со стороны нарушителя. Экспериментальные исследования проводились с использованием нескольких видов средств вычислительной техники, различающихся аппаратными и программными конфигурациями.

Экспериментально подтвержден вывод: длительность выполнения трассы — это случайная величина, зависящая от присутствия или отсутствия гипервизора.

Критерии присутствия гипервизора, основанные на различных статистических показателях выполнения трасс, были подробно рассмотрены в подпараграфе 15.3.6. Здесь они кратко представлены в табл. 38.

Таблица 38

Критерии обнаружения гипервизора

Критерий	Описание
Критерий присутствия гипервизора на основе дисперсии	Гипервизор присутствует, если $\vec{x}_n \in W = \{\vec{x}_n : \hat{\sigma}\ (\vec{x}_n) \geq d\}$, где $\hat{\sigma}^2$ — выборочная дисперсия; $d \in \mathbb{Z}$ определяется экспериментально
Критерий присутствия гипервизора на основе момента четвертого порядка	Гипервизор присутствует, если $\vec{x}_n \in W = \{\vec{x}_n : \hat{\upsilon}_2(\vec{x}_n) \geq \mu\}$. где $\hat{\upsilon}_2$ — выборочный центральный момент четвертого порядка; $\mu \in \mathbb{Z}$ определяется экспериментально

Окончание табл. 38

Критерий	Описание
Критерий присутствия гипервизора на основе длины вариационного ряда	Гипервизор присутствует, если $\vec{x}_n \in W = \{\vec{x}_n : \hat{l}(\vec{x}_n) \geq e\}$, где $\hat{l}$ — длина вариационного ряда, построенного по выборке; $e \in \mathbb{Z}$ определяется экспериментально

Примечание: $\vec{x}_n$ — выборка объема n из генеральной совокупности X; W — критическое множество, определяющее условия присутствия гипервизора; $\sigma^2(\vec{x}_n)$ — выборочная дисперсия значений выборки $\vec{x}_n$; d — экспериментально определенное пороговое значение выборочной дисперсии; $\upsilon_2(\vec{x}_n)$ — выборочный центральный момент четвертого порядка для выборки $\vec{x}_n$; μ — пороговое значение для выборочного момента четвертого порядка, определенное экспериментально; $l(\vec{x}_n)$ — длина вариационного ряда, построенного по выборке $\vec{x}_n$; e — пороговое значение для длины вариационного ряда, определенное экспериментально.

Экспериментальные исследования проводились по схеме однофакторных опытов, где варьируемым фактором выступала операционная среда вычислительной машины, а именно наличие или отсутствие гипервизора. В каждом опыте запускались программные трассы, которые выполнялись в условиях стандартной операционной системы без гипервизора и в условиях, когда гипервизор был активен.

Результатами опытов являлись матрицы 1000×10, содержащие данные измерений длительности выполнения трассы для случаев отсутствия и присутствия гипервизора ($T_{\text{ОТ}}$ и $T_{\text{ПР}}$). Пример фрагмента такой матрицы приведен в табл. 33.

Для обеспечения требований метрологического международного стандарта ISO 5725 [517–521] к сходимости и воспроизводимости данных измерений опыты выполнялись сериями по пять повторных опытов в течение 10 дней для обеспечения стабильности наблюдаемых данных. Для каждой конфигурации СВТ (см. табл. 31) были проведены тесты в целях определения пороговых значений статистических характеристик выполнения трасс и оценки вероятности ошибок I (α) и II (β) рода. На основе

полученных экспериментальных матриц были построены графики зависимости длительности выполнения трассы от номера повторных измерений.

Результаты экспериментов показали, что предложенные критерии пригодны для обнаружения гипервизора. На рис. 124 представлен пример графиков распределения длительности выполнения трасс для случаев наличия и отсутствия гипервизора с искажением показаний счетчика тактов. Видно, что в случае присутствия гипервизора регистрируемая длительность выполнения трассы оказывается достаточно близка к значениям для случая отсутствия. Кроме того, точки распределяются по фиксированным уровням (слоям), число которых в случае присутствия гипервизора значительно больше.

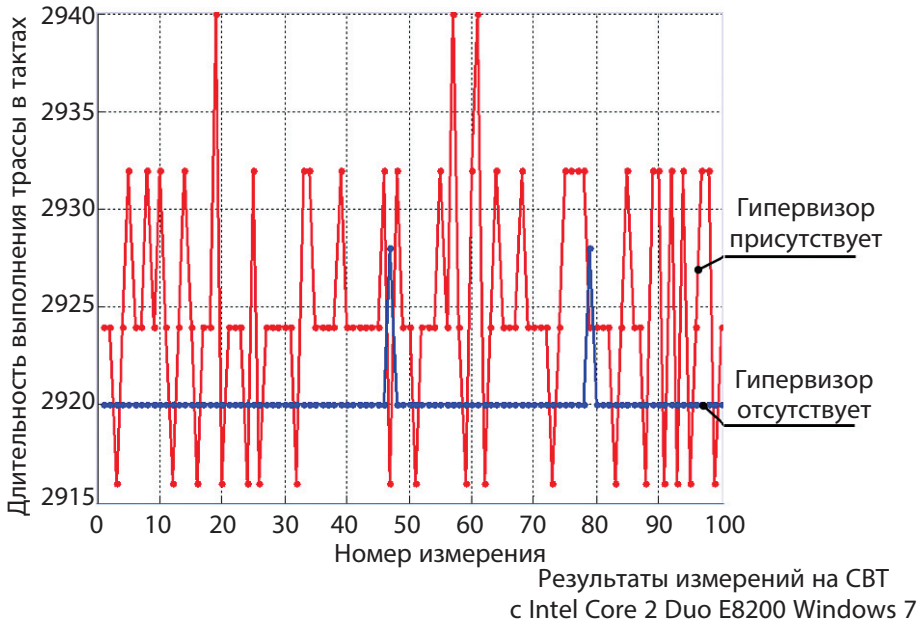


Рис. 124. Графики распределения длительности выполнения трасс для случаев наличия и отсутствия гипервизора

Экспериментальная проверка показала, что предложенные критерии присутствия гипервизора обеспечивают высокую достоверность обнаружения гипервизоров даже при искажении показаний счетчика тактов, что свидетельствует о правомерности

использования метода в условиях противодействия со стороны нарушителя.

В табл. 35 приведены экспериментально определенные пороговые условия тест-статистик длительности выполнения трассы (для систем, указанных в табл. 31). Таблица демонстрирует количественные показатели работы предложенного метода и его работоспособность в зависимости от характеристик используемых вычислительных систем и уровня фильтрации данных.

Для каждого компьютера и для каждой тест-статистики в табл. 35 представлены пороговые значения, при превышении которых можно судить о присутствии гипервизора. Уровень фильтрации влияет на пороговые значения: более высокий уровень фильтрации (например, $f = 0{,}2$) приводит к изменению пороговых значений и вероятности ошибок. Ошибки I рода (α) и II рода (β) в большинстве случаев близки к нулю, что свидетельствует о высокой точности предложенного метода. Наибольшие значения ошибок наблюдаются при использовании специализированного гипервизора (AMD Phenom X4 945 с ОС Windows Live CD XP (DDD)). Был использован специализированный скрытый образец гипервизора, получающий управление из BIOS.

Проведенные исследования позволили предложить комбинированный подход к обнаружению гипервизора, который включает два этапа: на предварительном происходит получение пороговых значений, а на оперативном — осуществляется обнаружение гипервизора с использованием полученных пороговых значений. Подробно эти этапы описаны в параграфе 16.4 и табл. 37.

Выявленные зависимости показателей распределения длительности выполнения трассы от отсутствия и присутствия гипервизора позволили разработать методы его обнаружения, помогающие принимать решение о наличии гипервизора в условиях противодействия.

Выводы

На основе детального анализа режимов работы процессора были разработаны модели, описывающие зависимость длительности выполнения трассы от параметров функционирования

процессора в условиях как присутствия, так и отсутствия гипервизора. Исследование этих моделей показало, что для обнаружения гипервизора основными статистическими характеристиками являются выборочное среднее длительности выполнения трассы, дисперсия, моменты четвертого порядка и длина вариационного ряда.

Был предложен критерий обнаружения гипервизора. Теоретический анализ, подтвержденный экспериментальными данными, показал, что статистические характеристики длительности выполнения трассы существенно зависят от наличия гипервизора. В частности, в случае присутствия гипервизора вариабельность длительности выполнения трассы значительно выше, чем в случае его отсутствия.

Для повышения различимости статистических характеристик длительности выполнения трассы необходима предварительная фильтрация экспериментальных данных от низкочастотных выбросов. Анализ показал, что даже при активных попытках нарушителя скрыть гипервизор из-за случайного характера длительности трассы работоспособность предложенных методов остается высокой, что делает их устойчивыми к противодействию.

Разработана методика обнаружения гипервизоров, использующих технологию аппаратной виртуализации, состоящая из двух этапов: предварительного и оперативного. На предварительном этапе производится вычисление пороговых значений статистических характеристик длительности выполнения трасс. Эти пороговые значения служат базой для последующего анализа и позволяют установить нормативы, с которыми будут сравниваться текущие показатели системы. На оперативном этапе осуществляются непрерывный мониторинг и обнаружение гипервизора в системе с использованием полученных на предварительном этапе пороговых значений.

Экспериментальные исследования подтвердили теоретические выводы, показав, что более 90% измерений значений длительности выполнения трассы распределяются по стабильным уровням, зависящим от присутствия или отсутствия гипервизора. Это позволило разработать методы обнаружения гипервизора.

зоров, стойкие к противодействию и применимые для различных условий, включая одиночные и вложенные гипервизоры.

Разработанная методика может быть использована специалистами по информационной безопасности для контроля отсутствия гипервизоров. Предложенные рекомендации и методы тестирования могут быть внедрены в практику для создания средств обнаружения гипервизоров.

Заключение

Wer mit Ungeheuern kämpft, mag zusehn,
dass er nicht dabei zum Ungeheuer wird.
Friedrich Nietzsche,
Beyond Good and Evil, 1886

Сражаясь с чудовищами, помни —
как легко самому стать чудовищем.
Фридрих Ницше,
«По ту сторону добра и зла», 1886

В данной монографии представлен системный подход к защите ядра ОС Windows от современных и эволюционирующих угроз, связанных с атаками со стороны драйверов и гипервизоров.

Анализ и эксперименты показали, что штатные средства защиты ядра, такие как PatchGuard, HVCI и KDP, не в полной мере соответствуют вызовам времени.

Нарастающая сложность и многообразие техник нарушителей — от драйверов-руткитов до гипервизор-руткитов и атак с использованием уязвимых драйверов, включая целенаправленные попытки обхода или компрометации средств защиты, требуют перехода к новым проактивным решениям, основанным на гипервизорных технологиях, способных работать в условиях активного противодействия со стороны нарушителя.

Практическая значимость работы подтверждается разработкой и успешной экспериментальной апробацией авторских гипервизорных решений MemoryMonRWX, AllMemPro и MemoryRanger. Они демонстрируют новые возможности по защите содержимого памяти ядра: кода драйверов и системных структур от несанкционированного доступа со стороны драйверов. Использование этих гипервизорных решений способствует существенному сокращению поверхности атаки и усложняет эксплуатацию уязвимостей ядра.

Важным вкладом является предложенная методика обнаружения скрытых гипервизоров в условиях противодействия, осно-

ванная на двухэтапном процессе фильтрации данных и применении статистических пороговых значений в режиме реального времени.

Разработанные методы и гипервизорные решения прошли всестороннюю экспериментальную проверку и были представлены на ведущих международных конференциях по кибербезопасности в Америке, Европе и Азии, включая Black Hat USA, Black Hat Europe, HITB Security Conference, EKOPARTY, LABScon, Texas Cyber Summit и IEEE Symposium on Security and Privacy Workshops, а также на профильных российских научных и отраслевых форумах. Это подтверждает не только научную новизну предложенных подходов, но и их высокую практическую значимость, а также признание в профессиональном и академическом сообществе.

Несмотря на то, что основное внимание в работе уделено платформе Windows, предложенные алгоритмы, методы и средства для обнаружения скрытых угроз могут быть адаптированы для других современных операционных систем, включая UNIX-подобные ОС, такие как macOS и различные дистрибутивы Linux, в том числе отечественные разработки. Это подтверждает их архитектурную универсальность и потенциал для создания кроссплатформенных решений по защите ядра.

Сочетание системного научного подхода к анализу атак на ядро с практическими результатами в виде разработок по мониторингу, обнаружению и предотвращению атак делает данную работу значительным вкладом в развитие технологий кибербезопасности и открывает перспективы для дальнейших исследований и создания средств защиты нового поколения для современных операционных систем.

Будет ли поставлена точка в обеспечении безопасности ОС? Вероятнее всего, нет: эксперты и нарушители вовлечены в бесконечную гонку на опережение в условиях агрессивной и постоянно изменяющейся информационной среды.

Библиографический список

1. *Yosifovich P.* Windows Internals: System architecture, processes, threads, memory management, and more [Text] / P. Yosifovich, M. Russinovich, D. Solomon, A. Ionescu. — Part 1. — 7th ed. — Redmond: Microsoft Press, 2017. — 800 p; Part 2. — 7th ed. — Redmond: Microsoft Press, 2021. — 912 p.
2. *Смекалин С.А.* Зачем нужен селектор. Определение понятия согласно архитектуре x86 / С.А. Смекалин // Проблемы Науки. — 2015. — № 7 (37). URL: https://cyberleninka.ru/article/n/zachem-nuzhen-selektor-opredelenie-ponyatiya-soglasno-arhitekture-x86 (дата обращения: 23.11.2025).
3. *Силаков Д.В.* Использование аппаратной виртуализации в контексте информационной безопасности / Д.В. Силаков // Труды ИСП РАН. — 2011. URL: https://cyberleninka.ru/article/n/ispolzovanie-apparatnoy-virtualizatsii-v-kontekste-informatsionnoy-bezopasnosti (дата обращения: 23.11.2025).
4. *Yosifovich P.* Windows Kernel Programming [Text] / P. Yosifovich. — NY: Apress, 2019. — 450 p.
5. *Schmidt A.* Teaching operating systems: windows kernel projects [Text] / A. Schmidt, A. Polze, D. Probert // SIGCSE '10: Proceedings of the 41st ACM technical symposium on Computer science education. — 2010. — P. 490–494.
6. Microsoft. Windows Operating System Internals Curriculum Resource Kit. URL: https://micksmix.wordpress.com/2009/06/18/windows-operating-system-internals-curriculum-resource-kit/ (дата обращения 23.11.2025).
7. *Болотов А.С.* Файловая система ntfs: обзор версий, производительность / А.С. Болотов, Т.П. Болотова // Вестник ГУУ. — 2012. — № 7. URL: https://cyberleninka.ru/article/n/faylovaya-sistema-ntfs-obzor-versiy-proizvoditelnost (дата обращения: 23.11.2025).
8. Workpackage6_Virtual_Secure_Mode.pdf // Bundesamt für Sicherheit in der Informationstechnik: URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Cyber-Sicherheit/SiSyPHus/Workpackage6_Virtual_Secure_Mode.pdf?__blob=publicationFile&v=1 (дата обращения: 23.11.2025).
9. *Коркин И.Ю.* Технологии сокрытия вредоносных программ и новые способы противодействия им / И.Ю. Коркин // Безопасность информационных технологий. — 2009. — URL: https://

bit.spels.ru/index.php/bit/article/view/825 (дата обращения: 23.11.2025).
10. *Rudd E.M.* A Survey of Stealth Malware Attacks, Mitigation Measures, and Steps Toward Autonomous Open World Solutions / E.M. Rudd, A. Rozsa, M. Günther, T.E. Boult. — 2016. — URL: https://arxiv.org/pdf/1603.06028 (дата обращения: 24.11.2025).
11. *Rutkowska J.* Introducing Stealth Malware Taxonomy / J. Rutkowska. — 2006. — URL: https://www.semanticscholar.org/paper/Introducing-Stealth-Malware-Taxonomy-Rutkowska/49f29dbaf9a0e2e72590006d155d10c76e6cf43f (дата обращения: 24.11.2025).
12. Picus Security. PICUS 10 critical MITRE ATT&CK techniques: T1036 — Masquerading // Picus Security Blog. — URL: https://www.picussecurity.com/resource/blog/picus-10-critical-mitre-attck-techniques-t1036-masquerading (дата обращения: 24.11.2025).
13. *Коркин И.Ю.* Методика обнаружения нелегитимного программного обеспечения, использующего технологию аппаратной виртуализации: дис. ... канд. техн. наук [Текст] / И.Ю. Коркин. — М., 2011. — 151 с.
14. *Nadim M.* Kernel-level Rootkit Detection, Prevention and Behavior Profiling: A Taxonomy and Survey / M. Nadim. — 2023. — URL: https://doi.org/10.48550/arXiv.2304.00473 (дата обращения: 24.11.2025).
15. *Blunden B.* The Rootkit Arsenal: Escape and Evasion in the Dark Corners of the System / B. Blunden. — 2nd ed. — Jones & Bartlett Publishers, Burlington, MA. 2012. — 783 p.
16. Driver compatibility with memory integrity and VBS. — URL: https://learn.microsoft.com/en-us/windows-hardware/test/hlk/testref/driver-compatibility-with-device-guard (дата обращения: 24.11.2025).
17. *Ionescu A.* Demystifying Modern Windows Rootkits / A. Ionescu // Black Hat USA 2020. — 2020. — URL: https://www.blackhat.com/us-20/briefings/schedule/#demystifying-modern-windows-rootkits-19633 (дата обращения: 24.11.2025).
18. *Хоглунд Г.* Руткиты: внедрение в ядро Windows [Текст] / Г. Хоглунд, Дж. Батлер — СПб.: Питер, 2007. — 295 с.
19. *Liu L.* Research and Design of Rootkit Detection Method [Text] / L. Liu, Z. Yin, Y. Shen, H. Lin // Physics Procedia. — 2012. — Vol. 33. — P. 852–857.
20. *Joy J.* Rootkit Detection Mechanism: A Survey [Text] / J. Joy, A. John, J. Joy // Advances in Parallel Distributed Computing. — 2011. — P. 366–374.

21. *Davis M.* Hacking Exposed™ Malware & Rootkits Reviews / M. Davis, S. Bodmer. — URL: https://archive.org/stream/hackingexposed.malwareandrootkitsmcgrawhill2010/Hacking%20Exposed.%20Malware%20and%20Rootkits%20%28McGraw-Hill%2C%202010%29_djvu.txt (дата обращения: 24.11.2025).
22. *Graziano M.* Subverting Operating System Properties through Evolutionary DKOM Attacks / M. Graziano, L. Flore, A. Lanzi, D. Balzarotti. — 2021. — URL: https://www.researchgate.net/publication/303912861_Subverting_Operating_System_Properties_Through_Evolutionary_DKOM_Attacks (дата обращения: 24.11.2025).
23. *Bailey B.D.* Winning the Race to Bare Metal / B.D. Bailey // Black Hat USA 2008. — 2008. — URL: https://www.blackhat.com/presentations/bh-usa-08/Bailey/BH_US_08_Bailey_Winning_the_Race_to_Bare_Metal_Slides.pdf (дата обращения: 24.11.2025).
24. *Tanda S.* Hypervisor development for security analysis / S. Tanda // Hexacon 2025. — 2025. — URL: https://www.hexacon.fr/trainer/tanda/ (дата обращения: 24.11.2025).
25. *Desnos A.* Detecting (and creating!) a HVM rootkit (aka BluePill-like) [Text] / A. Desnos, E. Filiol, I. Lefou // Journal in Computer Virology. — 2011. Vol. 7. — P. 23–49.
26. *King S.T.* SubVirt: implementing malware with virtual machines [Text] / S.T. King, P.M. Chen // 2006 IEEE Symposium on Security and Privacy (S&P'06). — Berkeley/Oakland, CA, 2006. — P. 14–327.
27. *Embleton S.* SMMrootkit: a new breed of OS independent malware [Text] / S. Embleton, S. Sparks, C.C. Zou // Proceedings of the 4th International Conference on Security and Privacy in Communication Networks (SecureComm 2008). — ACM, 2008. — P. 91–96.
28. *Schiffman J.* The SMMRootkit Revisited: Fun with USB [Text] / J. Schiffman, D. Kaplan // 2014 9th International Conference on Availability, Reliability and Security (ARES). — IEEE, 2014. — P. 532–539.
29. *Szaknis M.* The design of the simple SMM rootkit / M. Szaknis, K. Szczypiorski // Proceedings of the 2022 9th International Conference on Wireless Communication and Sensor Networks (icWCSN '22). — 2022. — URL: https://dl.acm.org/doi/10.1145/3514105.3514114 (дата обращения: 24.11.2025).
30. *Korkin I.* Applying Memory Forensics to Rootkit Detection / I. Korkin, I. Nesterov // Annual ADFSL Conference on Digital Forensics, Security and Law, 2014. Richmond, VA, USA. — URL: https://www.researchgate.net/publication/301843478_Applying_

Memory_Forensics_to_Rootkit_Detection (дата обращения: 24.11.2025).
31. *Ермолов М.М.* Недокументированные инструкции x86 для управления CPU на уровне микроархитектуры в современных процессорах Intel / М.М. Ермолов, Д.В. Скляров, М.С. Горячий // Безопасность информационных технологий. — 2022. — URL: http://dx.doi.org/10.26583/bit.2022.4.03 (дата обращения: 24.11.2025).
32. *Teodorescu C.* Veni, No Vidi, No Vici: Attacks on ETW Blind EDR Sensors / C. Teodorescu, I. Korkin, A. Golchikov // Black Hat Europe Conference. London, UK, November 10–11, 2021. — URL: https://i.blackhat.com/EU-21/Wednesday/EU-21-Teodorescu-Veni-No-Vidi-No-Vici-Attacks-On-ETW-Blind-EDRs.pdf (дата обращения: 24.11.2025).
33. *Korkin I.* Blasting Event Driven Cornucopia: WMI based User Space Attacks Blind SIEMs and EDRs / I. Korkin, C. Teodorescu, A. Golchikov // Black Hat USA. Las Vegas (USA), 10 Aug. 2022. — URL: https://www.blackhat.com/us-22/briefings/schedule/#blasting-event-driven-cornucopia-wmi-based-user-space-attacks-blind-siems-and-edrs-27211 (дата обращения: 24.11.2025).
34. *Pogonin D.* Microsoft Defender Will Be Defended: MemoryRanger Prevents Blinding Windows AV / D. Pogonin, I. Korkin // ADFSL 2022 Conference on Digital Forensics, Security and Law, Florida, USA. — URL: https://commons.erau.edu/adfsl/2022/presentations/7/ (дата обращения: 24.11.2025).
35. *Hand M.* Evading EDR: The Definitive Guide to Defeating Endpoint Detection Systems [Text] / M. Hand. — San Francisco: No Starch Press. 2023. — 312 p.
36. *Watson C.* EDR Evasion: Stranger Things In A Payload, GIAC (GREM) Gold Certification. Advisor: Domenica Crognale, The SANS Institute, 27 June 2021. — URL: https://1filedownload.com/edr-evasion-stranger-things-in-a-payload-cissp-note-2/ (дата обращения: 24.11.2025).
37. *Tsaur W.-J.* Exploring Rootkit Detectors' Vulnerabilities Using a New Windows Hidden Driver Based Rootkit / W.-J. Tsaur, Y.-C. Chen // Proceedings of The Second IEEE International Conference on Social Computing (SocialCom2010). — 2010. — URL: https://doi.org/10.1109/SocialCom.2010.127 (дата обращения: 24.11.2025).
38. Vomel S., Lenz H. Visualizing Indicators of Rootkit Infections in Memory Forensics / S. Vomel, H. Lenz // Paper presented at 7th International Conference on IT Security Incident Management and IT Forensics (IMF). Nuremberg, German, 12–14 March 2013. — https://www.researchgate.net/publication/261039414_Visualizing_

Indicators_of_Rootkit_Infections_in_Memory_Forensics (дата обращения: 24.11.2025).
39. *Smolár M.* BlackLotus UEFI bootkit: Myth confirmed / M. Smolár. — URL: https://www.welivesecurity.com/2023/03/01/blacklotus-uefi-bootkit-myth-confirmed/ (дата обращения: 24.11.2025).
40. *Dolan-Gavitt B.* Forensic analysis of the Windows registry in memory / B. Dolan-Gavitt // Digital Investigation. — 2008. Vol. 5 (Suppl.). — URL: https://doi.org/10.1016/j.diin.2008.05.003 (дата обращения: 24.11.2025).
41. *Hossain M.A.* Enhanced detection of obfuscated malware in memory dumps: a machine learning approach for advanced cybersecurity / M.A. Hossain, M.S. Islam // Cybersecurity. — 2024. Vol. 7. — URL: https://doi.org/10.1186/s42400-024-00205-z (дата обращения: 24.11.2025).
42. *Hewardt M.* Advanced Windows Debugging [Text] / M. Hewardt, D. Pravat. — Boston: Addison-Wesley Professional, 2007. — 840 p.
43. Microsoft Documentation. (n.d.). MiniDumpWriteDump function. — URL: https://docs.microsoft.com/en-us/windows/win32/api/minidumpapiset/nf-minidumpapiset-minidumpwritedump (дата обращения: 24.11.2025).
44. *Sikorski M.* Practical Malware Analysis [Text] / M. Sikorski, A. Honig. — San Francisco: No Starch Press, 2012. — 800 p.
45. XADM: How to Use Userdump.exe to Capture the State of the Information Store. — URL: https://mskb.pkisolutions.com/kb/250509 (дата обращения: 24.11.2025).
46. Как получить дамп процесса в Windows 7, 8, 8.1, 10, 11 и в Windows Server 2008, 2012, 2016, 2019. URL: https://support.kaspersky.ru/common/diagnostics/12401?ysclid=midmdtgnna744298137 (дата обращения: 24.11.2025).
47. PMDump: A Tool for Memory Dumping. — URL: https://vidstromlabs.com/freetools/pmdump/ (дата обращения: 24.11.2025).
48. *Vasudevan A.* MalTRAK: Tracking and Eliminating Unknown Malware / A. Vasudevan // Proceedings of the 2008 Annual Computer Security Applications Conference. —2008. — URL: https://www.computer.org/csdl/proceedings-article/acsac/2008/3447a311/12OmNx8Ouor (дата обращения: 24.11.2025).
49. *Casey E.* Digital Evidence and Computer Crime: Forensic Science, Computers, and the Internet [Text] / E. Casey. — 3d ed. — Burlington, MA: Elsevier Academic Press, 2010. — 840 p.

