DOI: 10.15514/ISPRAS-2025-37(6)-17

Тестирование подсистемы безопасности ОС Astra Linux на основе формализованного описания модели управления доступом

П.Н. Девянин, ORCID: 0000-0003-2561-794X <pdevyanin@astralinux.ru>
С.С. Жиляков, ORCID: 0009-0006-1831-697X <szhiliakov@astralinux.ru>
А.И. Смирнов, ORCID: 0009-0000-3483-110X <alesmirnov@astralinux.ru>

ООО «РусБИТех-Астра»,
117105, г. Москва, Варшавское шоссе, д. 26.

Аннотация. В операционной системе (ОС) Astra Linux кроме традиционного для большинства ОС дискреционного управления доступом ее подсистемой безопасности PARSEC реализуются механизмы мандатного контроля целостности (МКЦ) и мандатного управления доступом (МРД). С учетом многообразия имеющихся в данной ОС сущностей (объектов доступа, файлов, каталогов, сокетов и др.) и субъектов (процессов) эти механизмы имеют сложную логику функционирования, затрудняющую их тестирование с использованием вручную подготовленных тестов. Влияет на проблему необходимость выполнения процессов разработки безопасного программного обеспечения (ПО) для соответствия ОС Astra Linux требованиям высших классов защиты и уровней доверия. Вместе с тем в основе механизмов МКЦ и МРД этой ОС используется мандатная сущностно-ролевая ДП-модель управления доступом и информационными потоками в ОС семейства Linux (МРОСЛ ДП-модель), описанная в классической математической нотации и в формализованной нотации на языке формального метода Event-B. Авторами развивается рекомендованный ГОСТ Р 59453.4-2025 подход к тестированию механизмов управления доступом на основе сбора трасс системных вызовов ОС и их перевода на язык формальной модели с целью проверки соответствия ей логики функционирования механизма управления доступом ОС. Результатам этой работы посвящена настоящая статья, в которой, во-первых, изложены итоги разработки и верификации используемого для тестирования нижнеуровневого представления МРОСЛ ДП-модели (PARSEC-модели), выполненного на языке формального метода Event-B и представляющего функциональную спецификацию связанных с управлением доступом системных вызовов ОС. Во-вторых, описывается система тестирования, включающая модуль ядра ОС для сбора трасс системных вызовов, ПО для их преобразования в модельные трассы, аниматор модельных трасс, выполненный с применением инструментального средства ProB, и ПО для формирования результатов тестирования в формате инструментального средства Allure. В-третьих, в статье рассматривается подход к использованию для распараллеливания тестирования технологии eBPF.

Ключевые слова: операционная система; операционная система Astra Linux; МРОСЛ ДП-модель; метод Event-B; тестирование.

Для цитирования: Девянин П.Н., Жиляков С.С., Смирнов А.И. Тестирование подсистемы безопасности ОС Astra Linux на основе формализованного описания модели управления доступом. Труды ИСП РАН, том 37, вып. 6, часть 2, 2025, стр. 21–36. DOI: 10.15514/ISPRAS–2025–37(6)–17.

Testing the Astra Linux OS Security Subsystem Based on a Formalized Description of the Access Control Model

P.N. Devyanin, ORCID: 0000-0003-2561-794X <pdevyanin@astralinux.ru>
S.S. Zhiliakov, ORCID: 0009-0006-1831-697X <szhiliakov@astralinux.ru>
A.I. Smirnov, ORCID: 0009-0000-3483-110X <alesmirnov@astralinux.ru>

RusBITech-Astra,
26, Varshavskoe, Moscow, 117105, Russia.

