НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

НАУЧНЫЙ ДОКЛАД

по результатам подготовленной
научно-квалификационной работы (диссертации)

«Верификация ядра операционной системы для анализа выполнения высокоуровневых требований защиты информации»

ФИО Ефремов Денис Валентинович

Направление подготовки 02.06.01 Компьютерные и информационные науки

Профиль (направленность) программы 05.13.11 Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

Аспирантская школа по компьютерным наукам

Аспирант _____/ФИО/
подпись

Научный руководитель _____/ФИО/
подпись

Директор Аспирантской школы по компьютерным наукам_____/ФИО/
подпись

Москва, 2018

ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ

Актуальность работы

Повсеместная информатизация обуславливает рост числа угроз экономике и безопасности, тем самым актуализируя задачи обеспечения информационной безопасности. Системы информационной безопасности (СЗИ) и контроля доступа в своей программной части обыкновенно имеют резидентный компонент, реализованный как драйвер/модуль ядра операционной системы (ОС). Подобное решение предоставляет СЗИ необходимый уровень контроля над событиями в ОС, при этом обеспечивая самому компоненту базовый уровень изоляции от программного обеспечения (ПО) пользовательского уровня.

Современные массовые операционные системы — это большие и сложные системы. Обоснование корректности поведения данных систем является одной из наиболее важных задач программной инженерии. Тема верификации ядер операционных систем активно развивается последние 20 лет. Существуют успешные попытки верификации отдельных составляющих операционных систем, построения корректных микроядер и гипервизоров, однако существующие методы в нынешнем виде не всегда подходят для верификации готовых компонентов массовых операционных систем, в том числе модулей ядра операционной системы Linux.

Одним из самых высоких требований к разработке СЗИ является наличие формальной модели с доказательством непротиворечивости последней. Однако наличие формальной модели СЗИ само по себе не способно гарантировать корректность работы системы и соблюдения модельных функциональных требований, так как формальная модель — это не реализация самой системы. Полагается, что реализация пишется на основе имеющейся формальной модели, тем устанавливая неформальное соответствие между моделью и реализацией.

Существуют также отдельные требования на доказательство корректности реализации в понимании отсутствия в последней определенных классов ошибок (имеется ввиду, например, undefined behavior). Этими требованиями закрываются потенциальные ошибки, которые могут привносится на реализационном уровне за счет большей детализации последнего по сравнению с модельным.

Формальное или хотя бы полу-формальное соответствие между моделью и реализацией в большинстве случаев сложно показать за счет разницы в уровне абстракций. Однако подобная задача актуальна как с точки зрения разработчика программного обеспечения, так и с точки зрения разработчика модели. Разработчику необходимо быть уверенным в правильном понимании модели и на раннем уровне выявлять несоответствия в коде. Разработчику модели необходимо иметь обратную связь в том смысле, что порой модель может быть написана на слишком абстрактном уровне, опираться на предположения, не выполняющиеся в реальности, требовать большей детализации.

На текущей момент в стандартах, как правило, не требуется формальное обоснование соответствия между моделью и реализацией. Но развитие методов и технологий, а также повсеместная информатизация общественных институтов, подталкивает данное требование к включению в списки требований на высокие оценочные уровни доверия.

Предметом исследования работы являются монолитные ядра массовых операционных систем, модели их функционирования, их верификация и тестирование. Объектом исследования работы является модуль безопасности ядра Linux - СЗИ дистрибутива специального назначения AstraLinux, его соответствие мандатной сущностно-ролевой ДП-модели.

Актуальность данной темы определяется многообразием компонентов, используемых в операционной системе Linux и реализованных в виде модулей ядра, корректность которых является важнейшей составляющей надежности всей

системы, а также все возрастающими требования стандартов безопасности и сертификационными требованиями о защите информации, содержащейся, например, в государственных информационных системах.

Цель и задачи работы

Целью данной диссертационной работы является создание методов и инструментов, позволяющих производить оценку больших объёмов кода на соответствие формальным моделям, применение которых давало бы возможность повысить общий уровень безопасности, надёжности и устойчивости к ошибкам исследуемых систем; апробация созданных инструментов в ходе работ по верификации модуля безопасности операционной системы GNU/Linux.