50. *Carvey H.* Windows Forensic Analysis Toolkit: Advanced Analysis Techniques for Windows 8 [Text] / H. Carvey. — 4th ed. — Oxford: Syngress, 2014. — 350 p.
51. *Dykstra J.* Acquiring forensic evidence from infrastructure-as-a-service cloud computing: Exploring and evaluating tools, trust, and techniques [Text] / J. Dykstra, A. Sherman // Digital Investigation. — 2012. — Vol. 9. — P. S90–S98.
52. ProcDump v11.0. — URL: https://learn.microsoft.com/en-us/sysinternals/downloads/procdump (дата обращения: 24.11.2025).
53. *Milkovic L.* Defeating Windows memory forensics / L. Milkovic. — URL: https://storage.googleapis.com/google-code-archive-downloads/v2/code.google.com/dementia-forensics/Defeating%20Windows%20memory%20forensics.pdf?ysclid=midmscyov6376741169 (дата обращения: 24.11.2025).
54. *Okolica J.* A Compiled Memory Analysis Tool / J. Okolica, G. Peterson // Advances in Digital Forensics VI. DigitalForensics 2010: IFIP Advances in Information and Communication Technology. — URL: https://doi.org/10.1007/978-3-642-15506-2_14 (дата обращения: 25.11.2024).
55. *Graziano M.* Hypervisor Memory Forensics / M. Graziano, A. Lanzi, D Balzarotti // Research in Attacks, Intrusions, and Defenses: Paper presented at the 16th International Symposium. — URL: https://www.s3.eurecom.fr/docs/raid13_graziano.pdf (дата обращения: 25.11.2025).
56. Vis: Virtualization enhanced live forensics acquisition for native system. Digital Investigation / M. Yu, Z. Qi, Q. Lin [et al.]. — URL: https://www.sciencedirect.com/science/article/abs/pii/S1742287612000229?via%3Dihub (дата обращения: 25.11.2025).
57. *Kuhn S.* A forensic hypervisor for process tracking and exploit discovery / S. Kuhn, S. Taylor // Proceedings — IEEE Military Communications Conference MILCOM. — URL: https://www.researchgate.net/publication/261237031_A_forensic_hypervisor_for_process_tracking_and_exploit_discovery (дата обращения: 25.11.2025).
58. *Stüttgen J.* Anti-forensic resilient memory acquisition [Text] / J. Stüttgen, M. Cohen // Digital Investigation. — 2013. — Vol. 10. — P. S105–S115.
59. *Rutkowska J.* IsGameOver() Anyone? / J. Rutkowska, A. Tereshkin // Invisible Things Lab — URL: https://invisiblethingslab.com/resources/bh07/IsGameOver.pdf (дата обращения: 25.11.2025).
60. *Datta A.* A Logic of Secure Systems and its Application to Trusted Computing / A. Datta, J. Franklin, D. Garg, D. Kaynar // 30th IEEE

Symposium on Security and Privacy. — URL: https://people.mpi-sws.org/~dg/papers/oakland09.pdf (дата обращения: 25.11.2025).
61. *Lioy A.* Trusted-Computing Technologies for the Protection of Critical Information Systems / A. Lioy, G. Ramunno, D. Vernizzi // Proceedings of the International Workshop on Computational Intelligence in Security for Information Systems CISIS'08. Advances in Soft Computing. — 2009. — Vol. 53. — URL: https://doi.org/10.1007/978-3-540-88181-0_10 (дата обращения: 26.11.2025).
62. *Embleton S.* SMM Rootkits: A New Breed of OS Independent Malware / S. Embleton, S. Sparks, C. Zou // SECURECOMM ACM, 2008. — URL: https://eudl.eu/doi/10.1145/1460877.1460892 (дата обращения: 26.11.2025).
63. *Duflot L.* Using CPU System Management Mode to Circumvent Operating System Security Functions / L. Duflot, D. Etiemble, O. Grumelard. — URL: https://www.researchgate.net/publication/241643659_Using_CPU_System_Management_Mode_to_Circumvent_Operating_System_Security_Functions (дата обращения: 26.11.2025).
64. *Wang J.* Firmware-assisted Memory Acquisition and Analysis tools for Digital Forensics / J. Wang, F. Zhang, K. Sun, A. Stavrou // 2011 Sixth IEEE International Workshop on Systematic Approaches to Digital Forensic Engineering. — URL: https://ieeexplore.ieee.org/document/6159113?reload=true; (дата обращения: 26.11.2025).
65. When hardware meets software: A bulletproof solution to forensic memory acquisition / A. Reina, A. Fattori, F. Pagani [et al.] // ACM International Conference Proceeding Series. — URL: https://dl.acm.org/doi/abs/10.1145/2420950.2420962 (дата обращения: 26.11.2025).
66. *Tereshkin A.* Introducing Ring -3 Rootkits / A. Tereshkin, R. Wojtczuk // Black Hat USA. — URL: https://www.blackhat.com/presentations/bh-usa-09/TERESHKIN/BHUSA09-Tereshkin-Ring3Rootkit-SLIDES.pdf (дата обращения: 26.11.2025).
67. *Hill B.* Lenovo Rocked by Critical BIOS Vulnerability, Fingers Point to Shoddy Intel Reference Code / B. Hill. — URL: https://hothardware.com/news/lenovo-rocked-by-critical-bios-vulnerability (дата обращения: 26.11.2025).
68. Black Hat 2022: The Intel PPAM attack story. — URL: https://www.binarly.io/blog/black-hat-2022-the-intel-ppam-attack-story (дата обращения: 26.11.2025).
69. *Stewin P.* Understanding DMA Malware / P. Stewin, Iu. Bystrov. — URL: https://link.springer.com/chapter/10.1007/978-3-642-37300-8_2 (дата обращения: 26.11.2025).

70. *Ververis V.* Security Evaluation of Intel's Active Management Technology (Master of Science Thesis) / V. Ververis. — URL: https://people.kth.se/~maguire/DEGREE-PROJECT-REPORTS/100402-Vassilios_Ververis-with-cover.pdf (дата обращения: 26.11.2025).
71. *Armasu L.* Intel AMT Vulnerability Shows Intel's Management Engine Can Be Dangerous / L. Armasu // Tom's Hardware. — URL: https://www.tomshardware.com/news/intel-amt-vulnerability-me-dangerous,34300.html (дата обращения: 26.11.2025).
72. *Morgan B.* IOMMU protection against I/O attacks: A vulnerability and a proof-of-concept / B. Morgan, E. Alata, V. Nicomette, M. Kaaniche // Journal of the Brazilian Computer Society. — URL: https://link.springer.com/article/10.1186/s13173-017-0066-7 (дата обращения: 26.11.2025).
73. *Pietro R.* Virtualization Technologies and Cloud Security: Advantages, Issues, and Perspectives / R. Pietro, F. Lombardi // Essays Dedicated to Sushil Jajodia on the Occasion of His 70th Birthday. — URL: https://link.springer.com/chapter/10.1007/978-3-030-04834-1_9 (дата обращения: 26.11.2025).
74. *Гордиевских В.М.* Сущность, структура и классификация современных технологий виртуализации / В.М. Гордиевских // Вестник Шадринского государственного педагогического университета. — 2015. — № 1 (25). — URL: https://cyberleninka.ru/article/n/suschnost-struktura-i-klassifikatsiya-sovremennyh-tehnologiy-virtualizatsii (дата обращения: 26.11.2025).
75. *David F.* Cloaker: Hardware Supported Rootkit Concealment / F.M. David, E.M. Chan, J.C. Carlyle, R.H. Campbel. — URL: https://ieeexplore.ieee.org/document/4531160 (дата обращения: 26.11.2025).
76. *Patel A.* Analyzing different techniques and tools to recover data from volatile memory / A. Patel, N. Mistry // International Journal for Scientific Research & Development. — URL: https://www.researchgate.net/publication/286321690_An_Analyzing_of_different_Techniques_and_Tools_to_Recover_Data_from_Volatile_Memory (дата обращения: 26.11.2025).
77. Digital Forensics and Cyber Crime / Ed. by I. Baggili // Second International ICST Conference, ICDF2C 2010, Abu Dhabi, United Arab Emirates, October 4-6, 2010. — URL: https://doi.org/10.1007/978-3-642-19513-6 (дата обращения: 26.11.2025).
78. *Carrier B.* A hardware-based memory acquisition procedure for digital investigations / B. Carrier, J. Grand // Digital Investigation. — URL: https://grandideastudio.com/media/tribble_paper.pdf (дата обращения: 26.11.2025).

79. *Petroni Jr. N.L.* Copilot — A Coprocessor-based Kernel Runtime Integrity Monitor / N.L. Petroni, Jr., T. Fraser, J. Molina, W.A. Arbaugh. URL: https://www.semanticscholar.org/author/Jr-Timothy-Nick-L.-Petroni/2386047698 (дата обращения: 26.11.2025).
80. *Korkin I.* Applying Memory Forensics to Rootkit Detection / I. Korkin, I. Nesterow. — URL: https://www.academia.edu/7380266/Applying_Memory_Forensics_to_Rootkit_Detection (дата обращения: 26.11.2025).
81. *Parekh M.* Memory Forensic: Acquisition and Analysis of Memory and Its Tools comparison / M. Parekh, S. Jani // International Journal of Engineering Technologies and Management Research. — URL: https://www.granthaalayahpublication.org/ijetmr-ojms/index.php/ijetmr/article/view/IJETMR18-CINSP-13 (дата обращения: 26.11.2025).
82. *Boileau A.* Hit by a bus: Physical access attacks with Firewire / A. Boileau. URL: http://papers.put.as/papers/macosx/2006/ab_firewire_rux2k6-final.pdf (дата обращения: 26.11.2025).
83. *Breuk R.* Integrating DMA attacks in exploitation frameworks / R. Breuk, A. Spruyt. — URL: https://www.semanticscholar.org/paper/Integrating-DMA-attacks-in-exploitation-frameworks-Breuk-Spruyt/b8ab84315d0d87a4a03405316215da0be3ba0510 (дата обращения: 26.11.2025).
84. *Lawson N.* Don't Tell Joanna: The Virtualized Rootkit Is Dead / N. Lawson, P. Ferrie, T. Ptacek. — URL: http://www.orkspace.net/secdocs/Conferences/BlackHat/USA/2007/Dont%20Tell%20Joanna, %20The%20Virtualized%20Rootkit%20Is%20Dead.pdf (дата обращения: 26.11.2025).
85. *Bulygin Y.* Chipset Based Approach to Detect Virtualization Malware a.k.a. DeepWatch. Security Center of Excellence / Y. Bulygin, D. Samyde. — URL: infocon.org/cons/Black%20Hat/Black%20Hat%20USA/Black%20Hat%20USA%202008/conference%20cd/speakers/Bulygin_Detection_of_Rootkits/bh-us-08-bulygin_Chip_Based_Approach_to_Detect_Rootkits.pdf (дата обращения: 26.11.2025).
86. Advanced Micro Devices. AMD64 Technology: AMD64 Architecture Programmer's Manual, Volume 2: System Programming. — URL: https://tc.gtisc.gatech.edu/cs6265/2021/refs/amd64-vol2-sys.pdf (дата обращения: 26.11.2025).
87. *Rutkowska J.* Beyond The CPU: Defeating Hardware Based RAM Acquisition (part I: AMD case) / J. Rutkowska // COSEINC Advanced

Malware Labs. — URL: https://www.first.org/conference/2007/papers/rutkowska-joanna-slides.pdf (дата обращения: 26.11.2025).
88. MSDN (2010). Windows Research Kernel Source Code. — URL: https://github.com/sidyhe/wrk?ysclid=mimbruwuxy305120897 (26.11.2025).
89. *Ермишечкин А.Н.* Поиск ядра ОС Windows в виртуальном адресном пространстве для формирования отладочного дампа / А.Н. Ермишечкин // Вестник науки и образования. — 2020. — № 9-2 (87). — URL: https://cyberleninka.ru/article/n/poisk-yadra-os-windows-v-virtualnom-adresnom-prostranstve-dlya-formirovaniya-otladochnogo-dampa (дата обращения: 26.11.2025).
90. *Орленко Е.С.* Форензика и антифорензика: криминалистические аспекты [Текст] / Е.С. Орленко, И.Г. Гришаева // Вестник Воронежского государственного университета. Серия: Право. — 2023. — № 3 (54). — С. 254–257.
91. *Fu D.* A Windows Rootkit Detection Method Based on Cross-View / D. Fu, S. Zhou, C. Cao. — URL: https://www.researchgate.net/publication/238519435_A_Windows_Rootkit_Detection_Method_Based_on_Cross-View (дата обращения: 26.11.2025).
92. *Rutkowska J.* Thoughts about Cross-View based Rootkit Detection / J. Rutkowska. — URL: https://elhacker.info/manuales/Análisis%20de%20malware/crossview_detection_thoughts.pdf (дата обращения: 26.11.2025).
93. *Fortino G.* SigIL: A Signature-Based Approach of Malware Detection on Intermediate Language / G. Fortino, C. Greco, A. Guzzo, M. Ianni // Computer Security. ESORICS 2023 International Workshops: ESORICS 2023. Lecture Notes in Computer Science. — 2024. — Vol. 14399. — URL: https://doi.org/10.1007/978-3-031-54129-2_15 (дата обращения: 26.11.2025).
94. *Haruyama T.* One-byte modification for breaking memory forensic analysis / T. Haruyama, H. Suzuki // Proceedings of the Black Hat Europe,Amsterdam, The Netherlands, 14–16 March 2021. — URL: https://speakerdeck.com/takahiro_haruyama/one-byte-modification-for-breaking-memory-forensic-analysis (дата обращения: 27.11.2025).
95. *Jung S.* Memory Layout Extraction and Verification Method for Reliable Physical Memory Acquisition / S. Jung, S. Seo, Y. Kim, C. Lee // Electronics. — URL: https://doi.org/10.3390/electronics10121380 (дата обращения: 27.11.2025).
96. *Aumaitre D.* A little journey inside Windows memory / D. Aumaitre // J. Comput. Virol. — 2009. — Vol. 5. — URL: https://doi.org/10.1007/s11416-008-0112-2 (дата обращения: 27.11.2025).

97. *Pu W.* A Method for Hidden Process Detection Based on Routines of Thread Scheduling List / W. Pu, J. Yu, J. Chang, Y. Liu. — URL: https://www.researchgate.net/publication/251953342_A_Method_for_Hidden_Process_Detection_Based_on_Routines_of_Thread_Scheduling_List (дата обращения: 27.11.2025).
98. Cyber Security Essentials. Auerbach Publications / ed. by J. Graham, R. Olson, R. Howard. — URL: https://doi.org/10.1201/b10485 (дата обращения: 27.11.2025).
99. *Stuart M.* The Volatility Framework: Volatile memory artifact extraction utility framework / M. Stuart, A. Adair // Volatility Foundation. URL: https://www.volatilityfoundation.org/ (дата обращения: 27.11.2025).
100. *Komal B.* Step by step memory forensics / B. Komal // eForensics Magazine. — URL: https://www.cyberengage.org/post/memory-forensics-a-step-by-step-methodology (дата обращения: 27.11.2025).
101. *Cui W.* Tracking Rootkit Footprints with a Practical Memory Analysis System / W. Cui, M. Peinado, Z. Xu, E. Chan // 21st USENIX Security Symposium. — URL: https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/cui (дата обращения: 27.11.2025).
102. *Schuster A.* Searching for processes and threads in Microsoft Windows memory dumps / A. Schuster // Digital Investigation. — 2006. — Vol. 3, Supplement. — URL: https://www.sciencedirect.com/science/article/pii/S1742287606000727 (дата обращения: 27.11.2025).
103. SigGraph: Brute Force Scanning of Kernel Data Structure Instances Using Graph-based Signatures / Z. Lin, J. Rhee, X. Zhang. — URL: https://www.researchgate.net/publication/221655382_SigGraph_Brute_Force_Scanning_of_Kernel_Data_Structure_Instances_Using_Graph-based_Signatures (дата обращения: 27.11.2025).
104. Microsoft Learn. Снятие моментального снимка, просмотр процессов. — URL: https://learn.microsoft.com/ru-ru/windows/win32/toolhelp/taking-a-snapshot-and-viewing-processes (дата обращения: 27.11.2025).
105. Microsoft Learn. ZwQuerySystemInformation function — Win32 apps. — URL: https://learn.microsoft.com/ru-ru/windows/win32/sysinfo/zwquerysysteminformation (дата обращения: 27.11.2025).
106. Microsoft Learn. CreateToolhelp32Snapshot function (tlhelp32.h) — Win32 apps. — URL: https://learn.microsoft.com/ru-ru/windows/win32/api/tlhelp32/nf-tlhelp32-createtoolhelp32snapshot (дата обращения: 27.11.2025).
107. Microsoft Learn. Process32First function (tlhelp32.h) — Win32 apps. — URL: https://learn.microsoft.com/ru-ru/windows/

win32/api/tlhelp32/nf-tlhelp32-process32first (дата обращения: 27.11.2025).
108. Microsoft Learn. Process32Next function (tlhelp32.h) — Win32 apps. — URL: https://learn.microsoft.com/ru-ru/windows/win32/api/tlhelp32/nf-tlhelp32-process32next (дата обращения: 27.11.2025).
109. Microsoft Learn. EnumServicesStatusA function (winsvc.h) — Win32 apps. — URL: https://learn.microsoft.com/en-us/windows/win32/api/winsvc/nf-winsvc-enumservicesstatusa (дата обращения: 27.11.2025).
110. *Musavi S.* Back to Static Analysis for Kernel-Level Rootkit Detection / S. Musavi, M. Kharrazi // IEEE Transactions on Information Forensics and Security. — URL: https://www.researchgate.net/publication/264561621_Back_to_Static_Analysis_for_Kernel-Level_Rootkit_Detection (дата обращения: 27.11.2025).
111. *Hoglund G.* Rootkits: Subverting the Windows Kernel / G. Hoglund, J. Butler. — URL: https://www.oreilly.com/library/view/rootkits-subverting-the/0321294319/ (дата обращения: 27.11.2025).
112. *Ligh M.H.* The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory / M.H. Ligh, A. Case, J. Levy, A. Walters. — Indianapolis: Wiley, 2014. — 912 p.
113. Understanding Windows DKOM (Direct Kernel Object Manipulation) techniques (Part 1-EPROCESS). — URL: https://nixhacker.com/understanding-windows-dkom-direct-kernel-object-manipulation-attacks-eprocess/ (дата обращения: 27.11.2025).
114. *Probert D.* Architecture of the Windows Kernel / D. Probert. — URL: https://www.cs.fsu.edu/~zwang/files/cop4610/Fall2016/windows.pdf#:~:text=URL%3A%20https%3A%2F%2Fwww.cs.fsu.edu%2F~zwang%2Ffiles%2Fcop4610%2FFall2016%2Fwindows.pdf%0AVisible%3A%200%25%20 (дата обращения: 27.11.2025).
115. NtQueryDirectoryObject function — URL: https://learn.microsoft.com/en-us/windows/win32/devnotes/ntquerydirectoryobject (дата обращения: 27.11.2025).
116. *Tsaur W.-J.* New Protection of Kernel-level Digital Rights Management in Cloud-based Consumer Electronics Environments / W.-J. Tsaur, L.-Y. Yeh // Proceedings of the International Conference on Grid & Cloud Computing and Applications (GCA). — URL: https://worldcomp-proceedings.com/proc/p2015/GCA2880.pdf (дата обращения: 27.11.2025).
117. *Tsaur W.* New Windows Rootkit Technologies for Enhancing Digital Rights Management in Cloud Computing Environments / W. Tsaur, J. Wu // Proceedings of the 2014 International Conference on

eLearning, e-Business, Enterprise Information Systems, and e-Governm. — URL: https://worldcomp-proceedings.com/proc/p2014/EEE2315.pdf (дата обращения: 27.11.2025).
118. *Jakobsson M.* Crimeware: Understanding New Attacks and Defenses / M. Jakobsson, Z. Ramzan. — URL: https://www.researchgate.net/publication/31864971_Crimeware_Understanding_New_Attacks_and_Defenses_M_Jakobsson_Z_Ramzan (дата обращения: 27.11.2025).
119. *Pistelli D.* AntiMida 1.0 / D. Pistelli. — URL: http://www.ntcore.com/files/antimida_1.0.htm (дата обращения: 27.11.2025).
120. Windows 10 Control Flow Guard Internals (mj0011). — URL: https://github.com/tpn/pdfs/blob/master/Windows%2010%20Control%20Flow%20Guard%20Internals%20(mj0011).pdf (дата обращения: 27.11.2025).
121. IDebugDataSpaces::ReadDebuggerData method (dbgeng.h), URL: https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/dbgeng/nf-dbgeng-idebugdataspaces-readdebuggerdata (дата обращения: 28.11.2025).
122. The Web Application Hacker's Handbook: Discovering and Exploiting Security Flaws / D. Stuttard, M. Pinto, M.H. Ligh [et al.]. — 2^{d} ed. — John Wiley & Sons, Inc. 2011. — 912 p.
123. Malware Analyst's Cookbook and DVD: Tools and Techniques for Fighting Malicious Code, Chapter 16. Memory Forensics: Rootkits. — URL: https://apprize.best/security/web_1/41.html (дата обращения: 28.11.2025).
124. HKLM\SYSTEM\CurrentControlSet\Services Registry Tree. — URL: https://learn.microsoft.com/en-us/windows-hardware/drivers/install/hklm-system-currentcontrolset-services-registry-tree (дата обращения: 28.11.2025).
125. Using the NT Registry for Driver Install. — URL: https://www.osronline.com/article.cfm%5eid=170.htm (дата обращения: 28.11.2025).
126. Функция RegEnumKeyExA (winreg.h). — URL: https://learn.microsoft.com/ru-ru/windows/win32/api/winreg/nf-winreg-regenumkeyexa (дата обращения: 28.11.2025).
127. Swachchhanda Shrawan. AURORA — Leveraging ETW for Advanced Threat Detection // Nextron Systems. — URL: https://www.nextron-systems.com/2025/07/31/aurora-leveraging-etw-for-advanced-threat-detection/ (дата обращения: 28.11.2025).
128. *Mandt T.* Kernel Pool Exploitation on Windows 7 / T. Mandt // Black Hat USA. — URL: https://media.blackhat.com/bh-dc-11/

Mandt/BlackHat_DC_2011_Mandt_kernelpool-wp.pdf (дата обращения: 28.11.2025).
129. Windows NT 4.0 Full Free Source Code (912 MB). — URL: http://igorkorkin.blogspot.ru/2013/09/windows-nt-40-full-free-source-code-912_16.html (дата обращения: 28.11.2025).
130. *Ligh M.* Investigating Windows Threads with Volatility / M. Ligh. — URL: https://mnin.blogspot.com/2011/04/investigating-windows-threads-with.html (дата обращения: 28.11.2025).
131. *Schuster A.* Searching for processes and threads in Microsoft Windows memory dumps [Text] / A. Schuster // Digital Investigation. — 2006. — Vol. 3. — P. 10–16.
132. *Sylve J.* Pool tag quick scanning for windows memory analysis [Text] / J. Sylve, L. Marziale, Golden G. Richard III // Digital Investigation. — 2016.— Vol. 16. — P. S25–S32.
133. *Dorfman A.* FRODO: Format Reverser of Data Objects / A. Dorfman // Proceedings of the Hack In The Box Security Conference (HITB), Amsterdam, Netherlands, 2014. — URL: https://arxiv.org/pdf/1606.04662 (дата обращения: 28.11.2025).
134. *Wang T.-Y.* Detecting Unknown Malicious Executables Using Portable Executable Headers [Text] / T.-Y. Wang, C.-H. Wu, C.-C. Hsieh // 2009 Fifth International Joint Conference on INC, IMS and IDC, Seoul, Korea (South). — 2009. P. 278–284.
135. *Barabosch T.* Learn how to fix PE magic numbers with Malduck / T. Barabosch. — URL: https://archive.orkl.eu/3b6da142765deeb858a1b7806c8680edce8bda00.pdf (дата обращения: 28.11.2025).
136. *Calvet J.* Aligot: Cryptographic function identification in obfuscated binary programs [Text] / J. Calvet, J. Fernandez, J.-Y. Marion // Proceedings of the ACM Conference on Computer and Communications Security. — 2012. — P. 169–182.
137. *Chen R.* The history of calling conventions, part 3 / R. Chen // The Old New Thing. — URL: https://devblogs.microsoft.com/oldnewthing/20040108-00/?p=41163 (дата обращения: 28.11.2025).
138. Idea: Opcode-Sequence-Based Malware Detection [Text] / I. Santos, F. Brezo, J. Nieves [et al.] // Proc 2nd International Symposium on Engineering Secure Software and Systems. — 2010.— P. 35–43.
139. Opcode-Sequence-Based Semi-supervised Unknown Malware Detection / I. Santos, B. Sanz, C. Laorden [et al.] // Computational Intelligence in Security for Information Systems. Lecture Notes in Computer Science. — 2011. — Vol. 6694. — Springer, Berlin, Heidelberg. URL: https://doi.org/10.1007/978-3-642-21323-6_7 (дата обращения: 28.11.2025).

