DOI: 10.15514/ISPRAS-2021-33(6)-2

Мониторинг и тестирование модулей операционных систем на основе абстрактных моделей поведения системы

[1] *Д.В. Ефремов, ORCID: 0000-0002-9916-056X <efremov@ispras.ru>*
[1] *В.В. Копач, ORCID: 0000-0003-2461-9986 <vkopach@ispras.ru>*
[1,2] *Е.В. Корныхин, ORCID: 0000-0001-9303-3132 <kornevgen@ispras.ru>*
[1,2,3] *В.В. Кулямин, ORCID: 0000-0003-3439-9534 <kuliamin@ispras.ru>*
[1,2,3] *А.К. Петренко, ORCID: 0000-0001-7411-3831 <petrenko@ispras.ru>*
[1,2,3,4] *А.В. Хорошилов, ORCID: 0000-0002-6512-4632 <khorochilov@ispras.ru>*
[1,3] *И.В. Щепетков, ORCID: 0000-0002-5794-004X <shchepetkov@ispras.ru>*

[1] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25*
[2] *Московский государственный университет имени М.В. Ломоносова, 119991, Россия, г. Москва, Ленинские горы, д. 1*
[3] *НИУ Высшая школа экономики, 101978, Россия, г. Москва, ул. Мясницкая, д. 20*
[4] *Московский физико-технический институт, 114701, Россия, Московская область, г. Долгопрудный, Институтский пер., 9*

Аннотация. В связи с высокой сложностью современных операционных систем (ОС) для спецификации даже отдельных аспектов их функциональности, как, например, функций безопасности, приходится использовать достаточно сложные модели на высокоуровневых языках. При этом задача верификации соответствия таким моделям серьезно усложняется из-за необходимости установления связей между реализацией операционной системы и моделью, представленными на сильно отличающихся языках. В данной работе мы представляем методику установления и поддержания таких связей, которую можно эффективно использовать при тестировании и мониторинге операционных систем. Описанная методика была успешно применена при тестировании и мониторинге компонентов ядра операционной системы Linux с использованием моделей на Event-B.

Ключевые слова: верификация во время выполнения; ядро Linux; LSM; IMA/EVM; Event-B; ProB

Для цитирования: Ефремов Д.В., Копач В.В., Корныхин Е.В., Кулямин В.В., Петренко А.К., Хорошилов А.В., Щепетков И.В. Мониторинг и тестирование модулей операционных систем на основе абстрактных моделей поведения системы. Труды ИСП РАН, том 33, вып. 6, 2021 г., стр. 15-26. DOI: 10.15514/ISPRAS–2021–33(6)–2

Благодарности: Данная работа выполнена при поддержке гранта РФФИ 20-01-00568.

Runtime Verification of Operating Systems Based on Abstract Models

[1] *D.V. Efremov, ORCID: 0000-0002-9916-056X <efremov@ispras.ru>*
[1] *V.V. Kopach, ORCID: 0000-0003-2461-9986 <vkopach@ispras.ru>*
[1,2] *E.V. Kornykhin, ORCID: 0000-0001-9303-3132 <kornevgen@ispras.ru>*
[1,2,3] *V.V. Kuliamin, ORCID: 0000-0003-3439-9534 <kuliamin@ispras.ru>*
[1,2,3] *A.K. Petrenko, ORCID: 0000-0001-7411-3831 <petrenko@ispras.ru>*
[1,2,3,4] *A.V. Khoroshilov, ORCID: 0000-0002-6512-4632 <khorochilov@ispras.ru>*
[1,3] *I.V. Shchepetkov, ORCID: 0000-0002-5794-004X <shchepetkov@ispras.ru>*

[1] *Ivannikov Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.*
[2] *Lomonosov Moscow State University, GSP-1, Leninskie Gory, Moscow, 119991, Russia.*
[3] *National Research University, Higher School of Economics 20, Myasnitskaya ulitsa, Moscow, 101978, Russia.*
[4] *Moscow Institute of Physics and Technology (MIPT), 9, Campus per., Dolgoprudny, Moscow region, 114701, Russia.*

Abstract. High complexity of a modern operating system (OS) requires to use complex models and high-level specification languages to describe even separated aspects of OS functionality, e.g., security functions. Use of such models in conformance verification of modeled OS needs to establish rather complex relation between elements of OS implementation and elements of the model. In this paper we present a method to establish and support such a relation, which can be effectively used in testing and runtime verification/monitoring of OS. The method described was used successfully in testing and monitoring of Linux OS core components on conformance to Event-B models.

