УДК 004.056
doi: 10.26583/bit.2024.1.08

Павел А. Теплюк[1], Алексей Г. Якунин[2]
Алтайский государственный технический университет им. И. И. Ползунова,
пр-т Ленина, 46, Барнаул, 656038, Россия
[1]*e-mail: my@teplyukpavel.ru, https://orcid.org/0009-0002-8019-437X*
[2]*e-mail: almpas@list.ru, https://orcid.org/0000-0001-5103-3177*

МОДЕЛИ И ПОДХОДЫ К АНАЛИЗУ ПОВЕРХНОСТИ АТАКИ ДЛЯ ФАЗЗИНГ-ТЕСТИРОВАНИЯ ЯДРА LINUX*

Аннотация. Целью исследования явилось проведение анализа возможных способов определения поверхности атаки применительно к решению задачи фаззинг-тестирования ядра операционных систем семейства Linux и выбор из них наиболее подходящего. Для оценки и сравнения различных моделей и практических подходов к анализу поверхности атаки, а также оценки возможности их комбинирования использовались такие методы теоретического исследования как анализ, сравнение, дедукция. Проведены оценка и сравнение существующих моделей и подходов к анализу поверхности атаки ядра Linux. Предложено решение по практическому определению поверхности атаки для эффективного тестирования ядра методом фаззинга, которое объединяет в себе исследованные подходы. Результаты проведенного исследования могут быть использованы для практического построения поверхности атаки, которая позволит более точно определить цели фаззинг-тестирования ядра Linux.
Ключевые слова: операционная система, ядро Linux, динамический анализ, фаззинг, поверхность атаки, системные вызовы.
Для цитирования: ТЕПЛЮК, Павел А.; ЯКУНИН, Алексей Г. МОДЕЛИ И ПОДХОДЫ К АНАЛИЗУ ПОВЕРХНОСТИ АТАКИ ДЛЯ ФАЗЗИНГ-ТЕСТИРОВАНИЯ ЯДРА LINUX. Безопасность информационных технологий, [S.l.], т. 31, № 1, с. 135–145, 2024. ISSN 2074-7136. URL: https://bit.spels.ru/index.php/bit/article/view/1602. DOI: http://dx.doi.org/10.26583/bit.2024.1.08.

Благодарности. Исследование проводится при финансовой поддержке Московского технического университета связи и информатики в рамках научного проекта № 40469-26/23-К.

Pavel A. Teplyuk[1], Aleksei G. Yakunin[2]
Polzunov Altai State Technical University,
Lenina Ave., 46, Barnaul, 656038, Russia
[1]*e-mail: my@teplyukpavel.ru, https://orcid.org/0009-0002-8019-437X*
[2]*e-mail: almpas@list.ru, https://orcid.org/0000-0001-5103-3177*

Models and approaches to attack surface analysis for fuzz testing of the Linux kernel*

Abstract. The purpose of the study was to analyze possible methods for determining the attack surface in relation to solving the problem of fuzzing testing the kernel of operating systems of the Linux family and selecting the most suitable one. To evaluate and compare various models and practical approaches to attack surface analysis, as well as assess the possibility of combining them, theoretical research methods such as analysis, comparison, and deduction were used. Existing models and approaches to analyzing the attack surface of the Linux kernel are assessed and compared. A solution is proposed for the practical determination of the attack surface for effective testing of the kernel using the fuzzing method, which combines the studied approaches. The results of the study can be used to practically construct an attack surface, which will allow us to more accurately determine the goals of fuzz testing of the Linux kernel.
Keywords: operating system, Linux kernel, dynamic analysis, fuzzing, attack surface, system calls.

For citation: TEPLYUK, Pavel A.; YAKUNIN, Aleksei G. Models and approaches to attack surface analysis for fuzz testing of the Linux kernel. IT Security (Russia), [S.l.], v. 31, no. 1, p. 135–145, 2024. ISSN 2074-7136. URL: https://bit.spels.ru/index.php/bit/article/view/1602. DOI: http://dx.doi.org/10.26583/bit.2024.1.08.

**Acknowledgement.* The research is financially supported by the Moscow Technical University of Communications and Informatics as part of research project № 40469-26/23-K.

Введение

Широкое распространение операционных систем на базе ядра Linux в качестве основы системного программного обеспечения приводит к увеличению информационных систем, напрямую зависимых от надёжности и безопасности ядра. Такими системами, в частности, являются программно-аппаратные комплексы, обеспечивающие работу критически важных производственных и инфраструктурных автоматизированных систем [1].

