На правах рукописи

Фурсова Наталья Игоревна

МЕТОДЫ МОНИТОРИНГА ОБЪЕКТОВ ОПЕРАЦИОННОЙ СИСТЕМЫ, ВЫПОЛНЯЮЩЕЙСЯ В ВИРТУАЛЬНОЙ МАШИНЕ

Специальность 05.13.11 —
«Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей»

Автореферат

диссертации на соискание учёной степени
кандидата технических наук

Великий Новгород — 2017

Работа выполнена в Новгородском государственном университете имени Ярослава Мудрого

Научный руководитель: кандидат технических наук, доцент
Макаров Владимир Алексеевич

Официальные оппоненты: **Ильин Вячеслав Анатольевич,**
доктор физико-математических наук,
начальник отдела Курчатовского комплекса НБИКС-технологий Национального исследовательского центра «Курчатовский институт»

Козачок Александр Васильевич,
кандидат технических наук,
сотрудник Федерального государственного казенного военного образовательного учреждения высшего образования «Академия Федеральной службы охраны Российской Федерации» (г. Орел)

Ведущая организация: Межведомственный суперкомпьютерный центр Российской академии наук — филиал Федерального государственного учреждения «Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук»

Защита состоится 21 декабря 2017 г. в 16:00 часов на заседании диссертационного совета Д 002.087.01 на базе Института системного программирования имени В.П. Иванникова РАН по адресу: 109004, Москва, ул. Александра Солженицына, д. 25.

С диссертацией можно ознакомиться в библиотеке Федерального государственного бюджетного учереждения науки Института системного программирования им. В.П. Иванникова РАН.

Автореферат разослан «____» _____ 2017 года.

Ученый секретарь
диссертационного совета
Д 002.087.01, к-т физ.-мат. наук Зеленов Сергей Вадимович

Общая характеристика работы

Актуальность. Безопасность программного обеспечения связана с анализом недокументированных возможностей и вредоносного внешнего влияния. Программы как правило поставляются производителем в виде бинарного кода (исполняемых файлов), а их исходные коды при этом обычно недоступны. Анализ бинарного кода сопряжен с решением сложных задач, но взамен дает лучшее понимание за счет того, что анализируется именно тот код, который будет выполняться.

Существует много способов исследования бинарного кода, например, анализ помеченных данных (taint-анализ), поведенческий анализ, трассировка выполнения машинных команд. Для этого требуется наличие высокоуровневой информации об исследуемых приложениях и операционных системах (открываемые файлы, работающие процессы, вызываемые функции, загружаемые модули), получение которой сопряжено с проблемой, называемой семантический разрыв (semantic gap).

Семантический разрыв — это разрыв в представлении наблюдаемых в системе данных и данных, требуемых для проведения анализа. Например, анализатор требует информацию о файлах, а имеется только поток инструкций. В этой работе поведение операционных систем (ОС) исследуется в виртуальной машине, поэтому вопросы мониторинга на реальном аппаратном обеспечении не рассматриваются. Выполняемая в виртуальной машине ОС называется гостевой.

Виртуальная машина предоставляет для анализа поток инструкций виртуализированного процессора и ее оперативную память. Выделить из этого потока высокоуровневую информацию вручную практически невозможно. Поэтому возникает необходимость в автоматических методах мониторинга высокоуровневых событий, например, операций с файлами и процессами. Эти события будут накладываться на низкоуровневую трассу машинных команд, улучшая как понимание происходящего в исследуемой системе, так и поддержку работы других методов анализа: анализа помеченных данных, выявления аномалий в поведении и других.

Передовыми исследователями в области мониторинга объектов ОС являются команды, реализующие такие инструменты, как PANDA, DECAF, RTKDSM. Методы, заложенные в эти решения, имеют особенности: использование программ-агентов — приложений, загружаемых в систему для сбора данных о структурах, адресах функций и других данных.

Исходя из особенностей, можно определить что такие подходы имеют трудности с анализом систем с закрытым исходным кодом, а также с системами, не предусматривающими загрузку в них программ извне.

Несмотря на описанные особенности, эти инструменты можно использовать для анализа ОС настольных компьютеров, таких как Windows и Linux, но их тяжело применять для анализа встроенных систем. Как правило, исследования встроенных систем проводятся в виртуальной среде, в которую загружается ПО, извлеченное из ПЗУ исследуемого устройства. В этом случае подход с программами-агентами будет крайне трудозатратен или невозможен по причине того, что бинарный код агента может быть несовместим с системой (зачастую версия ОС и параметры сборки неизвестны), а исходный код в таких системах невозможно скомпилировать.

Чтобы обойти трудности с использованием программы-агента, необходимо использовать методы мониторинга, не предполагающие внедрение инструментального кода в окружение, в котором выполняется исследуемая программа.

В соответствии с вышесказанным можно выделить ряд требований, которые должны учитываться в новом методе мониторинга объектов ОС. Во-первых, не использовать загрузку в нее сторонних модулей, во-вторых, обеспечить работу анализатора с семействами исследуемых систем без внесения изменений в алгоритмы анализа. При выполнении этих требований также появляется возможность анализировать образы систем, извлеченных из ПЗУ. Под семействами ОС понимаются системы, разработанные на одном ядре, например, к семейству Windows относятся все пользовательские версии этой операционной системы.

В диссертационной работе предлагаются методы мониторинга объектов операционной системы без использования программных агентов. Объектами ОС в контексте этой работы называются файлы, процессы, модули, вызываемые функции. Как правило, объекты имеют характерные атрибуты, например, для файла это имя и дескриптор, для процесса идентификатор и адресное пространство. Для работы предлагаемых методов не требуются знания о внутреннем устройстве системы, а необходимая информация получается из открытых описаний программных интерфейсов. Кроме того, набор данных, собранный для одной ОС, применим ко всему семейству этой ОС.

Целью диссертационной работы является исследование и разработка методов и инструментальных средств, реализующих мониторинг объектов гостевой операционной системы при ее выполнении в полносистемном программном эмуляторе. Разработанные методы должны обеспечивать конфигурирование и работу анализатора без загрузки инструментального кода

в исследуемую систему. Единожды сконфигурированный анализатор должен работать для операционных систем из одного семейства без перенастройки.

Для достижения поставленной цели необходимо решить следующие **задачи**:

1. Предложить модель для операционных систем общего назначения, работающих на настольных компьютерах, мобильных устройствах и коммуникационном оборудовании.
2. Разработать метод мониторинга событий виртуальной машины для получения информации об объектах ОС без внедрения инструментального кода в исследуемую систему.
3. Разработать метод вызова системных функций по запросу анализатора для получения заданных атрибутов объектов ОС.
4. Реализовать инструмент для мониторинга объектов ОС по предложенным методам в виде плагинов к эмулятору QEMU.
5. Провести функциональное тестирование разработанного инструмента, определить масштаб накладных расходов, а также проверить достоверность получаемых результатов.

Научная новизна:

1. Предложен новый метод получения информации об объектах ОС через мониторинг событий виртуальной машины, позволяющий анализировать встроенные системы, а также семейства ОС.
2. Предложен новый метод получения заданных атрибутов объектов ОС по запросу анализатора через встраивание вызовов системных функций в поток инструкций виртуальной машины.