Для достижения поставленной цели в диссертационной работе должны быть решены следующие научно-практические задачи:

- Проведён аналитический обзор современной научно-технической, нормативной, методической литературы, затрагивающей научно-техническую проблему, исследуемую в рамках диссертационной работы;
- Разработана методология разработки формальных спецификаций на ядро Linux;
- Разработан метод выявления несоответствий между формальной моделью и тестируемой системой с её формальными спецификациями;
- Разработаны критерии оценки полноты верификации, покрытия событий операционной системы и событий формальной модели и её спецификаций;
- Создана автоматическая система тестирования крупного программного проекта с применением разработанных методов.

Научная новизна

Научная новизна диссертационной работы заключается в получении следующих оригинальных результатов:

- Метод дедуктивной верификации частей массовых операционных систем снизу-вверх («от реализации к модели») с интеграцией в непрерывный цикл разработки;
- Метод тестирования формальных спецификаций на ядро Linux, транслированных в проверки времени выполнения;
- Метод выявления несоответствий между формальной моделью и анализируемой системой посредством трансляции формальной модели в исполняемую;
- Алгоритм воспроизведения трассы обработки системных вызовов на исполняемой модели;
- Публично доступный пакет формальных спецификаций к библиотечным функциям ядра Linux.

Практическая значимость

Результаты диссертации могут быть использованы при разработке программного обеспечения в соответствии с требованиями к мерам защиты и подтверждению их корректности 3 части («Компоненты доверия к безопасности») стандарта ГОСТ Р ИСО/МЭК 15408 на оценочный уровень доверия 7.

Использование результатов работы

Результаты диссертационной работы использовались при прохождении сертификации операционной системы специального назначения Astra Linux Special Edition на соответствие требованиям документов «Требования безопасности информации к операционным системам» (ФСТЭК России, 2016 г.) и «Профиль защиты операционных систем типа «А» второго класса защиты ИТ.ОС.А2.ПЗ» (ФСТЭК России, 2016 г.).

Результаты диссертации внедрены в программное обеспечение дедуктивной верификации моделей и механизмов защиты ОС AstraVer Toolset Института системного программирования им. В.П. Иванникова РАН.

Также часть результатов диссертации использовалась в учебном процессе факультета компьютерных наук НИУ ВШЭ в бакалаврской программе «Программная инженерия» в курсе «Верификация программ».

Методология и методы исследования

При выполнении работы использовались формальные методы верификации, методы статического и динамического анализа, системного программирования, теории алгоритмов и сложности, аппарат теории множеств и логики высшего порядка. Разработанные алгоритмы программно реализованы с использованием технологий процедурного, функционального и объектно-ориентированного программирования. Формальные спецификации реализованы с использованием парадигмы смены состояний и логической парадигмы. Для оценки эффективности решений применяются тесты производительности.

Положения, выносимые на публичное представление

1. Метод дедуктивной верификации частей массовых операционных систем;
2. Метод тестирования формальных спецификаций на ядро Linux;
3. Алгоритм выявления несоответствий между формальной моделью и анализируемой системой;
4. Метод трансляции формальных моделей в исполняемые.

Апробация работы

Результаты работы входят в следующие проекты:

- НИР «Исследование и верификация модели управления доступом и информационными потоками, реализованной в операционной системе Astra Linux Special Edition». В рамках проекта автор занимался: анализом соответствия между требованиями формальной модели и функциями, реализованными в модуле безопасности ОС Astra Linux Special Edition, разработкой спецификаций для библиотечных функций ядра Linux, реализацией поддержки интеграции статической и динамической верификации, динамическим анализом модуля безопасности на предмет обработки нестандартных ситуаций (fault injection анализ).
- ПНИ «Разработка методов и инструментов для дедуктивной верификации модулей ядра операционной системы Linux». В рамках проекта автор занимался: анализом опасных приемов программирования на языке Си, разработкой тестовых пакетов для инструментов дедуктивной верификации, разработкой и публикацией материалов о возможностях системы дедуктивной верификации.