Keywords: runtime verification; linux kernel; LSM; IMA/EVM; Event-B; ProB

For citation: Efremov D.D., Kopach V.V., Kornykhin E.V., Kuliamin V.V., Petrenko A.K., Khoroshilov A.V., Shchepetkov I.V. Runtime Verification of Operating Systems Based on Abstract Models. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 6, 2021, pp. 15-26 (in Russian). DOI: 10.15514/ISPRAS–2021–33(6)–2

Acknowledgements. This work was supported by the RFBR grant No. 20-01-00568.

1. Введение

Сложность современных операционных систем (ОС) требует соответствующей поддержки со стороны методов верификации и валидации, использующихся при их разработке. Одним из многообещающих методов, способных справиться с возрастающей сложностью, является использование моделей, которые абстрагируются от многих низкоуровневых и специфичных деталей реализации ОС. Также помогает разделение отдельных аспектов функциональности, безопасности, производительности и др. по разным уровням или модулям моделей. Объединенные вместе, данные методы значительно упрощают процессы задания и проверки требований, а также использование различных инструментов анализа и доказательства.

Практика показывает, что для точного и полного описания таких аспектов современных индустриальных ОС, как, например, контроль доступа, нужны сложные модели, разработанные с использованием достаточно выразительных языков. Мы успешно использовали Event-B [1] в нескольких проектах для описания и формальной верификации моделей контроля доступа в ОС, основанных на Linux, включая реализации мультиуровневых защитных механизмов [2, 3]. Следующим шагом является использование полученных моделей для проверки их соответствия реальному поведению ОС с целью выявления возможных расхождений и противоречий.

Учитывая многочисленные сложности, связанные с эффективным применением подходов дедуктивной верификации при работе с кодом ядра ОС, подходящими для выполнения данной задачи кажутся методы тестирования и динамической верификации [4]. Для этого требуется разработать метод тестирования и мониторинга с использованием сравнительно большой Event-B модели в качестве оракула. Статья рассматривает применение формальных моделей в мониторинге и динамической верификации поведения ОС и содержит описание ключевых аспектов предлагаемого метода.

2. Event-B модели

Event-B [1] – это формальный метод, основанный на теории множеств и логике предикатов.

Он имеет простую нотацию и поддерживается платформой для разработки и верификации моделей Rodin [5]. Модель на языке Event-B – это *дискретная система переходов*, в которой компоненты моделируемой системы описываются с помощью *состояний*, переходов между состояниями, которые называются *событиями*, и свойствами, которые должны соблюдаться в каждом состоянии (*инварианты*). Модели можно декомпозировать на отдельные модули, называемые уровнями, с помощью техники *пошагового уточнения* (stepwise refinement) [6].

Rodin способен автоматически генерировать *условия верификации*: утверждения, которые должны быть доказаны, чтобы демонстрировать корректность модели. Rodin также предоставляет средства для их автоматического и интерактивного доказательства. В качестве расширения для Rodin, а также в виде независимого инструмента, доступен инструмент анимации и проверки моделей ProB [7].

Мы используем Event-B для создания моделей контроля доступа ядер ОС, основанных на Linux. Эти модели могут быть довольно абстрактны: в таком случае они называются моделями *политик безопасности*; или же они могут быть более полными и ближе к реализации ОС: такие модели называются *функциональными спецификациями*. Обычно мы создаем модели обоих типов для каждой ОС, с которой приходится работать. Это позволяет облегчить анализ свойств безопасности и доказательство их непротиворечивости, в то же время сохраняет возможность демонстрации соответствия моделей с кодом (реализацией). Мы описали процесс работы с несколькими моделями разного уровня абстракции и доказательством их соответствия в работе [8].

Функциональные спецификации достаточно являются низкоуровневыми и описывают *системные вызовы* ядра Linux. Мы предлагаем использовать формальные спецификации как оракулы в тестировании для проверки того, что результаты выполнения системных вызовов не ведут к нарушению требований политик безопасности.

3. Контроль доступа в Linux

Ядро Linux, начиная с версии 2.5 (2002 год) [9], включает в свой состав подсистему Linux Security Modules (LSM). Данная подсистема представляет собой набор модулей, усиливающих безопасность ядра и реализующих управление контролем доступа. Взаимодействие между модулями и ядром организовано с помощью API интерфейса LSM.

Этот API позволяет реализовывать дополнительные проверки доступа, усиливающие существующие. Он не предназначен для изменения состояния ядра, за пределами специально отведенных для этого интерфейса полей в структурах данных ядра.

Таким образом, побочный эффект модулей безопасности, корректно использующих этот интерфейс с архитектурной точки зрения, сводится только к досрочному завершению обработки какого-либо события в ядре. Ровно таким же образом работает механизм уточнений в Event-B.