Abstract. In Astra Linux operating system (OS), in addition to the traditional Discretionary Access Control, its PARSEC security subsystem implements Mandatory Access Control policies such as Mandatory Integrity Control (MIC) and Multilevel Security (MLS). Given the variety of entities (objects, files, directories, sockets, etc.) and subjects (processes) available in a given OS, these policies often have a complicated logic of functioning, which makes it difficult to test them using manual testing. This problem is especially aggravated in the context of the necessity to fulfill the Security Development Lifecycle in compliance with the requirements of the highest protection classes and trust levels established by the FSTEC of Russia regulatory documents. Besides, the MIC and MLS policies of this OS are based on the mandatory entity-role model of access and information flows security control in OS of Linux family (MROSL DP-model), described in the classical mathematical notation and in the formalized notation using the formal Event-B method. Therefore, the authors of this paper have developed and finalized the approach recommended by GOST R 59453.4-2025, taking into account the specifics of current releases of Astra Linux OS, which consists of tracing system calls and translating them into the language of the formal model in order to verify the compliance of the functioning access control policies with the model. The results of this work are described in this paper, which, firstly, outlines the results of the development and verification of the so-called lower-level representation of the MROSL DP-model (PARSEC-model) used for testing, performed in the Event-B and in essence represents a functional specification of access control-related system calls of the OS. Secondly, it describes a testing system that includes the Linux Kernel Module for tracing system calls, software for their translation into model traces, an animator of model traces using the ProB toolkit, and software for generating test results in the format of the Allure toolkit. Thirdly, the paper considers approach to using eBPF technology for parallelizing testing.

Keywords: operating system; Astra Linux; MROSL DP-model; Event-B; testing.

For citation: Devyanin P.N., Zhiliakov S.S., Smirnov A.I. Testing the Astra Linux OS security subsystem based on a formalized description of the access control model. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 6, part 2, 2025. pp. 21-36 (in Russian). DOI: 10.15514/ISPRAS-2025-37(6)-17.

1. Введение

Тестирование программного обеспечения (ПО) на выявление отличий между его реально существующими и требуемыми свойствами (функциональное тестирование) согласно ГОСТ Р 56939-2024 [1] является неотъемлемой частью процесса разработки безопасного ПО. Реализация этого процесса особенно важна, когда ПО выполняет функции средства защиты информации (СЗИ), например, являясь операционной системой (ОС). Ключевую роль в таких СЗИ часто выполняет механизм управления доступом. Для большинства ОС это дискреционное управление доступом [2-3]. Такой механизм достаточно прост для реализации и понимания, его тестирование, разработка соответствующих тестов, как правило, не представляют затруднений. Кроме того, в ОС семейств Linux или Windows он не претерпевал существенных изменений в течение десятилетий.

Вместе с тем, задача тестирования механизма управления доступом ОС существенно усложняется, когда в него включают реализацию других политик управления доступом таких, как мандатный контроль целостности (МКЦ) или мандатное управление доступом (МРД). Примером ОС с МКЦ и МРД является сертифицированная по установленным нормативными документами ФСТЭК России [4] требованиям высших классов защиты и уровней доверия ОС Astra Linux (операционная система специального назначения Astra Linux

Special Edition) [5-6]. Как во многих ОС семейства Linux в этой ОС управление доступом осуществляется между субъектами (процессами), функционирующими от имени десятков или даже сотен локальных, системных или зарегистрированных в домене учетных записей пользователей, к широкому спектру сущностей (объектов доступа, файлов, каталогов, сокетов и др.). При этом субъектам, учетным записям пользователей назначаются уровни целостности и уровни доступа, используемые МКЦ и МРД привилегии, а сущностям – уровни целостности и уровни доступа, специальные флаги. Все перечисленное затрудняет понимание логики функционирования МКЦ и МРД, их тестирование с использованием вручную подготовленных тестов.