Основные результаты диссертационной работы докладывались на:

- Весенний коллоквиум молодых исследователей в области программной инженерии (SYRCoSE: Spring Young Researchers Colloquium on Software Engineering), Санкт-Петербург, 2014 г. (не вошло в период обучения в аспирантуре НИУ ВШЭ);
- Семинар по технологиям разработки и анализа программ ВМиК МГУ, Москва, 2015 г.;
- Научно-практическая Открытая конференция ИСП РАН, Москва, 2016 г.;
- Конференции по безопасности Positive Hack Days 17, Москва, 2017 г.;

- Научно-практическом семинаре "Моделирование и верификация политик безопасности управления доступом в операционных системах" Москва, 2017 г.;
- Научно-практическая Открытая конференция ИСП РАН, Москва, 2017 г.;
- Зимней школе 2nd ARVI COST School on Runtime Verification, Праз-Сюр-Арли, 2018 г.;
- Научно-практической конференции OS DAY 2018, Москва, 2018 г.;

Запланированные доклады:

- Конференция 8th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation, Лимассол, ноябрь 2018 г.
- Научно-практическая Открытая конференция ИСП РАН, Москва, 2018 г.;

Публикации

Основные результаты по теме диссертации изложены в работах [1-7], 2 из которых изданы в журналах, рекомендованных ВАК [1, 2], 2 — находится в базе Scopus [6, 7], 1 — в тезисах докладов [3], 1 — отдельный изданием [5]. Автором также получено 1 свидетельство о государственной регистрации программы для ЭВМ [4]. За время обучения в аспирантуре НИУ ВШЭ были подготовлены работы [1-2, 5-7].

В статье [1] автором написаны разделы 3,4,6,7,8 в которых излагаются метод разработки спецификаций и метод тестирования инструментов верификации, описываются результаты работы. В статье [3] автором написаны разделы 3,4,5 в которых излагается метод дедуктивной верификации модуля ядра Linux, описываются разработанные автором инструменты сопровождения процесса верификации. В статье [6] автором написаны разделы 2 и 4, посвященные описанию стратегии верификации и результатам использования техники авто-

активных доказательств. В работе [7] автором написаны разделы 2,3,6,7,8, посвященные рассмотрению аналогичных работ, методу верификации и разработки формальных спецификаций, выявленным проблемам в инструментах дедуктивной верификации и их тестированию. В монографии [5] автором написаны главы 6 и 7, в которых рассматривается разработка формальных спецификаций для модуля безопасности, перенос функциональных требований с абстрактной модели на реализацию, метод динамической верификации соответствия реализации высокоуровневой модели. В свидетельстве о государственной регистрации программ для ЭВМ [4] вклад автора состоит в разработке кода программы.

СОДЕРЖАНИЕ РАБОТЫ

Во введении обосновывается актуальность темы исследования, его научная новизна, практическая значимость результатов, формулируются цели и задачи работы, представляются научные положения, выносимые на защиту.

В первой главе для определения уровня существующих методов и инструментов формальной верификации модулей ядра ОС Linux, их интеграции с динамическими инструментами анализа, и их применимости в требуемом контексте ядер ОС проводится обзор научно-технической, нормативной, методической литературы, затрагивающей научно-техническую проблему, исследуемую в рамках диссертационной работы.

В разделе 1 рассматриваются методы дедуктивной верификации. Такие методы основываются на том, что на языке спецификаций описывается состояние системы, задаются условия её корректности, описываются функциональные требования к ней. В зависимости от области применения, инструменты дедуктивной верификации уже могут содержать в себе определенные модели ошибки, например, деления на ноль и разыменования указателя. Чтобы показать

корректность исследуемого объекта, инструментами генерируются условия верификации в которых находят отражение как функциональные требования к системе, так и требования к её корректности (отсутствию ошибок). Посредством ручного или автоматического доказательства общезначимости условий верификации обосновывается отсутствие ошибок в системе и соответствие её спецификации. Используя дедуктивные методы верификации возможно доказать свойства программ/моделей на всех входных данных и всех путях выполнения. Дедуктивные методы верификации относятся к статическим методам анализа и позволяют рассуждать о корректности исследуемого объекта на уровне исходного кода.