140. *Zolotukhin M.* Detection of zero-day malware based on the analysis of opcode sequences [Text] / M. Zolotukhin, T. Hamalainen // 2014 IEEE 11th Consumer Communications and Networking Conference, CCNC 2014. — 2014. — P. 386–391.
141. *Honap A.M.* Hiding Kernel Level Rootkits Using Buffer Overflow and Return Oriented Programming / A.M. Honap, W. Lee // International Conferences on Information Science and System. — URL: https://www.researchgate.net/publication/321458260_Hiding_Kernel_Level_Rootkits_Using_Buffer_Overflow_and_Return_Oriented_Programming (дата обращения: 28.11.2025).
142. *Pareek H.* A Heuristics-based Static Analysis Approach for Detecting Packed PE Binaries / H. Pareek, R. Arora, A. Singh. — URL: https://www.researchgate.net/publication/257367777_A_Heuristics-based_Static_Analysis_Approach_for_Detecting_Packed_PE_Binaries (дата обращения: 28.11.2025).
143. *Lyda R.* Using Entropy Analysis to Find Encrypted and Packed Malware / R. Lyda, J. Hamrock. — URL: https://www.researchgate.net/publication/3437909_Using_Entropy_Analysis_to_Find_Encrypted_and_Packed_Malware (дата обращения: 28.11.2025).
144. *Vassilev A.* The Importance of Entropy to Information Security [Text] / A. Vassilev, T. Hall // Computer. — 2014. — Vol. 47. — P. 78–81.
145. *Davies S.R.* Comparison of Entropy Calculation Methods for Ransomware Encrypted File Identification / S.R. Davies, R. Macfarlane, W.J. Buchanan // Entropy. — 2022. — URL: https://arxiv.org/pdf/2210.13376 (дата обращения: 28.11.2025).
146. Binary File's Visualization and Entropy Features Analysis Combined with Multiple Deep Learning Networks for Malware Classification / H. Guo, S. Huang, C. Huang [et al.] // Security and Communication Networks. — URL: https://doi.org/10.1155/2020/8881760 (дата обращения: 28.11.2025).
147. *Maartmann-Moe C.* The persistence of memory: Forensic identification and extraction of cryptographic keys / C. Maartmann-Moe, S.E. Thorkildsen, A. Årnes // Digital Investigation. — 2009. — Vol. 6, suppl. — URL: https://www.sciencedirect.com/science/article/pii/S1742287609000486 (дата обращения: 28.11.2025).
148. *Матвеева В.С.* Энтропия и ее использование для решения задач информационной безопасности [Текст] / В.С. Матвеева // Безопасность информационных технологий (БИТ). — 2014. — № 4. — С. 30–36.
149. *Jochheim B.* On the Automatic Detection of Embedded Malicious Binary Code using Signal Processing Techniques / B. Jochheim //

Project Report, Faculty of Engineering and Computer Science. — URL: https://inet.haw-hamburg.de/teaching/ss-2012/master-projects/benjamin_jochheim_pr1.pdf (дата обращения: 28.11.2025).
150. *Pareek H.* Entropy and n-gram analysis of malicious PDF documents / H. Pareek // International Journal of Engineering Research and Technology. — URL: https://www.researchgate.net/publication/235974671_Entropy_and_n-gram_analysis_of_malicious_PDF_documents (дата обращения: 28.11.2025).
151. *Iwamoto K.* The Method for Shellcode Extraction from Malicious Document File Using Entropy and Emulation / K. Iwamoto, K. Wasaki // Journal of Information Processing. — URL: https://www.researchgate.net/publication/273576295_The_Method_for_Shellcode_Extraction_from_Malicious_Document_File_Using_Entropy_and_Emulation (дата обращения: 28.11.2025).
152. *Tabish S.* Malware detection using statistical analysis of byte-level file content / S. Tabish, M. Shafiq, M. Farooq. — URL: https://www.semanticscholar.org/paper/Malware-detection-using-statistical-analysis-of-Tabish-Shafiq/4828a78f56f40fa5fa35c23e540fc3171117ec12 (дата обращения: 28.11.2025).
153. *Schmidt T.* Context-adaptive Entropy Analysis as a Lightweight Detector of Zeroday Shellcode Intrusion for Mobiles / T. Schmidt, M. Wahlisch, M. Groning // The ACM Conference on Wireless Network Security (WiSec), Hamburg, Germany. — URL: https://www.sigsac.org/wisec/WiSec2011/wisec2011-poster-3-schmidt-mobile-ids.pdf, (дата обращения: 28.11.2025).
154. *Nguyen K.* An approach to detect network attacks applied for network forensics / K. Nguyen, D. Tran, W. Ma, D. Sharma // 11th International Conference on Fuzzy Systems and Knowledge Discovery. — URL: https://www.researchgate.net/publication/289092755_An_approach_to_detect_network_attacks_applied_for_network_forensics (28.11.2025).
155. *Devi D.* PE File Features in Detection of Packed Executables / D. Devi, S. Nandi // International Journal of Computer Theory and Engineering. — URL: https://www.researchgate.net/publication/269838084_PE_File_Features_in_Detection_of_Packed_Executables (дата обращения: 28.11.2025).
156. *Nataraja L.* Detecting packed executables based on raw binary data / L. Nataraja, G. Jacobb, B.S. Manjunatha. — URL: https://vision.ece.ucsb.edu/abstract/567 (дата обращения: 28.11.2025).
157. Countering Entropy Measure Attacks on Packed Software Detection / X. Ugarte-Pedrero, I. Santos, B. Sanz [et al.]. — URL: https://www.

researchgate.net/publication/234046681_Countering_Entropy_Measure_Attacks_on_Packed_Software_Detection (28.11.2025).
158. *Поддубный В.А.* Средство обнаружения скрытого исполнимого кода в памяти ОС Windows / В.А. Поддубный, И.Ю. Коркин // Вопросы кибербезопасности. — 2019. — № 5 (33). — URL: https://cyberleninka.ru/article/n/sredstvo-obnaruzheniya-skrytogo-ispolnimogo-koda-v-pamyati-oc-windows (дата обращения: 28.11.2025).
159. *Hall G.* Sliding Window Measurement for File Type Identification / G. Hall, W. Davis. — URL: https://www.researchgate.net/publication/237601448_Sliding_Window_Measurement_for_File_Type_Identification (дата обращения: 28.11.2025).
160. Automated mapping of large binary objects using primitive fragmenttype classification / G. Conti, S. Bratus, A. Shubina [et al.] // Digital Investigation. — 2010. — Vol. 7. — URL: https://www.researchgate.net/publication/242822686_Automated_mapping_of_large_binary_objects_using_primitive_fragmenttype_classification (дата обращения: 28.11.2025).
161. *Alshahwan N.* Detecting Malware with Information Complexity / N. Alshahwan, E. Barr, D. Clark, G. Danezis // Entropy. — 2015. — Vol. 22. — URL: https://www.researchgate.net/publication/272844860_Detecting_Malware_with_Information_Complexity (дата обращения: 28.11.2025).
162. *Matveeva V.* Searching for Random Data in File System During Forensic Expertise / V. Matveeva, A. Epishkina // Biosciences, Biotechnology Research Asia, BBRA. — 2015. — URL: https://www.researchgate.net/publication/283030528_Searching_for_Random_Data_in_File_System_During_Forensic_Expertise (дата обращения: 28.11.2025).
163. *Bat-Erdene M.* Dynamic classification of packing algorithms for inspecting executables using entropy analysis / M. Bat-Erdene, T. Kim, H. Li, H. Lee. — URL: https://www.researchgate.net/publication/261487633_Dynamic_classification_of_packing_algorithms_for_inspecting_executables_using_entropy_analysis (дата обращения: 28.11.2025).
164. *Ispahany J.* Ransomware Detection Using Machine Learning: A Review, Research Limitations and Future Directions / J. Ispahany, Md R. Islam, M. Khan, I. Zahidul // IEEE Access. — URL: https://www.researchgate.net/publication/380408103_Ransomware_detection_using_machine_learning_A_review_research_limitations_and_future_directions (дата обращения: 28.11.2025).

165. *Cortesi A.* Visualizing entropy in binary files / A. Cortesi. — URL: https://corte.si/posts/visualisation/entropy/ (дата обращения: 28.11.2025).
166. *Conti G.* Visual Reverse Engineering of Binary and Data Files / G. Conti, E. Dean, M. Sinda, B. Sangster // Visualization for Computer Security. VizSec 2008. : Lecture Notes in Computer Science, vol 5210. Springer, Berlin, Heidelberg, 2008. — URL: https://doi.org/10.1007/978-3-540-85933-8_1. (дата обращения: 28.11.2025).
167. *Merkel R.* Statistical Detection of Malicious PE-Executables for Fast Offline Analysis / R. Merkel, T. Hoppe, Ch. Kraetzer, J. Dittmann — URL: https://www.researchgate.net/publication/221521417_Statistical_Detection_of_Malicious_PE-Executables_for_Fast_Offline_Analysis (дата обращения: 28.11.2025).
168. *Kazi M.* Detecting the Zeus Banking Malware Using the Random Forest Binary Classification Algorithm and a Manual Feature Selection Process / M. Kazi, S. Woodhead, D. Gan. — URL: https://www.researchgate.net/publication/340920876_Detecting_the_Zeus_Banking_Malware_Using_the_Random_Forest_Binary_Classification_Algorithm_and_a_Manual_Feature_Selection_Process (дата обращения: 28.11.2025).
169. *Kazi M.A.* Comparing and analysing binary classification algorithms when used to detect the Zeus malware / M. Kazi, S. Woodhead, D. Gan // 2019 Sixth HCT Information Technology Trends (ITT): Ras Al Khaimah, United Arab Emirates, 2019. — URL: https://www.researchgate.net/publication/340896067_Comparing_and_analysing_binary_classification_algorithms_when_used_to_detect_the_Zeus_malware (дата обращения: 28.11.2025).
170. *Russinovich M.* Process Explorer. Sysinternals. Microsoft / M. Russinovich. — URL: https://learn.microsoft.com/en-us/sysinternals/downloads/process-explorer (дата обращения 28.11.2025).
171. *Espiner T.* Microsoft blocks Vista driver 'hack' tool / T. Espiner // ZDNet. — URL: https://www.zdnet.com/article/microsoft-blocks-vista-driver-hack-tool/ (дата обращения: 28.11.2025).
172. *Korkin I.* Windows 8 is Cyber Battlefield / I. Korkin. — URL: https://igorkorkin.blogspot.com/2012/09/windows-8-is-cyber-battlefield.html (дата обращения: 28.11.2025).
173. *Eresheim S.* The Evolution of Process Hiding Techniques in Malware — Current Threats and Possible Countermeasures / S. Eresheim, R. Luh, S. Schrittwieser // J. Inf. Process. — URL: https://www.researchgate.net/publication/319868952_The_Evolution_of_Process_Hiding_

Techniques_in_Malware_-_Current_Threats_and_Possible_Countermeasures (дата обращения: 28.11.2025).
174. Stealth Loader: Trace-Free Program Loading for API Obfuscation / Y. Kawakoya, E. Shioji, Y. Otsuki [et al.]. — URL: https://link.springer.com/chapter/10.1007/978-3-319-66332-6_10 (дата обращения: 29.11.2025).
175. *Bauman E.* Superset Disassembly: Statically Rewriting x86 Binaries Without Heuristics / E. Bauman, Z. Lin, K.W. Hamlen // Network and Distributed System Security Symposium. — 2018. — URL: https://www.researchgate.net/publication/323249543_Superset_Disassembly_Statically_Rewriting_x86_Binaries_Without_Heuristics (дата обращения: 29.11.2025).
176. *Harbour N.* Recognizing and Avoiding Disassembled Junk / N. Harbour // Mandiant. URL: https://cloud.google.com/blog/topics/threat-intelligence/recognizing-avoiding-disassembled-junk/ (дата обращения: 29.11.2025).
177. *Zipf G.K.* Human behavior and the principle of least effort [Text] / G.K. Zipf. — Addison-Wesley Press, 1949. — 597 p.
178. *Mandelbrot B.B.* An In-formational Theory of the Statistical Structure of Languages [Text] / B.B. Mandelbrot // Communication Theory. — Academic Press, Princeton, 1953. — P. 486–502.
179. *Barinova O.* On Detection of Multiple Object Instances Using Hough Transforms / O. Barinova, V. Lempitsky, P. Kohli // IEEE transactions on pattern analysis and machine intelligence. — URL: https://www.researchgate.net/publication/221982302_On_Detection_of_Multiple_Object_Instances_Using_Hough_Transforms (дата обращения: 29.11.2025).
180. *Nguyen A.Q.* Capstone: Next-Gen Disassembly Framework / A.Q. Nguyen // Blackhat USA. — URL: https://www.capstone-engine.org/BHUSA2014-capstone.pdf (дата обращения: 29.11.2025).
181. *Lekander A.* Inside the Underground Market for EV Code Signing Certificates / A. Lekander. — URL: https://cyberinsider.com/inside-the-underground-market-for-ev-code-signing-certificates/ (дата обращения: 29.11.2025).
182. *Toulas B.* Hackers exploit Windows policy to load malicious kernel drivers / B. Toulas. — URL: https://www.bleepingcomputer.com/news/security/hackers-exploit-windows-policy-to-load-malicious-kernel-drivers/ (дата обращения: 29.11.2025).
183. *Neal C.* Old certificate, new signature: Open source tools forge signature timestamps on Windows drivers / C. Neal. — URL: https://blog.talosintelligence.com/old-certificate-new-signature/ (дата обращения: 29.11.2025).

184. How much do criminals pay for certificates on the dark web? — URL: https://www.helpnetsecurity.com/2017/11/03/certificates-sale-dark-web/ (дата обращения: 29.11.2025).
185. *Zohdy M.* BlackCat ransomware deploys new signed kernel driver / M. Zohdy, S. Magdy, M. Fahmy. — URL: https://www.trendmicro.com/en_us/research/23/e/blackcat-ransomware-deploys-new-signed-kernel-driver.html (дата обращения: 29.11.2025).
186. *Svajcer V.* Exploring vulnerable Windows drivers / V. Svajcer, J. Nutland, N. Hoffman, C. Neal. — URL: https://blog.talosintelligence.com/exploring-vulnerable-windows-drivers/ (дата обращения: 29.11.2025).
187. Qilin Ransomware and the Hidden Dangers of BYOVD. — URL: https://blackpointcyber.com/blog/qilin-ransomware-and-the-hidden-dangers-of-byovd/ (дата обращения: 10.10.2025).
188. *Toulas B.* Ransomware gangs exploit Paragon Partition Manager bug in BYOVD attacks / B. Toulas. — URL: https://www.bleepingcomputer.com/news/security/ransomware-gangs-exploit-paragon-partition-manager-bug-in-byovd-attacks/ (дата обращения: 29.11.2025).
189. *Jadhav A.* Evolution of evasive malwares: A survey / A. Jadhav, D. Vidyarthi, M. Hemavathy. — URL: https://www.researchgate.net/profile/Ashish-Jadhav (дата обращения: 29.11.2025).
190. *Seals T.* Project Sauron has been spying on governments for 5 years / T. Seals. — URL: https://www.infosecurity-magazine.com/news/project-sauron-has-been-spying/ (дата обращения: 29.11.2025).
191. *Prakash A.* On the Trustworthiness of Memory Analysis —An Empirical Study from the Perspective of Binary Execution / A. Prakash, E. Venkataramani, H. Yin, Z. Lin // IEEE Transactions on Dependable and Secure Computing. — URL: https://www.researchgate.net/publication/285005808_On_the_Trustworthiness_of_Memory_Analysis_-An_Empirical_Study_from_the_Perspective_of_Binary_Execution (29.11.2025).
192. *Rhee J.* Kernel Malware Analysis with Un-tampered and Temporal Views of Dynamic Kernel Memory / J. Rhee, R. Riley, D. Xu, X. Jiang. — URL: https://www.researchgate.net/publication/221427681_Kernel_Malware_Analysis_with_Un-tampered_and_Temporal_Views_of_Dynamic_Kernel_Memory (29.11.2025).
193. *Matrosov A.* Rootkits and Bootkits: Reversing Modern Malware and Next Generation Threats [Text] / A. Matrosov, E. Rodionov, S. Bratus. — San Francisco, CA: No Starch Press, 2019. — 448 p.
194. *Block F.* Investigating Memory Analysis Tools — SSDT Hooking via Pointer Replacement / F. Block. — URL: https://insinuator.

net/2015/12/investigating-memory-analysis-tools-ssdt-hooking-via-pointer-replacement/ (дата обращения: 29.11.2025).
195. *Ahmed I.* Integrity Checking of Function Pointers in Kernel Pools via Virtual Machine Introspection / I. Ahmed, Richard III, G. Golden, A. Zoranic, V. Roussev. — URL: https://www.researchgate.net/publication/300111604_Integrity_Checking_of_Function_Pointers_in_Kernel_Pools_via_Virtual_Machine_Introspection (дата обращения: 29.11.2025).
196. *Sparks S.* Shadow Walker: Raising the Bar for Rootkit Detection / S. Sparks, J. Butler // Black Hat Japan 2005. — URL: https://www.blackhat.com/presentations/bh-jp-05/bh-jp-05-sparks-butler.pdf (дата обращения: 29.11.2025).
197. You Can Run but You Can't Read: Preventing Disclosure Exploits in Executable Code / M. Backes, T. Holz, B. Kollenda // Proceedings of the ACM Conference on Computer and Communications Security. — URL: https://dl.acm.org/doi/10.1145/3321705.3329819 (дата обращения: 29.11.2025).
198. Intel 64 and IA-32 Architectures Software Developer's Manual. Volume 3A: System Programming Guide, Part 1. — Intel Corporation, 2023. — URL: https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3a-part-1-manual.pdf (дата обращения: 29.11.2025).
199. *Hu K.* Data Protection Method, Apparatus, Storage Medium, and Computer Device / K. Hu, I.Y. Korkin // European patent EP4231181B1. Granted 23 Apr. 2025: Assignee: Huawei Technologies Co. Ltd. — URL: https://patents.google.com/patent/EP4231181B1/en (дата обращения: 29.11.2025).
200. *Mosberger D.* IA-64 Linux Kernel: Design and Implementation / D. Mosberger, S. Eranian. — URL: https://www.researchgate.net/publication/234785353_IA-64_Linux_Kernel_Design_and_Implementation (дата обращения: 29.11.2025).
201. *Seshadri A.* SecVisor: a tiny hypervisor to provide lifetime kernel code integrity for commodity OSes / A. Seshadri, M. Luk, N. Qu, A. Perrig // ACM SIGOPS Operating Systems Review. — URL: https://www.researchgate.net/publication/220910042_SecVisor_a_tiny_hypervisor_to_provide_lifetime_kernel_code_integrity_for_commodity_OSes (дата обращения: 29.11.2025).
202. *Ortolani S.* KLIMAX: Profiling Memory Write Patterns to Detect Keystroke-Harvesting Malware / S. Ortolani, C. Giuffrida, B. Crispo. — URL: https://www.researchgate.net/publication/221427552_KLIMAX_Profiling_Memory_Write_Patterns_to_Detect_Keystroke-Harvesting_Malware (дата обращения: 29.11.2025).

203. *Dang A.* System and method for kernel rootkit protection in a hypervisor environment (Patent No. EP2766844A1) / A. Dang, P. Mohinder, V. Srivastava // European Union: McAfee LLC. — URL: https://patents.google.com/patent/EP2766844A1 (дата обращения: 29.11.2025).
204. *Cheng Y.* DriverGuard: Virtualization-Based Fine-Grained Protection on I/O Flows / Y. Cheng, X. Ding, R.H. Deng // ACM Transactions on Information and System Security. — URL: https://doi.org/10.1145/2505123 (дата обращения: 29.11.2025).
205. *Suzaki K.* Kernel Memory Protection by an Insertable Hypervisor Which Has VM Introspection and Stealth Breakpoints / K. Suzaki, T. Yagi, K. Kobara, T. Ishiyama // Proceedings of the Conference. — 2014. — URL: https://doi.org/10.1007/978-3-319-09843-2_4 (дата обращения: 29.11.2025).
206. MOSKG: countering kernel rootkits with a secure paging mechanism: MOSKG countering rootkits with secure paging mechanism / G. Yan, S. Luo, F. Feng // Security and Communication Networks. — 2015. — Vol. 8. — URL: https://www.researchgate.net/publication/277336231_MOSKG_countering_kernel_rootkits_with_a_secure_paging_mechanism_MOSKG_countering_rootkits_with_secure_paging_mechanism (29.11.2025).
207. *Wang Z.* HyperSafe: A Lightweight Approach to Provide Lifetime Hypervisor Control-Flow Integrity / Z. Wang, X. Jiang. — URL: https://www.researchgate.net/publication/220713660_HyperSafe_A_Lightweight_Approach_to_Provide_Lifetime_Hypervisor_Control-Flow_Integrity (дата обращения: 29.11.2025).
208. *Luțaș A.* U-HIPE: hypervisor-based protection of user-mode processes in Windows / A. Luțaș, A. Coleșa, S. Lukacs, D. Lutas // Journal of Computer Virology and Hacking Techniques. — 2015. — URL: https://www.researchgate.net/publication/272029327_U-HIPE_hypervisor-based_protection_of_user-mode_processes_in_Windows (дата обращения: 29.11.2025).
209. *Srivastava A.* Efficient Monitoring of Untrusted Kernel-Mode Execution / A. Srivastava, J. Giffin. — URL: https://www.ndss-symposium.org/wp-content/uploads/2017/09/sriv.pdf (дата обращения: 29.11.2025).
210. *Martignoni L.* Live and Trustworthy Forensic Analysis of Commodity Production Systems / L. Martignoni, A. Fattori, R. Paleari, L. Cavallaro. — URL: https://www.researchgate.net/publication/221427657_Live_and_Trustworthy_Forensic_Analysis_of_Commodity_Production_Systems (29.11.2025).