Научной основой реализации МКЦ и МРД в ОС Astra Linux является соответствующая критериям ГОСТ Р 59453.1-2021 [2] мандатная сущностно-ролевая ДП-модель управления доступом и информационными потоками в ОС семейства Linux (МРОСЛ ДП-модель) [3]. Данная модель описана в двух нотациях: математической (аналогично классическим моделям [7]) и, что наиболее важно в контексте настоящей статьи, в формализованной на языке формального метода Event-B [8-10]. Машиночитаемый язык представления модели в этой нотации обеспечивает возможность выполнения рекомендаций ГОСТ Р 59453.4-2025 [11] и применения апробированного подхода [12-13] по автоматизированному тестированию механизма управления доступом ОС, заключающегося в сборе трасс системных вызовов ОС, их переводе на язык формальной модели с целью проверки соответствия ей логики функционирования этого механизма. Такой подход, во-первых, отвечает реализуемой «Группой Астра» (ООО «РусБИТех-Астра») единой методологии разработки безопасного системного ПО [14], во-вторых, сокращает ресурсы на тестирование МКЦ и МРД по мере расширения их функций, изменения их программного кода, а, в-третьих, позволяет учитывать модификации, вносимые в МРОСЛ ДП-модель, которая согласно рекомендациям ГОСТ Р 59453.3-2025 [15-16] также регулярно дорабатывается и развивается [17].

Следует отметить, что непосредственно соответствующее описанию МРОСЛ ДП-модели в математической нотации ее описание в формализованной нотации (названное верхнеуровневым) позволяет согласно рекомендациям ГОСТ Р 59453.2-2021 [18] дедуктивно верифицировать модель с применением инструментального средства Rodin [10, 19]. Однако использование такого верхнеуровневого описания модели напрямую для задачи автоматизированного тестирования не является возможным в силу его абстрактности, а, именно, в нем не обеспечивается точное соответствие программному коду системных вызовов ядра ОС и подсистемы безопасности PARSEC, реализующей МКЦ и МРД в ОС Astra Linux. В связи с этим на языке формального метода Event-B было разработано нижнеуровневое представление модели (названное PARSEC-моделью), которое позволило применить подход, изложенный в [12-13], и использовать разработанные его авторами инструментальные средства. В том числе, для монитора системных вызовов был применен специальный модуль ядра ОС, использующий механизмы инструментации исполняемого кода Kprobes, Kretprobes и отправляющий полученные им данные мониторинга в пользовательское пространство ОС для дальнейшего их преобразования в модельные трассы и их анимации с применением инструментального средства ProB [20].

Вместе с тем, данный подход имеет некоторые недостатки. Во-первых, разработка модулей ядра ОС всегда сопряжена с риском – ошибка в его программном коде может значительно повлиять на поведение ОС, в том числе привести к ее краху. Во-вторых, такие модули требуют компиляции под каждую версию ядра ОС, что значительно усложняет процесс автоматизированного тестирования ОС с несколькими поддерживаемыми версиями ядра (к таким ОС относится и ОС Astra Linux). В-третьих, монитор системных вызовов не обладает функцией одновременного сбора их трасс для нескольких процессов, что замедляет тестирование и затрудняет развертывание тестовой инфраструктуры в среде непрерывной разработки и интеграции ОС (CI). Наконец, текущая реализация монитора системных вызовов не отслеживает обращения прикладного ПО к подсистеме безопасности PARSEC

при изменении уровней конфиденциальности и целостности (меток безопасности) компонент ОС.

В связи с изложенным авторами статьи был рассмотрен подход к применению технологии eBPF, представляющей собой альтернативу загружаемым модулям ядра. Данная технология ранее применялась для трассировки сетевых пакетов [21], и для анализа производительности ОС [22]. В настоящее время eBPF также применяется в таких системах мониторинга средств виртуализации на уровне ядра ОС, как Tetragon, Falco и Tracee [23-25].

Таким образом, статья организована следующим образом. В следующем разделе рассматриваются результаты разработки и верификации нижнеуровневого представления МРОСЛ ДП-модели (PARSEC-модели). В разд. 3 описывается система тестирования подсистемы безопасности PARSEC ОС Astra Linux. В разд. 4 анализируется подход к применению технологии eBPF для расширения функциональных возможностей этой системы тестирования, в том числе за счет его распараллеливания. Заключение завершает статью, в нем подводятся итоги выполненных исследований и разработок, а также рассматриваются дальнейшие направления их развития.