В разделе 2 рассматриваются существующие инструменты дедуктивной верификации, потенциально позволяющие верифицировать модули ядра Linux, в том числе уже существующие модули без значимых ограничений по объему и по поддержке языка Си. Выявляются достоинства и недостатки существующих инструментов. Также рассматриваются вопросы индустриального использования инструментов с целью оценки их применимости к решению поставленной задачи.

В разделе 3 исследуется проекты по верификации системного программного обеспечения, такого как компиляторы, ядра операционных систем, микроядра, встроенные прошивки микросхем, гипервизоры. Рассматривается вопрос обоснования корректности реализации и соблюдения на ней модельных требований, если они были представлены в таком виде. В большом количестве проектов процесс верификации строится сверху вниз: создается модель, обосновывается её корректность, из модели разрабатывается реализация посредством все более и более детального уточнения последней, возможно, с применением кодогенерации. Такой подход обоснован и действует в ситуации, когда имеется полный контроль над кодовой базой проекта, когда разработка

ведется с нуля или когда имеется формализация интерфейса взаимодействия системы со своими подсистемами.

Большинство формально верифицированных систем разрабатываются «с нуля», изначально учитывая работы по формальной верификации. Однако для модуля ядра Linux это не возможно (или же сильно затруднено), так как последний зависим от самого ядра по многим параметрам, начиная от интерфейса взаимодействия и общих структур данных и заканчивая общим стилем кода, используемых конструкций языка и его нестандартных расширений. К сожалению, интерфейс LSM (Linux Security Modules) взаимодействия ядра с модулями безопасности не формализован, плохо документирован, подвержен частым изменениям, как в плане расширения, так и в плане изменения семантики функций. Функции взаимодействия модуля с ядром также подвержены всем перечисленным особенностям. Данные обстоятельства делают практически невозможным разработку кода модуля безопасности по схеме верификации «сверху вниз».

В разделе 4 подводятся основные итоги обзора и делаются выводы. Результаты обзора позволяют заключить, что основные усилия сейчас сосредоточены на формальном анализе отдельных алгоритмов операционных систем. Системный код ядер массовых операционных систем является слишком объемным и слишком сложным для полноценного применения дедуктивных методов анализа. Формальную связь моделей систем с реализацией удается показать в проектах, в которых имеется полный контроль над кодовой базой, и в которых работы по формальной верификации ведутся по принципу «сверху вниз».

Таким образом, можно заключить, что в настоящее время отсутствуют готовые методы работ по дедуктивной верификации системного кода, отсутствуют готовые методы и инструменты обоснования согласованности формальных

моделей и их реализаций в контексте ядер массовых операционных систем, разработка которых велась без учета нужд формального анализа.

Вторая глава посвящена доказательству корректности реализации модуля безопасности Linux, отсутствия в нем ошибок типа undefined behavior, с помощью метода дедуктивной верификации. На данном шаге разрабатываются формальные спецификации на языке ACSL на код LSM модуля безопасности ядра Linux. Спецификации представляют собой контракты на функции языка Си. Требования модели безопасности вручную переносятся на уровень спецификаций ACSL.

В разделе 1 описывается структура ядра Linux, взаимодействие его подсистем, рассматривается подсистема LSM, обработка ядром системных вызовов (запросов на доступ пользовательских приложений к ресурсам). В разделе описываются характерные для кода ядра особенности, способы работы с ними инструментов формальной верификации. Исходя из рассмотренной структуры ядра и модуля безопасности, интерфейсов их взаимодействия, очерчивается область кода, предназначенная для дедуктивного анализа.

Доказывается соответствие кода своей спецификации только для функций модуля. Для функций ядра, вызываемых модулем, также разрабатываются контрактные спецификации, которые остаются без доказательства. Соблюдение предусловий во всех точках вызова ядром функций модуля безопасности также остается без доказательства.

В разделе 2 описывается способ разработки спецификаций на языке ACSL на код ядра Linux. Описывается методология верификации ядерного кода с циклом работ по разработке спецификаций, их доказательству, обновлению и переноса спецификаций на новые версии кода, включению верификации в системы непрерывной интеграции для своевременного обнаружения расхождений в коде и спецификациях. В разделе также описываются созданные автором дополнительные инструменты для поддержки работы по формальной верификации

системного кода, основным предназначением которых является планирование работ по верификации, генерация отчетов о покрытии кода спецификациями и его метриках, перенос спецификаций между разными версиями кода, выделение отдельных частей исходного кода из общей кодовой базы для анализа. Приводятся алгоритмы их функционирования.