211. *Pfoh J.* Nitro: hardware-based system call tracing for virtual machines [Text] / J. Pfoh, C. Schneider, C. Eckert // Proceedings of the 6th International Conference on Advances in Information and Computer Security (IWSEC'11). — Berlin: Springer-Verlag, 2011. P. 96–112.
212. *Azab A.M.* HIMA: A Hypervisor-Based Integrity Measurement Agent / A.M. Azab, P. Ning, E.C. Sezer, X. Zhang // 2009 Annual Computer Security Applications Conference. Honolulu, HI, USA, 2009. — URL: https://www.researchgate.net/publication/221046631_HIMA_A_Hypervisor-Based_Integrity_Measurement_Agent (дата обращения: 29.11.2025).
213. *Yan L.K.* Transparent and Precise Malware Analysis Using Virtualization: From Theory to Practice / L.K. Yan // Electrical Engineering and Computer Science — Dissertations. — 2013. — URL: https://surface.syr.edu/eecs_etd/332 (дата обращения: 29.11.2025).
214. Nikhil Bhatia Performance Evaluation of AMD RVI Hardware Assist // VMware ESX 3.5. — 2008. — URL: https://www.vmware.com/docs/rvi_performance (дата обращения: 29.11.2025).
215. MMU Virtualization Via Intel EPT: Technical Details. — URL: https://revers.engineering/mmu-ept-technical-details/ (дата обращения: 29.11.2025).
216. *Natu N.* Nested Paging in bhyve / N. Natu, P. Grehan // The FreeBSD Project. — URL: https://people.freebsd.org/~neel/bhyve/bhyve_nested_paging.pdf (дата обращения: 29.11.2025).
217. How to hide a hook: A hypervisor for rootkits. Phrack Magazine, 0x0f(0x45). — URL: https://www.phrack.org/issues/69/15.html (дата обращения: 29.11.2025).
218. Intel Corporation. Intel 64 and IA-32 Architectures Software Developer's Manual. Volume 3C: System Programming Guide, Part 3. Chapter 28: VMX Support for Address Translation. — URL: https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3c-part-3-manual.pdf (дата обращения: 29.11.2025).
219. Scalability, fidelity and stealth in the DRAKVUF dynamic malware analysis system / T.K. Lengyel, S. Maresca, B.D. Payne [et al.] // Proceedings of the 30th Annual Computer Security Applications Conference (ACSAC '14). — Association for Computing Machinery, New York, NY, USA, 2014. — URL: https://doi.org/10.1145/2664243.2664252 (дата обращения: 29.11.2025).
220. *Willems C.* TR-HGI-2012-002 CXPInspector: Hypervisor-Based, Hardware-Assisted System Monitoring / C. Willems, R. Hund, T. Holz. — URL: https://www.semanticscholar.org/paper/TR-HGI-2012-002-CXPInspector-%3A-Hypervisor-Based-%2C-Willems-

Hund/e0b6046627cc708579df6a8091136865e0d8294c (дата обращения: 29.11.2025).
221. *Willems C.* Hypervisor-Based, Hardware-Assisted System Monitoring / C. Willems, R. Hund, T. Holz: URL: https://www.virusbulletin.com/uploads/pdf/conference_slides/2013/WillemsHund-VB2013.pdf (дата обращения: 29.11.2025).
222. Reliability and Security Monitoring of Virtual Machines Using Hardware Architectural Invariants / C. Pham, Z. Estrada, P. Cao [et al.] // Proceedings of the International Conference on Dependable Systems and Networks. — 2014. — URL: https://www.researchgate.net/publication/277518418_Reliability_and_Security_Monitoring_of_Virtual_Machines_Using_Hardware_Architectural_Invariants (дата обращения: 29.11.2025).
223. *Zhu B.* Monitor Trap Flag (MTF) Usage in EPT-based Guest Physical Memory Monitoring / B. Zhu. — URL: http://hypervsir.blogspot.com/2014/11/monitor-trap-flag-mtf-usage-in-ept.html (дата обращения: 29.11.2025).
224. *Tanda S.* Memory Monitor of Read, Write, and execute access / S. Tanda. — URL: https://github.com/tandasat/MemoryMon/tree/rwe_cdfs (дата обращения: 29.11.2025).
225. *Tanda S.* Monitoring & controlling kernel-mode events by HyperPlatform / S. Tanda, I. Korkin // Paper presented at the REcon conference, Montreal, Canada. — URL: recon.cx/2016/talks/Monitoring-and-controlling-kernel-mode-events-by-HyperPlatform.html (дата обращения: 29.11.2025).
226. *Tanda S.* User-mode program parsing logs created by HyperPlatform / S. Tanda. — URL: https://github.com/tandasat/hyperplatform_log_parser (дата обращения: 29.11.2025).
227. *Tanda S.* Monitor Device Driver Interfaces (DDIMon) / S. Tanda. — URL: https://github.com/tandasat/DdiMon/tree/master/DdiMon (дата обращения: 29.11.2025).
228. *Tanda S.* Elevation of privilege detector (EoPMon) / S. Tanda. — URL: https://github.com/tandasat/EopMon/tree/master/EopMon (дата обращения: 29.11.2025).
229. *Tanda S.* Hypervisor based tool for monitoring system register accesses / S. Tanda. — URL: https://github.com/tandasat/GuardMon/tree/master/GuardMon (дата обращения: 29.11.2025).
230. *Tanda S.* An exploit for CVE-2014-0816s / S. Tanda. — URL: https://github.com/tandasat/CVE-2014-0816 (дата обращения: 29.11.2025).
231. *Malliaros S.* Protecting sensitive information in the volatile memory from disclosure attacks / S. Malliaros, C. Ntantogian, C. Xenakis //

2016 11th International Conference on Availability, Reliability and Security (ARES). — 2016. — URL: https://ieeexplore.ieee.org/document/7784634 (дата обращения: 29.11.2025).
232. Thwarting Memory Disclosure with Efficient Hypervisor-enforced Intra-domain Isolation / Y. Liu, T. Zhou, K. Chen [et al.] // Proceedings of the 22d ACM SIGSAC Conference on Computer and Communications Security. — URL: https://www.researchgate.net/publication/286109587_Thwarting_Memory_Disclosure_with_Efficient_Hypervisor-enforced_Intra-domain_Isolation (дата обращения: 29.11.2025).
233. You Can Type, but You Can't Hide: A Stealthy GPU-based Keylogger / E. Ladakis, L. Koromilas, G. Vasiliadis. — URL: https://www.semanticscholar.org/paper/You-Can-Type%2C-but-You-Can't-Hide%3A-A-Stealthy-Ladakis-Koromilas/3a998058f7e784836364ac3d93818619d5998239 (дата обращения: 30.11.2025).
234. *Tanda S.* Proof-of-concept code disables PatchGuard (DisPG) / S. Tanda. — URL: https://github.com/tandasat/PgResarch/tree/master/DisPG/DisPG/ (дата обращения: 30.11.2025).
235. *Tanda S.* MemoryMon: Analyzing Turla Rootkit / S. Tanda. — URL: https://www.youtube.com/watch?v=O5_ocsplrfA (дата обращения: 30.11.2025).
236. *Adler B.* Windows Kernel Exploitation via Vulnerable Drivers / B. Adler // RIT Computing Security Blog. — URL: https://ritcsec.wordpress.com/2017/05/17/windows-kernel-exploitation-via-vulnerable-drivers/ (дата обращения: 30.11.2025).
237. NVIDIA Corporation. Security Bulletin: NVIDIA Windows GPU Display Driver contains multiple vulnerabilities in the kernel mode layer (nvlddmkm.sys) handler for DxgkDdiEscape. — URL: https://www.syxsense.com/syxsense-securityarticles/drivers_&_hardware/syx-1002-11662.html?agt=index (дата обращения: 30.11.2025).
238. *Gee H.* HP Conexant Audio Driver Vulnerability / H. Gee. — URL: http://harvey-gee.blogspot.ru/2017/05/hp-conexant-audio-driver-vulnerability.html (дата обращения: 30.11.2025).
239. *Cimpanu C.* Keylogger Found in HP Notebook Keyboard Driver. BleepingComputer / C. Cimpanu. — URL: https://www.bleepingcomputer.com/news/hardware/keylogger-found-in-hp-notebook-keyboard-driver/ (дата обращения: 30.11.2025).
240. *Hung K.-C.* Schneider Electric UnitelWay Device Driver Buffer Overflow / K.-C. Hung // SCADA Hacker. — URL: https://scadahacker.com/vulndb/2011/ics-vuln-schneider-11-277-01.html (дата обращения: 30.11.2025).

241. National Vulnerability Database. CVE-2017-0155 Detail. — URL: https://nvd.nist.gov/vuln/detail/CVE-2017-0155 (дата обращения: 30.11.2025).
242. *Kirda E.* Detecting and Analyzing Kernel-Based Malware / E. Kirda // Security Intelligence. — URL: https://securityintelligence.com/detecting-and-analyzing-kernel-based-malware/ (дата обращения: 30.11.2025).
243. HandleMaster // GitHub. — 2015. — URL: https://github.com/A5-/HandleMaster/tree/master/HandleMaster (дата обращения: 13.11.2025).
244. *Baranov A.* Windows exploitation in 2016 / A. Baranov // ESET. — 2016. — URL: https://cdn1.esetstatic.com/fileadmin/Images/AU/Image/blog/Windows-Exploitation-2016-A4.pdf (дата обращения: 30.11.2025).
245. *Korkin I.* Detect Kernel-Mode Rootkits via Real Time Logging & Controlling Memory Access / I. Korkin, S. Tanda. — URL: https://www.researchgate.net/publication/317040406_Detect_Kernel-Mode_Rootkits_via_Real_Time_Logging_Controlling_Memory_Access (30.11.2025).
246. *Perla E.* A Guide to Kernel Exploitation: Attacking the Core [Text] / E. Perla, M. Oldani. — Burlington: Syngress, 2011. — 464 p.
247. *Buckbee M.* What is Mimikatz? The Beginner's Guide / M. Buckbee // Varonis Blog. — 2023. — URL: https://www.varonis.com/blog/what-is-mimikatz (дата обращения: 30.11.2025).
248. *Oh M.* Hardening Windows 10 with zero-day exploit mitigations / M. Oh, E. Florio. — Microsoft Malware Protection Center. — 2017. — URL: https://vulners.com/mmpc/MMPC:11F96360F6FFA25D4AC7028A2E9CAA9D (дата обращения: 30.11.2025).
249. *Oh J.W.* Duqu 2.0 Win32k Exploit Analysis / J.W. Oh, E. Florio // Virus Bulletin Conference. — 2015. — URL: https://www.virusbulletin.com/uploads/pdf/conference_slides/2015/OhFlorio-VB2015.pdf (дата обращения: 30.11.2025).
250. *McKinley K.* Manually Exploiting MS17-010 / K. McKinley // LMG Security Blog. — 2018. — URL: https://www.lmgsecurity.com/manually-exploiting-ms17-010/ (дата обращения: 30.11.2025).
251. *Rodionov E.* The Evolution of TDL: Conquering x64 / E. Rodionov, A. Matrosov // ESET. — URL: https://web-assets.esetstatic.com/wls/200x/white-papers/The_Evolution_of_TDL.pdf (дата обращения: 30.11.2025).
252. *Rodionov E.* Mind the Gapz: The Most Complex Bootkit Ever Analyzed? / E. Rodionov, A. Matrosov // ESET. — URL: https://

web-assets.esetstatic.com/wls/2013/04/gapz-bootkit-whitepaper.pdf (дата обращения: 30.11.2025).
253. *Korkin I.* Hypervisor-Based Active Data Protection for Integrity and Confidentiality of Dynamically Allocated Memory in Windows Kernel / I. Korkin. — URL: https://www.researchgate.net/publication/325464287_Hypervisor-Based_Active_Data_Protection_for_Integrity_and_Confidentiality_of_Dynamically_Allocated_Memory_in_Windows_Kernel (дата обращения: 30.11.2025).
254. Cyber-Informed Engineering / R. Anderson, J. Benjamin, V. Wright [et al.]. — Idaho National Laboratory, 2017. — URL: https://inldigitallibrary.inl.gov/sites/sti/sti/7323660.pdf (дата обращения: 30.11.2025).
255. *Isakovic H.* A Mixed-Criticality Integration in Cyber-Physical Systems: A Heterogeneous Time-Triggered Architecture on a Hybrid SoC Platform / H. Isakovic, R. Grosu. — URL: https://www.researchgate.net/publication/325090689_A_Mixed-Criticality_Integration_in_Cyber-Physical_Systems_A_Heterogeneous_Time-Triggered_Architecture_on_a_Hybrid_SoC_Platform (30.11.2025).
256. *Mehnen J.* Practical Security Aspects of the Internet of Things / J. Mehnen, H. He, S. Tedeschi, N. Tapoglou. — URL: https://link.springer.com/chapter/10.1007/978-3-319-50660-9_9 (дата обращения: 30.11.2025).
257. *Burgess C.* Germany warns of nation-state cyber espionage threat / C. Burgess // CSO Online. — 2017. — URL: https://www.csoonline.com/article/562415/germany-warns-of-nation-state-cyber-espionage-threat.html (дата обращения: 30.11.2025).
258. *Maggi F.* Rogue Robots : Testing the Limits of an Industrial Robot's Security / F. Maggi, A.M. Zanchettin. — URL: https://www.blackhat.com/docs/us-17/thursday/us-17-Quarta-Breaking-The-Laws-Of-Robotics-Attacking-Industrial-Robots-wp.pdf (дата обращения: 30.11.2025).
259. *Srivastava A.* Kernel Data Integrity Protection via Memory Access Control / A. Srivastava, I. Erete, J.T. Giffin. — URL: https://www.semanticscholar.org/paper/Kernel-Data-Integrity-Protection-via-Memory-Access-Srivastava-Erete/b66ddfc033dd1c5a34c9bd80d024e5f74a44236f (дата обращения: 30.11.2025).
260. *Wang J.* HACS: A Hypervisor-Based Access Control Strategy to Protect Security-Critical Kernel Data / J. Wang, P. Zhao, H. Ma // DEStech Transactions on Computer Science and Engineering. — URL: https://www.semanticscholar.org/paper/HACS%3A-A-Hypervisor-

Based-Access-Control-Strategy-to-Wang-Zhao/91444e88a851c9662608e01822c7a91adf4fc4b6 (дата обращения: 30.11.2025).
261. Driver compatibility with Device Guard in Windows 10 | Microsoft Community Hub. — URL: https://techcommunity.microsoft.com/blog/windows-hardware-certification/driver-compatibility-with-device-guard-in-windows-10/364865 (дата обращения: 30.11.2025).
262. Mitigate threats by using Windows 10 security features. — URL: https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-10/security/threat-protection/overview-of-threat-mitigations-in-windows-10 (дата обращения: 30.11.2025).
263. Windows technical documentation for developers and IT pros. — URL: https://learn.microsoft.com/en-us/windows-hardware (дата обращения: 30.11.2025).
264. *Tian D.* A policy-centric approach to protecting OS kernel from vulnerable LKMs [Text] / D. Tian, X. Xiong, C. Hu, P. Liu // Software: Practice and Experience. — 2018. — Vol. 48, № 6. — P. 1269–1284.
265. *Xiong X.* Practical Protection of Kernel Integrity for Commodity OS from Untrusted Extensions / X. Xiong, D. Tian, P. Liu // Network and Distributed System Security Symposium. — 2011. — URL: https://www.researchgate.net/publication/221655502_Practical_Protection_of_Kernel_Integrity_for_Commodity_OS_from_Untrusted_Extensions (30.11.2025).
266. *Zabrocki A.* Linux Kernel Runtime Guard (LKRG) / A. Zabrocki. — URL: http://openwall.info/wiki/p_lkrg/Main (дата обращения: 30.11.2025).
267. *Field S.* An Introduction to Kernel Patch Protection / S. Field. — URL: https://blogs.msdn.microsoft.com/windowsvistasecurity/2006/08/12/an-introduction-to-kernel-patch-protection/ (дата обращения: 30.11.2025).
268. *Korkin I.* Acceleration of statistical detection of zero-day malware in the memory dump using CUDA-enabled GPU hardware. arXiv / I. Korkin, I. Nesterow. — URL: https://doi.org/10.48550/arXiv.1606.04662 (дата обращения: 30.11.2025).
269. Windows Kernel Opaque Structures — Windows drivers. — URL: https://learn.microsoft.com/en-us/windows-hardware/drivers/kernel/eprocess (дата обращения: 30.11.2025).
270. PLOAD_IMAGE_NOTIFY_ROUTINE callback function (ntddk.h). — URL: https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/ntddk/nc-ntddk-pload_image_notify_routine (дата обращения: 30.11.2025).

271. *Korkin I.* Source code of Hypervisor-based system for Allocated Memory Protection. GitHub repository / I. Korkin. — URL: https://github.com/IgorKorkin/AllMemPro (дата обращения: 30.11.2025).
272. *Desimone J.* Kernel Mode Threats and Practical Defenses / J. Desimone, G. Landau // Proceedings of Black Hat USA 2018. — URL: https://i.blackhat.com/us-18/Thu-August-9/us-18-Desimone-Kernel-Mode-Threats-and-Practical-Defenses.pdf (дата обращения: 30.11.2025).
273. *Shirole S.* Performance Optimizations for Isolated Driver Domains: Thesis submitted to the Faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of Master of Science / S. Shirole // Computer Science & Applications. — Blacksburg, VA, USA, 2014. — URL: https://vtechworks.lib.vt.edu/server/api/core/bitstreams/7c82ca9f-93d9-4206-98dd-0ac1f6ad8092/content (дата обращения: 30.11.2025).
274. *Oh M.* Return of the Kernel Rootkit Malware (on Windows 10) / M. Oh // Proceedings of the BlueHat v18 Security Conference. — Redmond, Washington, USA, 2018. — : URL: https://github.com/ohjeongwook/Publications/blob/master/Return%20of%20the%20kernel%20rootkit%20malware%20(on%20windows%2010).pdf (дата обращения: 30.11.2025).
275. *Corbató F.J.* Introduction and overview of the Multics system / F.J. Corbató, V.A. Vyssotsky // Proceedings of the November 30–December 1, 1965, Fall Joint Computer Conference, Part I (AFIPS '65, Fall, Part I). — Association for Computing Machinery, New York, NY, USA, 1965. — P. 185–196. — URL: https://doi.org/10.1145/1463891.1463912 (дата обращения: 30.11.2025).
276. *Ghosh S.* Windows XP is still the third most popular operating system in the world / S. Ghosh // Business Insider. — 2017. — URL: https://www.businessinsider.com (дата обращения: 30.11.2025).
277. Australian Cyber Security Centre. Hardening Microsoft Windows 10 version 1709 Workstations. — URL: https://arnaudloos.com/assets/files/ACSC-HardeningWin10.pdf (дата обращения: 30.11.2025).
278. *Manès V.* Domain Isolated Kernel: A lightweight sandbox for untrusted kernel extensions / V. Manès, D. Jang, C. Ryu, B. Kang // Computers & Security. — URL: https://www.researchgate.net/publication/322547536_Domain_Isolated_Kernel_A_lightweight_sandbox_for_untrusted_kernel_extensions (дата обращения: 30.11.2025).
279. *Qiang W.* PrivGuard: Protecting Sensitive Kernel Data From Privilege Escalation Attacks / W. Qiang, J. Yang, H. Jin, X. Shi // IEEE Access. — URL: https://www.researchgate.net/

publication/327146316_PrivGuard_Protecting_Sensitive_Kernel_Data_From_Privilege_Escalation_Attacks (30.11.2025).
280. *Brookes S.* ExOShim: preventing memory disclosure using execute-only kernel code [Text] / S. Brookes, R. Denz, M. Osterloh, S. Taylor // Int. J. Inf. Comput. Secur. — 2022. — Vol. 17. — P. 57–82.
281. *Buhren R.* The Threat of Virtualization: Hypervisor-Based Rootkits on the ARM Architecture / R. Buhren, J. Vetter, J. Nordholz // Information and Communications Security: ICICS 2016. Lecture Notes in Computer Science(). — URL: https://doi.org/10.1007/978-3-319-50011-9_29 (дата обращения: 30.11.2025).
282. InkTag: secure applications on an untrusted operating system. ASPLOS proceedings / O.S. Hofmann, S. Kim, A.M. Dunn [et al.] // International Conference on Architectural Support for Programming Languages and Operating Systems. — URL: https://www.researchgate.net/publication/259766268_InkTag_Secure_Applications_on_an_Untrusted_Operating_System (30.11.2025).
283. *Feng X.* BehaviorKI: Behavior Pattern Based Runtime Integrity Checking for Operating System Kernel / X. Feng, Q. Yang, L. Shi, Q. Wang // 2018 IEEE International Conference on Software Quality, Reliability and Security. — URL: https://www.researchgate.net/publication/326853017_BehaviorKI_Behavior_Pattern_Based_Runtime_Integrity_Checking_for_Operating_System_Kernel (30.11.2025).
284. Hypernel: A Hardware-Assisted Framework for Kernel Protection without Nested Paging / D. Kwon, K. Oh, J. Park [et al.] // 2018 55th ACM/ESDA/IEEE Design Automation Conference. — URL: https://www.researchgate.net/publication/325866816_Hypernel_a_hardware-assisted_framework_for_kernel_protection_without_nested_paging (30.11.2025).
285. Meltdown: Reading Kernel Memory from User Space / M. Lipp, M. Schwarz, D. Gruss [et al.] // Proceedings of the 27th USENIX Security Symposium (USENIX Security 18). — 2018. — URL: https://www.researchgate.net/publication/345037959_Meltdown_reading_kernel_memory_from_user_space (дата обращения: 01.12.2025).
286. EPTI: Efficient Defence against Meltdown Attack for Unpatched VMs / Z. Hua, D. Du, Y. Xia [et al.] // USENIX Annual Technical Conference. — URL: https://www.researchgate.net/publication/365019690_Efficient_Protection_Mechanism_for_CPU_Cache_Flush_Instruction_Based_Attacks (01.12.2025).
287. *Ge X.* Sprobes: Enforcing Kernel Code Integrity on the TrustZone Architecture / X. Ge, H. Vijayakumar, T. Jaeger // ArXiv. — URL: https://www.researchgate.net/publication/267515544_

Sprobes_Enforcing_Kernel_Code_Integrity_on_the_TrustZone_Architecture (01.12.2025).
288. *Tian D.* An Online Approach for Kernel-level Keylogger Detection and Defense [Text] / Tian D., Jia X., Chen J., Hu C. // J. Inf. Sci. Eng. — 2017. — Vol. 33. — P. 445–461.
289. *He Y.* TZ-SSAP: Security-Sensitive Application Protection on Hardware-Assisted Isolated Environment / Y. He, X. Zheng, Z. Zhu, G. Shi. // Security and Privacy in Communication Networks: SecureComm 2016. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. — 2017. — Vol. 198. — URL: https://doi.org/10.1007/978-3-319-59608-2_30 (дата обращения: 01.12.2025).
290. kR^X: Comprehensive Kernel Protection against Just-In-Time Code Reuse / M. Pomonis, T. Petsios, A.D. Keromytis [et al.] // Proceedings of the Twelfth European Conference on Computer Systems. — 2017. — URL: https://www.researchgate.net/publication/316176264_kRX_Comprehensive_Kernel_Protection_against_Just-In-Time_Code_Reuse (дата обращения: 01.12.2025).
291. *Pan J.* Digtool: A Virtualization-Based Framework for Detecting Kernel Vulnerabilities / J. Pan, G. Yan, X. Fan // USENIX Security Symposium. — 2017. — URL: https://archive.org/details/youtube-EOhIxpcyAiw (дата обращения: 01.12.2025).
292. SecPod: a Framework for Virtualization-based Security Systems / X. Wang, Y. Chen, Z. Wang [et al.] // USENIX Annual Technical Conference. — 2015. — URL: https://www.researchgate.net/publication/280832276_SecPod_A_Framework_for_Virtualization-based_Security_Systems (дата обращения: 01.12.2025).
293. *Baliga A.* Detecting kernel level rootkits using data structure invariants / A. Baliga, V. Ganapathy, L. Iftode // IEEE Transactions on Dependable and Secure Computing. — 2011. — Vol. 8, № 5. — URL: https://doi.org/10.1109/TDSC.2010.38 (дата обращения: 01.12.2025).
294. *Götzfried J.* HyperCrypt: Hypervisor-Based Encryption of Kernel and User Space / J. Götzfried, N. Dorr, R. Palutke, T. Müller // 2016 11th International Conference on Availability, Reliability and Security (ARES. — URL: https://www.researchgate.net/publication/303601761_RamCrypt_Kernel-based_Address_Space_Encryption_for_User-mode_Processes (дата обращения: 01.12.2025).
295. iKernel: Isolating Buggy and Malicious Device Drivers Using Hardware Virtualization Support / R. Tan, E. Chan, R. Farivar [et al.]. — URL: https://www.researchgate.net/publication/4282155_

iKernel_Isolating_Buggy_and_Malicious_Device_Drivers_Using_Hardware_Virtualization_Support (01.12.2025).
296. *Chen Q.* PrivWatcher: Non-bypassable Monitoring and Protection of Process Credentials from Memory Corruption Attacks / Q. Chen, A.M. Azab, G. Ganesh, P. Ning // Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security. — URL: https://doi.org/10.1145/3052973.3053029 (дата обращения: 01.12.2025).
297. Hypervision Across Worlds: Real-time Kernel Protection from the ARM TrustZone Secure World / A.M. Azab, P. Ning, J. Shah [et al.] // Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. — URL: https://www.researchgate.net/publication/286866420_Hypervision_Across_Worlds_Real-time_Kernel_Protection_from_the_ARM_TrustZone_Secure_World (01.12.2025).
298. *Zhang F.* HyperCheck: A Hardware-Assisted Integrity Monitor [Text] / F. Zhang, J. Wang, K. Sun, A. Stavrou // IEEE Transactions on Dependable and Secure Computing. — 2014. — Vol. 11, № 4. — P. 332–344.
299. *Gadaleta F.* HyperForce: Hypervisor-enForced Execution of Security-Critical Code / F. Gadaleta, N. Nikiforakis, J.T. Mühlberg, W. Joosen // IFIP International Information Security Conference. — 2012. — URL: https://www.researchgate.net/publication/262568710_HyperForce_Hypervisor-enForced_Execution_of_Security-Critical_Code (дата обращения: 01.12.2025).
300. *Han S.* Myth and Truth about Hypervisor-Based Kernel Protector: The Reason Why You Need Shadow-Box / S. Han, J. Kang // Black Hat Asia 2017 Conference Proceedings. — URL: https://pdfs.semanticscholar.org/480a/a489617eed62252375461d3a82ed7e715c3d.pdf (дата обращения: 01.12.2025).
301. *Sun Y.* SIDE: Isolated and efficient execution of unmodified device drivers / Y. Sun, T.-C. Chiueh // Proceedings of the 2013 43rd Annual : IEEE/IFIP International Conference on Dependable Systems and Networks. — URL: https://www.researchgate.net/publication/261211094_SIDE_Isolated_and_efficient_execution_of_unmodified_device_drivers (дата обращения: 01.12.2025).
302. *Korkin I.* Divide et Impera: MemoryRanger Runs Drivers in Isolated Kernel Spaces / I. Korkin // ArXiv. — URL: https://www.researchgate.net/publication/329945797_Divide_et_Impera_MemoryRanger_Runs_Drivers_in_Isolated_Kernel_Spaces (дата обращения: 01.12.2025).