Согласно утверждённой ФСТЭК России «Методике выявления уязвимостей и недекларированных возможностей в программном обеспечении»[1], одним из шагов разработки безопасных программных продуктов для обеспечения соответствия требованиям методики, начиная с минимального уровня доверия, является выполнение динамического анализа программного кода объекта оценивания, в том числе с применением тестирования методом фаззинга. Фаззинг (фаззинг-тестирование) [2] представляет собой метод динамического анализа кода, при котором на вход функциям или системным вызовам исследуемого программного обеспечения подаются случайные данные в попытке выявить дефекты и ошибки в логике.

Важной задачей для эффективного тестирования методом фаззинга является определение поверхности атаки, то есть выявление всех неконтролируемых входов в систему, которые неподконтрольны легитимному пользователю, но могут быть потенциально подконтрольны злоумышленнику.

В [3] приводятся три общие модели поверхности атаки ядра: GenSec, IsolSec и StaticSec. Каждая из них имеет свои особенности применения. Далее эти модели будут раскрыты более подробно, а. также рассмотрены существующие подходы к анализу и определению поверхности атаки в ядре:

- на основе анализа баз данных CVE;
- на основе метрик сложности;
- с помощью интроспекции виртуальных машин;
- с помощью отслеживания помеченных данных.

Целью работы является анализ моделей и подходов к определению поверхности атаки, и выбор наиболее эффективных с точки зрения корректного определения фаззинг-целей в ядре Linux. Кроме того, будут обозначены возможные направления дальнейших исследований.

1. Модели поверхности атаки

При определении поверхности атаки в ядре Linux необходимо обозначить универсальную модель безопасности, которая покрывает всё работающее ядро, а затем более конкретную модель, охватывающую локальные атаки из непривилегированного пользовательского пространства, направленные против ядра. Аппаратное обеспечение и конфигурацию ядра во время компиляции необходимо учитывать, так как от них будет

[1]Информационное сообщение ФСТЭК России от 10 февраля 2021 г. URL: https://fstec.ru/dokumenty/vse-dokumenty/informatsionnye-i-analiticheskie-materialy/informationnoe-soobshchenie-fstek-rossii-ot-10-fevralya-2021-g-n-240-24-647 (дата обращения: 28.08.2023).

напрямую зависеть количество точек входа со стороны пользователя. Основной задачей для обеспечения безопасности функционирования операционной системы (ОС), и, следовательно, её ядра, является обеспечение гарантий конфиденциальности, целостности и доступности процессов и данных. Так, злоумышленник может захватить контроль посредством выполнения произвольного кода в режиме ядра, либо с помощью более ограниченных атак. К таким атакам можно, например, отнести утечку памяти, которая позволяет нарушить конфиденциальность и вызвать отказ в обслуживании путем сбоя ядра с целью установления недоступности системы. Также предполагается, что оборудование и прошивка, на которой работает система, являются доверенными. Далее рассмотрены общедоступные модели поверхности атаки на ядро Linux.

Модель GenSec. Данная модель включает в поверхность атаки все возможные подсистемы ядра, предрасположенные к сбоям (рис. 1). Допускается, что злоумышленник имеет учетную запись в целевой системе и может взаимодействовать с аппаратным обеспечением. Кроме того, злоумышленник контролирует привилегированные приложения. Модель включает в себя сбои из-за недостатков в ядре, доступ к которым возможен только из привилегированного доступа. При этом дефекты могут быть в любой части работающего ядра – включая основное ядро и все загруженные модули, а также любые модули, которые могут быть загружены в будущем, например, при подключении нового оборудования. Модель GenSec является довольно обширной, поэтому учитывать все будущие действия во время функционирования ОС при ее применении будет очень сложно.