В разделе 3 описывается способ переноса функциональных требований с высокоуровневой модели безопасности в нотации Event-B на код модуля безопасности. Для этого на Event-B вручную создается дополнительная модель механизмов безопасности на уровне LSM, содержащая контракты операций интерфейса LSM и инварианты, описывающие гарантируемые свойства безопасности (помимо обычных инвариантов, описывающих корректность данных, используемых операциями LSM). Далее, контракты событий Event-B спецификации LSM вручную транслируются в контракты этих же операций на языке спецификаций ACSL. В завершение код на языке Си модуля LSM верифицируется, доказывается его соответствие контрактам, описанным на ACSL.

Результатом такой верификации является не только доказательство того, что функции LSM выполняют требования, заданные в спецификациях, но и доказательство выполнения общих требований к защите информации (инвариантов безопасности) модуля LSM, поскольку ACSL спецификация соответствует Event-B спецификации LSM, а из свойства «соответствия» следует сохранение в ACSL спецификации свойств Event-B спецификации.

В разделе 4 подводится итог дедуктивного анализа кода LSM модуля ядра Linux. Фиксируется область проанализированного кода, рассматриваются классы ошибок, потенциально оставшихся невыявленными в коде, описываются вопросы, оставшиеся за рамками анализа и вынесенные в разряд предположений. Также рассматриваются вопросы улучшения инструментов дедуктивной верификации, их моделей памяти и целых чисел, гибкости языка ACSL.

Третья глава посвящена интеграции дедуктивной верификации и динамического анализа спецификаций. Рассматривается вопрос проверки спецификаций, вынесенных в разряд предположений на стадии дедуктивного анализа: контрактов на функции ядра, вызываемых модулем, и предусловий для функций модуля, вызываемых ядром. Спецификации на языке ACSL транслируются в проверки времени выполнения (assertions). На работающей системе динамическим образом осуществляется проверка обоснованности данных предположений: на всех ли путях выполнения под тестовой нагрузкой соблюдаются данные спецификации.

В разделе 1 обосновывается необходимость данного вида анализа. Обоснование строится на изменчивой природе кода ядра Linux, невозможности применить дедуктивный вид анализа ко всему ядру, на том, что дедуктивная верификация — это статический вид анализа на уровне исходных кодов (возможны ошибки в бинарных кодах).

В заключении раздела делается вывод, что большинство из рассмотренных предположений и ограничений подхода можно ослабить, если в дополнении к дедуктивной верификации осуществлять проверки спецификаций в момент выполнения кода. Так, например, если оттранслировать предусловия к интерфейсу LSM в проверки времени выполнения, то это позволит в динамике проверить спецификации, при отсутствии возможности их доказать. Таким образом повышается общий уровень доверия к тем исходным предположениям, на основе которых осуществлялась дедуктивная верификация.

В разделе 2 рассматривается расширение E-ACSL для платформы Frama-C, которое на основе исходных кодов и спецификаций к ним на языке ACSL порождает новый исходный код, в котором часть спецификаций отображается в проверки времени выполнения. Расширение E-ACSL работает лишь с подмножеством языка ACSL. Так, например, невозможно отобразить леммы и

аксиомы в исполняемый код по очевидным причинам. Для моделирования работы со спецификационными типами чисел (неограниченными) используется библиотека libgmp, а также собственная библиотека для отслеживания состояний памяти. Данный инструмент рассматривается в контексте его применения к ядру.

В разделе делается вывод о том, что в неизменном виде инструмент малопригоден, так как существенным образом зависит от библиотек пользовательского уровня. Основная сложность состоит в том, что для работы инструмента, все обращения исследуемого кода к оперативной памяти должны быть специальным образом инструментированы E-ACSL для поддержки в актуальном состоянии структур данных библиотеки памяти, отвечающих за отслеживание валидных по доступу областей памяти и дающих возможность транслировать в проверки времени выполнения предикаты с прошлыми значениями переменных.