303. *Korkin I.* MemoryRanger Prevents Hijacking FILE_OBJECT Structures in Windows Kernel / I. Korkin // ArXiv. — URL: https://www.researchgate.net/publication/333338049_MemoryRanger_Prevents_Hijacking_FILE_OBJECT_Structures_in_Windows_Kernel (дата обращения: 01.12.2025).
304. *Korkin I.* Windows Kernel Hijacking Is Not an Option: MemoryRanger Comes to the Rescue Again / I. Korkin // J. Digit. Forensics Secur. Law. — URL: https://www.researchgate.net/publication/352373411_Windows_Kernel_Hijacking_Is_Not_an_Option_MemoryRanger_Comes_to_the_Rescue_Again (дата обращения: 01.12.2025).
305. *Korkin I.* Protected Process Light is not Protected: MemoryRanger Fills The Gap Again / I. Korkin. — URL: https://www.researchgate.net/publication/353113395_Protected_Process_Light_is_not_Protected_MemoryRanger_Fills_The_Gap_Again (дата обращения: 01.12.2025).
306. *Korkin I.* MemoryRanger source code / I. Korkin // GitHub repository. — URL: https://github.com/IgorKorkin/MemoryRanger (дата обращения: 01.12.2025).
307. *Korkin I.* MemoryRanger Demos: The Attack and the Attack Prevention / I. Korkin. — URL: https://www.youtube.com/watch?list=PL0Aerbf3kwULpVhoHyjMUeUFLwnvur5iu&v=HNxc-tjy3QA (дата обращения: 01.12.2025).
308. Microsoft CVE-2018-8120: Win32k Elevation of Privilege Vulnerability // Rapid7. — 2024. — URL: https://www.rapid7.com/db/vulnerabilities/msft-cve-2018-8120/ (дата обращения: 01.12.2025).
309. Microsoft Windows: CVE-2018-8611: Windows Kernel Elevation of Privilege Vulnerability // Rapid7. — 2024. — URL: https://www.rapid7.com/db/vulnerabilities/msft-cve-2018-8611/ (дата обращения: 01.12.2025).
310. Microsoft Windows: CVE-2018-8170: Windows Image Elevation of Privilege Vulnerability // Rapid7. — 2024. — URL: https://www.rapid7.com/db/vulnerabilities/msft-cve-2018-8170/ (дата обращения: 01.12.2025).
311. *Golden B.* Using Files in a Driver — Windows drivers / B. Golden // Learn.microsoft.com. — 2021. — URL: https://learn.microsoft.com/en-us/windows-hardware/drivers/kernel/using-files-in-a-driver (дата обращения: 01.12.2025).
312. *Таненбаум Э.С.* Современные операционные системы [Текст] / Э.С. Таненбаум, Х. Бос. — 4-е изд. — СПб.: Издательский дом «Питер». 2024. — 1120 с.

313. EaseFilter. The Windows File System File I/O Model — EaseFilter. — URL: https://www.easefilter.com/kb/File_IO.htm (дата обращения: 01.12.2025).
314. *Russinovich M.* Inside NT's Object Manager. ITPro Today / M. Russinovich. — URL: https://www.itprotoday.com/it-infrastructure/inside-nt-s-object-manager (дата обращения: 01.12.2025).
315. *Nagar R.* Windows NT File System Internals: A Developer's Guide [Text] / R. Nagar. — Cambridge: O'Reilly Media, 1997. — 774 p.
316. *Fernández E.B.* More Patterns for Operating System Access Control / E.B. Fernández, J. Sinibaldi // European Conference on Pattern Languages of Programs. — URL: https://www.researchgate.net/publication/221034452_More_Patterns_for_Operating_System_Access_Control (01.12.2025).
317. *McHoes A.M.* Understanding Operating Systems / A.M. McHoes, I.M. Flynn. — 6th ed. — Boston: Course Technology/Cengage Learning. — URL: https://archive.org/details/understandingope0000mcho (дата обращения: 01.12.2025).
318. *Probert D.* (2004). Windows Kernel Internals: Object Manager & LPC / D. Probert // Advanced Operating Systems Group, Windows Core Operating Systems Division, Microsoft Corporation. — URL: https://www.i.u-tokyo.ac.jp/edu/training/ss/msprojects/data/04-ObjectManagerLPC.pdf (дата обращения: 01.12.2025).
319. *Pistelli D.* AntiMida 1.0 / D. Pistelli. — URL: https://ntcore.com/files/antimida_1.0.htm (дата обращения: 01.12.2025).
320. Fyyre. WalkDirectory function. KernelDetective source code. — URL: https://github.com/Fyyre/kerneldetective/blob/master/module.cpp (дата обращения:01.12.2025).
321. *Abdalhalim A.* A Light on Windows 10's "OBJECT_HEADER->TypeIndex" / A. Abdalhalim. — URL: https://medium.com/@ashabdalhalim/a-light-on-windows-10s-object-header-typeindex-value-e8f907e7073a (дата обращения: 01.12.2025).
322. *Feldman L.* Windows NT: The Next Generation [Text] / L. Feldman. — Carmel, Ind: Sams, 1993. — 400 p.
323. *Yitbarek S.F.* Neverland: Lightweight Hardware Extensions for Enforcing Operating System Integrity. arXiv preprint / S.F. Yitbarek, T. Austin. — URL: http://arxiv.org/abs/1905.05975 (дата обращения: 01.12.2025).
324. CyberArk. GhostHook: Bypassing PatchGuard with Processor Trace-Based Hooking. — URL: https://www.cyberark.com/resources/threat-research-blog/ghosthook-bypassing-patchguard-with-processor-trace-based-hooking (дата обращения: 01.12.2025).

325. *Cimpanu C.* PoC published for new Microsoft PatchGuard (KPP) bypass / C. Cimpanu. — URL: https://therecord.media/poc-published-for-new-microsoft-patchguard-kpp-bypass (дата обращения: 01.12.2025).
326. GitHub — everdox. InfinityHook. — URL: https://github.com/everdox/InfinityHook (дата обращения: 01.12.2025).
327. *Bisson D.* Banking Trojan Banload Uses Signed Malware Driver to Remove Security Software / D. Bisson. — URL: https://www.itsec.ru/news/ghosthook-novaya-besfaylovaya-ugroza-dlia-poiskovikov (дата обращения: 01.12.2025).
328. Cybercrime: Groups Behind "Banload" Banking Malware Implement New Techniques. — URL: https://www.sentinelone.com/labs/cybercrime-groups-behind-banload-banking-malware-implement-new-techniques/ (дата обращения: 01.12.2025).
329. *Cimpanu C.* Ransomware installs Gigabyte driver to kill antivirus products / C. Cimpanu. — URL: https://www.zdnet.com/article/ransomware-installs-gigabyte-driver-to-kill-antivirus-products/ (дата обращения: 01.12.2025).
330. *Harpaz O.* The Nansh0u Campaign — Hackers Arsenal Grows Stronger / O. Harpaz, D. Goldberg. — URL: https://www.akamai.com/blog/security/the-nansh0u-campaign-hackers-arsenal-grows-stronger (дата обращения: 01.12.2025).
331. *Alibasa B.* Hackers Infect 50,000 Servers With Sophisticated Crypto Mining Malware / B. Alibasa. — URL: https://www.coindesk.com/markets/2019/06/03/hackers-infect-50000-servers-with-sophisticated-crypto-mining-malware (дата обращения: 01.12.2025).
332. *Michael J.* Screwed Drivers — Signed, Sealed, Delivered / J. Michael, M. Shkatov. — URL: https://eclypsium.com/wp-content/uploads/Screwed-Drivers.pdf (дата обращения: 01.12.2025).
333. From alert to driver vulnerability: Microsoft Defender ATP investigation unearths privilege escalation flaw. — URL: https://www.microsoft.com/en-us/security/blog/2019/03/25/from-alert-to-driver-vulnerability-microsoft-defender-atp-investigation-unearths-privilege-escalation-flaw/ (дата обращения: 01.12.2025).
334. Access Token Manipulation, Technique T1134 — Enterprise // MITRE ATT&CK®. — URL: https://attack.mitre.org/techniques/T1134/ (дата обращения: 01.12.2025).
335. *Bisht S.* Understanding and Abusing Access Tokens — Part II. Medium / S. Bisht. — URL: https://securitytimes.medium.com/understanding-and-abusing-access-tokens-part-ii-b9069f432962.

336. Sysnap. Hijacking kernel handle // 0tutorials. — 2011. — URL: http://0tutorials.blogspot.com/2011/08/hijacking-kernel-handle.html (дата обращения: 01.12.2025).
337. *Akhmetshyn A.* DarthTon/Blackbone / A. Akhmetshyn // GitHub. — 2013. — URL: https://github.com/DarthTon/Blackbone (дата обращения: 01.12.2025).
338. *Probert D.* Windows Kernel Architecture Internals / D. Probert // MSRA/UR Workshop — Beijing, China. — URL: https://repo.zenk-security.com/Linux%20et%20systemes%20d.exploitations/Windows%20Kernel%20Architecture%20Internals.pdf (дата обращения: 01.12.2025).
339. *Schreiber S.* Undocumented Windows 2000 Secrets: A Programmer's Cookbook [Text] / S. Schreiber. — Boston: Addison-Wesley, 2001. — 592 p.
340. *Monnappa K.A.* Learning Malware Analysis: Explore the concepts, tools, and techniques to analyze and investigate Windows malware [Text] / K.A. Monnappa. — Birmingham: Packt Publishing, 2018. — 510 p.
341. CodeMachine — Article — Windows Object Headers // CodeMachine. — URL: https://codemachine.com/articles/object_headers.html (дата обращения: 02.12.2025).
342. *Suma G.S.* A novel methodology for windows 7 × 64 memory forensics / G.S. Suma, D. Sulekha, K.L. Thomas. — URL: https://www.semanticscholar.org/paper/Forensic-reconstruction-of-executables-from-Windows-Dija-Suma/087ab55babfca0b1978407f45dc38bc6303851ed (дата обращения: 02.12.2025).
343. *Akhmetshyn A.* Blackbone/src/BlackBoneDrv/Routines.c at master / A. Akhmetshyn // GitHub. — URL: https://github.com/DarthTon/Blackbone/blob/master/src/BlackBoneDrv/Routines.c (дата обращения: 02.12.2025).
344. ReactOS. ExUnlockHandleTableEntry. ReactOS Kernel. — URL: https://doxygen.reactos.org/de/d51/ntoskrnl_2ex_2handle_8c_source.html#l00887 (дата обращения: 02.12.2025).
345. WRK. ExUnlockHandleTableEntry. The Windows Research Kernel: Retrieved from https://github.com/Aekras1a/Labs/blob/9c9121da3fcc34f840a3f67e14fcc2a76d4aa053/Labs/WRK/base/ntos/ex/handle.c (дата обращения: 02.12.2025).
346. Microsoft Corporation. NtfsStru.h. Модуль Windows NT Kernel Source Code. — URL: https://github.com/ZoloZiak/WinNT4/blob/master/private/ntos/cntfs/ntfsstru.h (дата обращения: 02.12.2025).
347. MSDN. FsRtlGetPerStreamContextPointer Macro. Programming reference for Windows Driver Kit. — URL: https://docs.microsoft.

com/en-us/windows-hardware/drivers/ddi/ntifs/nf-ntifs-fsrtlgetperstreamcontextpointer (дата обращения: 02.12.2025).
348. MSDN. FSRTL_COMMON_FCB_HEADER structure. Programming reference for Windows Driver Kit. — URL: https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/ntifs/ns-ntifs-_fsrtl_common_fcb_header (дата обращения: 02.12.2025).
349. MSDN. FSRTL_ADVANCED_FCB_HEADER structure. Programming reference for Windows Driver Kit. — URL: https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/ntifs/ns-ntifs-_fsrtl_advanced_fcb_header (дата обращения: 02.12.2025).
350. *A. Ismail M.* An Investigation into Access Control in Various Types of Operating Systems / M. A. Ismail, H. Aboelseoud M., M. Senousy // International Journal of Computer Applications. — 2014. — Vol. 98, № 10. — URL: https://research.ijcaonline.org/volume98/number10/pxc3897454.pdf (дата обращения: 02.12.2025).
351. *Johnson M.H.* Windows System Programming [Text] / M.H. Johnson. — 4th ed. — Massachusetts, US: Addison-Wesley Professional, 2015. — 537 p.
352. *Russinovich M.* Windows NT Architecture, Part 2. ItProToday / M. Russinovich. — URL: https://www.itprotoday.com/compute-engines/windows-nt-architecture-part-2 (дата обращения: 02.12.2025).
353. *Stallings W.* Operating System Security [Text] / W. Stallings // Computer Security Handbook. — John Wiley & Sons, Ltd, 2012. — P. 24.1–24.21.
354. *Barta C.* Access Token Stealing on Windows / C. Barta. — URL: https://www.offsec.com/awe/AWEPAPERS/Token_stealing.pdf (дата обращения: 02.12.2025).
355. *Oh M.* Detecting and mitigating elevation-of-privilege exploit for CVE-2017-0005 / M. Oh // Microsoft Security Blog. — 2017. — URL: https://www.microsoft.com/en-us/security/blog/2017/03/27/detecting-and-mitigating-elevation-of-privilege-exploit-for-cve-2017-0005 (дата обращения: 02.12.2025).
356. *Singh A.* Advanced detection of token impersonation attacks in modern operating systems [Text] / A. Singh, J. Kaplan, J. Feng, D. Sanossian // Proceedings of the 26th Annual Network and Distributed System Security Symposium (NDSS 2019). — San Diego, USA: Internet Society, 2019. — P. 1–15.
357. *Bui J.* Understanding and Defending Against Access Token Theft: Finding Alternatives to winlogon.exe / J. Bui. — URL: https://posts.specterops.io/understanding-and-defending-against-access-token-

theft-finding-alternatives-to-winlogon-exe-80696c8a73b (дата обращения: 02.12.2025).
358. Microsoft Windows Security | Microsoft Press Store. — URL: https://www.microsoftpressstore.com/articles/article.aspx?p=2228450&seqNum=2 (дата обращения: 02.12.2025).
359. *Blaauwendraad B.* Using Mimikatz' driver, Mimidrv, to disable Windows Defender in Windows / B. Blaauwendraad, T. Ouddeken, C.V. Bockhaven. — URL: https://rp.os3.nl/2019-2020/p61/report.pdf (02.12.2025).
360. *Delpy B.* A little tool to play with Windows security / B. Delpy. — URL: https://github.com/windwang/mimikatz-1 (дата обращения: 02.12.2025).
361. *Steube J.* Hashcat — advanced password recovery / J. Steube, G. Gristina. — URL: https://hashcat.net/hashcat/ (дата обращения: 02.12.2025).
362. *Forshaw J.* Microsoft Windows PPL Process Injection Privilege Escalation / J. Forshaw // Packet Storm. — URL: https://packetstormsecurity.com/files/143946/Microsoft-Windows-PPL-Process-Injection-Privilege-Escalation.html (дата обращения: 02.12.2025).
363. Notscimmy. pplib: Elevate processes to Protected Processes. — URL: https://github.com/notscimmy/pplib (дата обращения: 02.12.2025).
364. *Lavrijsen M.* Protected Processes Light Killer / M. Lavrijsen // GitHub. — 2019. — URL: https://github.com/Mattiwatti/PPLKiller (дата обращения: 02.12.2025).
365. FireF0X. Multi-purpose proof-of-concept tool based on CPU-Z CVE-2017-15303 // GitHub. — 2018. — URL: https://github.com/hfiref0x/Stryker (дата обращения: 02.12.2025).
366. *Bui H.* Weaponizing Gigabyte driver for priv escalation and bypass PPL / H. Bui // GitHub. — 2019. — URL: https://github.com/hoangprod/DanSpecial (дата обращения: 02.12.2025).
367. Harakirinox. Bypass PPL (Protected Process Light) from user-mode using a vulnerable driver // UnKnoWnCheaTs — Multiplayer Game Hacks and Cheats. — URL: https://www.unknowncheats.me/forum/anti-cheat-bypass/262766-bypass-ppl-protected-process-light-user-mode-using-vulnerable-driver.html (дата обращения: 02.12.2025).
368. *Van Impe K.* Mimikatz and hashcat in practice / K. Van Impe. — URL: https://www.vanimpe.eu/2019/03/07/mimikatz-and-hashcat-in-practice/ (дата обращения: 02.12.2025).
369. Use attack surface reduction rules to prevent malware infection // Microsoft Learn. URL: https://learn.microsoft.com/en-us/defender-endpoint/attack-surface-reduction (дата обращения: 02.12.2025).

370. Protected Processes in Windows Vista // CSDN. — URL: https://blog.csdn.net/linhanshi/article/details/805038 (дата обращения: 02.12.2025).
371. Astr0baby. Unloading AV from Windows 10 // WordPress. — 2017. — URL: https://astr0baby.wordpress.com/2017/09/11/unloading-av-from-windows-10/ (дата обращения: 02.12.2025).
372. *Son D.* Preventing Mimikatz steal Windows system password / D. Son // Cybersecurity News. — URL: https://securityonline.info/prevent-mimikatz/ (дата обращения: 02.12.2025).
373. *Ionescu A.* Unreal Mode: Breaking Protected Processes / A. Ionescu // NoSuchCon. — 2014. — URL: http://publications.alex-ionescu.com/NoSuchCon/NoSuchCon%202014%20-%20Unreal%20Mode%20-%20Breaking%20Protected%20Processes.pdf (дата обращения: 02.12.2025).
374. Achilles. The Birth of a Process Part-1 // Medium. — URL: https://medium.com/@Achilles8284/the-birth-of-a-process-part-1-bfb4fdac070e (дата обращения: 02.12.2025).
375. Bitdefender. Bitdefender GravityZone Administrator's Guide. — URL: https://www.bitdefender.co.th/resources/GravityZoneEnterprise/Current/Documentation/en_US/Bitdefender_GravityZone_AdministratorsGuide_7_enUS.pdf (дата обращения: 02.12.2025).
376. Cisco. AMP for Endpoints User Guide. — URL: https://cloudmanaged.ca/wp-content/uploads/2020/05/AMP-for-Endpoints-User-Guide.pdf (дата обращения: 17.12.2024).
377. ESET. Easy Protection: ESET Internet Security & ESET Smart Security Premium // Форум ESET. — URL: https://forum.eset.com/topic/19683-easy-protection/ (дата обращения: 17.12.2024).
378. KaspeskyLab. About Protected Process Light (PPL) technology for Windows. — URL: https://support.kaspersky.com/us/common/windows/13905#:~:text=Protected%20Process%20Light%20(PPL)%20technology%20is%20used%20for%20controlling%20and,Shutdown (дата обращения: 02.12.2025).
379. *Hopkins M.* Overview of Early Launch AntiMalware — Windows drivers / M. Hopkins, N. Bassett, E. Graff. — URL: https://learn.microsoft.com/en-us/windows-hardware/drivers/install/early-launch-antimalware (дата обращения: 02.12.2025).
380. Microsoft Defender Antivirus in Windows Overview — Microsoft Defender for Endpoint. — URL: https://learn.microsoft.com/en-us/defender-endpoint/microsoft-defender-antivirus-windows (дата обращения: 02.12.2025).
381. Configure added LSA protection. — URL: https://learn.microsoft.com/en-us/windows-server/security/credentials-protection-and-

management/configuring-additional-lsa-protection (дата обращения: 02.12.2025).
382. NtOpCode. Anatomy of the Process Environment Block (PEB) (Windows Internals) // Opcode. — 2018. — URL: https://ntopcode.wordpress.com/2018/02/26/anatomy-of-the-process-environment-block-peb-windows-internals/ (дата обращения: 02.12.2025).
383. ZwQueryInformationProcess function — Win32 apps // Microsoft Learn, 2021. — URL: https://learn.microsoft.com/en-us/windows/win32/procthread/zwqueryinformationprocess (дата обращения: 02.12.2025).
384. *Labro C.* Debugging Protected Processes / C. Labro // itm4n's blog. — 2022. — URL: https://itm4n.github.io/debugging-protected-processes/ (дата обращения: 02.12.2025).
385. *Ionescu A.* The Evolution of Protected Processes Part 1: Pass-the-Hash Mitigations in Windows 8.1 / A. Ionescu. — URL: https://www.alex-ionescu.com/the-evolution-of-protected-processes-pass-the-hash-mitigations-in-windows-8-1/ (дата обращения: 02.12.2025).
386. *Ionescu A.* The Evolution of Protected Processes Part 2: Exploit/Jailbreak Mitigations, Unkillable Processes and Protected Services / A. Ionescu. — URL: https://www.alex-ionescu.com/wip-draft-the-evolution-of-protected-processes-part-2-exploitjailbreak-mitigations-unkillable-processes-and-protected-services/?utm_source=chatgpt.com (дата обращения: 02.12.2025).
387. *Aquilino B.I.* Relevance of Security Features Introduced in Modern Windows OS / B.I. Aquilino. — URL: https://aaltodoc.aalto.fi/bitstream/handle/123456789/38990/master_Aquilino_Broderick_2019.pdf (дата обращения: 17.12.2024).
388. *Ionescu A.* Unknown Known DLLs and other Code Integrity Trust Violations: Breaking Signature Guarantees in Windows / A. Ionescu, J. Forshaw // REcon. — 2018. — URL: http://publications.alex-ionescu.com/Recon/Recon%202018%20-%20Unknown%20Known%20DLLs%20and%20other%20code%20integrity%20trust%20violations.pdf (дата обращения: 02.12.2025).
389. *Tworek G.* Creating Protection Service / G. Tworek // Github Source Code. — 2020. — URL: https://github.com/gtworek/PSBits/blob/master/Services/SpoilService.c (дата обращения: 02.12.2025).
390. Path, Experimenting with Protected Processes and Threat-Intelligence. — URL: https://blog.tofile.dev/2020/12/16/elam.html (дата обращения: 02.12.2025).
391. b33f. FuzzySecurity/IBM-RedCON-2020 // GitHub. — URL: https://github.com/FuzzySecurity/IBM-RedCON-2020 (дата обращения: 02.12.2025).

392. DarthTon. How-To Make Your External Hack 'Undetectable' (BattlEye Bypass) // UnKnoWnCheaTs — Multiplayer Game Hacks and Cheats. — 2014. — URL: https://www.unknowncheats.me/forum/904106-post5.html (дата обращения: 02.12.2025).
393. *Bui H.* Weaponizing vulnerable driver for privilege escalation — Gigabyte Edition! / H. Bui // Medium. — 2019. — URL: https://medium.com/@fsx30/weaponizing-vulnerable-driver-for-privilege-escalation-gigabyte-edition-e73ee523598b (дата обращения: 02.12.2025).
394. *Mulder J.* Mimikatz Overview, Defenses and Detection / J. Mulder // SANS Institute. — 2016. — URL: https://www.sans.org/white-papers/36780/ (дата обращения: 02.12.2025).
395. *Hand M.* Mimidrv In Depth: Exploring Mimikatz's Kernel Driver / M. Hand // Medium. — 2021. — URL: https://rp.os3.nl/2019-2020/p61/report.pdf (дата обращения: 02.12.2025).
396. *Chester A.* Exploring Mimikatz — Part 1 — Wdigest / A. Chester. — URL: https://blog.xpnsec.com/exploring-mimikatz-part-1/ (дата обращения: 02.12.2025).
397. *Chester A.* Exploring Mimikatz — Part 2 — SSP / A. Chester. — URL: https://blog.xpnsec.com/exploring-mimikatz-part-2/ (дата обращения: 02.12.2025).
398. *Patil D.N.* Analysis of Windows In-Memory Structures for Extracting Digital Evidence / D.N. Patil, B.B. Meshram // International Journal of Computer Science and Software Engineering. — 2017. — Vol. 6, № 6. — URL: https://search.proquest.com/openview/e1620fb9ff42b266bef18e27c78e6e84/1?pq-origsite=gscholar&cbl=2044552 (дата обращения: 02.12.2025).
399. *Gkatziourlis P.* Preventing Mimikatz Attacks / P. Gkatziourlis // Blue Team. — URL: https://freedium.cfd/ed283e7ebdd5 (дата обращения: 02.12.2025).
400. *Delpy B.* Mimikatz latest release / B. Delpy // GitHub. — 2022. — URL: https://github.com/gentilkiwi/mimikatz/releases/latest (дата обращения: 02.12.2025).
401. *Delpy B.* Antivirus software thinks mimikatz is malware / B. Delpy // GitHub. — 2016. — URL: https://github.com/gentilkiwi/mimikatz/issues/55 (дата обращения: 02.12.2025).
402. *Altermatt D.* How to spot PPL manipulations in Windows / D. Altermatt // SCIP. — URL: https://www.scip.ch/en/?labs.20200116 (дата обращения: 02.12.2025).
403. *Schneider M.* Local Security Authority. Keeping Secrets Safe / M. Schneider // SCIP. — URL: https://www.scip.ch/en/?labs.20191121 (дата обращения: 02.12.2025).