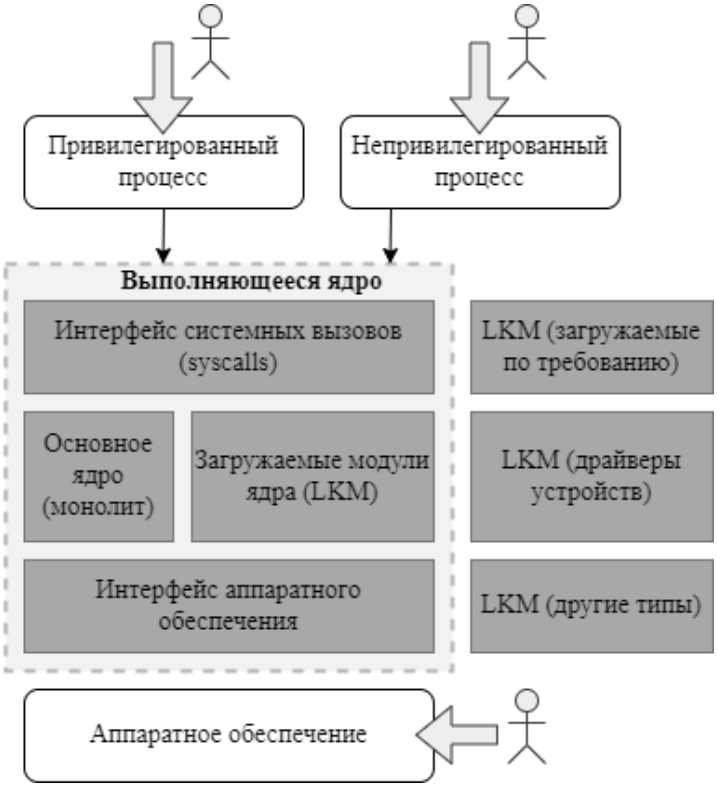

Рис. 1. Модель GenSec

Модель IsolSec. Модель IsolSec (рис. 2) отражает общую модель в многопользовательских системах и в системах, где предполагается, что злоумышленник имеет локальный доступ, например, путем компрометации непривилегированного процесса в системе, и пытается повысить свои привилегии.

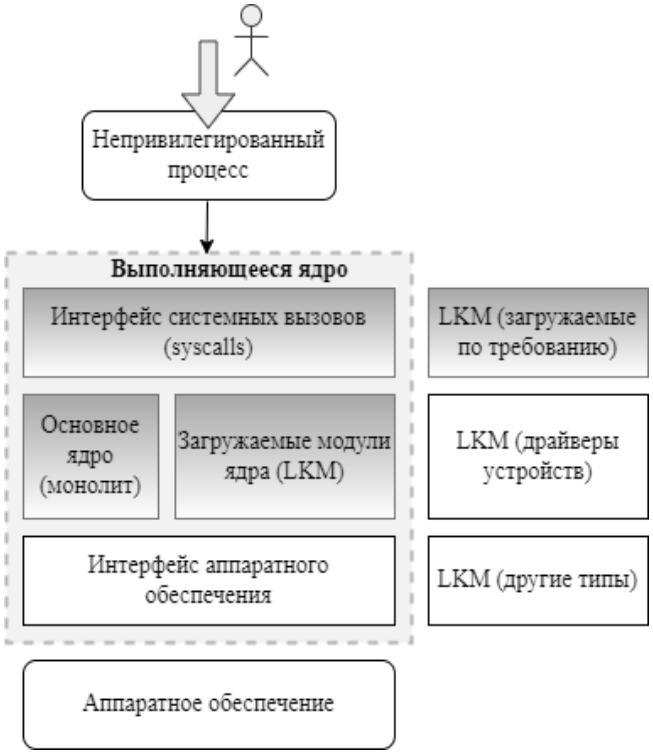

Рис. 2. Модель IsolSec

Злоумышленник имеет непривилегированный локальный доступ, поэтому он может использовать интерфейс системного вызова. Также предположим, что злоумышленник может внедрять код в загружаемые модули ядра (LKM – loadable kernel module), которые загружаются системой по требованию. Поскольку злоумышленник не может подключить оборудование к целевой системе, предполагаем, что ошибки в LKM, не связанные с установленным оборудованием, не могут привести к сбоям. Ядро Linux предлагает множество псевдофайловых систем, а именно procfs, debugfs, sysfs и securityfs, в которых операции с файловой системой передаются на конкретные пути кода в ядре, в основном в LKM, и часто используются для предоставления информации или тонкой настройки интерфейсов для привилегированных процессов пользовательского пространства. Однако проверки привилегий выполняются на уровне виртуальной файловой системы с использованием списков контроля доступа, реализованных по стандарту переносимого интерфейса операционных систем POSIX (Portable Operating System Interface for UNIX)

Кроме того, поскольку эти файловые системы должны быть доступны из непривилегированного приложения, которое помещено в «песочницу», включим все функциональные возможности, предоставляемые этими четырьмя псевдо-файловыми системами, в поверхность атаки.

Также включим модули ядра, которые либо не загружаются во время работы, либо могут быть загружены во время работы привилегированным пользователем без каких-либо проверок на легитимность модуля.

Модель StaticSec. Модель изоляции StaticSec (рис. 3) схожа с IsolSec, однако её отличие состоит в том, что в StaticSec злоумышленник не может модифицировать LKM.

Хотя загрузка модулей злоумышленником возможна из-за соответствующих настроек, установленных по умолчанию в многих дистрибутивах Linux, отключить это поведение довольно просто (путем включения параметра управления modules_disabled).

Модель StaticSec предполагает, что для взломщика доступны только изначально загруженные LKM, загруженные изначально для работы.