2. Нижнеуровневое представление МРОСЛ ДП-модели (PARSEC-модель)

МРОСЛ ДП-модель является основой для реализации механизма управления доступом в ОС Astra Linux. В математической и формализованной нотациях модель имеет иерархическое представление и включает согласованное описание ролевого управления доступом (РУД, представляющего штатное для ОС семейства Linux дискреционное управление доступом), МКЦ и МРД [3]. В обеих нотациях согласно ГОСТ Р 59453.1-2021 [2] модель включает описание состояний и правил перехода между состояниями абстрактного автомата, соответствующего политикам управления доступом, реализуемым в ОС Astra Linux.

Как уже было отмечено, непосредственно соответствующее описанию МРОСЛ ДП-модели в математической нотации ее верхнеуровневое представление в формализованной нотации на языке формального метода Event-B достаточно абстрактно. В нем не учитываются некоторые особенности программной реализации управления доступом в ядре ОС и подсистеме безопасности PARSEC, являющейся модулем безопасности ядра – Linux Security Module (LSM) [26]. Например, они оперируют не абстрактными сущностями (объектами доступа) и субъектами, а файловыми дескрипторами, индексными дескрипторами (inode) и другими системными компонентами ОС. Также реализованные в ядре ОС обработчики системных вызовов (выполняющих операции над файлами, каталогами, процессами, учетными записями пользователей) по своему функционалу отличаются от их описания в соответствующих правилах преобразования состояний абстрактного автомата в верхнеуровневом представлении модели.

Чтобы приблизить описание модели в формализованной нотации к программной реализации управления доступом в ядре ОС и подсистеме безопасности PARSEC, а также создать условия для применения этого описания в системе автоматизированного тестирования, на Event-B было разработано нижнеуровневое представление МРОСЛ ДП-модели (PARSEC-модель), которое по сути является функциональной спецификацией обработчиков системных вызовов в ядре ОС и подсистеме безопасности PARSEC. Выбор Event-B позволил, во-первых, сохранить преемственность технологий описания и верификации модели с использованными для ее верхнеуровневого представления, а, во-вторых, применить подход, изложенный в [12-13], который также базируется на Event-B.

PARSEC-модель аналогично верхнеуровневому представлению с применением техники пошагового уточнения (refinement) Event-B [8] имеет иерархическое представление, включающее три уровня, соответствующие ключевым для режимов защищенности ОС Astra Linux политикам управления доступом [6]:

- Первый уровень – «Базовый» («Орел»), дискреционное управление доступом;
- Второй уровень – «Усиленный» («Воронеж»), мандатный контроль целостности;
- Третий уровень – «Максимальный» («Смоленск»), мандатное управление доступом.

Каждый уровень представлен соответствующими контекстами (contexts), описывающими статические компоненты моделируемой ОС, и машинами (machines), включающими описания соответствующих системным вызовам событий (events) и инвариантов безопасности (invariants).

PARSEC-модель содержит описание следующих событий (системных вызовов): open, openat, mkdir, chmod, fchmod, fchmodat, chown, fchown, fchownat, chdir, fchdir, getxattr, setxattr, link, execve, umask, exit, close, unlink, rmdir, fork. В ней также присутствует описание события chlbl, которое соответствует функции sys_chlbl, входящей в интерфейс подсистемы безопасности PARSEC и позволяющей изменять уровни целостности и конфиденциальности сущности (файла или каталога).

Стоит отметить, что на текущий момент РУД, описанное в верхнеуровневом представлении МРОСЛ ДП-модели, явно не реализовано в ОС, и поэтому его формализация отсутствует в нижнеуровневой PARSEC-модели. Кроме того, в нем отсутствуют некоторые несущественные, но сложные для моделирования элементы дискреционного управления доступом (например, расширенные списки контроля доступа Access Control Lists – ACL), поскольку они не интересны при тестировании подсистемы безопасности PARSEC.