404. How to Obtain SeDebugPrivilege when Debug Program Policy is Enabled // Windows OS Hub. — URL: https://woshub.com/obtain-sedebugprivilege-debug-program-policy-enabled/ (дата обращения: 02.12.2025).
405. *Kennedy D.* Dumping Wdigest Creds With Meterpreter Mimikatz/ Kiwi In Windows 8.1 / D. Kennedy. — URL: https://www.trustedsec.com/blog/dumping-wdigest-creds-with-meterpreter-mimikatzkiwi-in-windows-8-1/ (дата обращения: 02.12.2025).
406. Restricted RDP for Admin (RestrictedAdmin) // MPCA Solutions GmbH. — URL: https://blog.ahasayen.com/restricted-admin-mode-for-rdp/ (дата обращения: 02.12.2025).
407. *Matarazzo P.* Credential Guard overview / P. Matarazzo // Microsoft Learn. — URL: https://learn.microsoft.com/en-us/windows/security/identity-protection/credential-guard/ (дата обращения: 02.12.2025).
408. *Dibley J.* Secure Privileged Credentials with Windows Defender Credential Guard / J. Dibley // Netwrix Blog. — URL: https://blog.netwrix.com/2023/02/06/using-windows-defender-credential-guard-to-protect-privileged-credentials/ (дата обращения: 02.12.2025).
409. *Korkin I.* YouTube channel / I. Korkin. — URL: https://www.youtube.com/c/IgorKorkin (дата обращения: 02.12.2025).
410. *Oki K.* A PoC for Mhyprot2.sys vulnerable driver that allows read/write memory in kernel/user via unprivileged user process / K. Oki. — URL: https://github.com/kkent030315/evil-mhyprot-cli (дата обращения: 02.12.2025).
411. Hfiref0x. Universal PatchGuard and Driver Signature Enforcement Disable. — URL: https://github.com/hfiref0x/UPGDSED (дата обращения: 02.12.2025).
412. *Lavrijsen M.* Patching the Windows boot manager, boot loader and kernel at boot time / M. Lavrijsen. — URL: https://github.com/Mattiwatti/EfiGuard (дата обращения: 02.12.2025).
413. *Allievi A.* Introducing Kernel Data Protection, a new platform security technology for preventing data corruption. Security Kernel Core Team / A. Allievi. — URL: https://www.microsoft.com/security/blog/2020/07/08/introducing-kernel-data-protection-a-new-platform-security-technology-for-preventing-data-corruption (дата обращения: 02.12.2025).
414. *Bulazel A.* Windows Offender: Reverse Engineering Windows Defender's Antivirus Emulator / A. Bulazel // BlackHat USA. — 2018. — URL: https://i.blackhat.com/us-18/Thu-August-9/us-

18-Bulazel-Windows-Offender-Reverse-Engineering-Windows-Defenders-Antivirus-Emulator.pdf (дата обращения: 02.12.2025).
415. *Dekel K.* Illusion Gap — Antivirus Bypass Part 1 / K. Dekel. — URL: https://www.cyberark.com/resources/threat-research-blog/illusion-gap-antivirus-bypass-part-1 (дата обращения: 02.12.2025).
416. Narib. Dissecting the Windows Defender Driver — WdFilter. Series 1–4. — URL: https://www.n4r1b.com/posts/2020/01/dissecting-the-windows-defender-driver-wdfilter-part-1 (дата обращения: 02.12.2025).
417. IredTeam. Evading Windows Defender with 1 Byte Change. — URL: https://www.ired.team/offensive-security/defense-evasion/evading-windows-defender-using-classic-c-shellcode-launcher-with-1-byte-change (дата обращения: 02.12.2025).
418. *Maude J.* TrickBot Attack Chain: Deconstructed & Mitigated / J. Maude // BeyondTrust. — 2021. — URL: https://www.beyondtrust.com/blog/entry/trickbot-attack-chain-deconstructed-mitigated (дата обращения: 02.12.2025).
419. *Cocomazzi A.* Hide and Seek | New Zloader Infection Chain Comes with Improved Stealth And Evasion Mechanisms / A. Cocomazzi, A. Pirozzi // SentinelLABS Research Team. — URL: https://assets.sentinelone.com/sentinellabs/SentinelLabs-Zloader (дата обращения: 02.12.2025).
420. *Abrams L.* IObit forums hacked to spread ransomware to its members. BleepingComputer / L. Abrams. — URL: https://www.bleepingcomputer.com/news/security/iobit-forums-hacked-to-spread-ransomware-to-its-members (дата обращения: 02.12.2025).
421. *Lakshmanan R.* Ransomware Hackers Using AuKill Tool to Disable EDR Software Using BYOVD Attack / R. Lakshmanan. — URL: https://thehackernews.com/2023/04/ransomware-hackers-using-aukill-tool-to.html (дата обращения: 02.12.2025).
422. *Blaauwendraad B.* Using Mimikatz' driver, Mimidrv, to disable Windows Defender in Windows / B. Blaauwendraad, T. Ouddeken, C. Bockhaven. — URL: https://rp.os3.nl/2019-2020/p61/report.pdf (дата обращения: 02.12.2025).
423. Bs. Removing Kernel Callbacks Using Signed Drivers. — URL: https://br-sn.github.io/Removing-Kernel-Callbacks-Using-Signed-Drivers (дата обращения: 02.12.2025).
424. Yasser. A tool to kill antimalware protected processes. — URL: https://github.com/Yaxser/Backstab (дата обращения: 02.12.2025).
425. Shaw J. Process Herpaderping — Windows Defender Evasion / J. Shaw. — URL: https://pentestlaboratories.com/2021/01/18/

process-herpaderping-windows-defender-evasion (дата обращения: 02.12.2025).
426. *Lagrasta F.* The dying knight in the shiny armour. Killing Defender through NT symbolic links redirection while keeping it unbothered / F. Lagrasta. — URL: https://aptw.tf/2021/08/21/killing-defender.html (дата обращения: 02.12.2025).
427. *Sandker C.* A Windows Authorization Guide / C. Sandker. — URL: https://csandker.io/2018/06/14/AWindowsAuthorizationGuide.html (дата обращения: 02.12.2025).
428. *Laiho S.* How Windows Security really works / S. Laiho // Workplace Ninja User Group Switzerland. — URL: https://res.cloudinary.com/digicomp-academy-ag/images/Digiblog-exasys/20329-How-Windows-Security-really-works-CMCE1602-Sami-Laiho/20329-How-Windows-Security-really-works-CMCE1602-Sami-Laiho.pdf (дата обращения: 02.12.2025).
429. *Utin M.* Malicious Hypervisor Threat — Phase Two: How to Catch the Hypervisor / M. Utin // DeepSec. — URL: https://infocondb.org/con/deepsec/deepsec-2016-ten/malicious-hypervisor-threat-phase-two-how-to-catch-the-hypervisor (дата обращения: 02.12.2025).
430. *Naraine R.* Vitriol Rootkit to Demo at MS BlueHat Hacker Summit / R. Naraine. — URL: https://www.eweek.com/security/vitriol-rootkit-to-demo-at-ms-bluehat-hacker-summit/ (дата обращения: 02.12.2025).
431. *Fannon C.* An Analysis of Hardware-assisted Virtual Machine based Rootkits. (Master's thesis) Naval Postgraduate School / C. Fannon. — URL: http://hdl.handle.net/10945/42621 (дата обращения: 03.12.2025).
432. *Sierra-Arriaga F.* Security Issues and Challenges for Virtualization Technologies / F. Sierra-Arriaga, R. Branco, B. Lee // ACM Computing Surveys. — URL: https://www.researchgate.net/publication/341628086_Security_Issues_and_Challenges_for_Virtualization_Technologies (дата обращения: 03.12.2025).
433. *Athreya M.B.* (2010). Subverting Linux on-the-fly using hardware virtualization technology (Master's thesis) / M.B. Athreya // Georgia Institute of Technology, School of Electrical and Computer Engineering. — URL: https://repository.gatech.edu/server/api/core/bitstreams/a68b3708-7e02-4941-b9a8-ccb487ee959f/content (дата обращения: 03.12.2025).
434. *Fritsch H.* Analysis and detection of virtualization-based rootkits / H. Fritsch //Analyse und Erkennung von virtualisierungsbasierten Rootkits. Bachelor's thesis, Technische Universität München, Fakultät

für Informatik. — URL: https://www.nm.ifi.lmu.de/pub/Fopras/frit08/PDF-Version/frit08.pdf (дата обращения: 03.12.2025).
435. *Mistry N.R.* Signature based volatile memory forensics: a detection based approach for analyzing sophisticated cyber attacks / N.R. Mistry, M.S. Dahiya // Int. J. Inf. Tecnol. — 2019. — Vol. 11. — URL: https://doi.org/10.1007/s41870-018-0263-4 (03.12.2025).
436. *Medley D.* Virtualization Technology Applied to Rootkit Defense / D. Medley. — URL: https://www.researchgate.net/publication/235185358_Virtualization_Technology_Applied_to_Rootkit_Defense (дата обращения: 03.12.2025).
437. *Myers M.* An Introduction to Hardware-Assisted Virtual Machine (HVM) Rootkits / M. Myers, S. Youndt — URL: https://www.semanticscholar.org/paper/An-Introduction-to-Hardware-Assisted-Virtual-(-HVM-Myers-Youndt/b2e5e191be99cb21e6ce3449478ba5eb29eb419c (дата обращения: 03.12.2025).
438. *Petroni N.L.* 2004. Copilot — a coprocessor-based kernel runtime integrity monitor / N.L. Petroni, T. Fraser, J. Molina, W.A. Arbaugh // Proceedings of the 13th conference on USENIX Security Symposium. — 2004. — Vol. 13. — URL: https://www.researchgate.net/publication/221260490_Copilot-A_Coprocessor-based_Kernel_Runtime_Integrity_Monitor (дата обращения: 03.12.2025).
439. *Egele M.* A survey on automated dynamic malware-analysis techniques and tools / M. Egele, T. Scholte, E. Kirda, C. Krügel // ACM Comput. Surv. — 2012. Vol. 44. — URL: https://www.researchgate.net/publication/254007303_A_Survey_on_Automated_Dynamic_Malware-Analysis_Techniques_and_Tools (дата обращения: 03.12.2025).
440. The Evolution of Volatile Memory Forensics / H. Nyholm, K. Monteith, S. Lyles [et al.] // J. Cybersecur. Priv. — 2022. — Vol. 2. URL: https://www.researchgate.net/publication/362147648_The_Evolution_of_Volatile_Memory_Forensics (дата обращения: 03.12.2025).
441. *Сорокина С.И.* Программирование драйверов и систем безопасности [Текст] / С.И. Сорокина, А.Ю. Тихонов, А.Ю. Щербаков. — СПб.: БХВ-Петербург, 2003. — 256 с.
442. *Becher M.* FireWire: all your memory are belong to us / M. Becher, M. Dornseif, C. Klein. — URL: https://www.researchgate.net/publication/236221577_FireWire_all_your_memory_are_belong_to_us (дата обращения: 03.12.2025).
443. *Latzo T.* All your System Memory are belong to us: From Low-Level Memory Acquisition to High-Level Forensic Event Reconstruction / T. Latzo. — URL: https://www.researchgate.net/

publication/353943452_All_your_System_Memory_are_belong_to_us_From_Low-Level_Memory_Acquisition_to_High-Level_Forensic_Event_Reconstruction (дата обращения: 03.12.2025).
444. *Rabin R.* Direct Memory Access (DMA) attacks. Risks, techniques, and mitigations in hardware hacking / R. Rabin // Pen Test Partners. — URL: https://www.pentestpartners.com/security-blog/direct-memory-access-dma-attacks-risks-techniques-and-mitigations-in-hardware-hacking/ (дата обращения: 03.12.2025).
445. *Fisher-Ogden J.* Hardware Support for Efficient Virtualization / J. Fisher-Ogden. — URL: https://course.ece.cmu.edu/~ece845/sp16/docs/hardwareVirt.pdf (дата обращения: 03.12.2025).
446. *Grand J.* Method and apparatus for preserving computer memory using expansion card (U.S. Patent No. 7,181,560) / J. Grand, B. Carrier. — https://patents.google.com/patent/US7181560B1/en (дата обращения: 03.12.2025).
447. *Morabito D.B.* Detecting Hardware-assisted Hypervisor Rootkits within Nested Virtualized Environments / D.B. Morabito // Theses and Dissertations. — 2012. — URL: https://www.semanticscholar.org/paper/Detecting-Hardware-assisted-Hypervisor-Rootkits-Morabito/7ffe8a29a8e045144281e3ee7f37f7d9dba4100e (дата обращения: 03.12.2025).
448. *Adams K.* BluePill detection in two easy steps / K. Adams // Gimme Hardware/Software Interface. — 2007. — URL: http://x86vmm.blogspot.com/2007/07/bluepill-detection-in-two-easy-steps.html (дата обращения: 03.12.2025).
449. *Barbosa E.* Detection of hardware virtualization rootkits / E. Barbosa // SyScan 2007, COSEINC Advanced Malware Labs. — 2007. — URL: https://www.slideshare.net/slideshow/detecting-virtualized-hardwarerootkitsppt/22177790 (дата обращения: 03.12.2025).
450. Oiaga M. Download Free Hardware Virtualization-Based Windows Rootkit Detector. Softpedia / M. Oiaga. — URL: https://news.softpedia.com/news/Download-Free-Hardware-Virtualization-Based-Windows-Rootkit-Detector-79624.shtml (дата обращения: 03.12.2025).
451. *Луценко А.* Красная пилюля — аппаратная виртуализация в контексте информационной безопасности / А. Луценко // VM Guru. — 2010. — URL: https://vm-guru.com/articles/red-pill-virtualization-security (дата обращения: 03.12.2025).

452. Broadcom Inc. Symantec Endpoint Protection 14. — URL: https://www.broadcom.com/support/security-center/definitions/download/detail?gid=sep14 (дата обращения: 03.12.2025).
453. *Park J-O.* A Study on Detection of Hacking and Malware Codes in Bare Metal Hypervisor for Virtualized Internal Environment of Cloud Service / Park J-O. — URL: http://paper.ijcsns.org/07_book/201310/20131012.pdf (дата обращения: 03.12.2025).
454. *Юзбашев А.В.* Способ обнаружения вложенных мониторов виртуальных машин [Текст] / А.В. Юзбашев // Безопасность информационных технологий. — 2015. — Т. 22, № 1. — С. 117–118.
455. Tamper-Resistant Execution in an Untrusted Operating System Using A Virtual Machine Monitor / H. Chen, F. Zhang, C. Chen [et al.]. — URL: https://ipads.se.sjtu.edu.cn/_media/publications/chaos-ppi-tr.pdf (дата обращения: 03.12.2025).
456. *Nie C.* Dynamic root of trust in trusted computing / C. Nie. — URL: https://www.researchgate.net/publication/228419133_Dynamic_root_of_trust_in_trusted_computing (дата обращения: 03.12.2025).
457. TPM + Internet Virtual Disk + Platform Trust Services = Internet Client. National Institute of Advanced Industrial Science and Technology / K. Suzaki, K. Iijima, T. Yagi [et al.] // IBM Japan. — URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=6d3eefb231803288dbb9fd957779804bcd9bdccb (дата обращения: 03.12.2025).
458. *Wojtczuk R.* Attacking Intel® Trusted Execution Technology / R. Wojtczuk, J. Rutkowska // Invisible Things Lab. — URL: https://invisiblethingslab.com/resources/bh09dc/Attacking%20Intel%20TXT%20-%20paper.pdf?ref=blog.golem.network (дата обращения: 03.12.2025).
459. *Wojtczuk R.* Another Way to Circumvent Intel ® Trusted Execution Technology / R. Wojtczuk, J. Rutkowska, A.I. Tereshkin. — URL: https://www.semanticscholar.org/paper/Another-Way-to-Circumvent-Intel-%C2%AE-Trusted-Execution-Wojtczuk-Rutkowska/b386a777e80fcf6674176686bc77ad4a84f24fff (дата обращения: 03.12.2025).
460. Trusted Platform Module (TPM) Summary, White Paper. — URL: https://trustedcomputinggroup.org/resource/trusted-platform-module-tpm-summary/ (дата обращения: 03.12.2025).
461. *Brossard J.* Hardware Backdooring is practical / J. Brossard // Blackhat Briefings (USA) 2012, Blackhat Briefings, Jul 2012, Las Vegas, United States. — URL: https://hal.science/hal-04672193 (дата обращения: 03.12.2025).

462. *Чекин Р.Н.* Современные угрозы безопасности обработки информации со стороны встроенного программного обеспечения BIOS / Р.Н. Чекин // Доклады ТУСУР. 2010. — № 1-1 (21). — URL: https://cyberleninka.ru/article/n/sovremennye-ugrozy-bezopasnosti-obrabotki-informatsii-so-storony-vstroennogo-programmnogo-obespecheniya-bios (дата обращения: 03.12.2025).
463. *Никитин Е.* Всегда ли на замке? Как обезопасить компьютер модулем доверенной загрузки? / Е. Никитин, В. Шрамко // Системный администратор.— 2010. — Вып. № 2 (2). — URL: https://samag.ru/archive/article/1068 (дата обращения: 03.12.2025).
464. *Марков А.С.* Программный метод обеспечения безопасности загрузки операционной среды [Текст] / А.С. Марков, И.Б. Пугачев // Известия Института инженерной физики. — 2009. — № 1 (11). — С. 7–9.
465. *Kyte I.* Enhanced side-channel analysis method to detect hardware virtualization based rootkits [Text] / I. Kyte, K.P. Zavarsky, D. Lindskog, R. Ruhl // World Congress on Internet Security (WorldCIS-2012), Guelph, ON, Canada.— 2012. — P. 192–201.
466. *Kyte I.* Detection of Hardware Virtualization Based Rootkits by Performance Benchmarking / I. Kyte, P. Zavarsky, D. Lindskog, R. Ruhl. — URL: https://www.semanticscholar.org/paper/Detection-of-Hardware-Virtualization-Based-Rootkits-Kyte-Zavarsky/468760741fcacb07c8742f789ef8a64ec051b8d5 (дата обращения: 03.12.2025).
467. *Pék G.* Towards the automated detection of unknown malware on live systems [Text] / G. Pék, L. Buttyán // 2014 IEEE International Conference on Communications (ICC), Sydney, NSW, Australia. — 2014. — P. 847–852.
468. *Коркин И.Ю.* Метод обнаружения аппаратной виртуализации в компьютерных системах / И.Ю. Коркин, Т.В. Петрова, А.Ю. Тихонов // Безопасность информационных технологий. — 2010. — Т. 17, № 1. — URL: https://www.elibrary.ru/item.asp?id=18812947 (дата обращения: 03.12.2025).
469. *Ferrie P.* Attacks on More Virtual Machine Emulators / P. Ferrie. — URL: https://www.semanticscholar.org/paper/Attacks-on-More-Virtual-Machine-Emulators-Ferrie/a72d98d5a478efa62383a63862fc07dba831c8a5 (дата обращения: 03.12.2025).
470. Improving Virtualization in the Presence of Software Managed Translation Lookaside Buffers / Chang X., Hubertus F., Yi G. [et al.] // SIGARCH Comput. Archit. — 2013. — URL: https://doi.org/10.1145/2508148.2485933 (дата обращения: 03.12.2025).

471. *Garfinkel T.* Compatibility Is Not Transparency: VMM Detection Myths and Realities / T. Garfinkel, K. Adams, A. Warfield, J. Franklin. — URL: https://www.researchgate.net/publication/221150742_Compatibility_Is_Not_Transparency_VMM_Detection_Myths_and_Realities (дата обращения: 03.12.2025).
472. *Rutkowska J.* Virtualization Detection vs. Blue Pill Detection / J. Rutkowska. — URL: https://blog.invisiblethings.org/2007/08/03/virtualization-detection-vs-blue-pill.html (дата обращения: 03.12.2025).
473. *Smith A.J.* A study of replacement algorithms for a virtual-storage computer [Text] / A.J. Smith, J.R. Goodman // IBM Journal of Research and Development. — 1985. — Vol. 19 (3). — P. 244–256.
474. *Patterson D.A.* Computer Organization and Design: The Hardware/Software Interface [Text] / D.A. Patterson, J.L. Hennessy. — 5th ed. — Elsevier, 2013. — 689 p.
475. *Silberschatz A.* Operating System Concepts [Text] / A. Silberschatz, P.B. Galvin, G. Gagne. — 10th ed. — John Wiley & Sons, 2018. — 972 p.
476. *Fog A.* How to optimize for the Pentium family of microprocessors / A. Fog. — URL: https://cr.yp.to/bib/2004/fog.pdf (дата обращения: 03.12.2025).
477. *Фог А.* Оптимизация для процессоров семейства Pentium: 22. Команды передачи управления и переходов (все процессоры), (пер. Aquila) / А. Фог. — URL: https://wasm.in/blogs/optimizacija-dlja-processorov-semejstva-pentium-22-komandy-peredachi-upravlenija-i-perexodov-vse-processory.162/ (дата обращения: 03.12.2025).
478. *Erez M.* EE482: Advanced Computer Organization Lecture #3: Branch Prediction / M. Erez. — URL: http://cva.stanford.edu/classes/ee482a/scribed/lect03.pdf (дата обращения: 03.12.2025).
479. *Fog A.* The microarchitecture of Intel, AMD, and VIA CPUs: An optimization guide for assembly programmers and compiler makers / A. Fog. — URL: https://www.agner.org/optimize/microarchitecture.pdf (дата обращения: 03.12.2025).
480. *Bulygin Y.* CPU side-channels vs. virtualization malware: The good, the bad, or the ugly / Y. Bulygin. — URL: https://www.slideserve.com/reia/cpu-side-channels-vs-virtualization-malware-the-good-the-bad-or-the-ugly (дата обращения: 03.12.2025).
481. *Shields T.* Survey of Rootkit Technologies and Their Impact on Digital Forensics / T. Shields. — URL: https://www.researchgate.net/publication/228950074_Survey_of_Rootkit_Technologies_and_Their_Impact_on_Digital_Forensics (дата обращения: 03.12.2025).

482. Blue pill/red pill — the matrix has windows longhorn. — URL: https://evilfingers.com/publications/research_RU/blue-pill.pdf (дата обращения: 03.12.2025).
483. Remote Detection of Virtual Machine Monitors with Fuzzy Benchmarking / Franklin J., Luk M., McCune J.M. [et al.] // Proceedings of the USENIX Workshop on Offensive Technologies (WOOT). — URL: https://netsec.ethz.ch/publications/papers/franklin_mccune_luk_seshadri_perrig_vandoorn_osr08.pdf (дата обращения: 03.12.2025).
484. *Greenstadt R.* Security in Virtualized Environments / Greenstadt R. — URL: https://www.researchgate.net/publication/280095977_Security_Challenges_of_Virtualization_Hypervisors_in_Virtualized_Hardware_Environment (дата обращения: 03.12.2025).
485. *Rutkowska J.* Introducing Blue Pill. Presented at the SyScan'06 Conference, Singapore / J. Rutkowska // Advanced Malware Labs, COSEINC Research. — URL: https://web.cecs.pdx.edu/~wuchang/courses/cs492/files/BluePill.pdf (дата обращения: 03.12.2025).
486. *Dai Zovi D.A.* Hardware Virtualization Rootkits: Project CHINASHOP / D.A. Dai Zovi // Black Hat USA 2006, Las Vegas, NV. Matasano Security. — URL: https://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Zovi.pdf (дата обращения: 03.12.2025).
487. *Коркин И.Ю.* Метод выявления аппаратной виртуализации в компьютерных системах на основе механизма кэширования / И.Ю. Коркин // Безопасность информационных технологий. — 2011. — Т. 18, № 1. — URL: https://bit.spels.ru/index.php/bit/article/view/667 (дата обращения: 03.12.2025).
488. *Егоров В.Ю.* Регионы времени как объекты операционной системы общего назначения [Текст] / В.Ю. Егоров, Е.А. Матвеев // Информатика и ее применения. — 2008. — Т. 2, № 4. — С. 74–84.
489. *Таненбаум Э.* Архитектура компьютера [Текст] / Э. Таненбаум. — 5-е изд. — СПб.: Питер, 2007. — 844 с.
490. *Dinaburg A.* TSC Frequency For All: Better Profiling and Benchmarking / A. Dinaburg // Trail of Bits. — 2019. — URL: https://blog.trailofbits.com/2019/10/03/tsc-frequency-for-all-better-profiling-and-benchmarking/ (дата обращения: 03.12.2025).
491. *Касперски К.* Разгон и торможение Windows NT, URL: https://samag.ru/archive/article/327 (дата обращения: 03.12.2025).
492. *Bovet D.P.* Understanding the Linux Kernel / D.P. Bovet, M. Cesati. — 3d ed. O'Reilly Media, 2005. — URL: https://doc.lagout.org/operating%20system%20/linux/Understanding%20Linux%20Kernel.pdf (дата обращения: 03.12.2025).

493. *Marcho C.* What is IRQL and why is it important / Marcho C. — URL: https://techcommunity.microsoft.com/t5/ask-the-performance-team/what-is-irql-and-why-is-it-important/ba-p/372666 (дата обращения: 03.12.2025).
494. *Клименко А.В.* Обзор аппаратных средств и API-сервисов определения времени в персональном компьютере / А.В. Клименко, К.Д. Алексеев // ММС. — 2009. — № 3. — URL: https://cyberleninka.ru/article/n/obzor-apparatnyh-sredstv-i-api-servisov-opredeleniya-vremeni-v-personalnom-kompyutere (дата обращения: 03.12.2025).
495. Intel Corporation. 12th Generation Intel® Core™ Processors Datasheet. — Vol. 1. — URL: https://edc.intel.com/content/www/us/en/design/ipla/software-development-platforms/client/platforms/alder-lake-desktop/: 12th-generation-intel-core-processors-datasheet-volume-1-of-2/009/advanced-configuration-and-power-interface-acpi-states-supported/ (дата обращения: 03.12.2025).
496. High Precision Event Timers — Lenovo ThinkSystem. — URL: https://download.lenovo.com/km/media/attachment/HPET_December_2017.pdf , (дата обращения: 03.12.2025).
497. *Mills D.L.* Network Time Protocol (NTP) (RFC 958) / D.L. Mills. — URL: https://www.ietf.org/rfc/rfc958.txt (дата обращения: 03.12.2025).
498. Mills D.L. Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6, and OSI (RFC 2030). / D.L. Mills. — URL: https://datatracker.ietf.org/doc/html/rfc2030 (дата обращения: 03.12.2025).
499. Методический документ «Методика оценки угроз безопасности информации». — URL: https://fstec.ru/dokumenty/vse-dokumenty/spetsialnye-normativnye-dokumenty/metodicheskij-dokument-ot-5-fevralya-2021-g?ysclid=mip65s0o6j452335275 (дата обращения: 03.12.2025).
500. *Ports D.R.* Towards Application Security on Untrusted Operating Systems / Ports D.R., Garfinkel T. // Proceedings of the 3d conference on Hot topics in security (HOTSEC'08). — URL: https://dl.acm.org/doi/10.5555/1496671.1496672 (дата обращения: 03.12.2025).2
501. R_T_T. Китайские закладки. Непридуманная история о виртуализации, безопасности и шпионах // Хакер. — 2011. — № 12 (155). — С. 30–35. — URL: https://xakep.ru/2011/12/26/58104/ (дата обращения: 04.12.2025).
502. *Paoloni G.* How to Benchmark Code Execution Times on Intel® IA-32 and IA-64 Instruction Set Architectures / G. Paoloni. — URL: https://cis.temple.edu/~qzeng/cis3207-spring18/files/ia-32-ia-64-benchmark-code-execution-paper.pdf (дата обращения: 04.12.2025).