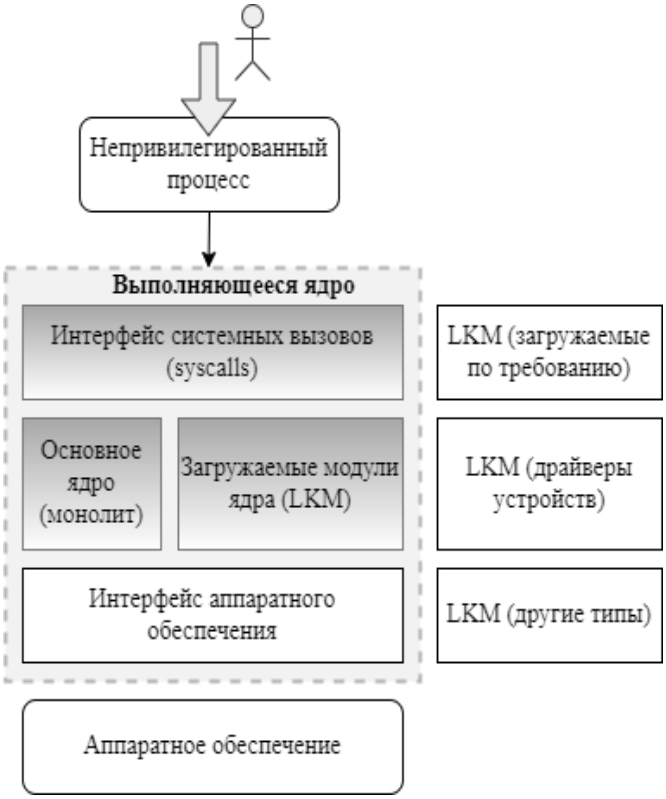

Рис. 3. Модель StaticSec

2. Подходы к определению поверхности атаки

Анализ поверхности атаки с использованием открытых баз данных CVE. Одной из наиболее известных проблем фаззинга ядра Linux является его чрезвычайно высокая сложность. По состоянию на ноябрь 2023 г. репозиторий ядра[2] в основной ветке кода содержит более 1.2 млн коммитов, а объем кодовой базы исчисляется более 30 млн строк кода.

В [4] предлагается подход к определению поверхности атаки для фаззинга с помощью автоматизированного анализа открытых баз данных общеизвестных уязвимостей CVE (Common Vulnerabilites and Exposures). Задачей исследования является выявление уязвимых подсистем ядра и, соответственно, исключение наиболее защищенных из поверхности атаки с целью повышения эффективности фаззинг-тестирования.

Определение поверхности атаки состоит из следующих этапов:

– сбор существующих CVE ядра Linux;

– поиск коммитов, в которых предлагается исправление CVE и определение файлов, содержащих соответствующие уязвимости;

– классификация подсистем ядра и сопоставление найденных уязвимых файлов с этими подсистемами;

– обобщение полученных данных: определение наиболее распространенных типов уязвимостей и соответствующих подсистем ядра.

[2]Kernel.org git repositories. URL: https://git.kernel.org/ (дата обращения: 04.11.2023).

В [4] также приводится практическая реализация выбранного подхода. Для сбора базы CVE ядра Linux из открытых источников сети Интернет, авторы разработали веб-сканер, который использует механизм парсинга HTML-страниц. В качестве источников были выбраны ресурсы National Vulnerability Database (NVD)[3] и CVEDetails[4].

Для последующего анализа инструмент собирает следующие параметры CVE:

– ID (идентификатор уязвимости);

– type (тип уязвимости);

– CVSS score (оценка уязвимости по стандарту Common Vulnerability Scoring System[5]).

Кроме того, дополнительно осуществляется сбор ссылок на коммиты в репозитории ядра Linux, которые содержат исправления уязвимых участков кода.

Классификация подсистем ядра основывается на интерактивной карте ядра Linux (Linux Kernel Map)[6]. Сопоставление уязвимых файлов к подсистемам происходит за счет поиска ключевых слов, идентифицирующих подсистему, в содержимом, либо именах файлов или директорий, в которых они расположены.

Всего было обнаружено 2162 уязвимостей ядра Linux в период с 1999 по 2018 гг., причем более 50% – это уязвимости, приводящие к отказу в обслуживании (Denial Of Service – DoS). Наибольшее количество уязвимостей в себе собрали такие подсистемы ядра, как synchronization (механизмы синхронизации), logical memory (управление логическими адресами памяти) и threads (управление потоками выполнения). Согласно подходу, эти подсистемы необходимо включить в поверхность атаки в первую очередь.