Практическая и теоретическая ценность работы. Разработанные методы представляют интерес для разработчиков инструментов динамического анализа, требующих для своей работы высокоуровневую информацию, например, анализ помеченных данных, поведенческий анализ, кроме того, они подходят для анализа встроенных систем. Такими работами занимается Институт системного программирования РАН.

Также представленные методы могут использоваться в исследованиях по информационной безопасности, а разработанный инструмент — лабораториями, проводящими исследования поведения программных систем.

Методология и методы исследования. Результаты диссертационной работы получены на базе использования методов динамического анализа и обратной инженерии бинарного кода, системного анализа, а также математического моделирования. При проведении экспериментов и оценке их результатов применялись методы теории вероятностей.

Основные положения, выносимые на защиту:

1. Метод мониторинга событий виртуальной машины для получения информации об объектах гостевой операционной системы без внедрения инструментального кода на уровне исследуемой системы.
2. Метод вызова системных функций по запросу анализатора для получения заданных атрибутов объектов операционной системы.
3. Инструментальная среда, работающая с тремя семействами операционных систем общего назначения (Windows, Linux, FreeBSD).

Апробация работы. Основные результаты работы опубликованы в статьях [1-7], из них работы [1-4] опубликованы в изданиях из перечня ВАК, статьи [2, 6, 7] индексированы Scopus. В статье [5] представлена концептуальная модель разработанного автором метода мониторинга объектов ОС. В статье [6] описаны предложенные автором методы и их реализации для мониторинга объектов операционных систем. В статьях [1, 2, 7] описан разработанный автором метод мониторинга событий виртуальной машины для получения информации об объектах ОС. В статье [3] описаны разработанные автором плагины для перехвата системных вызовов, отслеживания файлов и процессов. В работе [4] упоминается разработанное автором средство мониторинга файлов, используемое для осуществления анализа помеченных данных.

Результаты работы обсуждались на следующих конференциях:

1. Международная конференция РусКрипто, Солнечногорск, Россия, 25-28 марта, 2014.
2. Открытая конференция по компиляторным технологиям, Москва, Россия, 2 декабря, 2015.
3. SAC '16 Proceedings of the 31st Annual ACM Symposium on Applied Computing, Пиза, Италия, 4-8 апреля, 2016.
4. Международная конференция РусКрипто, Солнечногорск, Россия, 21-24 марта, 2017.
5. Международная Ершовская конференция по информатике PSI–2017, Москва, Россия, 27–29 июня 2017.
6. Научно-исследовательский семинар Института системного программирования РАН.
7. 11th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2017).

Личный вклад. Все представленные в диссертации результаты получены лично автором.

Объем и структура работы. Диссертация состоит из введения, пяти глав, заключения и приложения. Полный объем диссертации составляет

120 страниц текста с 10 рисунками и 9 таблицами. Список литературы содержит 54 наименования.

Краткое содержание работы

Во **введении** обосновывается актуальность исследований, проводимых в рамках данной диссертационной работы, формулируется цель, ставятся задачи работы, сформулированы научная новизна и практическая значимость представляемой работы.

Первая глава посвящена обзору работ, имеющих непосредственное отношение к теме диссертации. Рассматриваются существующие подходы и программные средства, связанные с мониторингом объектов исследуемой системы. Для решения этой проблемы используют статический и динамический анализ.

В разделе 1.1 описываются характеристики, которыми должны обладать инструменты для мониторинга объектов ОС вне зависимости от области применения. В их число входит минимизация вмешательства в исследуемую систему и гипервизор. Также необходимо оказывать минимальное влияние на производительность и, по возможности, присутствие в системе анализирующего инструмента не должно быть заметно.

В разделе 1.2 представлена классификация методов мониторинга объектов ОС существующих на данный момент. Выделяются две большие группы: методы, использующие статический анализ и методы, использующие динамический анализ. Так же выделены такие критерии, как требования к исследуемой системе, область применения и извлекаемая информация.

В разделе 1.2.1 рассмотрены методы и инструменты, использующие статический анализ. Наиболее важным инструментом этой категории является Volatility. Этот фреймворк обладает рядом положительных качеств, таких как кроссплатформенность, модульное строение, возможность расширения, открытая архитектура. Поддерживает разные операционные системы и процессоры. Работает со снимками оперативной памяти.

Также в этой группе представлены инструменты FatKit и InSight. Эти инструменты имеют ограниченный, по сравнению с Volatility, ряд возможностей.

В разделе 1.2.2 представлены методы и инструменты, использующие методы динамического анализа. Здесь следует выделить такие работы, как DECAF, PANDA и RTKDSM.

DECAF представляет собой фреймворк для полносистемного бинарного анализа, основанный на QEMU. Предлагает мониторинг объектов JIT,

совмещенное с анализом помеченных данных. Имеет модульную структуру и открытый исходный код.

PANDA — основанный на QEMU инструмент, состоящий из плагинов, которые могут быть загружены в любой момент. Для своей работы требует загрузки программы-агента в исследуемую систему. Для каждой версии каждой операционной системы программа-агент генерирует конфигурационный файл.

RTKDSM использует в своей работе некоторые плагины Volatility, но исследует память напрямую, не используя дампы. Является фреймворком с открытым исходным кодом, упрощает и автоматизирует анализ состояний исследуемой системы. Работает на основе гипервизора Xen, что ограничивает используемые архитектуры до x86. Поддерживает операционные системы Windows и Linux.

Все вышеописанные инструменты получают данные, используя структуры ядра исследуемой системы.

В разделе 1.2.4 приведена классификация методов мониторинга объектов ОС и представлены характеристики инструментов-аналогов в виде таблицы. Ввиду большого объема, в тексте автореферата приведена сокращенная версия классификации, и в таблице 1 представлен сравнительный анализ ограниченного набора инструментов.

Классификация методов мониторинга объектов операционной системы:

- Метод мониторинга объектов ОС (Method - M)
  - Динамический анализ «на лету» (**MF**)
  - Динамический анализ по снимкам системы (**MD**)
  - Динамический анализ по записанным трассам (**MT**)
  - Статический анализ по снимкам системы (**MS**)
- Особенности метода (Feature - F)
  - Внедрение внешних модулей в систему (**FM**)
  - Наличие исходных кодов (**RS**)
  - Наличие описания ABI (**RI**)

Выводы по главе 1 представлены в разделе 1.3. Определены основные ограничения рассмотренных методов и инструментов:

1. Использование структур ядра. Данные о структурах ядра не всегда доступны, например, когда используются ОС с закрытым исходным кодом. Получение этой информации сопряжено со сложным анализом системы. Кроме того, в различных версиях эти структуры могут быть разными, следовательно, необходимо иметь профиль под каждую версию каждой операционной системы.

Таблица 1

Анализ инструментов для мониторинга объектов ОС

	Volatility	FATKit	InSight	PANDA	DECAF	RTKDSM	**Новый метод**
Метод мониторинга объектов ОС							
MF						+	+
MD					+	+	
MT				+			+
MS	+	+	+				
Особенности метода							
FM			+	+	+		
FS	+	+	+	+	+	+	
FI	+			+	+	+	+

2. Невозможность работы с неизменяемыми образами исследуемых систем. В связи с использованием программ-агентов, рассмотренные методы не предполагают анализ систем такого рода.
3. Трудности сопровождения. Учитывая изменяемость операционных систем, а именно их внутреннего устройства, возникают трудности с поддержкой работоспособности инструмента анализа. Чем меньше изменений терпят затронутые части исследуемой системы в процессе развития, тем проще сопровождать продукт для ее анализа.