В настоящий момент (ядро Linux версии 5.15) интерфейс представляет собой список из 238 (см. LSM_HOOK в include/linux/lsm_hook_defs.h) возможных типов обработчиков событий в ядре. Модуль безопасности при инициализации явным образом регистрирует собственные обработчики, которые необходимы ему для реализации заложенных в нём политик безопасности. Ядро Linux в процессе своей работы и при наступлении определенных условий вызывает зарегистрированные обработчики либо для оповещения о каком-то из событий, например, о выделении нового индексного дескриптора в файловой подсистеме, либо для контроля операции со стороны модулей безопасности. В последнем случае в зависимости от результата, который возвращает модуль безопасности, обработка операции может быть либо продолжена ядром, либо завершена с кодом ошибки запрета доступа.

LSM интерфейс постоянно меняется в процессе разработки ядра, туда добавляются как новые контролирующие и оповещающие функции, так и убираются неиспользуемые и избыточные операции (см. рис. 1).

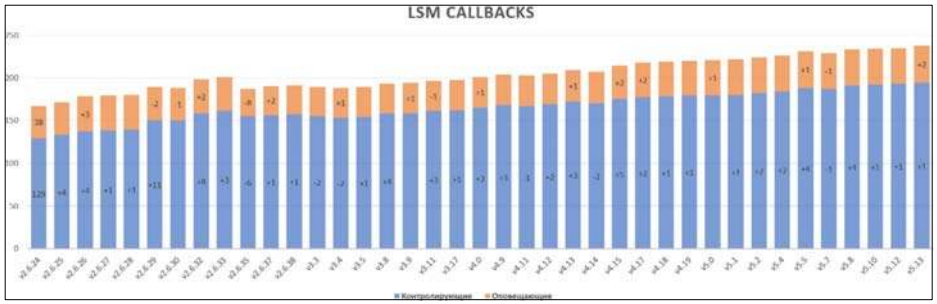

Рис. 1. Изменения LSM интерфейса в ядре Linux
Fig. 1. LSM interface changes in the Linux kernel

Изначально несколько LSM модулей не могли работать совместно. Позднее, разработчики ядра доработали интерфейс [10], чтобы каждому LSM обработчику соответствовал список в ядре. В таком случае ядро вызывает все зарегистрированные обработчики LSM по списку, в соответствии с приоритетом регистрации их модулей безопасности в ядре. Это дает возможность включать один основной (major) модуль безопасности, который прикрепляет собственные данные и метки к структурам ядра, и несколько вспомогательных (minor), которые не требуют хранения дополнительных данных для реализации политик безопасности. Постепенно разработчики ядра дорабатывают интерфейс [11] и структуры ядра с той целью, чтобы хранимые модулями безопасности данные также отделялись друг от друга, что позволит включать несколько основных модулей одновременно. Приоритет вызова этих модулей по-прежнему останется.

Модуль IMA/EVM (Integrity Measurement Architecture and Extended Verification Module) контроля целостности Linux также представляет собой модуль безопасности LSM, но его проверки всегда вызываются последними. Так сложилось исторически из-за необходимости обеспечивать одновременную работу модуля контроля целостности с одним из модулей мандатного контроля (SELinux, Apparmor, Smack, Tomoyo) до того, как интерфейс LSM позволил включать несколько модулей одновременно. На текущий момент это также остается актуальным, так как проверки целостности (IMA) требуют сверки контрольной суммы, записанной в расширенных атрибутах файла, с контрольной суммой содержимого файла. Подсчет контрольных сумм является затратной операцией и её рационально откладывать до того момента, когда все остальные проверки уже выполнены. Модуль также позволяет контролировать целостность не только содержимого файла (IMA), но и его метаданных (EVM). Под метаданными в данном случае подразумеваются атрибуты файла,

расширенные атрибуты, права, владелец, группы. Модуль также поддерживает проверку криптографической подписи контрольных сумм.

IMA/EVM имеет несколько режимов работы. Для понимания принципов работы модуля важны только два из них. В первом для файлов высчитываются контрольные суммы, которые затем записываются в расширенные атрибуты. Проверок доступа при этом не выполняется. Во втором режиме выполняется сверка контрольных сумм при доступах к файлу на открытие, запись, выполнение. В случае их несовпадения доступ не предоставляется. Эти режимы не переключаются динамически: ядро ОС должно загрузиться в одном из них.

Важной особенностью LSM интерфейса является то, что ядро вызывает зарегистрированные обработчики только после базовых проверок параметров и после дискреционных проверок доступа. Политику дискреционного доступа отменить нельзя, и реализуется она не в отдельном модуле безопасности, а самим ядром.

