Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование информационных систем
Системное программирование

Небогатиков Иван Юрьевич

Мониторинг работы операционной системы в реальном времени на основе гипервизора

Дипломная работа

Научный руководитель:
старший преподаватель Ханов А. Р.

Рецензент:
доцент Воробьева А. А.

Санкт-Петербург
2019

SAINT-PETERSBURG STATE UNIVERSITY

Software and Administration of Information Systems
Software Engineering

Nebogatikov Ivan

Monitoring of operating system in real time using hypervisor

Graduation Thesis

Scientific supervisor:
senior lecturer Khanov Arthur

Reviewer:
associate professor Vorobyova Alisa

Saint-Petersburg
2019

Оглавление

Введение

Системные вызовы — это запросы к сервисам операционной системы, например запросы к файловой системе. Их анализ — эффективный способ поиска вредоносного програмного обеспечения, поскольку с помощью него можно обнаружить вирусы, которые обходят обычные системы защиты [1].

Такой подход использует система КОДА как основной метод поиска и блокировки работы вирусов в ОС Windows. Для анализа ей необходимо изменять обработчик системных вызовов, но такой способ вызывает срабатывание системы Kernel Patch Protection, которая защищает ядро операционной системы от нежелательных модификаций. Это происходит, поскольку тем же методом пользуются [2] руткиты — вирусы, способные повысить свой уровень привилегий.

Поскольку система Kernel Patch Protection не документирована и ее поведение часто меняется, взаимодействие с ней не эффективно. Решение этой проблемы — использование технологий виртуализации. Современные процессоры поддерживают их на аппаратном уровне, это позволяет создать еще одно кольцо защиты с номером -1. В нем можно хранить и обрабатывать информацию, доступ к которой должен быть ограничен. Минус такого подхода — дополнительные расходы по времени.

Такое решение позволяет гарантировать, что обработчик прерываний был изменен только КОДОЙ и не модифицирован после руткитами.

Также КОДА не защищена от атак на файлы жесткого диска программами, которые обходят контроль учетных записей Windows, и вирусов, которые пытаются найти и выгрузить системы защиты. Использование ее методов для самозащиты не эффективно, поскольку ее поведение зависит от загруженных моделей подозрительных системных вызовов, которые могут не срабатывать во всех необходимых случаях.

Важным параметром систем защиты является избыточная нагрузка, поскольку необходимо или проводить периодические проверки, или добавлять дополнительные действия в обработчики событий, что нега-

тивно сказывается на производительности.

КОДУ необходимо защитить от описанных выше угроз: системные вызовы должны обрабатываться антивирусом, запретить модификацию файлов КОДЫ, она должна быть скрыта от попыток поиска и выгрузки. При этом избыточная нагрузка на производительность процессора и работы с жестким дискам не должна превышать 10%.

1. Постановка задачи

Целью данной работы является разработка системы защиты для антивируса КОДА.

Для её достижения были поставлены следующие задачи:

- разработать систему защиты обработчика системных вызовов, устанавливаемого КОДОЙ;
- защитить файлы антивируса от программ, обходящих контроль учетных записей Windows;
- защитить КОДУ от поиска систем защиты;
- провести тестирование производительности разработанной системы.

2. Обзор существующих решений

Поскольку предметная область данной работы — гипервизорная защита, в данном разделе рассматриваются похожие системы, созданные для разных целей: комплексная защита, защита памяти, защита исполняемого кода.

2.1. Shadow-Box

Гипервизорный фреймворк Shadow-Box для защиты ядра Linux, разработанный в National Security Research Institute [4], использует два способа мониторинга процессов гостевой операционной системы: событийно-ориентированый и периодические проверки. На рис. 1 показана архитектура системы: логически она состоит из двух изолированных частей, но компонент гипервизора Light-Box позволяет обоим компонентам работать в одном ядре. Тем самым снижается избыточная нагрузка на систему и уменьшается семантическое различие между гипервизором и ОС.