Таким образом, возникает задача разработки методов мониторинга объектов ОС, обеспечивающих построение анализатора без использования исходных кодов и настроек компиляции исследуемой системы. Анализатор не должен загружать в исследуемую систему инструментальный код, а также должен обеспечивать переносимость между ОС одного семейства.

Вторая глава посвящена описанию модели исследуемой операционной системы.

В разделе 2.1 представлена модель исследуемой системы. Предлагаемая модель представляет собой набор сущностей и источников информации. Выделены такие сущности, как файл, процесс, адресное пространство, отображение, модуль. Для получения информации о сущностях используется поток инструкций, из которого выделяются системные вызовы и API вызовы. Описанный набор сущностей являются достаточным для решения задач, поставленных в этой работе, однако, при необходимости модель может быть расширена и другими компонентами систем, например, объектами синхронизации.

Сущность **Файл** представляет все использующиеся в работе исследуемой системы файлы, как системные, так и пользовательские.

Файл обладает следующими атрибутами: имя, дескриптор, права доступа. Так же эта сущность наделена рядом операций, а именно: создать, открыть, читать, писать, закрыть.

В контексте этой работы выделены два типа файлов: существующих на диске и открытых в данный момент одним из процессов.

Процесс — это отдельная программа, работающая в системе. Каждый процесс имеет свое виртуальное адресное пространство, описываемое таблицей трансляции адресов.

Например, в архитектуре x86 в качестве идентификатора адресного пространства выступает регистр `cr3`. Его значение уникально для каждого процесса. Благодаря этому регистру можно определить в каком процессе работает исследуемое приложение.

В некоторых операционных системах процессы представляют собой иерархию, где четко определены родитель и потомок, в других как таковой иерархии нет, но объединяет разные модели то, что все процессы изначально порождаются главным процессом системы.

Процессы обладают такими атрибутами, как: имя образа, идентификатор процесса. Из операций над процессами можно выделить создание процесса и его уничтожение.

Адресное пространство — набор адресов, который может быть использован процессом для обращения к памяти.

Каждый процесс в многозадачной операционной системе выполняется в своем виртуальном адресном пространстве. Соответствие между виртуальными и физическими адресами описывается с помощью таблицы страниц. Таблицы создает и заполняет ядро, а процессор обращается к ним при необходимости осуществить трансляцию адреса. Каждый процесс работает со своим набором таблиц.

Адресное пространство можно назвать неофициальным идентификатором процесса. Это сущность, которая связана с определенным процессом.

Отображение — это способ работы с файлами в некоторых операционных системах, при котором всему файлу или некоторой непрерывной его части ставится в соответствие определённый участок памяти. В контексте этой работы отображение является сущностью.

Атрибутами отображения являются: имя отображенного файла, дескриптор файла. Операции, связанные с отображениями: создать, открыть и удалить.

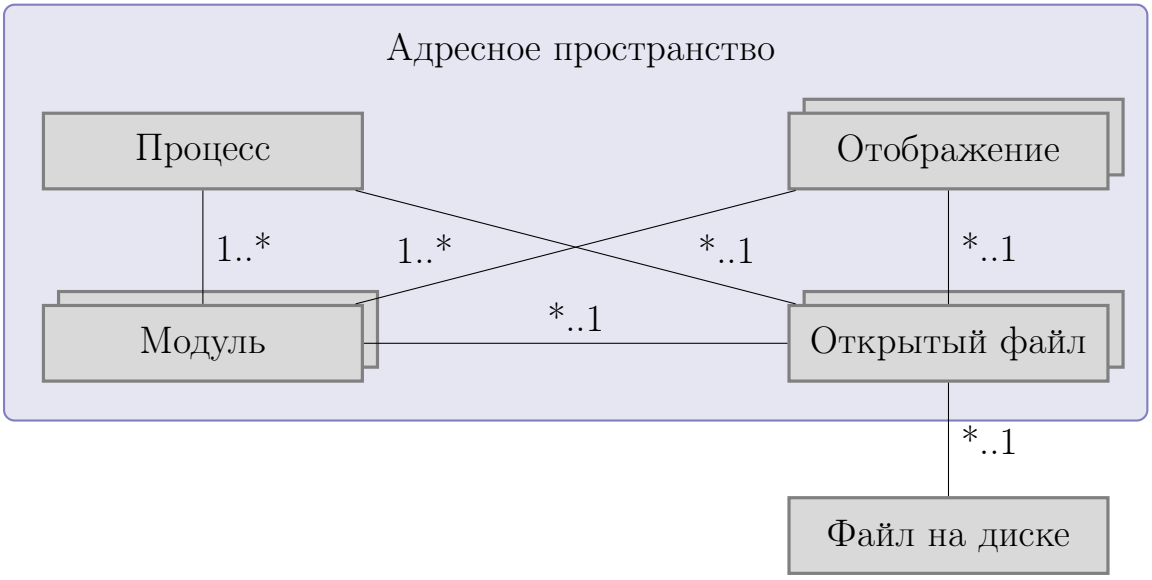

Рис. 1 — Структурная схема модели

Под **Модулем** понимается бинарный код, который загружается в систему динамически. В контексте этой работы модулями являются исполняемые файлы (*.exe), библиотеки (*.dll, *.so). Модули, как правило, предоставляют функции для вызова извне и отладочную информацию. В привычном виде для модулей не определены атрибуты и операции.

На рисунке 1 представлена структурная схема модели, определяющая взаимосвязь сущностей между собой.

Для поставленной в работе цели выделенных сущностей достаточно. Это подтверждается опытом рассмотренных в главе 1 инструментов динамического анализа (PANDA, DECAF, RTKDSM).

Входными данными для исследования является поток инструкций. Именно инструкции позволяют определить, что произошел системный вызов и только по адресу инструкции можно распознать нужный API вызов.

Системный вызов — это механизм, позволяющий взаимодействовать пользовательским программам с ядром исследуемой операционной системы. Системный вызов предполагает выполнение системной функции, которая по сути является обычной функцией с параметрами и возвращаем значением.

Системный вызов характеризуется следующими атрибутами: идентификатор системного вызова, имя системного вызова, параметры, возвращаемое значение. С системными вызовами связаны две операции: начать выполнение (например, инструкция sysenter), закончить выполнение (sysexit).

В разных операционных системах для начала и окончания системного вызова могут использоваться разные инструкции.

Поток данных в представленной модели изображен на рисунке 2.