Таким образом, на текущий момент контроль доступа в ядре Linux при обработке системного вызова выглядит следующим образом. Сначала выполняется базовая проверка аргументов системного вызова на корректность, далее самим ядром осуществляется проверка привилегий (capabilities), если это подразумевается системным вызовом и его аргументами (например, работа с сырыми сокетами требует привилегии CAP_NET_RAW у процесса), затем также самим ядром выполняется дискреционная проверка прав, после этого ядром вызываются модули контроля доступа LSM в порядке приоритета их регистрации. После того как все LSM модули разрешили доступ, ядро обрабатывает системный вызов и возвращает результат. Если на каком-то из этапов доступ запрещается, то ядро досрочно завершает обработку системного вызова и возвращает процессу, который его инициировал, код ошибки (например, -EPERM или -EACCES). Как можно видеть, реализация контроля доступа в ядре не монолитна, а состоит из базовой неотключаемой части и из дополнительных модулей, которые только могут вводить лишь новые проверки, ограничивающие доступ, никак не влияя на результаты прошлых проверок. Такое разделение хорошо ложится на Event-B модели и механизм уточнений.

Модули безопасности явно полагаются на то, что ядро корректным образом вызывает функции LSM интерфейса во время обработки событий. Политики безопасности, которые реализуются этими модулями, подразумевают, что LSM интерфейс достаточен (содержит необходимые контролирующие обработчики) для их реализации в виде модулей безопасности. При динамической верификации мы проверяем не только то, что модуль безопасности реализует политику безопасности корректным образом, но также то, что ядро вызывает модуль безопасности во время обрабатываемых событий и достаточность самого LSM интерфейса для реализации требуемой политики безопасности.

С точки зрения выявления дефектов реализации политик безопасности наиболее приоритетны расхождения вида "модель запрещает"<==>"ядро ОС разрешает". На практике расхождения, связанные с нехваткой физических ресурсов, крайне редки, а большинство выявляемых расхождений связано с тем, что модель недостаточно полно отражает поведение реальной системы. Многие пограничные случаи не описаны в официальной документации системных вызовов и выявляются на практике в виде расхождений.

4. Динамическая верификация

Далее, под моделью понимается функциональная спецификация поведения системных вызовов ядра ОС на языке Event-B.

В общем виде, схема динамической верификации выглядит следующим образом.

1) Подготавливается начальное окружение для теста. Сюда входят файлы, директории, ссылки, владельцы и группы, их права доступа, расширенные атрибуты модуля мандатного контроля доступа и модуля контроля целостности и другие данные, необходимые для запуска теста.
2) Снимаются данные начального окружения, подготовленные на прошлом шаге, как и данные самого файла теста. Они записываются в базу данных.
3) Осуществляется контролируемое тестовое воздействие на ядро ОС в изолированной среде.
4) Параллельно с прошлым шагом в базу данных записываются результаты обработки системных вызовов ядром ОС с включенными модулями безопасности.
5) Собранные данные о начальном окружении теста и результатах обработки системных вызовов ядром ОС преобразовываются в вид (трассу), пригодный для воспроизведения на модели.
6) Запускается процесс анимации модели с использованием инструмента ProB.
7) Состояние модели наполняется данными, снятыми с начального окружения теста на шаге 2 и преобразованными на шаге 5.
8) Системные вызовы с их аргументами и результатами, снятые на шаге 4 и преобразованные на шаге 5, анимируются на модели. В случае расхождения результатов реальных операций и модельных операций воспроизведение трассы прекращается, результаты о причинах расхождения записываются в журнал тестирования. В некоторых случаях расхождение не критично. Тогда анимация откатывается до состояния прошлого системного вызова и продолжается с пропуском системного вызова, на котором возникло расхождение. Процесс анимации завершается успехом, когда все системные вызовы были воспроизведены на модели и результаты их обработки ядром ОС соответствуют модельным.
9) Процесс повторяется для каждого теста.

Шаги 1-4 и шаги 5-8 могут быть разнесены по времени и выполняться отдельно друг от друга на разных машинах.

На шаге 8 возможны ситуации, когда результаты обработки системных вызовов совпадают и когда они не совпадают. В первом случае процесс анимации продолжается.

Лишь в случае, когда модель запрещает доступ, а ядро операционной системы запрещает доступ с кодом ошибки, не связанным с его контролем (например, код нехватки памяти -ENOMEM), в журнал теста заносится предупреждение. В ситуации, когда модель запрещает доступ, а ядро ОС его разрешило, анимация прекращается, в журнал теста заносятся ошибка, состояние модели, нарушенное в модели условие.