В разделе 3 описываются модификации инструмента E-ACSL, произведенные автором работы, для возможности его функционирования на уровне ядра Linux. Библиотека работы с длинными числами libgmp реализуется как самостоятельный модуль ядра Linux с экспортом необходимых функций. Библиотека отслеживания состояния памяти заменяется библиотекой собственной реализации. Ввиду специфики ядра Linux, становится невозможным отобразить некоторые предикаты на память в проверки времени выполнения. Новая библиотека отслеживания состояний памяти существенным образом опирается на встроенные в ядро механизмы проверки, такие как kasan и kmsan. Меняется способ инструментации плагином E-ACSL исследуемого кода.

В разделе 4 рассматриваются эксперименты по динамической проверке спецификаций. Описываются условия проведения экспериментов, потери в производительности, полученные результаты в виде найденных ошибок в предусловиях к интерфейсу LSM. Делается заключение о необходимости

использования метода, даются рекомендации по разработке спецификаций ACSL в виде, наиболее удобном для такого рода анализа.

В четвертой главе рассматривается вопрос интегральной проверки соответствия поведения ядра Linux требованиям модели политики безопасности управления доступом (например, МРОСЛ ДП-модели). Соответствие проверяется посредством сверки результатов реальных операций по запросу доступа при помощи системных вызовов ядра ОС с правилами предоставления доступа (или отказа в доступе) модели политики безопасности. Для этого во время выполнения приложений (программ пользовательского пространства), в динамике трассируются системные вызовы, фиксируются их параметры и результаты. Далее на основе полученных данных воспроизводится обработка запросов доступа в ядре на формальной функциональной спецификации на языке Event-B (ФСП). На функциональной спецификации проверяется допустимость результатов обработки системного вызова.

В разделе 1 рассматривается формальная функциональная спецификация Event-B, её предназначение, задача, строение и связь с спецификацией модели безопасности Event-B. Главным предназначением ФСП является связывание точек входа в ядро (системные вызовы) с МРОСЛ ДП-моделью. Между ФСП и МРОСЛ ДП-моделью существует отношение уточнения между их состояниями и операциями, причем на уровне ФСП за переход между состояниями отвечают системные вызовы, а на уровне модели политики безопасности – правила МРОСЛ ДП-модели.

В разделе 2 рассматривается вопрос целесообразности применения данного вида анализа. Отмечается, что в отечественной практике анализ имеет смысл применять в рамках выполнения требований класса доверия «Тестирование» ГОСТ Р ИСО/МЭК 15408-3, который требует проведения тестирования объекта оценки на соответствие функциональной спецификации (семейство доверия

ATE_COV «Покрытие»). Автоматический сбор трассы выполнения и проведение проверки её соответствия формальной функциональной спецификации на языке Event-B позволяет избавиться от необходимости вручную вычислять ожидаемый результат каждой операции, что, в свою очередь, даёт возможность значительно повысить количество и разнообразие проводимых тестов. Сбор информации о покрытии различных ситуаций в формальной функциональной спецификации также позволяет автоматизировать оценку покрытия интерфейсов функций безопасности, что является еще одним требованием семейства доверия ATE_COV «Покрытие». Рассматриваются потенциальные ошибки, которые должен выявлять данный вид анализа.

В разделе 3 рассматривается алгоритм работы анализа. Алгоритм разбивается на две последовательных стадии – сбора информации о поведении объекта оценки и её анализ.

На первой стадии осуществляется сбор трассы выполнения ядра Linux с модулем защиты в отдельную базу событий. В ОС в этот момент запускается либо специальный тестовый набор, либо система работает в стандартном режиме (работают некоторые пользовательские приложения).

На второй стадии осуществляется анализ собранной трассы с целью выявления в ней ошибок предоставления доступа. Для этого обработка системных вызовов из собранной трассы перепроверяется на ФСП, т. е. обработка системных вызовов воспроизводится на ФСП. Результаты обработки системных вызовов в ядре и в спецификации сверяются. В разделе описывается алгоритм воспроизведения собранной трассы на модели с учетом разных кодов возврата обработки системных вызовов, разности в уровне абстракций и возможных ошибок воспроизведения.