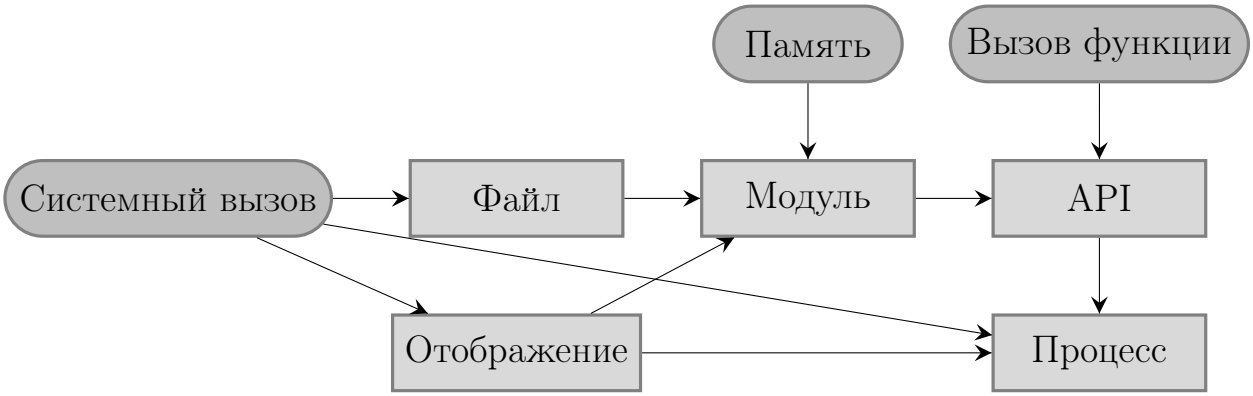

Рис. 2 — Схема потоков данных

В **третьей главе** описаны методы мониторинга объектов операционной системы, которые выносятся на зищиту.

В разделе 3.1 описан метод мониторинга объектов ОС, основанный на вышеописанной модели системы.

Ключевыми отличиями существующих методов являются затрагивание внутренних структур систем, а также использование программ-агентов. Первое требует доступа к исходным кодам исследуемой системы (что иногда проблематично), второе же не может быть использовано при анализе встроенных систем. Разрабатываемый метод должен обходить эти недостатки: использовать только те знания о системе, которые есть в открытом доступе, и не прибегать к программам-агентам.

Поэтому недокументированные структуры данных ядра системы использоваться не будет, вместо них задействован ABI (Application Binary Interface). ABI — это набор соглашений для доступа приложения к операционной системе и другим низкоуровневым сервисам, спроектированный для переносимости исполняемого кода между машинами, имеющими совместимые ABI. ABI можно рассматривать как набор правил, позволяющих компоновщику объединять откомпилированные модули компонента без перекомпиляции всего кода. Он регламентирует: использование регистров процессора, состав и формат системных вызовов и вызовов одного модуля другим, формат передачи аргументов и возвращаемого значения при вызове функции.

Для реализации подхода по вышеописанной модели необходимо иметь набор данных, предоставляемый ABI: как происходит системный вызов, идентификаторы системных вызовов, имена и параметры системных функций, размер указателя на область памяти, возвращаемое значение. Например, для ОС Windows на архитектуре x86 данные выглядят следующим образом: [sysenter, sysexit] — инструкции для входа в системный вызов и выхода из него, [cr3] — регистр, определяющий адресное пространство, [eax] — регистр, хранящий номер системного вызова на входе и возвращаемое значение функ-

ции на выходе, параметры функций передаются слева направо, коды системных вызовов и параметры функций определены документацией.

Идея метода заключается в том, чтобы получать высокоуровневую информацию из низкоуровневой, имея минимальные знания об исследуемой системе. Имеется поток инструкций, которые генерирует эмулятор QEMU, из этих инструкций отфильтровываются те, что отвечают за системные вызовы. Для этой операции необходимо знать опкоды (часть машинного языка, называемая инструкцией и определяющая операцию, которая должна быть выполнена) для начала и окончания системного вызова, идентификатор адресного пространства, значение указателя стека для верного сопоставления пары начало–конец, а также идентификаторы системных вызовов.

Системный вызов предполагает вызов системной функции, которая имеет параметры и возвращаемое значение. Далее из потока системных вызовов выбираются те, что представляют интерес, а именно вызовы, связанные с файлами, процессами, отображениями. Список интересующих вызовов может быть расширен по мере необходимости. При входе в системную функцию уже доступны входные параметры функций, но так как функция еще не выполнилась, то выходные параметры и возвращаемые значения сразу узнать не представляется возможным. Для их получения необходим перехват инструкции окончания системного вызова. Требуется сопоставить начало и конец выполнения функции, после чего можно считать выходные данные. Информация, полученная от системных функций (название функции, значения параметров и возвращаемых значений), собирается в журналы и может быть использована как самостоятельная единица анализа, как отправная точка для дальнейших исследований более сложных элементов системы, а также может быть передана другим анализаторам (например, анализу помеченных данных).

Для анализа некоторых частей системы, например, файловых операций, достаточно только системных функций. Но информации от системных функций недостаточно для получения всех параметров процессов. В этом случае необходимо исследовать библиотечные функции. Идея остается прежней — перехват интересующих функций из общего потока, однако теперь из общего потока необходимо выделять инструкцию вызова функции (например, call) и идентифицировать функции по адресу.

Таким образом можно получить высокоуровневую информацию о системе, например, журнал файловых операций, список запущенных процессов, список загруженных модулей, вызываемых функций.

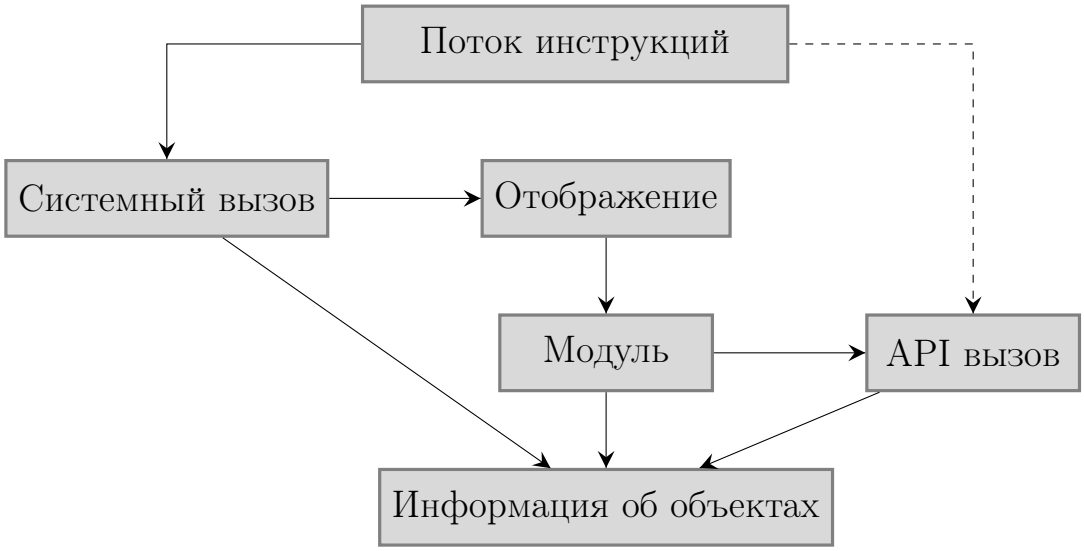

Рис. 3 — Схема процесса мониторинга объектов ОС

На рисунке 3 представлена схема процесса мониторинга объектов операционной системы. В следующих подразделах главы описан каждый шаг этой схемы.

Подраздел 3.1.1 посвящен описанию входных данных, необходимых для работы метода. Набор данных может быть получен с помощью документации или, в случае ее отсутствия, с помощью дизассемблирования.

Основным источником данных о системе является ABI. Из этого интерфейса необходимо получить следующую информацию: набор системных вызовов (количество системных вызовов, их идентификаторы, описание системных функций (параметры и возвращаемые значения)), соглашения о вызовах.

Помимо ABI необходимо определить инструкции для реализации системных вызовов, идентификаторы адресного пространства, форматы динамических библиотек, а также набор библиотечных функций.