Обратная ситуация (модель разрешает, ядро ОС запрещает) также может быть связана с ошибкой реализации политики безопасности. Но она менее критична с точки зрения безопасности, поэтому в таком случае процесс анимирования продолжается с пропуском системного вызова, на котором произошло расхождение. Если в реальной системе доступ был запрещен по причине нехватки ресурсов, например, код ошибки -ENOMEM, то событие просто пропускается при воспроизведении. Если же код ошибки связан с контролем доступа (например, -EPERM или -EINVAL), то это может свидетельствовать о некритичной ошибке в реализации, приводящий к запрету доступа там, где это не требуется с точки зрения политики безопасности. Информация о таком расхождении заносится в журнал теста, воспроизведение продолжается.

4.1 Реализация

Для реализации динамической верификации были разработаны компоненты мониторинга системных вызовов в ядре ОС, сбора информации о начальном окружении теста, трансляции трасс, а также компонент воспроизведения трасс на модели.

Компонент сбора информации о начальном окружении для теста собирает данные о пользователях в системе, их группах, информацию о файлах теста, передаваемых ему в качестве аргументов. С файлов и самого теста снимается информация о дискретных правах, владельце, группе, индексном дескрипторе, томе монтирования, полная информация о файлах и директориях до корневого раздела.

Монитор системных вызовов реализован в виде модуля ядра Linux и программы пользовательского пространства. Модуль мониторинга использует механизмы ядра Linux kprobes, kretprobes для перехвата аргументов системных вызовов и результатов их обработки, а также сокеты netlink для отправки сообщений с событиями системных вызовов программе пользовательского пространства. Последняя управляет процессом мониторинга и записывает его результаты в базу данных.

Компонент трансляции трасс осуществляет преобразование трасс системных вызовов с реальной системы в трассу для воспроизведения на модели. В качестве аргументов использует данные о начальном окружении теста, а также трассу, которая ранее была собрана при мониторинге. При трансляции трассы все реальные сущности транслируются в номерные элементы состояния модели. В модельную трассу также добавляются дополнительные события, которые соответствуют шагам подготовки начального окружения из тестов. Например, добавляется событие init, позволяющее за один раз обновить состояние модели до соответствия начальному окружению теста. Если бы его не было, требовалось бы реконструировать последовательность системных вызовов, необходимую для достижения этого состояния модели. Также, например, в спецификацию Event-B добавляется дополнительное псевдособытие login, которое позволяет в трассе отделить подготовительные действия с моделью от начала воспроизведения самого теста.

Компонент анимации осуществляет воспроизведение трасс на модели и сверку результатов доступов, полученных на реальной системе, на соответствие модели. В конце работы выводит статистику покрытия элементов модели и результат тестирования, нарушенные в модели условия, в случае наличия таковых. Компонент реализован на основе библиотеки ProB de.prob2.kernel.

4.2 Тесты

Далее мы опишем тесты, используемые в рамках динамической верификации. Тест – это программа, осуществляющая тестовое воздействие на тестируемую ОС. Тестовым воздействием является один из системных вызовов ядра Linux, включенных в функциональную спецификацию. С тестом связано описание состояния ОС перед тестовым воздействием (какие должны существовать файлы, директории, пользователи и группы, метки и т.п.). Это описание вместе с целевым системным вызовом хранится в базе данных и используется для подготовки начального окружения.

Основой для построения тестового набора были функциональные требования системных вызовов, представленные в модели. Альтернативой было бы использование самой модели, ее структуры, для построения тестового набора. Использование требований позволяет при помощи тестового набора проверить не только реализацию ОС, но и модель.

Апробация подхода проводилась на модели системных вызовов, взаимодействующих с файловой системой. Список этих системных вызовов такой: open, creat, mkdir, getdents, getxattr, setxattr, link, symlink, chmod, chown, execve, unlink.

Функциональные требования были поделены на 3 категории: свойства дискретного контроля доступа (DAC), свойства мандатного контроля доступа (MAC) и свойства целостности (IMA/EVM) [12]. Каждая категория имеет свои подкатегории, например, доступа к объекту, обхода пути и т.д. Тестами покрывается доступ для каждой из категорий для каждого системного вызова для каждого типа объекта (файл, директория, ссылка) для каждого типа

пользователя (привилегированный, непривилегированный, со специальными возможностями) для каждого режима системного вызова (если применимо).

Тесты минималистичны, независимы и воспроизводимы: они выполняют только один системный вызов под мониторингом. Данный подход позволяет упростить отладку ядра ОС вместе с системой контроля доступа и выявлять неточности/неполноту в функциональной спецификации. Также подход упрощает трансляцию тестовой трассы в модельную и упрощает процесс воспроизведения трассы на модели.