Результатом работы анализа является журнал «аномалий», в котором содержатся пункты расхождения поведения реальной системы с ФСП. Данные

журнала анализируются ручным образом. По результатам выявляются ошибки либо в исходном коде ядра и модуля безопасности, либо в формализации спецификации.

В разделе 4 рассматривается способ сбора трасс с ядра Linux. Для этого используются существующие инструменты с гибкими возможностями конфигурации. Описываются точки мониторинга (probe points) в ядре, способы сбора информации о системных вызовах с их аргументами и результатами обработки, дополнительной информации, которая позволила бы отображать состояние структур данных ядра на состояние структур данных спецификации, глобальное состояние структур данных ядра на момент начала и конца мониторинга.

В разделе 5 рассматривается механизм воспроизведения трасс на Event-B спецификации. Инструменты для работы с Event-B спецификациями не предоставляют возможности проверки допустимости определенной последовательности событий. Для этого требуется интерпретатор спецификации. Добиться аналогичного результата возможно, если осуществить трансляцию Event-B спецификации в нотацию одного из интерпретируемых или исполняемых языков.

Для анализа трассы системных вызовов на воспроизводимость на ФСП используется разработанный автором транслятор Event-B спецификации в язык Python. Статическая часть спецификации Event-B транслируется в систему типов (отображения, множества, перечисления), константы, а динамическая часть в функции. Переменные состояния спецификации в соответствии их инвариантами транслируются в глобальные переменные Python. При трансляции задаются две модели управления: когда на вход подается трасса, где уже полностью записаны переходы между событиями спецификации, и вторая модель – когда осуществляется подбор следующего события из текущего (для ФСП).

В разделе 6 рассматриваются вопросы сбора тестового покрытия на ядре Linux, оценки тестового покрытия на модели Event-B. За счет своего устройства модель позволяет гораздо больше переходов между состояниями, чем это допускается реализацией. В связи с этим, на модели оценивается покрытие по предусловиям событий, а не по возможным переходам между событиями. В разделе рассматриваются экспериментальные данные запусков предложенных инструментов и метода, производится оценка полученных результатов.

В заключении сформулированы основные результаты диссертационной работы. Результаты и положения, выносимые на защиту, совпадают.

СПИСОК РАБОТ

Статьи в журналах перечня ВАК

1. Ефремов Д.В., Мандрыкин М.У. Формальная верификация библиотечных функций ядра Linux. Труды Института системного программирования РАН, том 29, вып. 6, 2017
2. Ефремов Д.В. Технологическая инфраструктура для дедуктивной верификации модульных программных систем. Труды Института системного программирования РАН, том 30, вып. 5, 2018

Другие издания

3. Efremov D., Komarov N. Tools Support for Linux Kernel Deductive Verification Workflow. Proceedings of the Spring/Summer Young Researchers' Colloquium on Software Engineering, 2014 DOI: 10.15514/SYRCOSE-2014-8-6

Свидетельство о государственной регистрации программ для ЭВМ

4. Ефремов Д.В., Хорошилов А.В. Свидетельство о государственной регистрации программы для ЭВМ №2015617942 «Программа

преобразования структуры исходного кода на языке C для его последующей верификации»

Приняты к публикации

5. Девянин П.Н., Ефремов Д.В., Кулямин В.В., Петренко А.К., Хорошилов А.В., Щепетков И.В. Монография. Моделирование и верификация политик безопасности управления доступом в операционных системах. 2018, 178 с. Будет опубликована отдельным изданием в конце 2018 — начале 2019 года. Монография доступна по адресу: http://www.ispras.ru/publications/2018/security_policy_modeling_and_verification/
6. Volkov G., Mandrykin M., Efremov D. Lemma Functions for Frama-C: C programs as Proofs. Принята на конференцию ISP RAS Open 2018. Будет опубликована в сборнике «Proceedings of Ivannikov ISPRAS Open Conference» в конце 2018 года, входить в индексы Scopus и Web of Science.
7. Efremov D., Mandrykin M., Khoroshilov A. Deductive Verification of Unmodified Kernel Library Functions. Принята на конференцию ISoLA18. Будет опубликована LNCS сборнике трудов конференции IsoLA 2018, входить в индекс Scopus. Препринт доступен по адресу: https://arxiv.org/abs/1809.00626