Идентификатор адресного пространства важен, поскольку применяется для сопоставления начала и конца системной или библиотечной функции. В качестве этого идентификатора могут выступать некоторые регистры процессора.

Информацию из функций можно получить из двух источников: входные параметры и выходные параметры с возвращаемым значением. Некоторые функции предоставляют все необходимое на входе, тогда как другие имеют смысл только на выходе. В общем случае перехватывается выход, где можно получить информацию из обоих источников, однако, существуют функции, которые не возвращают управление (execve, ExitProcess), поэтому их отслеживать нужно только на входе. Как правило, такие функции предоставляют всю необходимую информацию во входных параметрах.

В подразделе 3.1.2 описан механизм перехвата системных вызовов.

Перехватчик системных вызовов — это один из двух основных механизмов метода. Он выделяет из потока инструкций виртуального процессора те, которые связаны с реализацией системных вызовов.

Системный вызов — это запрос из пользовательского приложения к ядру. Для этих запросов обычно предусмотрены специальные инструкции (например, sysenter/syscall для x86/64 или svc #0 для ARM). Эти инструкции переводят процессор в режим ядра и выполняют соответствующий системному вызову код.

Перехват системных вызовов происходит в процессе выполнения кода виртуальной машины. Перехватчик сканирует поток выполняющихся инструкций с целью найти инструкции, реализующие системные вызовы. Когда они обнаружены, осуществляется переход в обработчик системных вызовов, соответствующий его идентификатору.

В процессе перехвата системных вызовов появляется задача сопоставления начала и конца функции. Зачастую инструкции начала и конца, последовательно встретившиеся в потоке, могут не быть парой, потому что архитектуры современных систем многопоточные и в разных потоках вызываются разные функции. Поэтому нельзя ориентироваться только на опкод, для сопоставления необходимо использовать дополнительные данные.

Определить что инструкция начала и конца соответствуют одной функции можно с помощью адресного пространства, заранее определив что оно будет из себя представлять (например, это может быть значение регистра, указывающего на каталог виртуальных страниц памяти).

Таким образом, на входе в системную функцию необходимо сохранить идентификатор адресного пространства. Когда в потоке появляется инструкция окончания системного вызова из существующих входов по вышеописанным параметрам выбирается соответствующий. Когда пара восстановлена, можно получать выходные данные и возвращаемое значение функции.

С помощью перехвата системных вызовов можно получать информацию о таких объектах, как файлы, процессы и отображения.

В подразделе 3.1.3 представлено получение информации об отображениях. Отображение в этом методе имеет характер вспомогательной сущности, промежуточного звена между потоком инструкций и модулем. Информация об отображении становится доступной с помощью цепочки системных вызовов, то есть применяется механизм, описанный в подразделе 3.1.2.

В контексте этой работы рассматриваются случаи отображения динамических библиотек и исполняемых файлов. Результатом отображения этих

файлов должен стать адрес загрузки файла в память виртуальной машины и размер загруженного образа.

Библиотека или исполняемый файл представляются в системе в виде файлов, поэтому для них работают обычные файловые операции. В случае с обнаружением библиотеки цепочка системных вызовов в общем случае может выглядеть как последовательность двух вызовов: открытие файла и отображение в память (маппинг). В некоторых системах могут присутствовать промежуточные шаги (например, в ОС Windows для файлов такого рода создаются секции и только потом происходит отображение).

Адрес загрузки необходим для получения информации о модулях, которые рассмотрены в подразделе 3.1.4.

Модули способны предоставить такие данные, как экспортируемые функции, а если исполняемый файл был скомпилирован с отладочной информацией, то можно получить и ее. В этой работе рассматриваются экспортируемые функции.

Для того, чтобы получить информацию о функциях, содержащихся в модулях, необходимо провести ряд действий. Во-первых, получить информацию от отображения про базовый адрес загрузки модуля и его размер. Во-вторых, определить момент загрузки библиотеки в память. В-третьих, считать нужные секции библиотеки и получить адреса и имена экспортируемых функций.

Первый шаг описан в подразделе 3.1.3. Второй этап заключается в проверке загрузки необходимых частей файла. Дело в том, что после отображения файл не сразу и не полностью загружается в память, он подгружается частями, когда они необходимы для работы системы. Как правило, для извлечения информации о функциях весь файл не требуется, необходимо прочитать заголовок и с его помощью определить какие секции библиотеки содержат нужную информацию. Чтобы удостовериться что заголовок файла уже загружен, необходимо дождаться, когда управление перейдет на адрес из диапазона адресов библиотеки. Этот диапазон начинается с адреса загрузки и заканчивается адресом загрузки, увеличенным на размер образа.

Когда заголовок загружен и можно получить адреса секций, где хранится информация об экспортируемых функциях, можно пробовать запрашивать память по этим адресам. В случае успешной загрузки можно получить список функций и адресов. Если память не загружена, то эта операция повторяется до тех пор, пока нужные секции не будут загружены. Данные о вызываемых из этой библиотеки функцях не утрачиваются, потому что если секция не загрузилась, значит и функции из нее не вызывались.

Таким образом может быть получена информация о загруженных в систему модулях и экспортированных функциях.

В подразделе 3.1.5 описан механизм перехвата API вызовов, который практически идентичен механизму перехвата системных вызовов. Разница заключается в том, что для перехвата API вызовов необходимо предварительно получить информацию об адресах и именах функций, содержащихся в библиотеках. Когда эта информация определена и сохранена, из потока инструкций выделяются инструкции для вызова функции и для возврата управления. С помощью информации об адресном пространстве происходит сопоставление этих пар, как при работе с системными вызовами. Параметром инструкции вызова функции является адрес, по которому и осуществляется переход в соответствующий обработчик. Данные из функций — входные и выходные параметры и возвращаемые значения. С помощью API вызовов можно получить информацию о процессах и вызываемых функциях.

В разделе 3.2 представлено описание метода вызова системных функций для получения заданных атрибутов объектов ОС.

В подразделе 3.2.1 приведено описание инструмента Crosscut, использующего похожий метод.

В подразделе 3.2.2 описаны основная идея метода и частный случай применения, соответствующий контексту этой работы.

В процессе работы вышеописанного метода мониторинга состояний могут возникать ситуации, когда невозможно определить необходимую информацию, например, сопоставить между собой все полученные данные о процессах. В конкретно этой ситуации обычно не хватает системного вызова запроса идентификатора процесса, который возникает по запросу от какого-либо приложения, работающего в системе. Таким образом, нужно иметь возможность вызывать некоторые системные вызовы, когда это необходимо для анализа.

Идея метода заключается в том, что в основной поток инструкций эмулятора должны быть встроены инструкции, выполняющие нужные в данный момент действия. Если осуществлять встраивание на работающей в обычном режиме виртуальной машине, то это грозит изменением поведения системы. Допускать такую ситуацию неприемлемо, поэтому этот метод лучше использовать в связке с детерминированным воспроизведением. Детерминированное воспроизведение — это технология, позволяющая записывать и воспроизводить сценарии работы исследуемой системы. Полученный сценарий можно запускать многократно, применяя различные виды анализа. В этом случае анализ не сможет повлиять на систему, например, нарушив ее работу замедлением. С помощью детерминированного воспроизведения становится воз-

можным вызывать необходимые инструкции, не нарушая обычной работы эмулятора.