Результаты данного исследования показывают, что сбор и анализ данных об известных уязвимостях ядра Linux позволяют выявить наиболее «проблемные» подсистемы, которые потенциально могут иметь дефекты и сбои в процессе выполнения ядра. Эти подсистемы в рамках фаззинг-тестирования необходимо исследовать в первую очередь.

Анализ поверхности атаки на основе метрик сложности. В [5] предлагается подход к анализу поверхности атаки на основе метрик сложности (complexity metrics).

Суть подхода заключается в измерении цикломатической сложности кода ядра Linux в различных конфигурациях с целью определения того, как конкретные компоненты ядра влияют на поверхность атаки.

Цикломатическая сложность кода – это количественная метрика для расчета числа логических ветвей в коде. Всякий раз, когда поток управления разделяется, счетчик метрики увеличивается на 1. Минимальная сложность каждой функции равна 1.

В соответствии с моделями GenSec и IsolSec, описанными выше, в поверхность атаки включены монолитное ядро и модули, которые позволяют расширить функциональность системы.

Для измерения цикломатической сложности необходимо выбрать модули, общие для разных конфигураций ОС на базе Linux, а также аппаратно-зависимые разделы исходного кода ядра. В исследовании были протестированы следующие модули и подсистемы: SELinux, AMDGPU, KVM, ext4, xfs, btrfs и namespaces.

[3]National vulnerability database. URL: https://nvd.nist.gov/ (дата обращения: 08.11.2023).
[4]CVE security vulnerability database. URL: https://cvedetails.com/ (дата обращения: 09.11.2023).
[5]CVSS v2 Complete Documentation. URL: https://www.first.org/cvss/v2/guide (дата обращения: 10.11.2023).
[6]Interactive map of Linux kernel. URL: https://makelinux.github.io/kernel/map/ (дата обращения: 09.11.2023).

Для сбора метрик сложности предполагается использование инструментов статического анализа кода, например, SonarQube[7]. Последовательность операций для сбора метрик сложности представлена на рис. 4.

Получение метрик цикломатической сложности кода позволяет оценить, как различные подсистемы ядра Linux влияют на общую поверхность атаки. Например, использование модуля AMDGPU (графический драйвер для видеокарт AMD) увеличивает показатель сложности на 16.09%.

Тем не менее, полученные метрики не всегда коррелируют с поверхностью атаки. Согласно [5], значительная часть цикломатической сложности представлена в таких разделах исходного кода, как «arch», «driver» и «fs». Однако в развернутой системе, как правило, используется только актуальный для текущего аппаратного обеспечения код, поэтому большая часть исходного кода не будет задействована при выполнении ядра и, таким образом, не будет включена в поверхность атаки.

Рис. 4. Последовательность операций для сбора метрик сложности

Определение поверхности атаки с помощью интроспекции виртуальных машин. Использование виртуальных машин для запуска ядра Linux в процессе фаззинга имеет такие преимущества, как простота управления и масштабируемость. Поэтому именно такой способ запуска предусматривают фаззеры ядра, например, Syzkaller[8].

Тем не менее, анализ поверхности атаки ядра, запущенного в виртуальной машине, сопряжен с некоторыми проблемами, одна из которых – семантический разрыв, т.е. разрыв между представлением наблюдаемых в системе данных и данных, необходимых для анализа. Для сокращения такого разрыва применяют интроспекцию виртуальных машин (virtual machine introspection, VMI) [6].

Интроспекция виртуальных машин – это способ исследования текущего состояния виртуальной машины на системном уровне. Иными словами, интроспекция – это извлечение данных из ОС, которые используются системой для ее работы, но они скрыты от пользователя.

В рамках задачи по определению поверхности атаки предлагаются два подхода к интроспекции:

[7]Code Quality, Security & Static Analysis Tool with SonarQube Sonar. URL: https://www.sonarsource.com/products/sonarqube (дата обращения: 05.09.2023).
[8]Syzkaller. URL: https://github.com/google/syzkaller (дата обращения: 07.09.2023).

1. Подход на основе сопоставления исходного кода ядра ОС с выполняемыми инструкциями. С помощью этого можно получить информацию о размещении структур данных, а также об их содержимом [7]. Подход предполагает привязку алгоритмов анализа к конкретным ОС и даже к их недокументированным внутренним структурам, если недоступны исходные коды ОС.

2. Подход на основе встраивания специальных модулей в гостевую систему [8]. Здесь предполагается, что подключение модуля анализа может менять работу ОС. Также необходимо наличие SDK (software development kit) для сборки гостевого агента. Внедрение модуля не дает возможности применять детерминированное воспроизведение работы системы.