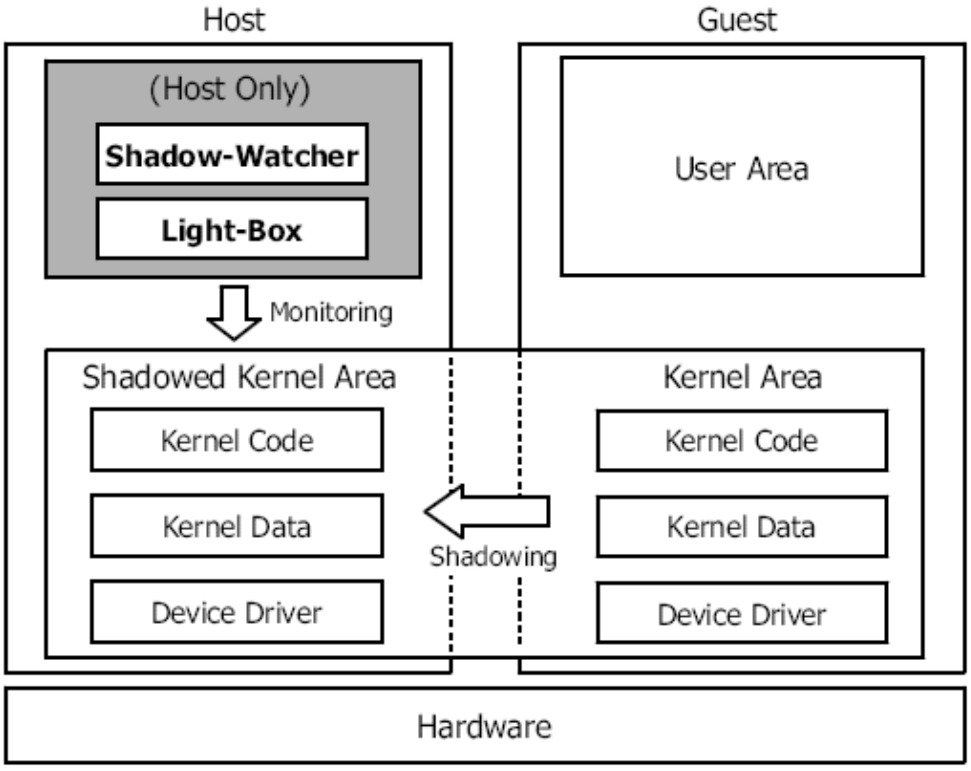

Рис. 1: Архитектура Shadow-Box

Безопасность работы гипервизора в ядре гостевой ОС обеспечивается следующими способами: изначальная проверка безопасности при загрузке, маскировка физических страниц процесса, защита регистров GDTR, LDTR, IDTR, MSR и независимый поток управления.

С помощью периодического анализа проверяются различные системные объекты, например, списки работающих модулей. На рис. 2 показано, что гипервизор копирует и сохраняет у себя необходимые списки, а функции модификации списков изменены таким образом, что изменяют списки в гипервизоре. Такой подход защищает от руткитов, которые скрывают свое присутствие с помощью удаления своих записей из этих списков.

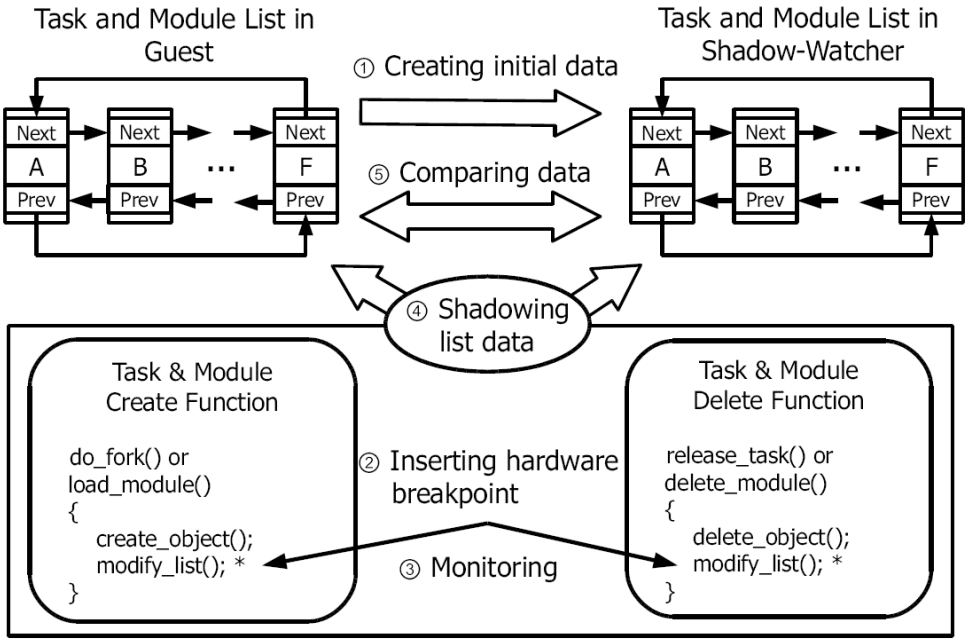

Рис. 2: Мониторинг списков задач в Shadow-Box

Тесты, проведенные создателями, показали увеличение нагрузки на 5.3% в системе с одним ядром и на 6.2% в в многоядерной системе.