Т.к. наибольшее значение при тестировании управления доступом в ОС Astra Linux представляют ее собственные механизмы защиты, то далее будут рассмотрены результаты разработки второго и третьего уровней PARSEC-модели, соответствующих МКЦ и МРД, тем более, что большая часть первого уровня была описана в рамках исследований [12-13].

Первым шагом при моделировании МКЦ в PARSEC-модели стало описание уровней целостности. В ОС программная реализация каждого уровня целостности состоит из двух частей: неиерархического уровня (категорий) целостности (маски 32 бит) и линейного уровня целостности (знаковых целых чисел от -128 до 127). Например, уровень целостности 0x00000002:-128 соответствует неиерархическому уровню «Виртуализация» 0x00000002 (в соответствующей маске бит ненулевой только второй бит) и минимальному линейному уровню -128. Поэтому в модели для субъектов (процессов), сущностей (файлов или каталогов) и учетных записей пользователей заданы по паре соответствующих функций, например, функции EntityInt и EntityLinearInt для сущностей (Листинг 1).

Поскольку формальный метод Event-B не поддерживает битовые операции над целыми числами, то в PARSEC-модели множество неиерархических уровней целостности Integrity задано как множество всех подмножеств несущего множества (carrier set) Int, элементы которого представляют собой позиции битов в двоичном представлении целого числа. Такое описание позволяет выразить программную реализацию битовой маски в терминах теории множеств, где каждому биту будет соответствовать нумерованная константа из множества Int. Например, значению неиерархической уровня целостности 0x0000003f (двоичному значению 0b111111, а десятичному значению 63 – максимальному по умолчанию уровню целостности max_ilev в ОС Astra Linux) будет соответствовать множество констант {Int1, Int2, Int3, Int4, Int5, Int6} ∈ Integrity. Сравнение любых двух неиерархических уровней целостности в таком случае естественно выражается с помощью операций из теории множеств. Так как в ОС значение линейного уровня целостности ограничено одним байтом и является знаковым числом, метод Event-B позволяет определить тип такой константы как интервал -128 .. 127, и сравнение линейных уровней целостности в данном случае будет соответствовать сравнению целых чисел.

Вторым шагом при моделировании МКЦ в PARSEC-модели стало описание инвариантов состояний. Для примера рассмотрим инвариант, который устанавливает требования к уровням

целостности субъекта и сущности при доступе на запись (Листинг 2). В этом фрагменте используются следующие функции:

- SubjectFDs – задает для процесса его файловые дескрипторы;
- FDFile – отображает файловый дескриптор на соответствующий файл;
- FDFlags – задает для файлового дескриптора флаги, с которыми был открыт соответствующий файл.

```
sets
  Int
  ENTITIES
constants
  Integrity
  LinearIntegrity
axioms
  @Integrity_type:  Integrity = ℙ(Int)
  @LinearIntegrity_type:  LinearIntegrity = -128..127
variables
  Entities
  EntityInt
  EntityLinearInt
invariants
  @Entities_type:              Entities ⊆ ENTITIES
  @EntityInt_type:             EntityInt ∈ Entities → Integrity
  @EntityLinearInt_type: EntityLinearInt ∈ Entities → LinearIntegrity
```

Листинг 1. Функции EntityInt и EntityLinearInt.
Listing 1. EntityInt и EntityLinearInt functions.

```
@SubjectAccessObject_Write:
  ∀ s,fd   s ↦ fd ∈ SubjectFDs ∧ FDFile(fd) ∈ Entities \ Containers ∧
  (O_WRONLY ∈ FDFlags(fd) ∨ O_RDWR ∈ FDFlags(fd)) ⇒
  EntityInt(FDFile(fd)) ⊆ SubjectInt(s) ∧ EntityLinearInt(FDFile(fd))
                        ≤ SubjectLinearInt(s)
```

Листинг 2. Инвариант установки требований к уровням целостности.
Listing 2. Invariant for setting integrity level requirements.