Встраивать можно любые инструкции, которые могут привести к получению необходимой на данный момент информации. В контексте этой работы встраиваются системные вызовы, потому что в процессе работы эмулятора не всегда возникают те вызовы, с помощью которых можно составить полную картину, например, о процессах, работающих в системе. Возникает необходимость вызывать такие функции принудительно, встраивая свой код в общий поток инструкций. После того, как сценарий работы записан, воспроизведение можно останавливать и вызывать нужные функции, предварительно сохранив состояние виртуальной машины, затем получать информацию и продолжать воспроизведение с сохраненного состояния. Таким образом возможно дополнить недостающие данные и получить более полную картину. Этот процесс представлен на рисунке 4.

В подразделе 3.2.3 приведен пример использования этого метода для ОС Android.

Раздел 3.3 содержит выводы по главе. А именно рассмотрены преимущества и недостатки разработанного метода мониторинга событий виртуальной машины для получения информации об объектах ОС.

Преимущества перед рассмотренными инструментами следующие:

1. Метод не использует структуры ядра, как источник информации о системе. Благодаря этому инструмент позволяет исследовать любую версию любой системы без ее предварительной настройки. Например, если ABI (Application Binary Interface) разных систем совпадает, то они обе могут быть исследованы без переконфигурирования анализатора.
2. Представленный метод обладает высокой производительностью. Это связано с небольшим потоком данных между гостевой ОС и анализатором. Производительность анализатора исследуется в главе 5.
3. Для мониторинга объектов ОС по предложенному методу не требуется загрузка внешних программ или модулей в исследуемую систему, что позволяет анализировать встроенные системы.
4. Совмещение разработанного метода с детерминированным воспроизведением снимает часть ограничений по производительности, что важно для некоторых приложений, например, работающих с сетевым трафиком.

Представленный метод мониторинга объектов ОС имеет ограничения:

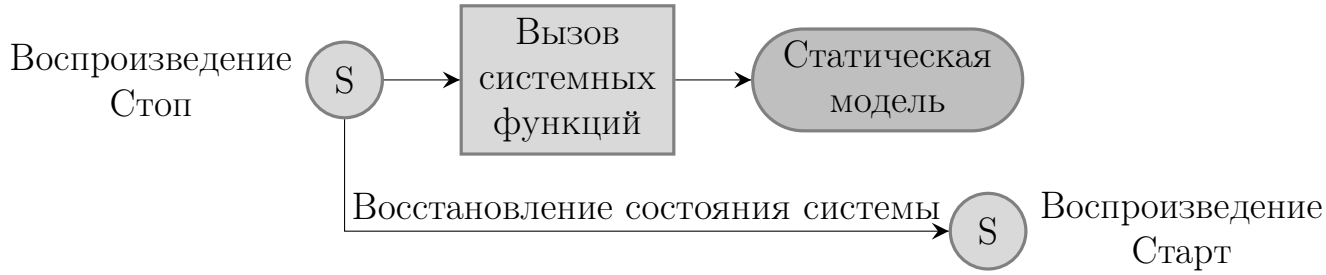

Рис. 4 — Вызов системных функций по запросу анализатора

1. Полная информация о работе системы для некоторого вида анализа (API вызовы, процессы) может быть получена только в том случае, если система и анализ будут запущены одновременно. В противном случае загрузка гостевых модулей может быть упущена и трассировка по ним осуществляться не будет.
2. Не предусмотрена работа с микроядерными системами.
3. Метод работает с учетом того, что ядро системы работает согласно документации, следовательно, измененное поведение ядра не рассматривается.

В **четвертой главе** описано применение разработанных методов на практике. Иструментальная среда представляет собой набор плагинов для эмулятора QEMU. Плагины — это динамические библиотеки, которые могут быть загружены при запуске эмулятора или подключены к нему в процессе работы.

Плагин может работать как самостоятельно, так и взаимодействовать с другими плагинами. Для взаимодействия плагинов существует механизм событий-подписок, который заключается в том, что каждый плагин может генерировать события и подписываться на события других плагинов. Например, плагин генерирует события о файловых операциях, а другому плагину нужно знать когда происходит открытие файла. Второй плагин подписывается на событие открытия файла, которое генерируется первым плагином, и получает всю информацию.

Кроме плагинов, события генерирует и эмулятор. Примерами событий являются прерывание, трансляция инструкции, выполнение инструкции, получение сетевого пакета, чтение/запись ячеек памяти. Все эти события также могут быть использованы плагинами.

Инструментальная среда разработана для операционных систем Windows, Linux, FreeBSD, работающих на архитектуре x86.

Раздел 4.1 посвящен описанию построения инструмента для операционной системы Windows. Он включает в себя описание входных данных и процесс мониторинга таких объектов, как файлы, модули и процессы.

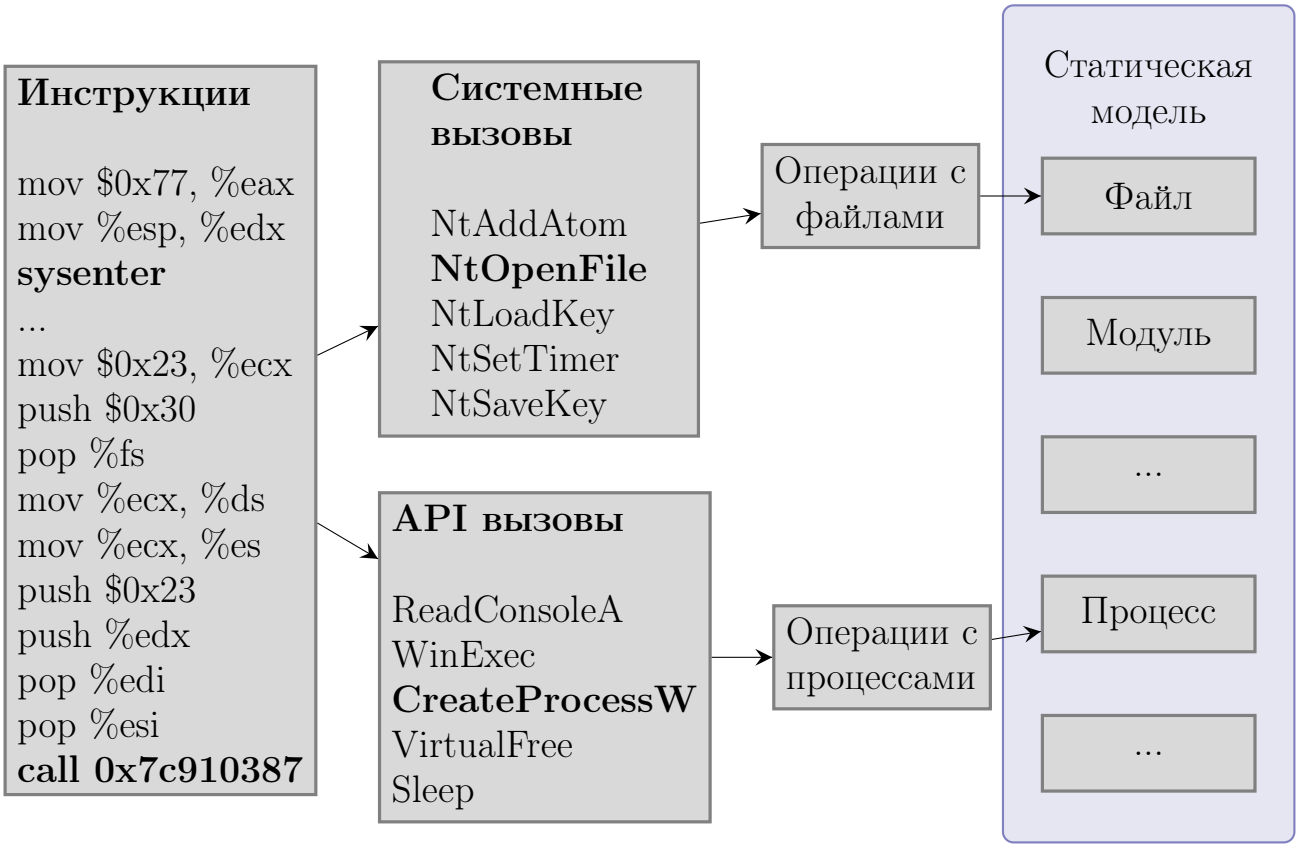

Рис. 5 — Механизм работы метода на примере получения информации о файлах и процессах