Табл. 1. Количество тестов
Table 1. Number of tests

syscall	DAC	MAC	IMA/EVM	all
open	141	14084	45	14270
Creat	57	9268	15	9340
mkdir	57	9268	15	9340
getdents	0	26	0	26
getxattr	110	9372	30	9512
setxattr	110	9372	30	9512
link	57	9268	15	9340
symlink	55	9268	15	9338
chmod	110	9372	30	9512
chown	110	9372	30	9512
execve	55	4712	15	4782
unlink	3015	27804	15	30834
all	3877	121186	255	125318

Используется комбинаторный подход для генерации тестов. Тесты DAC проверяют доступ к объекту (файлу, директории), обход пути до объекта, директории с установленными битами SetGID и Sticky. В тестах перебираются тип объекта (файл/директория), системный вызов, владельцы объекта, пользователь и группа субъекта доступа (процесса), списки ACL, дополнительные группы субъекта, длина пути до объекта и права доступа директорий в пути.

Тесты MAC проверяют мандатные правила доступа к объекту, к его родительской директории, доступа к объекту при наличии специальных атрибутов. В этих тестах перебираются тип объекта (файл/директория), мандатная метка объекта и субъекта (процесса), системный вызов, владелец объекта, владелец субъекта, длина пути к объекту, мандатные метки директорий в пути к объекту, специальные атрибуты.

В тестах подсистемы целостности IMA/EVM варьируются следующие параметры тестов:

- подпадает ли объект доступа под контроль целостности;
- атрибут неизменности (immutable bit) объекта доступа;
- контрольная сумма файла, хранящаяся в расширенных атрибутах файла.

Для минимизации тестового набора не генерируются все пары мандатных меток для субъектов и объектов. Вместо этого используются только наборы пар из меток, находящихся в различных важных соотношениях – метки эквивалентны, первая доминирует, вторая доминирует, метки несравнимы.

По итогу воспроизведения трассы на модели печатается статистика покрытия элементов модели. Планируется использовать эту информацию для дополнения тестового набора для более полного покрытия элементов модели и частей условий в модели.

5. Результаты

В рамках данной работы был реализован прототип системы динамической верификации, сверяющий поведение ядра Linux вместе с закрытым модулем мандатного контроля доступа

и модулем контроля целостности IMA/EVM с абстрактными моделями поведения системы на языке Event-B, формализующих политики безопасности и требования контроля доступа к операционной системе в целом. Прототип был апробирован на наборе системных вызовов Linux, связанных с доступом к файловой системе.

Это позволило выявить ошибки в модели системного вызова setxattr, когда разрешались только флаги XATTR_CREATE и XATTR_REPLACE и не учитывалось, что переменная флага может не иметь специального значения и быть равной нулю. Также модель избыточно требовала наличия прав доступа на чтение и на выполнение к исполняемому файлу при системном вызове execve. Ядро операционной системы наличие права на чтение не проверяет. В модели некорректно проверялись права дискреционного доступа на запись в каталоге, в котором пользователь создает ссылку на файл, что приводило к ситуации, когда доступ запрещался в реальной системе, но разрешался моделью.

В поставляемой вместе с модулем мандатного контроля доступа утилите редактирования меток мандатного контроля доступа была найдена ошибка, проявлявшаяся в некорректном назначении прав доступа на файл при использовании флага игнорирования части проверок.

В реализации модуля мандатного контроля доступа была выявлена логическая ошибка, связанная с тем, что при создании ссылок не проверялись права доступа к директории файла, на который создается ссылка. Это приводило к ситуации, когда ссылка успешно создавалась, чего не могло происходить с точки зрения политики безопасности. Также в реализации модуля контроля целостности метаданных EVM была выявлена ошибка, связанная с тем, что метки безопасности мандатного контроля доступа не измерялись (не вычислялась контрольная сумма). Изменение мандатной метки должно было привести к выявлению нарушения целостности метаданных файла, чего в реализации не происходило.

6. Аналогичные работы

В работе [13] рассматривается вопрос корректности вызовов функций LSM интерфейса ядром, нет ли путей в коде, в которых пропущены эти вызовы. Авторы используют тот факт, что в большинстве случаев вызовы LSM функций в ядре расставлены корректно. Они собирают трассы системных вызовов ядра, их аргументов, результаты LSM функций и функций доступа к основным структурам данных ядра. На основе собранной информации выявляются аномалии, когда в большинстве случаев при системном вызове LSM интерфейс контролирует доступ к данным ядра, но есть пути в коде ядре, где требуемые проверки отсутствуют.