В ОС факт доступа на запись процесса к некоторому файлу можно определить по наличию дескриптора этого файла в множестве файловых дескрипторов этого процесса, а также наличия флагов O_WRONLY или O_RDWR, с которыми был открыт этот файл. Кроме того, т.к. ядро ОС ограничивает получение файлового дескриптора к каталогу с доступом на запись (в таком случае выдается ошибка EISDIR), то рассматриваемый инвариант включает только проверку доступов на запись процесса к файлам. Таким образом, инвариант позволяет выразить проверяемое в ОС условие согласованности уровней целостности процесса и файла при наличии к нему доступа на запись – уровень целостности процесса должен быть не ниже уровня целостности файла.

После контекста и инвариантов были описаны события (системные вызовы). Для удобства верификации PARSEC-модели и ее использования в системе тестирования для некоторых системных вызовов ОС задавались по два события. Например, для системного вызова sys_chlbl были описаны события: chlbl_container – для изменения уровня целостности сущности-контейнера (каталога), chlbl_object – для изменения уровня целостности сущности-объекта (файла). Такое разделение обусловлено наличием у каталогов и файлов специфичных параметров МКЦ. В их числе флаг SILEV исполняемого файла, который задает порядок активизации из него процесса с учетом уровней целостности этого файла и учетной записи

пользователя, от имени которой активизируется процесс (это позволяет процессам в сессии учетной записи пользователя с низким уровнем целостности запускать некоторые процессы, которым для выполнения их функций необходим высокий текущий уровень целостности; так с помощью флага SILEV помечается исполняемый файл консольной утилиты passwd, запускаемой для изменения пароля низкоцелостной учетной записи пользователя, образ которого хранится в обладающем высоким уровнем целостности файле «/etc/shadow»). Поэтому событие chlbl_container содержит специальную переменную passed_silev, используемую для предотвращения установки флага SILEV на каталоги, и соответствующие два охранные условия (guard):

```
@grd10: passed_silev ∈ BOOL
@grd11: passed_silev = FALSE
```

Таким образом, если в системе тестирования на этапе трансляции системной трассы в события PARSEC-модели значение переменной passed_silev будет установлено TRUE (что в ОС соответствует тому, что на каталог делается попытка установить флаг SILEV), то во время анимации трассы условие @grd11 не будет выполнено, что будет сигнализировать о расхождении программной реализации подсистемы безопасности PARSEC и PARSEC-модели. Аналогично при моделировании МКЦ описываются другие флаги файлов (SSI) и каталогов (IINH, IRELAX, SSI) [27].

Для выполнения инвариантов безопасности в PARSEC-модели в событиях описаны соответствующие охранные условия. Так истинность рассмотренного инварианта @SubjectAccessObject_Write в событии открытия сущности (файла или каталога) open_exists обеспечивается охранным условием, показанным в Листинге 3.

```
@grd24:
  (O_WRONLY ∈ flags ∨ O_RDWR ∈ flags) ⇒
  (file ∈ Containers ∧ IRELAX(file) = TRUE) ∨
  ((file ∈ Entities \ Containers ∨ (file ∈ Containers ∧ IRELAX(file) = FALSE)) ∧
   EntityInt(file) ⊆ SubjectInt(proc) ∧
   EntityLinearInt(file) ≤ SubjectLinearInt(proc)
  )
```

Листинг 3. Охранное условие открытия файла.
Listing 3. File open guard condition.

Третий уровень PARSEC-модели соответствует МРД. В его рамках уровни конфиденциальности (доступа) сущностей, субъектов и учетных записей пользователей были заданы аналогично уровням целостности, т. к. они также состоят из двух частей: линейных уровней конфиденциальности и неиерархических уровней (категорий) конфиденциальности. Также аналогично уровню МКЦ заданы специальные флаги сущностей (CCNR, EHOLE, WHOLE) [6].