2.2. AllMemPro

Гипервизорная система защиты AllMemPro [5], работающая в ОС Windows, предназначена для активной защиты данных, но не позволяет работать с процессами системы.

Фреймворк состоит из трех частей: контроллер, переключатель и диспетчер. Контроллер позволяет защитить загрузку драйверов и аллокацию памяти, защита осуществляется с помощью проверки контрольной суммы CRC.

Переключатель управляет способом защиты: гипервизорный и управление таблицами страниц. При работе второго способа выполняется мониторинг всех страниц ядра системы, но обращения к страницам, занятым приложениями, которые мы дожны защитить, дополнительно обрабатываются диспетчером. Если защита осуществляется с помощью гипервизора, то для нужной области памяти ставится специальный флаг-ловушка, и при обращении к этому участку памяти гипервизор дополнительно проверяет права доступа.

Диспетчер осуществляет логику доступа к защищаемой памяти и проверяет права доступа.

Поскольку фреймворк осуществляет проверки при каждом обращении к памяти, скорость работы замедляется в 5 раз в режиме гипервизора.

2.3. HyperForce

Фреймворк HyperForce [6] сочитает в себе особенности обычных систем защиты и средств на основе гипервизора и используется для выполнения критических по безпасности участков кода.

Несмотря на то, что защищаемый код запускается в гипервизоре, ему доступны все ресурсы гостевой операционной системы, например память и API ядра. Так как не надо передавать все необходимые параметры, значительно увеличивается производительность выполнения защищенной части.

Критический код инкапсулируется в функцию, которая загружается в виде модуля ядра Linux в виртуализированную операционную систему. HyperForce создает виртуальное устройство, которое эмулирует некоторое настоящее устройство, например, видеокарту. Такой подход поддерживается технологией Virtual Machine Monitor (VMM). После

того как виртуальное устройство создается загружается в ядро Linux, HyperForce регистрирует адрес функции, которая работает с защищенным кодом, как обработчик прерывания нового устройства.

Так как виртуальное устройство контролируется гипервизором, именно гипервизор, а не гостевая ОС, решает, когда генерируются прерывания. Такой подход защищает от попыток вызова защищенного кода из скомпроментированного ядра ОС, так как из гостевой ОС невозможно понять, когда именно будет сгенерировано прерывание и будет выполнен критическая часть.

Тестирование с помощью архивирования методом bunzip показало увеличение нагрузки на 5%.

3. Описание используемых инструментов и библиотек

3.1. HyperPlatform

HyperPlatform — гипервизор с открытым исходным кодом [7], использующий технологию виртуализации Intel VT-x. Разрабатывается для исследований в операционной системе Windows, в частности для исследования руткитов и реверс-инжиниринга ядра.

Гипервизор позволяет отслеживать множество событий в гостевой операционной системе, такие как:

1. доступ к виртуальной/физической памяти;
2. обращения к системным регистрам;
3. срабатывания прерываний и вызов обработчиков;
4. исполнение конкретной инструкции процессора.

Был выбран именно этот гипервизор, так как он работает в Windows 7, 8.1 и 10 под архитектурами x86 и x64. Единственное дополнительное условие — должна поддерживаться процессором и быть включена технология Intel-VT.

3.2. Windows Driver Model

Фреймворк Windows Driver Model (WDM) [8] разрабатывается компанией Microsoft для унификации драйверов в операционной системе Windows. WDM поддерживает три модели драйверов:

1. Драйвер шины — драйвер устройства с собственной шиной ввода/вывода, например с интерфейсами PCI, PnpISA, SCSI или USB. Обязанности драйвера:
    - знание о текущих устройствах, подключенных к шине;

- реализация Plug and Play;
- общение устройств на шине;
- передача устройству на шине команд от функционального драйвера.

2. Функциональный драйвер — драйвер для конкретного устройства, обычно поставляется производителем.