В инструменте Natch[9], предназначенном для определения поверхности атаки, в основе подхода к интроспекции лежит перехват редко меняющихся событий (например, системные вызовы) и анализ структур данных ядра в памяти виртуальной машины.

В рамках отдельной системы системные вызовы изменяются достаточно редко. Как правило, могут дополняться варианты функций или же перестают использоваться старые.

Использование данных, которые системные вызовы берут в качестве параметров или возвращают в результате выполнения, недостаточно, чтобы восстановить полную информацию о запущенных процессах и загруженных модулях. К примеру, невозможно сопоставить системный вызов execve, который получает на вход имя программы, и порожденный им процесс.

Более подробные данные об ОС необходимо извлекать из структур данных ядра. Структуры, а также адреса памяти, в которых они располагаются, могут не совпадать из-за различий в версии или конфигурации сборки ядра. Поэтому необходимо в рамках данного подхода применять профиль интроспекции, который будет содержать смещения и адреса структур и глобальных переменных конкретной ОС.

Определение поверхности атаки с помощью отслеживания помеченных данных. Динамический анализ помеченных данных – это подход к определению поверхности атаки, при котором отслеживаются данные, полученные из внешних источников, во время выполнения программы [9].

Задачей такого анализа является автоматическое выявление процессов, модулей и функций, которые непосредственно связаны с обработкой данных из внешних источников. В рамках этой задачи необходимо решить три следующих подзадачи:

1) Определение данных, которые необходимо отслеживать. Например, вносить пометку на данные из недоверенных источников или на чувствительные данные.

2) Определение способа продвижения пометки при распространении данных в процессе выполнения программы.

3) Определение случаев, требующих вывод предупреждения об ошибке.

Инструменты DECAF [10] и Panda [8] используют динамическую бинарную трансляцию для отслеживания пометок. DECAF производит инструментирование на уровне внутреннего представления TCG. Panda использует преобразование внутреннего представления TCG в LLVM-биткод, который отслеживается для продвижения пометок.

В ранее упомянутом Natch используется метод «разбавленных меток», при котором с увеличением преобразований с помеченными данными снижается «сила» полученной пометки. Благодаря этому имеется возможность описывать поверхность атаки, поскольку в первую очередь целями злоумышленников являются функции, получающие слабо преобразованные данные.

[9]Инструмент для определения поверхности атаки Natch. URL: https://ispras.ru/technologies/natch/ (дата обращения: 06.09.2023).

3. Обсуждение результатов

Проанализированные общие модели поверхности атаки ядра Linux ориентированы на разный уровень привилегий в системе, доступный злоумышленнику.

Современные подходы к фаззингу ядра [11] предполагают генерирование случайных входных данных не только со стороны процессов пространства пользователя, но и со стороны интерфейсов аппаратных средств путем их эмулирования (например, интерфейса USB [12]). Таким образом, фаззинг позволяет имитировать действия злоумышленника, имеющего доступ к аппаратному обеспечению. Отсюда следует, что для построения поверхности атаки применительно к фаззинг-тестированию ядра Linux лучше всего подходит модель GenSec. Данная модель, охватывающая значительную часть подсистем ядра и интерфейсы аппаратных средств, позволяет максимизировать количество фаззинг-целей, что потенциально допускает обнаружение ранее неизвестных уязвимостей в коде ядра.

Тем не менее, модель GenSec в равной степени, как и другие рассмотренные модели, описывает поверхность атаки в общем виде и не учитывает многие детали. Поэтому важной задачей является исследование различных практических подходов к определению поверхности атаки, которые могут детализировать ее. Ввиду того, что, большой объем кода ядра, который растет с каждым релизом, значительно влияет на эффективность фаззинга, необходимо сократить поверхность атаки, выделив для начала наиболее потенциально уязвимые подсистемы.

Подход на основе анализа базы данных CVE как раз позволяет на основе статистики найденных уязвимостей ядра Linux в разные годы определить такие подсистемы. Достоинством подхода является несложная практическая реализация: нет необходимости разворачивать виртуальные машины и конфигурировать ядро. Однако достоинство определяет и недостаток данного подхода: исследуется не актуальное, запущенное в текущий момент ядро, а общие недостатки ядер различных версий, многие из которых уже не актуальны сегодня.

Подход на основе метрик сложности, который представляет собой по большей части статический анализ кода, позволяет численно оценить, как различные подсистемы ядра влияют на поверхность атаки. В отличие от предыдущего подхода, здесь предполагается исследование актуальной версии ядра. Подход не лишен недостатков: метрики сложности не всегда коррелируют с реальной поверхностью атаки в развернутой системе.