Для построения плагинов необходимо определить описанные в подразделе 3.1.1 входные данные. Набор получается следующим: инструкции для реализации системных вызовов `sysenter/sysexit`, адресное пространство представляется в виде регистра указателя на каталог страниц (`Cr3`), параметры функций передаются слева направо, формат динамических объектов PE (Portable Executable), идентификаторы системных вызовов, а также описания системных и API функций определены документацией.

На рисунке 5 представлен механизм работы метода на примере мониторинга информации о файлах и процессах для ОС Windows. Аналогичным образом он работает и для других систем.

Подраздел 4.1.1 посвящен описанию мониторинга файлов. Трассировка файловых операций основана на представленном в разделе 3.1 механизме перехвата системных вызовов. Для перехвата системных вызовов разработан набор платформо-зависимых плагинов для эмулятора QEMU (`syscalls`).

Вся необходимая информация о файлах (дескриптор, имя, тип доступа, буфер прочитанных/записанных данных) может быть получена на уровне системных функций.

Для трассировки файловых операций разработан плагин (`file_monitor`). Так как операции с файлами в рассматриваемых операционных системах организованы схожим образом, разработан один плагин. Он не зависит от гостевой операционной системы и гостевого аппаратного обеспечения.

Для анализа файловых операций перехватываются следующие функции: NtCreateFile, NtOpenFile, NtReadFile, NtWriteFile, NtDuplicateObject, NtClose в Windows.

Когда плагин `syscalls` обнаруживает платформо-зависимый системный вызов, он преобразовывает параметры и возвращает данные системного вызова в платформо-независимой форме.

Извлекаются следующие параметры операций: Create/Open file (возвращается дескриптор файла, имя файла, тип доступа), Close file (дескриптор файла), Read/Write file (дескриптор файла, адрес буфера, количество прочитанных/записанный байт), Duplicate object (дескриптор файла).

В подразделе 4.1.2 описано как извлекается информация о модулях. Основной задачей извлечения данных о модулях в этой ОС стало получение данных о работе процессов. Помимо этого модули предоставляют сведения и о других частях системы. В этой разделе под модулем понимается динамическая библиотека.

Для получения данных из библиотеки необходимо определить ее устройство. Она включает в себя заголовок и различные секции, одна из которых содержит информацию об экспортируемых функциях. Эти функции могут вызываться приложениями, загружающими библиотеку. Таким образом появляется задача получения адресов и имен экспортируемых функций для каждого загруженного модуля.

Важными данными при обнаружении библиотеки являются имя, базовый адрес загрузки ее в память и размер образа. Находить эти сведения можно с помощью перехвата ряда системных функций. Динамическая библиотека — это прежде всего файл, который в определенный момент открывается с помощью системной функции `NtOpenFile`. На этом этапе становится известным имя файла и его дескриптор. Следующий шаг, который проделывается с библиотекой, это создание секции, а именно вызывается системная функция `NtCreateSection`. Во входных параметрах этой функции присутствует дескриптор файла, для которого будет создана секция, а на выходе дескриптор секции. Далее вызывается функция `NtMapViewOfFile`, определяющая файл по дескриптору секции. Эта функция возвращает искомые данные: адрес загрузки библиотеки в память и размер образа (`baseAddress` и `viewSize`), а, следовательно, диапазон адресов, принадлежащих библиоте-

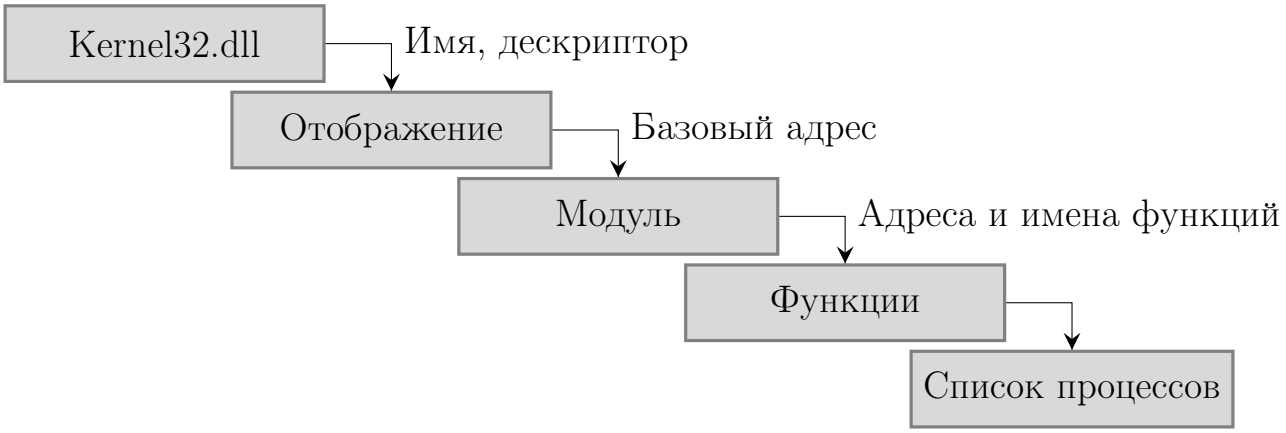

Рис. 6 — Получение списка процессов

ке. Далее необходимо отслеживать загрузку нужных секций библиотеки в память. Как только секции загружены, можно считать таблицу экспортируемых функций. Это реализовано в плагине `api_monitor`. Он предназначен для трассировки вызовов функций из динамических библиотек.

В подразделе 4.1.3 рассмотрено получение информации о процессах. Исследование процессов может осуществляться через системные вызовы и динамические библиотеки.

В первой части подраздела рассматривается мониторинг процессов с помощью системных вызовов. Результат получился неудовлетворительным, поэтому остальная часть подраздела посвящена описанию получения этой информации через API функции.

Задача анализа процессов предполагает получение следующих сведений о них: идентификатор процесса, имя образа (приложения) и идентификатор адресного пространства. Все эти сведения можно получить из библиотечных функций.

Библиотечные функции более высокоуровневые, чем системные и, их параметры обладают большей информативностью и понятны человеку. Так, например, с помощью функции GetCurrentProcessId в Windows можно точно получить идентификатор процесса в текущем адресном пространстве. Помимо информации, полученной из параметров, отслеживая вызывающиеся функции, можно делать выводы о работе исследуемой системы.