В продолжении [14] своей прошлой работы авторы проверяют не только необходимость LSM вызовов на определенных путях исполнения в ядре Linux, но и их достаточность. В прошлой работе авторы делали предположение о том, что в большей части ядра необходимые LSM вызовы уже расставлены корректным образом. В данной работе авторы моделируют необходимые LSM вызовы при определенных типах доступа к структурам данных ядра. Выявляются дополнительные ошибки, когда LSM вызов есть, но часть данных им не контролируется и не проверяется, что может приводить к изменению критических структур данных без соответствующих проверок и к полному игнорированию их в будущем за счет неавторизованного повышения привилегий. Авторами используется тот же подход с динамическим сбором трасс выполнения с ядра Linux и их последующим статическим анализом.

В работе [15] используется динамическая верификация для анализа модулей ядра Linux, обсуждаются подходы к инструментации загруженных модулей ядра Linux и представляется инструмент динамической верификации KEDR. Данный инструмент позволяет перехватывать выполнение функций, отслеживать утечки памяти, имитировать (fault injection) коды ошибок некоторых функций ядра для проверок путей их обработки в модулях.

Работа [16] использует автоматы для анализа состояний потоков ядра в рамках ядра реального времени PREEMPT_RT Linux. Важным отличием от остальных работ является то, что весь анализ производится в самом ядре, так как частота перехватываемых событий настолько велика, что невозможно производить запись о них в журнал событий, записывать его из ядра в файл и анализировать после. Весь анализ производится непосредственно в момент наступления событий в ядре. Авторы описывают несколько небольших автоматных моделей состояний ядра, транслируют их в код на языке Си, собирают в модули ядра и прикрепляют события переходов из одного состояния автомата в другое к вызовам функций ядра с помощью механизмов трассировки Linux. Самого факта вызова определенных функции без анализа его аргументов и контекста его вызова достаточно в рамках рассматриваемых автоматных моделей.

7. Заключение

В данной работе мы представляем метод проведения тестирования и мониторинга компонентов ядра ОС на основе спецификаций части функциональности, зафиксированной в формальной модели на языке Event-B. Описанный метод позволяет эффективно верифицировать реализацию ОС на соответствие достаточно сложным формальным моделям, представленным на языке, отличном от языка реализации, при помощи установления соответствия между событиями модели и системными вызовами, а также переменными модели и специфическими данными, получаемыми из трассировки параметров системных вызовов. Особенностью метода является построение достаточно сложной исходной конфигурации ОС, в рамках которой становится возможно реализовать необходимое разнообразие тестовых ситуаций, покрывающих разнообразные условия, присутствующие в спецификациях событий в формальной модели.

Метод был успешно применен для тестирования нескольких разных реализаций модуля ядра ОС Linux LSM, отвечающего за выполнение дополнительных к традиционным для Linux процедур контроля доступа. В тестируемых компонентах LSM были реализованы правила многоуровнего или мандатного управления доступом.

Список литературы / References

[1]. J.-R. Abrial. Modeling in Event-B: System and Software Engineering. Cambridge University Press, 2010, 612 p.

[2]. P.N. Devyanin, A.V. Khoroshilov et al. Using Refinement in Formal Development of OS Security Model. Lecture Notes in Computer Science, vol. 9609, 2016, pp. 107-115.

[3]. V. Kuliamin, A. Khoroshilov и D. Medveded. Formal Modeling of Multi-Level Security and Integrity Control Implemented with SELinux. In Proc. of the International Conference on Actual Problems of Systems and Software Engineering (APSSE), 2019, pp. 131-136.

[4]. А.К. Петренко, Д.В. Ефремов и др. Мониторинг и тестирование на основе многоуровневых спецификаций программ. Труды ИСП РАН, том 32, вып. 6, 2020 г., стр. 7-18 / A.K. Petrenko, D.V. Efremov et al. Monitoring and testing based on multi-level program specifications. Trudy ISP RAN/Proc. ISP RAS, vol. 32, issue 6, 2020, pp. 7-18 (in Russian). DOI: 10.15514/ISPRAS–2020–32(6)–1.

[5]. J.-R. Abrial, M. Butler et al. Rodin: an open toolset for modelling and reasoning in Event-B. International Journal on Software Tools for Technology Transfer, vol. 12, issue 6, 2010, pp. 447-466.

[6]. N. Wirth. Program Development by Stepwise Refinement. Communications of the ACM, vol. 14, issue 4, 1971, pp. 221-227.

[7]. M. Leuschel, M. Butler. ProB: A Model Checker for B. Lecture Notes in Computer Science, vol. 2805, 2003, pp. 855-874.

[8]. A. Khoroshilov, V. Kuliamin et al. A State-based Refinement Technique for Event-B. In Proc. of the Ivannikov Memorial Workshop (IVMEM), 2020, pp. 49-54.