Подходы на основе интроспекции виртуальных машин и отслеживания помеченных данных позволяют достаточно точно описывать поверхность атаки, поскольку здесь уже предполагается мониторинг и динамический анализ развернутого ядра.

Для выбора подходов были определены следующие критерии:

- сложность практической реализации;
- необходимость развертывания ядра;
- актуальность версий ядра;
- необходимость использования проприетарного программного обеспечения;
- точность определения поверхности атаки.

В табл. 1 представлено сравнение рассмотренных в статье подходов к анализу поверхности атаки.

Из анализа приведенных в данной таблице сведений следует, что для определения поверхности атаки применительно к решению задачи фаззинг-тестирования ядра будет целесообразно использовать комбинацию следующих подходов:

- на основе измерения метрик сложности;
- на основе интроспекции виртуальных машин;

– на основе анализа помеченных данных.

Подход на основе анализа баз данных CVE не является достаточно эффективным с точки зрения актуальности версий кода ядра и поэтому не будет применяться в рамках исследований.

Таблица 1. Сравнение подходов к определению поверхности атаки ядра Linux

Подход к определению поверхности атаки	**Сложность практической реализации**	**Необходимость развёртывания ядра**	**Актуальность исследуемого ядра**	**Необходимость использования проприетарного программного обеспечения**	**Точность определения поверхности атаки**
Анализ баз данных CVE	Низкая	Нет	Не всегда актуально	Нет	Средняя
Измерение метрик сложности	Средняя	Нет	Актуальная версия ядра	Да (SonarQube)	Средняя
Применение интроспекции виртуальных машин	Высокая	В виртуальной машине	Актуальная версия ядра	Да (Natch)	Высокая
Динамический анализ помеченных данных	Высокая	В виртуальной машине	Актуальная версия ядра	Да (Natch)	Высокая

Заключение

Определение поверхности атаки является важной задачей в ходе подготовки к процессу фаззинга ядра. Существующие подходы к ее построению позволяют анализировать подсистемы ядра разными способами. В дальнейших исследованиях планируется применить на практике комбинацию исследованных в статье подходов с целью более точного описания поверхности атаки применительно к решению задачи фаззинг-тестирования ядра Linux. Одним из направлений работы является поиск дефектов и уязвимостей методом фаззинга в актуальных ветках ядра на основе построенной поверхности атаки.

СПИСОК ЛИТЕРАТУРЫ:

1. Хорошилов, А. Международный проект по разработке ядра Linux. Системный администратор. 2022, № 3(232), с. 32–37. – EDN: FSDCWC.
2. Козырский Б.Л., Комаров Т. И., Иванов М. А. Использование фаззинга для поиска уязвимостей в программном обеспечении. Безопасность информационных технологий, [S.l.], т. 21, № 4, 2014. ISSN 2074-7136. – EDN: TSEIMN.
3. Kurmus A., Dechand S., Kapitza R. (2014). Quantifiable Run-Time Kernel Attack Surface Reduction. In: Dietrich, S. (eds) Detection of Intrusions and Malware, and Vulnerability Assessment. DIMVA 2014. Lecture Notes in Computer Science, v. 8550. Springer, Cham. DOI: 10.1007/978-3-319-08509-8_12.
4. Peng L. Attack surface analysis and code coverage improvement for fuzzing. Master's thesis, Nanyang Technological University, Singapore, 2019. – 80 p. DOI: 10.32657/10356/105642.
5. Bavendiek S. Attack surface analysis of the Linux kernel based on complexity metrics, 2021. – 88 p. DOI: http://dx.doi.org/10.13140/RG.2.2.29943.70561.
6. Довгалюк П.М., Климушенкова М.А., Фурсова Н.И., Степанов В.М., Васильев И.А., Иванов А.А., Иванов А.В., Бакулин М.Г., Егоров Д.И. Natch: Определение поверхности атаки программ с помощью отслеживания помеченных данных и интроспекции виртуальных машин. Труды Института системного программирования РАН. 2022;34(5):89-110. DOI: 10.15514/ISPRAS-2022-34(5)-6.
7. Jennia Hizver and Tzi-cker Chiueh. 2014. Real-time deep virtual machine introspection and its applications. SIGPLAN Not. 49, 7 (July 2014), 3–14. DOI: 10.1145/2674025.2576196.