При описании инвариантов состояний акцентировалось внимание на те из них, которые отражают МРД. Например, в любой момент функционирования ОС никакой процесс не должен иметь доступ на чтение к файлу или каталогу с уровнем конфиденциальности большим, чем уровень доступа процесса (за исключением случаев наличия у процесса либо специальных привилегий PARSEC_CAP_IGNMACLVL и PARSEC_CAP_IGNMACCAT, объединенных в PARSEC-модели в множество PARSEC_CAP_IGNMAC, либо полномочий суперпользователя root и максимального уровня целостности HighI, позволяющих ему нарушать МРД). Иллюстрация дана на Листинге 4.

Также для примера выполнение этого инварианта в событии открытия сущности (файла или каталога) open_exists было обеспечено охранным условием, показанным на Листинге 5.

В итоге разработки PARSEC-модели, выполнение всех инвариантов состояний было дедуктивно доказано при верификации с применением инструментального средства Rodin [19].

```
@SubjectAccessesConf_Read:
  ∀s,fd · s ↦ fd ∈ SubjectFDs ∧ FDFile(fd) ∈ Entities ∧ s ∈ Subjects ∧
   (O_RDONLY ∈    FDFlags(fd) ∨ O_RDWR ∈ FDFlags(fd)) ∧
  ¬(PARSEC_CAP_IGNMAC ⊆ SubjectCaps(s)) ∧
  ¬(SubjectUser(s) = ROOT_USER ∧ SubjectInt(s) = HighI ∧ SubjectLinearInt(s) ≥ 0)
  ⇒ (EntityLev(FDFile(fd)) ≤ SubjectLev(s) ∧ EntityCat(FDFile(fd)) ⊆ SubjectCat(s))
```

Листинг 4. Проверка прав доступа и наличия привилегий.
Listing 4. Checking access rights and privileges.

```
@grd26: (O_RDONLY ∈ flags ∨ O_RDWR ∈ flags) ⇒
      (EntityLev(file) ≤ SubjectLev(proc) ∧ EntityCat(file) ⊆ SubjectCat(proc)) ∨
            (PARSEC_CAP_IGNMAC ⊆ SubjectCaps(proc)) ∨
      (SubjectUser(proc) = ROOT_USER ∧ SubjectInt(proc)=HighI ∧
       SubjectLinearInt(proc) =0)
```

Листинг 5. Охранное условие открытия сущности (файла или каталога).
Listing 5. Entity (file or directory) open guard condition.

В заключении раздела отметим, что в [28] было сказано о потенциальной возможности разработки PARSEC-модели путем уточнения напрямую верхнеуровневого представления МРОСЛ ДП-модели в формализованной нотации. Однако такой подход кратно увеличивает сложность модели. Поэтому при разработке PARSEC-модели авторами настоящей статьи был использован экспертный подход, включающий в себя построение модели на основе анализа верхнеуровневого представления МРОСЛ ДП-модели в обеих нотациях, а также анализа исходного кода ядра ОС Astra Linux и подсистемы безопасности PARSEC.

3. Система тестирования подсистемы безопасности PARSEC ОС Astra Linux

Система тестирования (рис. 1) реализует автоматическую проверку соответствия поведения подсистемы безопасности PARSEC и нижнеуровневого представления МРОСЛ ДП-модели – PARSEC-модели. При этом за ее основу была взята система, описанная в [12-13].

Применение системы тестирования включает выполнение следующих шагов:

1. Подготовка начального окружения системы тестирования, включающая настройку ОС Astra Linux, создание соответствующих файлов, каталогов, учетных записей пользователей, задание их параметров МКЦ и МРД.
2. Сбор и сохранение инструментом gather_info информации о начальном окружении системы, которое было сформировано на шаге 1.
3. Загрузка модуля monitor в пространство ядра, и запуск вспомогательного процесса umonitor для взаимодействия с ним.
4. Запуск теста, включающего воздействия на подсистему безопасности PARSEC, и запись результатов мониторинга тестируемого процесса в базу данных (получение системной трассы тестируемого процесса).
5. Создание модельной трассы с помощью инструмента mediator из системной трассы, полученной на шаге 4, и начального окружения, полученного на шаге 2.
6. Анимация трассы, полученной на шаге 5, с использованием PARSEC-модели и инструментального средства ProB [20], преобразование ее результатов в формат платформы Allure [29] с помощью скрипта allure_report.py.
7. Получение вывода о соответствии поведения тестируемого процесса в ОС Astra Linux и результатами анимации на шаге 6 его модельной трассы.