[9]. G. Kroah-Hartman. LSM: Add all of the new security/* files for basic task control. URL: https://git.kernel.org/pub/scm/linux/kernel/git/tglx/history.git/commit?id=2b15fe6334aebd7d3340f8b826acb79b138afa74, accessed 15.01.2022.

[10]. J. Edge. Progress in security module stacking. URL: https://lwn.net/Articles/635771/, accessed 15.01.2022.
[11]. J. Edge. LSM stacking and the future. URL: https://lwn.net/Articles/804906/, accessed 15.01.2022.
[12]. Linux Integrity Subsystem. URL: http://linux-ima.sourceforge.net/, accessed 15.01.2022.
[13]. A. Edwards, T. Jaeger, X. Zhang. Runtime verification of authorization hook placement for the Linux security modules framework. In Proc. of the 9th ACM Conference on Computer and Communications Security, 2002, pp. 225-234.
[14]. T. Jaeger, A. Edwards, X. Zhang. Consistency analysis of authorization hook placement in the Linux security modules framework. ACM Transactions on Information and System Security (TISSEC), vol. 7, issue 2, 2004, pp. 175-205.
[15]. V.V. Rubanov, E. A. Shatokhin. Runtime verification of linux kernel modules based on call interception. In Proc. of the Fourth IEEE International Conference on Software Testing, Verification and Validation, 2011, pp. 180-189.
[16]. D.B. de Oliveira, T. Cucinotta, R.S. de Oliveira. Efficient formal verification for the Linux kernel. Lecture Notes in Computer Science, vol. 11724, 2019, pp. 315-332.

Информация об авторах / Information about authors

Денис Валентинович ЕФРЕМОВ – научный сотрудник. Сфера научных интересов: формальная верификация, статический и динамический анализ.

Denis Valentinovich EFREMOV – researcher. Research interests: formal verification, static and dynamic analysis.

Виктория Владимировна КОПАЧ – научный сотрудник. Сфера научных интересов: методы верификации и валидации, анализ полноты требований.

Viktoria Vladimirovna KOPACH – researcher. Research interests: verification and validation methods, requirements completeness analysis.

Евгений Валерьевич КОРНЫХИН – кандидат физико-математических наук, доцент кафедры системного программирования МГУ, старший научный сотрудник ИСП РАН. Область интересов: формальная дедуктивная верификация моделей, тестирование на основе моделей.

Eugeny Valerievich KORNYKHIN – Ph.D. in Physics and Mathematics, Associate Professor of system programming departments at MSU and Senior Researcher at ISP RAS. Fields of Interest: formal deductive verification, model-based testing.

Виктор Вячеславович КУЛЯМИН – кандидат физико-математических наук, ведущий научный сотрудник ИСП РАН, доцент кафедр системного программирования МГУ и ВШЭ. Область интересов: формальная дедуктивная верификация моделей, тестирование на основе моделей.

Viktor Vyacheslavovich KULIAMIN – Ph.D. in Physics and Mathematics, leading researcher at ISP RAS, associate professor of system programming departments at MSU and the HSE. Fields of Interest: formal deductive model verification, model-based testing.

Александр Константинович ПЕТРЕНКО – доктор физико-математических наук, профессор, заведующий отдела ИСП РАН, профессор МГУ и ВШЭ. Область интересов: формальные методы программной инженерии, тестирование программного и аппаратного обеспечения, формальная спецификация требований.

Alexander Konstantinovich PETRENKO – Doctor of Physical and Mathematical Sciences, Professor, Head of the Department of ISP RAS, Professor of MSU and HSE. Areas of interest: formal methods of software engineering, testing of software and hardware, formal specification of requirements.

Алексей Владимирович ХОРОШИЛОВ – кандидат физико-математических наук, ведущий научный сотрудник, директор Центра верификации ОС Linux в ИСП РАН, доцент кафедр системного программирования МГУ, ВШЭ и МФТИ. Основные научные интересы: методы

проектирования и разработки ответственных систем, формальные методы программной инженерии, методы верификации и валидации, тестирование на основе моделей, методы анализа требований, операционная система Linux.

Alexey Vladimirovich KHOROSHILOV – Ph.D. in Physics and Mathematics, Leading Researcher, Director of the Linux OS Verification Center at ISP RAS, Associate Professor of System Programming Departments at MSU, HSE, and MIPT. Main research interests: design and development methods for critical systems, formal methods of software engineering, verification and validation methods, model-based testing, requirements analysis methods, Linux operating system.

Илья Викторович ЩЕПЕТКОВ – младший научный сотрудник. Область интересов: разработка и верификация формальных моделей компьютерных систем.

Ilya Viktorovich SHCHEPETKOV – Junior Researcher. Fields of Interest: development and verification of formal models of computer systems.