503. *Zmudzinski K.* Methods for selecting cores to execute system management interrupts (US Patent No. US20090172229A1) / K. Zmudzinski. — URL: https://patents.google.com/patent/US20090172229A1/en?oq=US+2009%2f0172229+A1 (дата обращения: 04.12.2025).
504. *Коркин И.Ю.* Обнаружение вложенных мониторов виртуальных машин методами математической статистики [Текст] / И.Ю. Коркин // Сб. материалов XX Общероссийской научно-технической конференции «Методы и Технические Средства Обеспечения Безопасности Информации». — СПб.: 2011. — С. 146–147.
505. *Харари Ф.* Теория графов: пер. с англ. [Текст] / Ф. Харари. — 6-е изд., стереотип. — М.: Издательская группа URSS, 2022. — 304 с.
506. *Вентцель Е.С.* Теория вероятностей [Текст] / Е.С. Вентцель. — 5-е изд., испр. — М.: Издательский центр «Академия», 2003. — 576 с.
507. *Елисеева И.И.* Общая теория статистики: учебник [Текст] / И.И. Елисеева, М.М. Юзбашев; под ред. И.И. Елисеевой. — 5-е изд., перераб. и доп. — М.: Финансы и статистика, 2004. — 656 с.
508. *Тюрин Ю.Н.* Анализ данных на компьютере [Текст] / Ю.Н. Тюрин, А.А. Макаров. — М.: ИНФРА-М: Финансы и статистика, 1995. — 384 с.
509. *Буховец А.Г.* Статистический анализ данных в системе R / А.Г. Буховец, П.В. Москалев, В.П. Богатова, Т.Я. Бирючинская. — URL: http://gis-lab.info/docs/books/moskalev2010_statistical_analysis_with_r.pdf (дата обращения: 04.12.2025).
510. *Васнев С.А.* Статистика: учеб. пособие [Текст] / С.А. Васнев. — М.: МГУП, 2001. — 170 с.
511. Прикладная математическая статистика: учеб. пособие [Текст] / сост. А.А. Мицель. — Томск: ТУСУР, 2022. — 118 с.
512. *Allen M.E.* Kolmogorov-Smirnov Test for Discrete Distributions / M.E. Allen // Master's thesis, Naval Postgraduate School, Monterey, CA, 1976. — URL: https://archive.org/details/kolmogorovsmirno1094517830 (дата обращения: 04.12.2025).
513. *Chiodini P.* Exact Critical Values of Kolmogorov-Smirnov Test for Discrete Random Variables [Text] / P. Chiodini, S. Facchinetti // Statistica & Applicazioni. — 2011. — Vol. IX. — P. 63–77.
514. *Бекман И.Н.* Метод моментов (Параметрические моменты от кинетических кривых в обработке и интерпретации результатов диффузионных экспериментов / И.Н. Бекман. — URL: https://profbeckman.narod.ru/MOM.htm (дата обращения: 04.12.2025).

515. *Коркин И.Ю.* Обоснование критериев присутствия программного обеспечения, использующего технологию аппаратной виртуализации, в компьютерных системах [Текст] / И.Ю. Коркин // Безопасность информационных технологий. — 2012. — Вып. 2. — С. 90–92.
516. *Korkin I.* Strong approach to hardware-VM rootkits detection [Text] / I. Korkin // Hakin9 Extra Magazine (English Edition). — 2011. — № 6. — P. 30–35.
517. ГОСТ Р ИСО 5725–2002 «Точность (правильность и прецизионность) методов и результатов измерений, части 1–6» [Текст]. — М.: ИПК Издательство стандартов, 2002. — 24 с.
518. ISO 5725-1:2023. Точность (правильность и прецизионность) методов и результатов измерений. Часть 1: Общие принципы и определения [Текст]. — Женева: Международная организация по стандартизации, 2023. — 22 с.
519. ISO 5725-2:2019. Точность (правильность и прецизионность) методов и результатов измерений. Часть 2: Основной метод определения повторяемости и воспроизводимости стандартного метода измерений [Текст]. — Женева: Международная организация по стандартизации, 2019. — 31 с.
520. ISO 5725-3:2023. Точность (правильность и прецизионность) методов и результатов измерений. Часть 3: Промежуточные меры прецизионности стандартного метода измерений [Текст]. — Женева: Международная организация по стандартизации, 2023. — 20 с.
521. ISO 5725-4:2020. Точность (правильность и прецизионность) методов и результатов измерений. Часть 4: Основные методы определения правильности стандартного метода измерений [Текст]. — Женева: Международная организация по стандартизации, 2020. — 26 с.
522. *Бурдаев О.В.* Ассемблер в задачах защиты информации [Текст] / О.В. Бурдаев, М.А. Иванов, И.И. Тетерин; под ред. И.Ю. Жукова. — М.: Кудиц-Образ, 2002. — 318 с.
523. *Fog A.* Instruction Tables: Lists of Instruction Latencies, Throughputs and Micro-Operation Breakdowns for Intel, AMD, and VIA CPUs / A. Fog. — URL: https://www.agner.org/optimize/instruction_tables.pdf (дата обращения: 04.12.2025).
524. Intel Corporation. Using the RDTSC Instruction for Performance Monitoring. — URL: https://www.ccsl.carleton.ca/~jamuir/rdtscpm1.pdf (дата обращения: 04.12.2025).

525. *Корнфельд М.И.* Погрешность и надежность простейших экспериментов / М.И. Корнфельд. — URL: https://ufn.ru/ru/articles/1965/3/e/ (дата обращения: 04.12.2025).
526. *Светозаров В.В.* Основы статистической обработки результатов измерений: учеб. пособие [Текст] / В.В. Светозаров. — М.: Изд-во МИФИ, 2005. — 40 с.
527. *Щиголев Б.М.* Математическая обработка наблюдений [Текст] / Б.М. Щиголев. — 3-е изд. — М.: Наука, 1969. — 344 с.
528. Acronis True Image: A familiar name returns! — URL: https://www.acronis.com/en-us/blog/posts/acronis-cyber-protect-home-office-renames-to-true-image/ (дата обращения: 04.12.2025).
529. *Коркин И.Ю.* Выявление вложенных мониторов виртуальных машин / И.Ю. Коркин // Системы высокой доступности. — 2011. — № 2. — URL: http://radiotec.ru/ru/journal/Highly_available_systems/number/2011-2/article/10539 (дата обращения: 04.12.2025).
530. *Коркин И.Ю.* Новые статистические показатели и методы для обнаружения мониторов виртуальных машин в компьютерных системах [Текст] / И.Ю. Коркин // Естественные и технические науки. — 2011. — № 4 (54). — С. 498–502.

Список публикаций автора

Диссертация

Коркин И.Ю. Методика обнаружения нелегитимного программного обеспечения, использующего технологию аппаратной виртуализации: дис. ... канд. техн. наук: 05.13.19 «Методы и системы защиты информации, информационная безопасность» / И.Ю. Коркин; Национальный исследовательский ядерный университет «МИФИ». — М., 2012. — URL: https://elibrary.ru/item.asp?id=19366381 (дата обращения: 04.12.2025).

Сводный список публикаций в журналах и докладов на конференциях

1. *Коркин И.Ю.* Технологии сокрытия вредоносных программ и новые способы противодействия им / И.Ю. Коркин // Безопасность информационных технологий. — 2009. — Т. 16, № 4. — С. 43–46.
2. *Коркин И.Ю.* Способ обнаружения скрытых процессов в операционной системе Windows / И.Ю. Коркин // Проблемы информационной безопасности в системе высшей школы // Сб. науч. тр. 16-ой Всерос. конф. — М., 2009. — С. 111–112.
3. *Коркин И.Ю.* Метод обнаружения аппаратной виртуализации в компьютерных системах / И.Ю. Коркин, Т.В. Петрова, А.Ю. Тихонов // Проблемы информационной безопасности в системе высшей школы: сб. науч. тр. 17-ой Всерос. конф. — М., 2010. — С. 114–115.
4. *Коркин И.Ю.* Метод обнаружения аппаратной виртуализации в компьютерных системах / И.Ю. Коркин, Т.В. Петрова, А.Ю. Тихонов // Безопасность информационных технологий. — 2010. — Т. 17, № 1. — С. 80–81.
5. *Коркин И.Ю.* Критерий обнаружения монитора виртуальных машин в компьютерных системах / И.Ю. Коркин // Методы и технические средства обеспечения безопасности информации: сб. мат. 19-ой Общерос. науч.-техн. конф. — СПб., 2010. — С. 113–114.
6. *Коркин И.Ю.* Новый подход к выявлению аппаратной виртуализации в компьютерных системах / И.Ю. Коркин,

Т.В. Петрова, А.Ю. Тихонов // Молодежь и наука: тез. докл. 14-ой Междунар. телекоммуник. Конф. студентов и молодых ученых. — Ч. 3. — М.: Национальный исследовательский ядерный университет «МИФИ», 2010. — С. 241–242.
7. *Коркин И.Ю.* Метод выявления аппаратной виртуализации в компьютерных системах на основе механизма кэширования / И.Ю. Коркин // Безопасность информационных технологий. — 2011. — Т. 18, № 1. — С. 101–103.
8. *Коркин И.Ю.* Статистическая идентификация режимов работы компьютерных систем / И.Ю. Коркин // Телекоммуникации и новые информационные технологии в образовании: сб. науч. тр. 15-ой конф. — М.: Национальный исследовательский ядерный университет «МИФИ», 2011. — С. 163–165.
9. *Коркин И.Ю.* Выявление вложенных мониторов виртуальных машин / И.Ю. Коркин // Системы высокой доступности. — 2011. — Т. 7, № 2. — С. 76–77.
10. *Коркин И.Ю.* Выявление вложенных мониторов виртуальных машин: доклад на конференции «РусКрипто» / И.Ю. Коркин. — 2011. — URL: https://ruscrypto.ru/accociation/news/2011-03-21.html (дата обращения: 04.12.2025).
11. *Коркин И.Ю.* Обнаружение вложенных мониторов виртуальных машин методами математической статистики / И.Ю. Коркин // Методы и технические средства обеспечения безопасности информации: сб. мат. 20-ой Общерос. науч.-техн. конф. — СПб., 2011. — С. 146–147.
12. *Коркин И.Ю.* Новые статистические показатели и методы для обнаружения мониторов виртуальных машин в компьютерных системах / И.Ю. Коркин // Естественные и технические науки. — 2011. — № 4 (54). — С. 498–502.
13. *Коркин И.Ю.* Обоснование критериев присутствия программного обеспечения, использующего технологию аппаратной виртуализации, в компьютерных системах / И.Ю. Коркин // Безопасность информационных технологий. — 2012. — Т. 19, № 2. — С. 90–92.
14. *Жуков А.Е.* Модели выполнения процессорных инструкций в условиях противодействия со стороны нарушителя для компьютерных систем с поддержкой технологии аппаратной виртуализации / А.Е. Жуков, И.Ю. Коркин, Б.М. Сухинин // Безопасность информационных технологий. — 2012. — Т. 19, № 2. — С. 85–89.

15. *Коркин И.Ю.* Уровневый метод обнаружения гипервизоров в компьютерных системах / И.Ю. Коркин // Методы и технические средства обеспечения безопасности информации: сб. мат. 21-ой Общерос. науч.-техн. конф. — СПб., 2012. — С. 110–113.
16. *Коркин И.* Руткиты: проблемы безопасности и тенденции развития / И. Коркин // Хакер. — 2013. — Вып. 5 (172). — С. 74–79.
17. *Korkin I.* Anti-Rootkits in the Era of Cyber Wars / I. Korkin // Exploiting Software: SamuraiWTF Toolkit. — 2012. — Is. 7. — P. 26–29.
18. *Korkin I.* Applying Memory Forensics to Rootkit Detection / I. Korkin, I. Nesterov // Proceedings of the 9th Annual ADFSL Conference on Digital Forensics, Security and Law. — Richmond, Virginia, USA, 2014. — P. 115–141.
19. *Korkin I.* Strong Approach to Hardware-VM Rootkits Detection / I. Korkin // Hakin9 Extra — Rootkit. — 2011. — Is. 6. — P. 30–35.
20. *Korkin I.* Two Challenges of Stealthy Hypervisors Detection: Time Cheating and Data Fluctuations / I. Korkin // Proceedings of the 10th Annual ADFSL Conference on Digital Forensics, Security and Law. — Daytona Beach, Florida, USA, 2015. — P. 33–57.
21. *Korkin I.* Two Challenges of Stealthy Hypervisors Detection: Time Cheating and Data Fluctuations / I. Korkin // Journal of Digital Forensics, Security and Law. — 2015. — Vol. 10, № 2. — P. 7–38.
22. *Korkin I.* Acceleration of Statistical Detection of Zero-day Malware in the Memory Dump Using CUDA-enabled GPU Hardware / I. Korkin, I. Nesterov // Proceedings of the 11th Annual ADFSL Conference on Digital Forensics, Security and Law. — Daytona Beach, Florida, USA, 2016. — P. 47–82.
23. *Tanda S.* Monitoring & controlling kernel-mode events by HyperPlatform / S. Tanda, I. Korkin // REcon Conference. — Montreal, Canada, 2016. — URL: https://recon.cx/2016/talks/Monitoring-and-controlling-kernel-mode-events-by-HyperPlatform.html (дата обращения: 04.12.2025).

24. *Korkin I.* Detect Kernel-Mode Rootkits via Real-Time Logging & Controlling Memory Access / I. Korkin, S. Tanda // Proceedings of the 12th Annual ADFSL Conference on Digital Forensics, Security and Law. — Daytona Beach, Florida, USA, 2017. — URL: https://commons.erau.edu/adfsl/2017/papers/5/ (дата обращения: 04.12.2025).
25. *Korkin I.* Hypervisor-Based Active Data Protection for Integrity and Confidentiality of Dynamically Allocated Memory in Windows Kernel / I. Korkin // Proceedings of the 13th Annual ADFSL Conference on Digital Forensics, Security and Law. — San Antonio, Texas, USA, 2018. — URL: https://commons.erau.edu/adfsl/2018/presentations/13/ (дата обращения: 04.12.2025).
26. *Korkin I.* Divide et Impera: MemoryRanger Runs Drivers in Isolated Kernel Spaces / I. Korkin // Black Hat Europe Conference. — London, UK, 2018. — URL: https://www.blackhat.com/eu-18/briefings/schedule/index.html#divide-et-impera-memoryranger-runs-drivers-in-isolated-kernel-spaces-12668 (дата обращения: 04.12.2025).
27. *Korkin I.* MemoryRanger Prevents Hijacking FILE_OBJECT Structures in Windows Kernel / I. Korkin // Proceedings of the 14th Annual ADFSL Conference on Digital Forensics, Security and Law. — Daytona Beach, Florida, USA, 2019. — URL: https://commons.erau.edu/adfsl/2019/paper-presentation/7/ (дата обращения: 04.12.2025).
28. *Korkin I.* MemoryRanger Prevents Hijacking FILE_OBJECT Structures in Windows Kernel / I. Korkin // Journal of Digital Forensics, Security and Law. — 2019. — URL: https://commons.erau.edu/jdfsl/vol14/iss3/1/ (дата обращения: 04.12.2025).
29. *Korkin I.* Windows Kernel Hijacking is Not an Option: MemoryRanger Comes to the Rescue Again / I. Korkin // Hack In The Box Security Conference (HITB Lockdown 002). — Singapore, 2020. — URL: https://conference.hitb.org/hitb-lockdown002/sessions/kernel-hijacking-is-not-an-option-memoryranger-comes-to-rescue-again/ (дата обращения: 04.12.2025).
30. *Korkin I.* Windows Kernel Hijacking is Not an Option: MemoryRanger Comes to the Rescue Again / I. Korkin //

Journal of Digital Forensics, Security and Law. — 2021. — URL: https://commons.erau.edu/jdfsl/vol16/iss1/4/ (дата обращения: 04.12.2025).
31. *Korkin I.* Protected Process Light is not Protected: MemoryRanger Fills The Gap Again / I. Korkin // Systematic Approaches to Digital Forensic Engineering (SADFE) International Workshop in conjunction with the 42d IEEE Symposium on Security and Privacy: Proceedings of 2021 IEEE Symposium on Security and Privacy Workshops. — San Francisco, CA, USA, 2021. — URL: https://ieeexplore.ieee.org/document/9474278 (дата обращения: 04.12.2025).
32. *Korkin I.* Protected Process Light will be Protected — MemoryRanger Fills the Gap Again / I. Korkin // Texas Cyber Summit IV. — San Antonio, Texas, USA, 2021. — URL: https://www.youtube.com/watch?v=oY28U6U2moo (дата обращения: 04.12.2025).
33. *Korkin I.* Your Linux Passwords Are in Danger: MimiDove Meets the Challenge / I. Korkin, S. Golub // Texas Cyber Summit IV. — San Antonio, Texas, USA, 2021. — URL: https://www.youtube.com/watch?v=R4yWYtTqH4I (дата обращения: 04.12.2025).
34. *Korkin I.* Your Linux Passwords Are in Danger: MimiDove Meets the Challenge / I. Korkin, S. Golub // IOSR Journal of Computer Engineering. — 2021. — URL: https://www.iosrjournals.org/iosr-jce/papers/Vol23-issue6/Ser-1/C2306012728.pdf (дата обращения: 04.12.2025).
35. *Голуб С.А.* Анализ безопасности подсистем локальной аутентификации ОС семейств Windows и Linux / С.А. Голуб, И.Ю. Коркин // Безопасность информационных технологий. — 2022. — Т. 29, № 1. — С. 57–69.
36. *Korkin I.* Veni, No Vidi, No Vici: Attacks on ETW Blind EDR Sensors / I. Korkin, C. Teodorescu, A. Golchikov // Black Hat Europe Conference. — London, UK, 2021. — URL: https://www.blackhat.com/eu-21/briefings/schedule/#veni-no-vidi-no-vici-attacks-on-etw-blind-edr-sensors-24842 (дата обращения: 04.12.2025).
37. *Grishin M.* Human-Controlled Fuzzing With AFL / M. Grishin, I. Korkin // Proceedings of the 15th Annual ADFSL Conference

on Digital Forensics, Security and Law. — Florida, USA, 2022. — URL: https://commons.erau.edu/adfsl/2022/presentations/3/ (дата обращения: 04.12.2025).
38. *Grishin M.* Human-Controlled Fuzzing with AFL / M. Grishin, I. Korkin // ROOTCON 16. — Philippines, 2022. — URL: https://www.youtube.com/watch?v=__5KbGy2cHk (дата обращения: 04.12.2025).
39. *Pogonin D.* Microsoft Defender Will Be Defended: MemoryRanger Prevents Blinding Windows AV / D. Pogonin, I. Korkin // Proceedings of the 15th Annual ADFSL Conference on Digital Forensics, Security and Law. — Florida, USA, 2022. — URL: https://commons.erau.edu/adfsl/2022/presentations/7/ (дата обращения: 04.12.2025).
40. *Pogonin D.* Microsoft Defender Will Be Defended: MemoryRanger Prevents Blinding Windows AV / D. Pogonin, I. Korkin // ROOTCON 16. — Philippines, 2022. — URL: https://www.youtube.com/watch?v=dF7TP1V6wTM&list=PLpJXs8BYnI-bwj-NP_bD4mazhI_h3zQ5E (дата обращения: 04.12.2025).
41. *Korkin I.* Blasting Event-Driven Cornucopia: WMI-based User-Space Attacks Blind SIEMs and EDRs / I. Korkin, C. Teodorescu, A. Golchikov // Black Hat USA Conference. — Las Vegas, USA, 2022. — URL: https://www.blackhat.com/us-22/briefings/schedule/#blasting-event-driven-cornucopia-wmi-based-user-space-attacks-blind-siems-and-edrs-27211 (дата обращения: 04.12.2025).
42. *Korkin I.* Blasting Event Driven Cornucopia: WMI Edition / I. Korkin, C. Teodorescu, A. Golchikov // LABScon. — Scottsdale, Arizona, USA, 2022. — URL: https://www.sentinelone.com/labs/labscon-replay-blasting-event-driven-cornucopia-wmi-based-user-space-attacks-blind-siems-and-edrs/ (дата обращения: 04.12.2025).
43. *Korkin I.* Blinding Endpoint Security Solutions: WMI attack vectors / I. Korkin, C. Teodorescu, A. Golchikov // EKOPARTY. — Buenos Aires, Argentina, 2022. — URL: https://www.youtube.com/watch?v=Q35snDTxb6w (дата обращения: 04.12.2025).

44. *Калинкин А.О.* Обнаружение программ-шифровальщиков на основе данных механизма трассировки событий и применения метода машинного обучения / А.О. Калинкин, С.А. Голуб, И.Ю. Коркин, Д.Н. Пятовский // Безопасность информационных технологий. — 2022. — Т. 29, № 3. — С. 82–93.
45. *Kropova A.* ALPChecker — Detecting Spoofing and Blinding Attacks / A. Kropova, I. Korkin // Hack In The Box Security Conference (HITBSecConf2023), CommSec Track. — Phuket, Thailand, 2023. — URL: https://www.youtube.com/watch?v=DV0dls206zc (дата обращения: 04.12.2025).
46. *Поддубный В.А.* Средство обнаружения скрытого исполнимого кода в памяти ОС Windows / В.А. Поддубный, И.Ю. Коркин // Вопросы кибербезопасности. — 2019. — № 5 (33). — С. 75–82.
47. *Поддубный В.А.* Средство обнаружения скрытого исполнимого кода в памяти ОС Windows / В.А. Поддубный, И.Ю. Коркин // Управление информационной безопасностью в современном обществе: VII Междунар. науч.-практ. конф. — М., 2019. — URL: https://www.elibrary.ru/item.asp?id=41517996 (дата обращения: 04.12.2025).
48. *Казаков О.А.* Обнаружение криптомайнеров в ОС Windows инструментами технологии ETW / О.А. Казаков, И.Ю. Коркин // Вопросы кибербезопасности. — 2019. — № 5 (33). — С. 83–88.
49. *Казаков О.А.* Обнаружение криптомайнеров в ОС Windows инструментами технологии ETW / О.А. Казаков, И.Ю. Коркин // Управление информационной безопасностью в современном обществе: VII Междунар. науч.-практ. конф. — М., 2019. — URL: https://www.elibrary.ru/item.asp?id=41517997 (дата обращения: 04.12.2025).
50. Анализ способов получения копии оперативной памяти компьютера под управлением MAC OS / В.А. Ляшенко, А.А. Липницкий, М.А. Пак [и др.] // Актуальные научные исследования в современном мире. — URL: https://www.elibrary.ru/item.asp?id=43092509 (дата обращения: 04.12.2025).

51. *Рудик М.В.* Расширение возможностей Clang Static Analyzer для обнаружения новых уязвимостей / М.В. Рудик, И.Ю. Коркин // Естественные и технические науки. — 2021. — № 7 (158). — С. 225–230.
52. *Кропова А.В.* Исследование вопросов безопасности технологии ALPC в условиях атак с использованием драйверов Windows / А.В. Кропова, И.Ю. Коркин // ИНТЕРЭКСПО ГЕО-СИБИРЬ. — 2023. — Т. 7, № 2. — С. 43–50.
53. *Счастливцев К.Д.* Анализ защиты операционной системы Windows 11 от внедрения кода в процессы / К.Д. Счастливцев, И.Ю. Коркин // Кибернетика и информационная безопасность: сб. науч. тр. Всерос. науч.-техн. конф. «КИБ-2023». — М., 2023. — С. 30–31.
54. *Гришин М.А.* Гибридная схема направленного фаззинг тестирования ядра Linux с использованием инструмента Syzkaller / М.А. Гришин, И.Ю. Коркин // Кибернетика и информационная безопасность: сб. науч. тр. 2-ой Всерос. науч.-техн. конф. «КИБ-2024». — М., 2024. — С. 238–239.
55. *Коркин И.Ю.* О подходе к защите приложений класса «менеджер паролей» от перехвата клавиатурного ввода / И.Ю. Коркин // Кибернетика и информационная безопасность: сб. науч. тр. 3-ой Всерос. науч.-техн. конф. «КИБ-2025». — М., 2025. — С. 70-71.

Патент

Ни К. Data Protection Method, Apparatus, Storage Medium, and Computer Device / K. Hu, I.Y. Korkin; Assignee: Huawei Technologies Co. Ltd. // European Patent Office. — 2022. — Патент EP4231181B1. — URL: https://patents.google.com/patent/EP4231181B1/en (дата обращения: 04.12.2025).