8. Brendan Dolan-Gavitt, Tim Leek, Josh Hodosh, and Wenke Lee. 2013. Tappan Zee (north) bridge: mining memory accesses for introspection. In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security (CCS '13). Association for Computing Machinery, New York, NY, USA, p. 839–850. DOI: 10.1145/2508859.2516697.
9. Schwartz E.J., Avgerinos T. and Brumley D. All You Ever Wanted to Know about Dynamic Taint Analysis and Forward Symbolic Execution (but Might Have Been Afraid to Ask). IEEE Symposium on Security and Privacy, Oakland, CA, USA. 2010, p. 317–331. DOI: 10.1109/SP.2010.26.
10. Davanian A., Qi Z., Qu Y., Yin H. DECAF++: Elastic Whole-System Dynamic Taint Analysis. Proc. of the 22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID). 2019, p. 31–45. URL: https://www.usenix.org/conference/raid2019/presentation/davanian (дата обращения: 11.09.2023).
11. Ruffling the penguin! How to fuzz Linux kernel. URL: https://hackmag.com/security/linux-fuzzing/ (дата обращения: 10.09.2023).
12. Peng H., Payer M. USBFuzz: A Framework for fuzzing USB Drivers by Device Emulation. Proc. of the 29th USENIX Security Symposium. 2020, p. 2559–2575. URL: https://www.usenix.org/conference/usenixsecurity20/presentation/peng (дата обращения: 11.09.2023).

REFERENCES:

[1] Khoroshilov A. International Linux Kernel Development Project. Sistemnyj administrator. 2022, no. 3(232), p. 32–37 (in Russian). – EDN: FSDCWC.
[2] Kozirsky B.L., Komarov T. I.; Ivanov M.A. Using Fuzz Testing for Searching Software Vulnerabilities. IT Security (Russia), [S.l.], v. 21, no. 4, 2014. ISSN 2074-7136 (in Russian). – EDN: TSEIMN.
[3] Kurmus A., Dechand S., Kapitza R. (2014). Quantifiable Run-Time Kernel Attack Surface Reduction. In: Dietrich, S. (eds) Detection of Intrusions and Malware, and Vulnerability Assessment. DIMVA 2014. Lecture Notes in Computer Science, v. 8550. Springer, Cham. DOI: 10.1007/978-3-319-08509-8_12.
[4] Peng L. Attack surface analysis and code coverage improvement for fuzzing. Master's thesis, Nanyang Technological University, Singapore, 2019. – 80 p. DOI: 10.32657/10356/105642.
[5] Bavendiek S. Attack surface analysis of the Linux kernel based on complexity metrics, 2021. – 88 p. DOI: http://dx.doi.org/10.13140/RG.2.2.29943.70561.
[6] Dovgalyuk P.M., Klimushenkova M.A., Fursova N.I., Stepanov V.M., Vasiliev I.A., Ivanov A.A., Ivanov A.V., Bakulin M.G., Egorov D.I. Natch: using virtual machine introspection and taint analysis for detection attack surface of the software. Proceedings of the Institute for System Programming of the RAS (Proceedings of ISP RAS). 2022;34(5):89-110. DOI: 10.15514/ISPRAS-2022-34(5)-6 (in Russian).
[7] Jennia Hizver and Tzi-cker Chiueh. 2014. Real-time deep virtual machine introspection and its applications. SIGPLAN Not. 49, 7 (July 2014), 3–14. DOI: 10.1145/2674025.2576196.
[8] Brendan Dolan-Gavitt, Tim Leek, Josh Hodosh, and Wenke Lee. 2013. Tappan Zee (north) bridge: mining memory accesses for introspection. In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security (CCS '13). Association for Computing Machinery, New York, NY, USA, p. 839–850. DOI: 10.1145/2508859.2516697.
[9] Schwartz E.J., Avgerinos T. and Brumley D. All You Ever Wanted to Know about Dynamic Taint Analysis and Forward Symbolic Execution (but Might Have Been Afraid to Ask). IEEE Symposium on Security and Privacy, Oakland, CA, USA. 2010, p. 317–331. DOI: 10.1109/SP.2010.26.
[10] Davanian A., Qi Z., Qu Y., Yin H. DECAF++: Elastic Whole-System Dynamic Taint Analysis. Proc. of the 22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID). 2019, p. 31–45. URL: https://www.usenix.org/conference/raid2019/presentation/davanian (accessed: 11.09.2023).
[11] Ruffling the penguin! How to fuzz Linux kernel. URL: https://hackmag.com/security/linux-fuzzing/ (accessed: 10.09.2023).
[12] Peng H., Payer M. USBFuzz: A Framework for fuzzing USB Drivers by Device Emulation. Proc. of the 29th USENIX Security Symposium. 2020, p. 2559–2575. URL: https://www.usenix.org/conference/usenixsecurity20/presentation/peng (дата обращения: 11.09.2023).

Поступила в редакцию – 26 ноября 2023 г. Окончательный вариант – 12 февраля 2024 г.
Received – November 26, 2023. The final version – February 12, 2024.