3. Драйвер-фильтр — опциональный драйвер, который может модифицировать поведение устройства. Бывают трех видов:

   - фильтр шины;
   - низкоуровневый фильтр, модифицирующий поведение аппаратной части устройства;
   - высокоуровневый фильтр, предоставляющий новую функциональность для устройства.

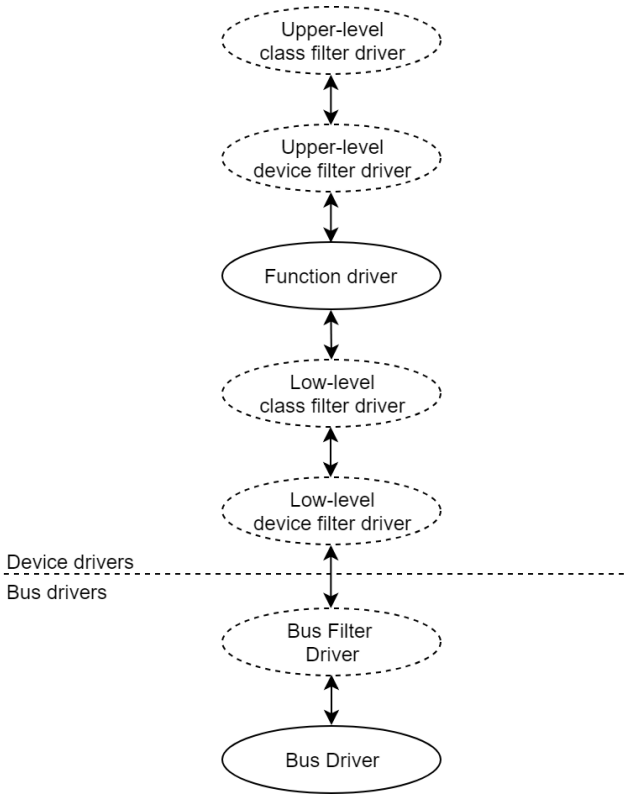

Рис. 3: Взаимодействие в WDM

Схема взаимодействия драйверов показана на рис. 3.
Драйверы-фильтры для файловой системы разделяются на

- legacy драйвера — устаревшая модель драйвера, заблокирована по умолчанию в Windows 7 и поздних версиях;
- minifilter драйвера.