Рецензии

РЕЦЕНЗИЯ НА МОНОГРАФИЮ КОРКИНА ИГОРЯ ЮРЬЕВИЧА«ЗАЩИТА ЯДРА ОПЕРАЦИОННЫХ СИСТЕМ В УСЛОВИЯХ ПРОТИВОДЕЙСТВИЯ»

Актуальность и научная новизна

Монография И.Ю. Коркина посвящена одной из важных проблем современной кибербезопасности — защите ядра операционных систем, в первую очередь Windows, от сложных угроз: руткитов, гипервизор-руткитов, атак на EDR/AV и техник скрытия вредоносного ПО с использованием аппаратной виртуализации. Актуальность темы монографии не вызывает сомнений.

Научная новизна работы заключается в изложенном автором комплексном подходе к решению проблемы, основанном на полученных лично автором новых научных результатах, изложенных в авторских публикациях в высокорейтинговых научных изданиях. Автор не только системно анализирует эволюцию угроз и защит, но и предлагает собственные гипервизорные решения, которые были представлены на ведущих мировых конференциях, а также математические методы обнаружения скрытых гипервизоров. Особо следует подчеркнуть, что автор рассматривает проблему защиты ядра ОС в условиях активного противодействия со стороны подготовленного нарушителя.

Структура и логика изложения

Монография состоит из трех глав. Каждая глава логично продолжает предыдущую, обеспечивая целостное раскрытие темы: от анализа угроз до реализации научно обоснованных авторских решений.

В главе 1 «Анализ угроз и механизмов защиты ядра Windows»:

- подробно рассмотрены архитектура Windows, эволюция защитных механизмов, современные техники атак со стороны

ядра, содержится исторический экскурс по развитию защиты Windows, что позволяет понять эволюцию угроз и ответных мер;
- описаны методы получения и анализа дампов памяти, техники сокрытия объектов и их обнаружения.

В главе 2 «Разработка гипервизоров для мониторинга и защиты памяти ядра»:
- представлены авторские гипервизорные проекты, их архитектура, реализация и практические результаты;
- рассмотрены вопросы защиты памяти ядра, изоляции драйверов, предотвращения атак на важные структуры и сущности ядра (FILE_OBJECT, токены процессов, PPL), в том числе пример атак на штатный антивирус Microsoft Defender.

В главе 3 «Математические основы обнаружения скрытых гипервизоров»:
- описаны сигнатурные, поведенческие, временные методы, а также модели нарушителя, способы компрометации таймеров и статистические методы обнаружения гипервизоров;
- приведены экспериментальные результаты апробации предложенной методики обнаружения.

Научная обоснованность содержания и практическая значимость монографии

Автор опирается на собственные многолетние исследования, результаты которых опубликованы более чем в 40 научных статьях, патентах и докладах на 30 международных конференциях по кибербезопасности, что подтверждает высокий уровень научной достоверности и признания в профессиональном сообществе. Практическая значимость работы проявляется в предложении конкретных инструментов и методик, пригодных для внедрения в реальные системы защиты информации, включая уникальные гипервизорные решения для защиты ядра Windows и математические модели обнаружения скрытых угроз.

Качество изложения и стиль

Текст монографии написан профессиональным, технически грамотным языком. Используется современная терминология, приведены многочисленные схемы, таблицы, примеры, что об-

легчает восприятие сложного материала. Каждый раздел сопровождается выводами, что помогает читателю структурировать знания и выделять ключевые идеи.

Заключение и рекомендация к изданию

Считаю, что монография И.Ю. Коркина — это актуальный и значимый научный труд, вносящий существенный вклад в развитие методов защиты ядра операционных систем от современных угроз. Автор демонстрирует глубокое понимание темы, владение современными технологиями и способность к комплексному анализу сложных проблем информационной безопасности.

Рекомендация

Монография рекомендуется к изданию. Она будет полезна специалистам и разработчикам решений по защите информации, студентам и аспирантам, а также всем, кто интересуется современными угрозами и методами защиты операционных систем. Полагаю, что она может стать справочником и настольной книгой для всех специалистов в области безопасности операционных систем.

Рецензент (подпись) Запечников Сергей Владимирович

Ученая степень: доктор технических наук.
Ученое звание: доцент
Должность: Профессор кафедры криптологии и кибербезопасности Института интеллектуальных кибернетических систем, Национальный исследовательский ядерный университет «МИФИ»

Подпись заверяю
НАЧАЛЬНИК ОТДЕЛА
АУП и УВП
МИКИРОВА Н.О.
ДАТА
10 СЕН 2025

РЕЦЕНЗИЯ НА МОНОГРАФИЮ И.Ю. КОРКИНА «ЗАЩИТА ЯДРА ОПЕРАЦИОННЫХ СИСТЕМ В УСЛОВИЯХ ПРОТИВОДЕЙСТВИЯ»

Монография И.Ю. Коркина посвящена весьма актуальной и в то же время исключительно сложной задаче — обеспечению безопасности ядра операционной системы Windows в условиях непрерывно изменяющихся угроз, включая атаки драйверов ядра и скрытые гипервизоры. Являясь результатом многолетних исследований автора, работа объединяет глубокий теоретический анализ предметной области и подробное изложение авторских методик.

Автор последовательно раскрывает проблематику безопасности ядра Windows начиная с анализа его архитектуры и эволюции встроенных механизмов защиты — от первых версий Kernel Patch Protection до современных решений HVCI и KDP. При этом обоснованно демонстрируется фундаментальная уязвимость штатного средства защиты ядра Kernel Patch Protection, работоспособность которого может быть нарушена атаками драйверов. Для построения средств противодействия этой уязвимости автор предлагает два принципиально разных, но взаимодополняющих подхода:

1. Гипервизорная защита памяти ядра, реализованная в виде собственных решений MemoryMonRWX, AllMemPro и MemoryRanger. Эти разработки используют технологию аппаратной виртуализации Intel VT-x и технологию EPT (Extended Page Tables) для мониторинга и контроля доступа к памяти ядра, что позволяет предотвращать несанкционированные модификации со стороны вредоносных драйверов.

2. Обнаружение скрытых гипервизоров, включая несколько вложенных образцов (nested hypervisors), на основе статистического анализа длительности выполнения инструкций. Разработанные математические модели и алгоритмы позволяют выявлять присутствие скрытых гипервизоров даже в условиях активного противодействия со стороны нарушителей.

Научная новизна работы заключается в предложенных автором решениях, которые были представлены на ведущих ми-

ровых конференциях по кибербезопасности. Особое внимание уделено условиям активного противодействия, что делает разработки особенно актуальными в условиях современных APT-атак.

Структура монографии логична и последовательна: от анализа угроз и архитектуры ядра (глава 1) к разработке и апробации гипервизорных решений по защите памяти ядра (глава 2) и, наконец, к методам обнаружения скрытых гипервизоров (глава 3). Такое построение обеспечивает целостное и многогранное раскрытие темы, подчеркивая научную зрелость и концептуальную завершенность исследования.

Следует отметить, что в работе убедительно обоснована достоверность полученных результатов, которые, кроме того, признаны мировым научным сообществом. Результаты многолетних исследований автора неоднократно докладывались на ведущих международных конференциях в области кибербезопасности, включая Black Hat USA/Europe, HUB Security Conference, IEEE Symposium on Security and Privacy Workshops, REcon, EKOPARTY и др. Наличие более чем 40 научных публикаций, изобретения, защищенного международным патентом, а также большой практический опыт работы в компаниях уровня Positive Technologies, Huawei и Binarly — все это свидетельствует о высоком профессиональном уровне автора и значимом вкладе его работ в мировую практику защиты операционных систем.

Заключение

Монография И.Ю. Коркина является значительным вкладом в развитие теории и практики защиты операционных систем. Она не только систематизирует современные угрозы и критически анализирует ограничения существующих решений, но и предлагает научно обоснованные, экспериментально подтвержденные и практически применимые методы защиты ядра. Работа соответствует требованиям к научным монографиям, обладает научной новизной, актуальностью и практической значимостью.

Изложенное позволяет рекомендовать монографию к изданию без замечаний.

Рецензент,
профессор кафедры
«Информационная безопасность»
МГТУ им. НL Э. Баумана,
доктор технических наук Ключарёв Петр Георгиевич

(подпись)

«ВЕРНО»
НАЧАЛЬНИК ОТДЕЛА
ЖИНЯПИНА А.Н.
ОТДЕЛ ПООРГАНИЗАЦИИ РАБОТЫ
ЕДИНОЙ ПРИЁМНОЙ УКСИА
МГТУИМЕНИ Н.Э.БАУМАНА

Оглавление

Список принятых сокращений **5**

Предисловие **13**

Благодарности **16**

Введение **17**

Раздел 1. Анализ вопросов безопасности ядра Windows ОС: механизмы защиты и техники атак **22**

Глава 1. Введение в архитектуру Windows ОС с позиции информационной безопасности **22**

1.1. Понятие защищенного режима работы процессора, технологии аппаратной виртуализации и их возможности с позиции информационной безопасности 25
1.2. Архитектура и ядро Windows ОС 29
1.3. История развития встроенных механизмов защиты Windows ОС 33

Глава 2. Анализ существующих техник сокрытия вредоносного программного обеспечения **45**

2.1. Стеганографические техники сокрытия 46
2.2. Сокрытие объектов «внутри» ОС: изменение пути выполнения и структур памяти 47
2.2.1. Изменение структур объектов 50
2.2.2. Изменение пути выполнения 50
2.3. Гипервизор-руткиты 54
2.4. SMM-руткиты 56
2.5. AMT/ME-руткиты 59

Глава 3. Возможности DKOM-техник: от сокрытия объектов до противодействия EDR **61**

3.1. Сокрытие процессов, потоков, драйверов и сервисов (служб) с помощью DKOM 63
3.2. Использование DKOM для обхода и противодействия EDR/AV 67

Глава 4. Анализ техник получения дампа памяти для выявления скрытых объектов **70**

4.1. Техники получения дампа виртуальной памяти процессов 70
4.2. Техники получения дампа физической памяти на различных уровнях привилегий 71
4.2.1. Средства получения дампа физической памяти на уровне ядра 71
4.2.2. Средства получения дампа физической памяти на уровне аппаратной виртуализации 73
4.2.3. Средства получения дампа физической памяти на уровне режима системного управления 74

4.2.4. Средства получения дампа физической памяти на уровне аппаратной технологии AMT 74
4.2.5. Средства получения дампа физической памяти на уровне аппаратного обеспечения 75
4.3. Техника получения дампа памяти с помощью дерева структур страниц памяти 76
4.3.1. Подход к получению дампа памяти 76
4.3.2. Процесс отображения виртуальной и физической памяти 77
4.3.3. Обработка данных, полученных из дампа памяти 81
4.3.4. Использование инструмента MASHKA в задачах исследования памяти 83

Глава 5. Методы обнаружения скрытых объектов в дампе памяти 87

5.1. Обнаружение скрытых процессов и потоков 89
5.2. Обнаружение скрытых драйверов ядра 92
5.3. Подходы на основе сравнения содержимого различных списков драйверов 97
5.4. Подходы на основе сигнатурного поиска фрагментов структур и модулей драйверов в памяти 103
5.4.1. Методы обнаружения структур драйверов на основе сигнатур 104
5.4.2. Методы обнаружения загруженных драйверов в памяти на основе сигнатур 106
5.5. Методы обнаружения драйверов с использованием атрибутов страниц памяти 108
5.6. Статистические методы обнаружения загруженных драйверов в памяти 110
5.6.1. Применение энтропии для поиска сжатых и зашифрованных данных 110
5.6.2. Метод «скользящего окна» для обнаружения кода в памяти 113
5.6.3. Техники визуализации для ручного анализа памяти 115
5.6.4. Анализ техник нарушителей для противодействия статистическим методам обнаружения исполнимого кода в памяти 116
5.7. Применение статических и динамических сигнатур для обнаружения процессов и драйверов 118
5.7.1. Пример динамической битовой сигнатуры для обнаружения процессов 118
5.7.2. Пример статической балловой сигнатуры для обнаружения драйверов 121

Глава 6. Перспективы развития техник сокрытия и обнаружения 128

6.1. Демонстрация новых методов сокрытия объектов в памяти 129
6.1.1. Первый способ на основе случайных диапазонов 130
6.1.2. Второй способ на основе циклических диапазонов 131
6.1.3. Использование ассемблерных вставок 133
6.2. Предлагаемые подходы и эксперименты по обнаружению новых техник 136
6.3. Анализ некоторых экспериментальных техник обнаружения драйверов в условиях сложных случаев сокрытия кода 140
6.3.1. Обнаружение скрытого драйвера методом уменьшения размера «скользящего окна» 141
6.3.2. Альтернативные подходы к обнаружению скрытого кода в памяти в условиях противодействия 142
Выводы 147

Раздел 2. Разработка гипервизоров для мониторинга и защиты памяти ядра Windows от атак драйверов 148

Глава 7. Проект MemoryMonRWX: обнаружение и анализ работы руткит-драйверов путем логирования их обращений к глобальным переменным ядра 148

7.1. Сравнительный анализ методов мониторинга доступа к памяти 151
7.1.1. Методы на основе операционной системы 152
7.1.2. Методы на основе гипервизора 158
7.1.3. Использование технологии Extended Page Tables 163
7.2. Архитектура и реализация гипервизора MemoryMonRWX для мониторинга обращений драйверов к памяти ядра 169
7.2.1. Применение EPT-технологии для отслеживания и контроля доступа к памяти 170
7.2.2. Алгоритм управления доступом к памяти с использованием MemoryMonRWX 172
7.2.3. Архитектура и основные компоненты MemoryMonRWX 176
7.3. Практическое применение MemoryMonRWX для обнаружения и анализа руткитов 181
7.3.1. Обеспечение целостности памяти. Предотвращение модификации кода и данных 182
7.3.2. Обеспечение конфиденциальности памяти. Предотвращение чтения данных 183
7.3.3. Применение MemoryMonRWX для анализа руткита Turla 184
7.3.4. Оценка производительности 185

Глава 8. Проект AllMemPro: применение гипервизора для защиты конфиденциальности и целостности динамически выделяемой памяти в ядре от несанкционированного доступа 187

8.1. Анализ предметной области 187
8.1.1. Угрозы для выделенной памяти в ОС Windows 187
8.1.2. Современные кибератаки: анализ, примеры и стратегии защиты для ядра и промышленного программного обеспечения 189
8.1.3. Модель угроз: описание и анализ 191
8.1.4. Анализ существующих методов и инструментов защиты памяти ядра 193
8.2. Архитектура и реализация гипервизора AllMemPro для защиты памяти 196
8.2.1. Основные сценарии атак на динамически выделяемую память 197
8.2.2. Архитектура системы AllMemPro 201
8.3. Практическое применение гипервизора AllMemPro: защита выделенной памяти драйвера Windows от несанкционированного доступа 205
8.3.1. Первый сценарий — демонстрация несанкционированного доступа к выделенной памяти без защиты гипервизора 205
8.3.2. Второй сценарий — гипервизор блокирует несанкционированный доступ к выделенной памяти в ядре 207
8.3.3. Третий сценарий — гипервизор обеспечивает безопасный совместный доступ к выделенной памяти со стороны драйвера и функций ядра операционной системы 209

Глава 9. Проект MemoryRanger: применение гипервизора для запуска нескольких драйверов в изолированных анклавах памяти ядра 213

9.1. Анализ существующих подходов к защите памяти ядра для ОС Windows и Linux 217
9.1.1. Эволюция встроенных средств безопасности Windows 217
9.1.2. Гипервизорные системы защиты памяти 217
9.2. Разработка и реализация гипервизора MemoryRanger для изолированного запуска драйверов 227
9.2.1. EPT: идея изоляции памяти 227
9.2.2. Архитектура MemoryRanger 232
9.3. Практическое применение MemoryRanger: предотвращение несанкционированного доступа к памяти ядра для случая нескольких независимых драйверов 235
9.3.1. Результаты тестирования производительности MemoryRanger 235
9.3.2. Ограничения MemoryRanger 237
9.3.3. Направления дальнейшего развития 237

Глава 10. Проект MemoryRanger: предотвращение несанкционированного доступа к структурам FILE_OBJECT в ядре Windows 239

10.1. Анализ архитектурных уязвимостей файловой подсистемы Windows: исследование возможностей несанкционированного доступа к файлам, открытым в режиме монопольного использования 240
10.1.1. Обзор файловой системы в режиме ядра 240
10.1.2. Встроенные в Windows механизмы безопасности 242
10.1.3. Анализ структуры FILE_OBJECT 243
10.1.4. Доступ к содержимому файла, открытого в монопольном режиме, путем перехвата его FILE_OBJECT 244
10.2. Модификация гипервизора MemoryRanger для защиты структур FILE_OBJECT 247
10.2.1. Архитектура MemoryRanger 247
10.2.2. Настройка MemoryRanger для защиты структур FILE_OBJECT 247

Глава 11. Проект MemoryRanger: предотвращение атак типа hijacking на структуры ядра Windows 251

11.1. Анализ новых атак на структуры ядра Windows: исследование методов несанкционированного доступа к файлам и повышения привилегий процессов 251
11.1.1. Анализ новых атак на структуры ядра Windows 251
11.1.2. Модель угроз 253
11.1.3. Две атаки на файловые объекты для случая файлов, открытых в режиме без совместного доступа 254
11.1.4. Перехват структур данных NTFS 260
11.1.5. Перехват токена доступа 266
11.2. Модификация гипервизора MemoryRanger для защиты структур ядра 270
11.2.1. Краткий обзор MemoryRanger 270
11.2.2. Основные компоненты MemoryRanger 271
11.3. Практическое применение модифицированного MemoryRanger: предотвращение атак на файловые объекты и токены процессов 272
11.3.1. Предотвращение атак перехвата дескрипторов 272
11.3.2. Предотвращение атак на структуры NTFS 273
11.3.3. Предотвращение атак на токены доступа 273

11.3.4. Эмпирическая проверка и тестирование ... 273
11.3.5. Производительность MemoryRanger ... 275
11.3.6. Сравнение MemoryRanger с Virtual Secure Mode ... 275

Глава 12. Проект MemoryRanger: предотвращение атак на защитный механизм Protected Process Light со стороны драйверов Windows ... 276

12.1. Анализ архитектурных уязвимостей механизма Protected Process Light (PPL): исследование возможностей несанкционированного отключения и повышения уровня защиты процессов ... 277
12.1.1. Защита памяти процессов: модель PPL, методы атаки и защиты ... 277
12.1.2. Внутреннее устройство PPL ... 278
12.1.3. Пример отключения защиты PPL и извлечение паролей с использованием Mimikatz ... 284
12.2. Модификация гипервизора MemoryRanger для защиты структур PPL ... 289
12.2.1. Расширенные возможности MemoryRanger для защиты механизма PPL ... 289
12.2.2. Защита PPL: обновления MemoryRanger ... 290
12.3. Практическое применение модифицированного MemoryRanger: предотвращение атак на PPL и сравнение с Windows Defender Credential Guard ... 294
12.3.1. Оценка влияния MemoryRanger на производительность процессора ... 294
12.3.2. Сравнение MemoryRanger и Windows Defender Credential Guard (WDCG) ... 295

Глава 13. Проект MemoryRanger: защита Microsoft Defender от атак на уровне ядра Windows ... 298

13.1. Анализ популярных атак на Microsoft Defender: исследование техник отключения и обхода антивирусной защиты ... 299
13.1.1. Проблемы безопасности ядра Windows и современные технологии защиты ... 299
13.1.2. Анализ атак на Microsoft Defender ... 300
13.2. Анализ работы антивируса Microsoft Defender в условиях атак драйверов: манипуляции с уровнем целостности и привилегиями антивирусного процесса ... 304
13.2.1. Обзор внутреннего устройства антивируса MD ... 304
13.2.2. Исходное поведение антивируса MD при обнаружении вредоносного программного обеспечения ... 306
13.2.3. Механизм мандатного контроля целостности ... 307
13.3. Модификация гипервизора MemoryRanger для защиты структур Microsoft Defender ... 314
13.4. Практическое применение модифицированного MemoryRanger: предотвращение атак на Microsoft Defender ... 316
13.4.1. Экспериментальное исследование работоспособности MemoryRanger ... 316
13.4.2. Преимущества обновленного MemoryRanger ... 317
Выводы ... 318

Раздел 3. Математические основы обнаружения скрытых гипервизоров в условиях противодействия ... 320

Глава 14. Введение в технологию виртуализации и примеры программного обеспечения, использующего технологию аппаратной виртуализации ... 320

14.1. Методы обнаружения гипервизора ... 323

14.2. Сигнатурные методы обнаружения ... 325
14.2.1. Программные методы обнаружения ... 326
14.2.2. Аппаратные методы обнаружения ... 327
14.3. Поведенческие методы обнаружения ... 328
14.3.1. Метод обнаружения на основе буфера ассоциативной трансляции (Translation Lookaside Buffer) ... 328
14.3.2. Методы обнаружения гипервизоров на основе ошибок в процессорах и реализациях гипервизоров ... 331
14.4. Обнаружение угроз на основе доверенного монитора виртуальных машин ... 332
14.5. Временные методы обнаружения гипервизоров ... 334
14.5.1. Метод обнаружения на основе буфера ассоциативной трансляции (Translation Lookaside Buffer) ... 334
14.5.2. Метод обнаружения на основе буфера стека возврата (Return Stack Buffer) ... 336
14.5.3. Метод обнаружения на основе списка демаскирующих событий ... 338
14.5.4. Метод обнаружения на основе обращения к памяти ... 340
14.6. Применение компьютерных счетчиков для оценки времени выполнения процессорных инструкций ... 341
14.6.1. Локальные счетчики ... 341
14.6.2. Сетевые счетчики ... 343

Глава 15. Методы математической статистики обнаружения гипервизора в условиях противодействия со стороны нарушителя ... 344

15.1. Модель нарушителя ... 344
15.2. Возможные способы компрометации счетчика тактов ... 346
15.2.1. Компрометация при перехвате БП-инструкции ... 346
15.2.2. Компрометация при перехвате инструкций чтения счетчика ... 348
15.2.3. Компрометация с использованием поля TSC_OFFSET в структуре управления виртуальной машиной ... 351
15.3. Построение моделей выполнения трассы ... 351
15.3.1. Анализ режимов работы процессора и влияния гипервизора на время выполнения инструкций ... 352
15.3.2. Особенности выполнения набора процессорных инструкций ... 355
15.3.3. Допущения для построения моделей выполнения трассы ... 360
15.3.4. Модель выполнения трассы в случае отсутствия гипервизора ... 360
15.3.5. Валидация разработанных моделей на основе экспериментальных данных ... 363
15.3.6. Критерии обнаружения гипервизора ... 370

Глава 16. Статистические характеристики длительности выполнения набора процессорных инструкций. Методика обнаружения скрытых гипервизоров ... 376

16.1. Экспериментальные исследования измерений длительности выполнения трассы ... 380
16.2. Характеристики выполнения процессорных инструкций и методы обработки экспериментальных данных ... 382
16.3. Анализ экспериментальных данных и выделение признаков для обнаружения скрытых гипервизоров ... 390
16.3.1. Фильтрация и обработка данных ... 391
16.3.2. Уровневый метод выявления признаков гипервизоров ... 391
16.3.3. Метод выборочных статистик ... 392

16.3.4. Пороговые условия обнаружения гипервизоров: результаты экспериментальной проверки ... 399
16.4. Методика обнаружения скрытых гипервизоров ... 400
16.4.1. Алгоритмы проверки на наличие гипервизора ... 400
16.4.2. Порядок тестирования на оперативном этапе ... 402
16.4.3. Пошаговая методика обнаружения нелегитимного гипервизора ... 403
16.5. Результаты апробации предлагаемой методики обнаружения гипервизоров в условиях противодействия со стороны нарушителя ... 404
Выводы ... 408

Заключение ... 411

Библиографический список ... 413

Список публикаций автора ... 467

Рецензии ... 475

По вопросам приобретения книг обращайтесь:
Отдел продаж «ИНФРА-М» (оптовая продажа):
127214, Москва, ул. Полярная, д. 31В, стр.1
Тел. (495) 859-48-60 (доб. 222, 564)
E-mail: books@infra-m.ru

•

Отдел «Книга—почтой»:
тел. (495) 859-48-60 (доб. 222)

ФЗ № 436-ФЗ	Издание не подлежит маркировке в соответствии с п. 1 ч. 2 ст. 1

Научное издание

Коркин Игорь Юрьевич

ЗАЩИТА ЯДРА ОПЕРАЦИОННЫХ СИСТЕМ В УСЛОВИЯХ ПРОТИВОДЕЙСТВИЯ

МОНОГРАФИЯ

Оригинал-макет подготовлен в НИЦ ИНФРА-М

ООО «Научно-издательский центр ИНФРА-М»
127214, Москва, ул. Полярная, д. 31В, стр. 1
Тел.: (495) 859-48-60
E-mail: books@infra-m.ru http://www.infra-m.ru

Подписано в печать 19.02.2026.
Формат 70×100/16. Бумага офсетная. Гарнитура Petersburg.
Печать цифровая. Авт. л. 22,0. Усл. печ. л. 39,65.
Тираж 500 экз. (I — 50). Заказ № 00000

ТК 865376-2228735-190226

Отпечатано в типографии ООО «Научно-издательский центр ИНФРА-М»
127214, Москва, ул. Полярная, д. 31В, стр. 1
Тел.: (495) 859-48-60