Процессы в Windows создаются с помощью API функций семейства `CreateProcess`, которые в ОС Windows находятся в библиотеке `kernel32.dll`.

На рисунке 6 представлен процесс получения списка процессов. Из рисунка видно, что для получения информации о процессах необходимо получить информацию о модулях. Когда она доступна, то адреса и имена функций

передаются в перехватчик API вызовов, где происходит обработка библиотечных функций.

Список процессов, полученный таким образом, соответствует списку процессов, отображенных системной утилитой Windows «Диспетчер задач».

В разделе 4.2 рассмотрена разработка инструмента для мониторинга файлов, модулей и процессов для операционных семейств семейства Linux.

Раздел 4.3 посвящен разработке инструментального средства для ОС FreeBSD.

В разделе 4.4 описана возможность конфигурирования трассировки системных вызовов для разных операционных систем.

В связи с существованием множества операционных систем и архитектур процессоров, приходится учитывать интерфейсы взаимодействия с ядром для каждого случая, и создавать отдельный плагин-перехватчик системных вызовов.

Был разработан один плагин для перехвата системных вызовов, работающий с конфигурационным файлом. Он предусматривает различные варианты осуществления системных вызовов в разных системах и подставляет нужные данные. В файле должны быть описаны входные данные, представленные в подразделе 3.1.1, относящиеся к системным вызовам.

Результатом работы этого плагина является трасса выполнявшихся системных вызовов с их параметрами и возвращаемыми значениями. Также этот плагин имеет возможность генерировать события, чтобы взаимодействовать с другими плагинами.

Конфигурационный файл делает трассировщик более гибким и настраиваемым. Он позволяет составлять собственные списки системных вызовов, которые необходимо трассировать, кроме того, нет необходимости пересобирать плагин для перехода на исследование другой ОС. Вместе с тем разработанный механизм облегчает поддержку новых операционных систем и архитектур процессоров.

Пятая глава посвящена функциональному тестированию разработанного инструмента и тестированию производительности. Также представлено сравнение настроечных параметров с инструментом DECAF.

В таблице приведены сравнение времени работы QEMU и разработанного инструмента с включенным перехватом системных вызовов (Qemu_syscalls) и с включенным плагином отслеживания процессов (Qemu_process). Для тестирования использовались операционные системы Windows XP, Ubuntu 9.1 и Arch Linux 4.2. Из таблицы 2 видно, что замедление работы с применением разработанных механизмов составляет от 3 до 14% в зависимости от ОС и операции и является несущественным.

Таблица 2

Сравнение скорости работы QEMU и разработанного инструмента

	Qemu, мин	Qemu_syscalls, мин	Qemu_process, мин
Windows XP			
Загрузка ОС	2:31	2:44	2:51
Замедление		9%	13%
Ubuntu 9.1			
Загрузка ОС	1:11	1:21	1:22
Замедление		14%	14%
Скачивание файла	1:37	2:41	2:46
Замедление		4%	9%
Arch Linux 4.2			
Загрузка ОС	0:27	0:29	0:30
Замедление		7%	11%
Скачивание файла	1:14	1:16	1:18
Замедление		3%	5%
Архивирование файла	2:08	2:13	2:14
Замедление		4%	5%

Сравнить напрямую скорость работы DECAF и нового инструмента является нетривиальной задачей, поскольку в этих проектах использованы разные кодовые базы, механизмы реализации плагинов и их функциональность.

Для сравнения сложности конфигурирования инструмента под новую гостевую систему был проведен анализ профилей (набора параметров, необходимых для работы системы). Выяснилось, что предлагаемые в этой работе профили применимы к семействам ОС, а не только к конкретным версиям ОС, включают меньше параметров и пишутся программистом вручную, в то время как профили DECAF генерируются автоматически программой-агентом и при ее отсутствии составить такой профиль вручную является весьма трудной задачей.

Корректность метода была проверена путем сравнения списка процессов ОС Windows с выводом системной утилиты «Диспетчер задач», а также для ОС Linux сравнивались результаты работы утилиты strace с журналом системных вызовов, полученных с помощью разработанного инструмента. Результаты оказались идентичными.

К положительным сторонам разработанного инструмента можно отнести его универсальность (работа с семействами ОС, встроенными системами),

простой набор входных данных для разработки плагинов для новых ОС, достоверность получаемых результатов. К недостаткам относится замедление работы виртуальной машины, оно небольшое (от 3 до 14%), но присутствует.

В **заключении** перечисляются основные особенности реализованного инструмента, отличающие его от существующих аналогов. Описываются его возможные применения и получаемые с его помощью результаты, а также планы развития разработанных методов.

Основные результаты работы:

1. Разработан метод мониторинга событий виртуальной машины для получения информации об объектах гостевой операционной системы без внедрения инструментального кода на уровне исследуемой системы.
2. Разработан метод вызова системных функций по запросу анализатора для получения заданных атрибутов объектов операционной системы.
3. На основе предложенных автором методов разработана и реализована инструментальная среда, работающая с тремя семействами операционных систем общего назначения (Windows, Linux, FreeBSD). По результатам проведенного тестирования продемонстрировано, что разработанный инструмент может быть использован для анализа встроенных систем, накладные расходы не превышают 14%, а результаты анализа являются достоверными.

Публикации автора по теме диссертации

1. *Фурсова Н. И.*, *Довгалюк П. М.*, *Васильев И. А.* Использование ABI для интроспекции виртуальных машин // Труды Института системного программирования РАН. — 2015. — № 27. — С. 159—168.
2. *Фурсова Н. И.*, *Довгалюк П. М.*, *Васильев И. А.*, *Макаров В.* Легковесный метод интроспекции виртуальных машин // Programming and Computer Software. — 2017. — № 5. — С. 307—313.
3. *Васильев И. А.*, *Фурсова Н. И.*, *Довгалюк П. М.*, *Климушенкова М. А.*, *Макаров В.* Модули для инструментирования исполняемого кода в симуляторе QEMU // Проблемы информационной безопасности. Компьютерные системы. — 2015. — № 4. — С. 194—203.
4. *Климушенкова М.*, *Бакулин М.*, *Падарян В.*, *Довгалюк П.*, *Фурсова Н. И.*, *Васильев И.* О некоторых ограничениях полносистемного анализа помеченных данных // Труды Института системного программирования РАН. — 2016.
5. *Фурсова Н. И.* Интроспекция виртуальных машин // Ученые записки Новгородского государственного университета имени Ярослава Мудрого. — 2015. — № 3. — С. 1—3.
6. *Fursova N.* Introspection of the Virtual Machines with System Calls Monitoring: Student Research Abstract // Proceedings of the 31st Annual ACM Symposium on Applied Computing. — Pisa, Italy : ACM, 2016. — С. 1582—1583. — (SAC '16). — ISBN 978-1-4503-3739-7. — DOI: 10.1145/2851613.2852008. — URL: http://doi.acm.org/10.1145/2851613.2852008.
7. *Dovgalyuk P.*, *Fursova N.*, *Vasiliev I.*, *Makarov V.* QEMU-based Framework for Non-intrusive Virtual Machine Instrumentation and Introspection // Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. — Paderborn, Germany : ACM, 2017. — С. 944—948. — (ESEC/FSE 2017). — ISBN 978-1-4503-5105-8. — DOI: 10.1145/3106237.3122817. — URL: http://doi.acm.org/10.1145/3106237.3122817.