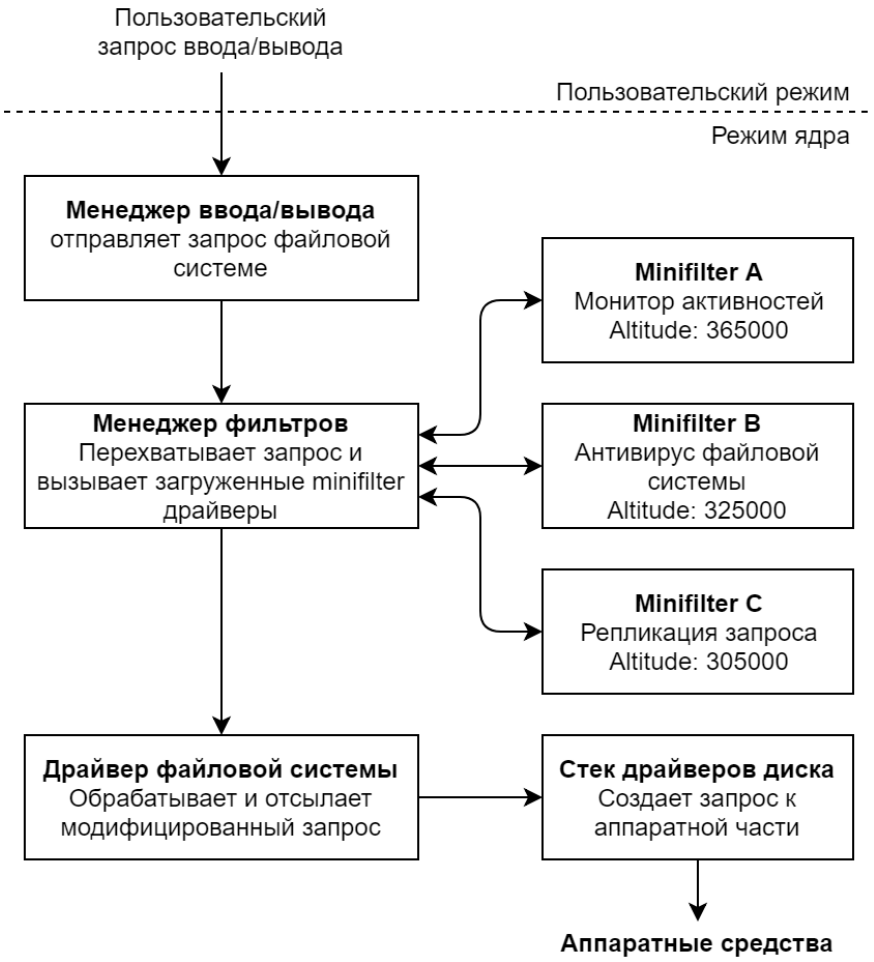

Рис. 4: Обработка запроса к файловой системе

Как показано на рис. 4, minifilter драйвера фильтруют I/O request packet (IRP) события файловой системы. Для каждой операции драйвер может включить обработчики двух видов:

- Preoperation Callback Routine — функция, которая будет вызываться при обращении к файловой системе до обработки его на устройстве;

- Postoperation Callback Routine — обработчик, вызываемый после завершения операции ввода/вывода на устройстве.

Каждый драйвер имеет параметр высоты (Altitude), который задает порядок вызова minifilter драйверов. Функции Preoperation Callback Routine вызываются по убыванию значения этого параметра. Postoperation Callback Routine — в обратном порядке, от меньшего к большему.

4. Реализация

4.1. Общая архитектура

Архитектура системы с разделением по кольцам защиты показана на рис. 5.

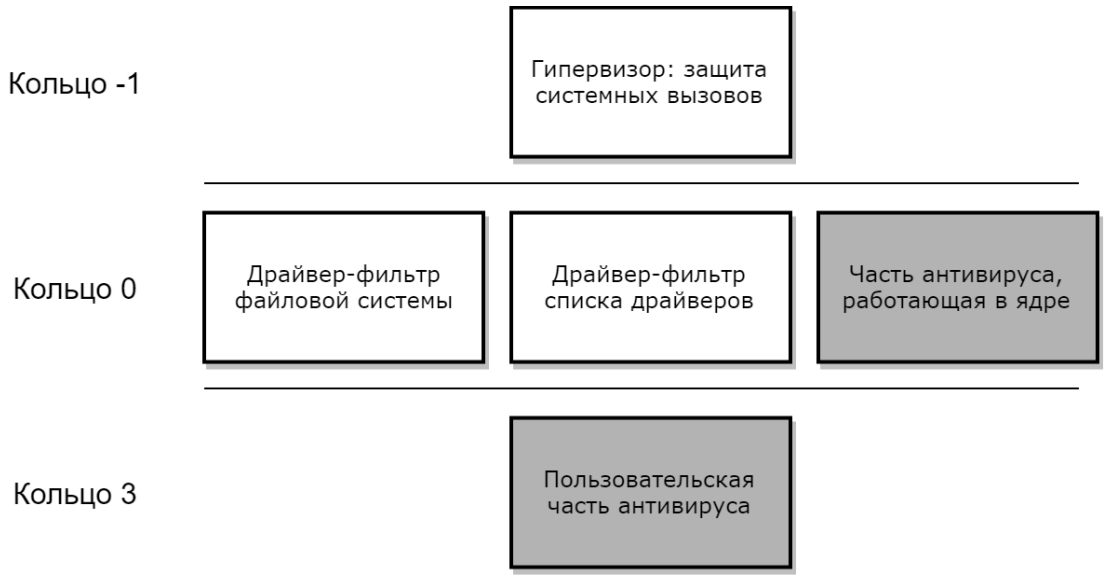

Рис. 5: Архитектура

Серым цветом выделены части антивируса КОДА: в ядре системы находится функциональная часть, которая обрабатывает и анализирует системные вызовы, в 3 кольце — часть для взаимодействия с пользователем.

В ядре системы отмеченные белым цветом элементы защиты антивируса: драйвер-фильтр файловой системы и драйвер-фильтр списка драйверов. Они были реализованы в виде отдельных драйверов, так как пользователю может быть критична производительность системы, и он мог отключить отдельные элементы защиты.

В -1 кольце находится гипервизор, обеспечивающий корректное обращение к обработчику системных вызовов.

4.2. Гипервизорная защита

Процессорная иснструкция SYSCALL, предназначенная для вызова обработчика системных вызовов, была реализована в архитектуре AMD64 как замена инструкции SYSENTER и вызовам через программное прерывание, например int 80h в ОС Linux.

Она использует три моделезависимых регистра (MSR), в которых находится информация об обработчике системных вызовов:

- в IA32_LSTAR MSR находится указатель на инструкции функции (значение, которое будет установлено в регистр RIP);
- IA32_CSTAR MSR содержит значения сегментов кода и стека (CS и SS);
- IA32_FMASK MSR — значения флагов, которые будут установлены в регистр RFLAGS.

Чтение в MSR регистров осуществляется с помощью команды RDMSR, принимающей на вход номер MSR регистра в ECX и возвращающей значение в EDX:EAX. Команда WRMSR наоборот, записывает значения EDX:EAX в MSR регистр, указанный в ECX. В 64-битном режиме старшие разряды регистров игнорируются. Исполнение этих инструкций требует 0 уровень привилегий, в противном случае возникает ошибка типа General Protection Fault.

Для изменения обработчика системных вызовов необходимо поменять значения в регистрах IA32_LSTAR и IA32_CSTAR, это происходит при старте КОДЫ. Система Kernel Patch Protection делает периодические проверки этих регистров.

С помощью гипервизора можно вмешаться в процесс обращения к MSR регистрам, так как RDMSR и WRMSR — инструкции условного выхода из виртуальной машины. Для этого необходимо выставить значения 1 в битовых картах структуры VMCS (virtual-machine control data structure) для чтения и записи.

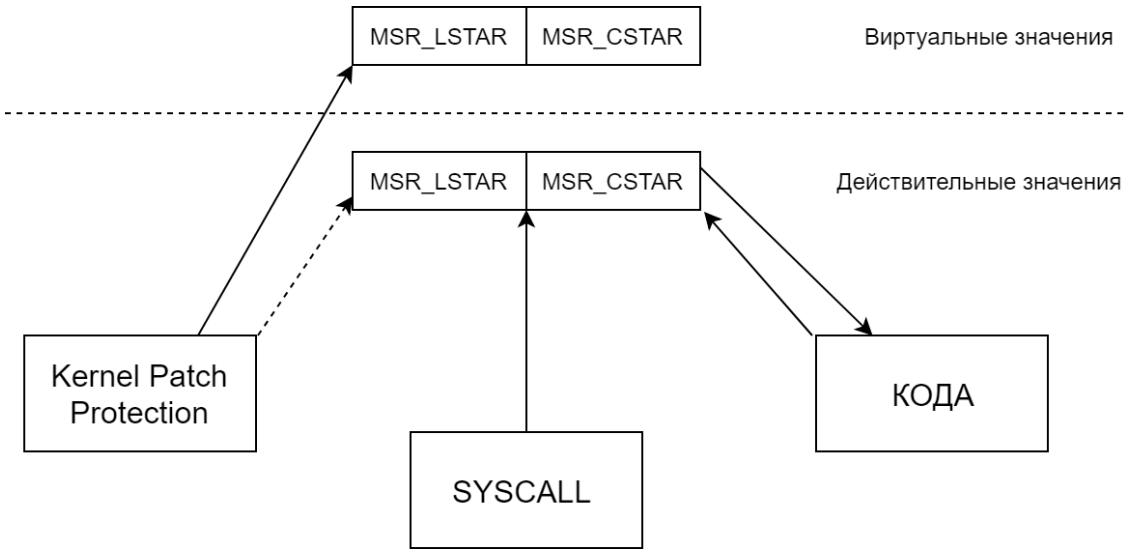

Рис. 6: Защита системных вызовов

Во время события VM Exit известна причина выхода из виртуальной машины, были добавлены обработчики обращений к необходимым регистрам.

Схема работы защиты показана на рис 6. Для регистров IA32_LSTAR и IA32_CSTAR в гипервизоре хранятся значения, которые были записаны в них до модификаций. Они возвращаются всегда при чтении, если к ним обращается не КОДА. Для антивируса есть возможность читать действительные значения в регистрах и писать в них.

Чтобы понять, что именно КОДА обращается к регистрам, гипервизор проверяет, выставлены ли флаги. В качестве флагов используются регистры общего назначения, не задействованные в операциях чтения и записи. Значения регистров и информация, какие из них использовать для флагов, задается на этапе компиляции. Команда SYSCALL будет использовать актуальные значения регистров, поэтому системные вызовы будут обрабатываться функцией, которую устанавливает КОДА.

Тестирование с КОДОЙ показало, что проблема со срабатыванием системы Kernel Patch Protection была устранена.

4.3. Драйвер-фильтр файловой системы

Для того, чтобы злоумышленники не могли прочитать или изменять файлы КОДЫ, обращения к файлам и папкам антивируса должны фильтроваться независимо от уровня привилегий учетной записи пользователя.

Реализованный minifilter драйвер фильтрует обращения к файлам и папкам, заданным на этапе компиляции. Для следующих событий в файловой системе были реализованы функции Preoperation Callback Routine:

- IRP_MJ_DIRECTORY_CONTROL — доступ к директории, делится на 2 типа:
  - IRP_MN_NOTIFY_CHANGE_DIRECTORY — событие изменения в папке;
  - IRP_MN_QUERY_DIRECTORY — получение файлов и папок в директории.
- IRP_MJ_CREATE — создание нового файла или директории;
- IRP_MJ_WRITE — событие записи в файл;
- IRP_MJ_READ — чтение из файла.

В обработчиках данных событий сравнивается полный путь до исполняемого файла и путь, к которому обращается программа. Если обращение происходит не к одному из необходимых объектов, или программа находится внутри одной из необходимых папок, то возвращается значение FLT_PREOP_SUCCESS_NO_CALLBACK, которое означает, что функция отработала корректно и у нее нет Postoperation Callback Routine. В противном случае — FLT_PREOP_COMPLETE, при котором считается, что запрос IRP обработан, и он не передается дальше другим драйверам и жесткому диску.

Информация о полном пути исполняемого файла получается с помощью функции ZwQueryInformationProcess, которая принимает на вход значение информацию о процессе, полученную из NtCurrentProcess.

Значение параметра Altitude для данного драйвера не важно, поскольку результат ограничения доступа не зависит от порядка вызова обработчика.

Тестирование проводилось с помощью приложения, которое читает указанные файлы файлы и пишет в них. У него достаточно привилегий для доступа к ним, так как можно считать, что вирус обошел этот уровень защиты.

4.4. Драйвер-фильтр списка драйверов

Данный драйвер использует метод прямого изменения объектов ядра (Direct kernel object manipulation, DKOM).

Одним из аргументов функции установки драйвера является структура PDRIVER_OBJECT. Его поле DriverSection содержит официально недокументированную структуру LDR_DATA_TABLE_ENTRY. Не смотря на это, во всех текущих версиях Windows она одинакова.

В поле InLoadOrderLinks (самое начало структуры) находится связный список всех загруженных драйверов. Из него извлекаются все вхождения, связанные с КОДОЙ. Имена этих драйверов известны на этапе компиляции.

При выгрузке драйвера все удаленные элементы заново включаются в список драйверов.

Такой подход скрывает системы защиты от обнаружения с помощью функции NtQuerySystemInformation с аргументом SystemModuleInformation (элемент перечисления SYSTEM_INFORMATION_CLASS). В данном случае она возвращает список всех загруженных драйверов.

4.5. Тестирование производительности

Тестирование производительности проводилось двумя методами:

1. встроенным в Windows средством оценки индекса производительности компьютера. Консольная команда winsat formal запускает тестирование следующими способами:

- тестирование диска с помощью последовательного и случайного чтения;
- оценка производительности процессора алгоритмами хеширования и сжатия;
- тестирование видеокарты и библиотеки Direct3d.

2. тестирование браузера с помощью Octane 2.0 JavaScript Benchmark. Сайт оценивает производительность выполнения JavaScript и Typescript кода, открытие PDF, работу с регулярными выражениями, массивами и числами с плавающей точкой.

Таблица 1: Результаты тестирования winsat formal

	Производительность процессора	Производительность жесткого диска
Среднее выборочное без системы защиты	414.59 МБайт/с	105.51 МБайт/с
Выборочная дисперсия без системы защиты	7.17 (МБайт/с)^2	6.41(МБайт/с)^2
Длина доверительного интервала без системы защиты	1.21	1.144
Среднее выборочное с системой защиты	403.84 МБайт/с	99.28 МБайт/с
Выборочная дисперсия с системой защиты	6.09 (МБайт/с)^2	5.90 (МБайт/с)^2
Длина доверительного интервала с системой защиты	1.116	1.098
Избыточная нагрузка	2.59%	5.90%

Тестирование обоими способами было запущено по 10 раз с вклю-

ченной системой защиты и 10 раз с выключенной.

Таблица 2: Результаты тестирования браузера

	Производительность браузера
Среднее выборочное без системы защиты	22426
Выборочная дисперсия без системы защиты	1340274.22
Длина доверительного интервала без системы защиты	529.294
Среднее выборочное с системой защиты	20873.8
Выборочная дисперсия с системой защиты	1103975.28
Длина доверительного интервала с системой защиты	503.284
Избыточная нагрузка	6,92%

Результаты тестирования показаны в таблице 1, в качестве производительности процессора была выбрана скорость сжатия методом LZW, для дисков — последовательное чтение. В таблице 2 показано изменение производительности браузера, единицы измерения результата — очки Octane 2. Избыточная нагрузка считалась как отношение выборочных средних. Доверительный интервал считался для распределения Стьюдента с доверительной вероятностью 0.975.

Сравнение с другими системами защиты по накладным расходам на производительность процессора показаны в таблице 3.

Тестирование проводилось на компьютере с процессором Intel(R) Core(TM) i5-2500 2.70GHz, 4 Гб ОЗУ, жестким диском Seagate Barracuda

Таблица 3: Сравнение избыточных нагрузок

система защиты	КОДА	HyperPlatform	AllMemPro	HyperForce
процессорное время, %	4%	6.2%	-	5%

ST31000524AS, видеокартой NVIDIA GeForce 210 и операционной системой Windows 10 x64.

Заключение

В результате данной работы является была разработана система защиты для антивируса КОДА.

Были решены следующие задачи:

- разработана система защиты обработчика системных вызовов на основе гипервизора;
- реализован драйвер-фильтр, защищающий файлы антивируса от программ, которые обходят контроль учетных записей Windows;
- реализован драйвер-фильтр, скрывающий КОДУ от поиска и выгрузки;
- проведено тестирование производительности разработанной системы, тесты показали досустимое значение дополнительных накладных расходов.

Список литературы

[1] Raymond Canzanese, Spiros Mancoridis, Moshe Kam. System Call-based Detection of Malicious Processes – 2015 IEEE International Conference on Software Quality, Reliability and Security. – URL: http://ieeexplore.ieee.org/abstract/document/7272922/ (дата обращения: 04.05.2019).

[2] Manuel Egele, Theodor Scholte, Engin Kirda, Christopher Kruegel. SA Survey on Automated Dynamic Malware-Analysis Techniques and Tools. – URL: https://www.sba-research.org/wp-content/uploads/publications/malware_survey.pdf (дата обращения: 04.05.2019).

[3] Intel 64 and IA-32 Architectures Developer's Manual: Combined Vols. 1, 2, and 3. Intel, 2018.

[4] Kelly S., Tolvanen J.-P. Myth and Truth about Hypervisor-Based Kernel Protector: The Reason Why You Need Shadow-Box. — Black Hat Asia 2017 — URL: https://www.blackhat.com/docs/asia-17/materials/asia-17-Han-Myth-And-Truth-about-Hypervisor-Based-Kernel-Protect
pdf (дата обращения: 04.05.2019).

[5] Korkin I. Hypervisor-Based Active Data Protection for Integrity and Confidentiality of Dynamically Allocated Memory in Windows Kernel. — 13th annual Conference on Digital Forensics, Security and Law — URL: https://arxiv.org/ftp/arxiv/papers/1805/1805.11847.pdf (дата обращения: 04.05.2019).

[6] Francesco Gadaleta, Nick Nikiforakis, Jan Mühlberg, Wouter Joosen. HyperForce: Hypervisor-enForced Execution of Security-Critical Code. — 27th Information Security and Privacy Conference (SEC) – – URL: https://arxiv.org/pdf/1405.5648.pdf (дата обращения: 04.05.2019).

[7] Репозиторий HyperPlatform. — URL: https://github.com/tandasat/HyperPlatform/tree/master/HyperPlatform (дата обращения: 04.05.2019).

[8] Документация Windows Driver Model. — URL: https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/windows-driver-model (дата обращения: 04.05.2019).

[9] Репозиторий. — URL: https://github.com/Ivan-Nebogatikov/WindowsHypervisorProtection (дата обращения: 04.05.2019).