В контексте настоящего исследования по сравнению с [12-13] были доработаны ряд шагов. Так при сборе данных о начальном окружении системы на шаге 2 важной составляющей являются значения специальных флагов МКЦ и МРД, уровней целостности и

конфиденциальности (доступа) файлов и каталогов, учетных записей пользователей (на этом этапе не собираются параметры процессов). Для этого данные о значениях флагов, уровней целостности и конфиденциальности файла или каталога извлекаются из их расширенных атрибутов; а для учетной записи пользователя – из соответствующих ее идентификатору uid файлов в каталогах /etc/parsec/micdb (уровень целостности) и /etc/parsec/macdb (уровень доступа).

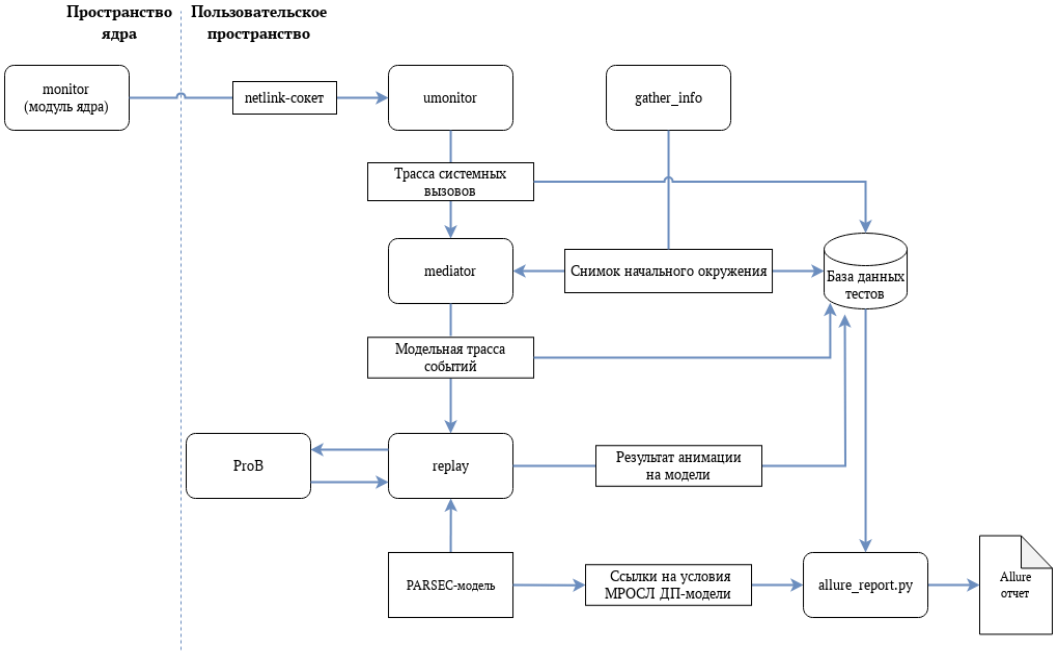

Рис. 1. Схема системы тестирования.
Fig. 1. Schematic diagram of the testing system.

Мониторинг на шаге 4 осуществляется с помощью модуля ядра ОС с применением механизмов инструментации исполняемого кода Kprobes и Kretprobes. Примечательным для рассматриваемой системы является то, что помимо стандартных системных вызовов был реализован перехват функции подсистемы безопасности PARSEC – sys_chlbl. Как было отмечено в предыдущем разделе, эта функция является интерфейсом для изменения уровней целостности и конфиденциальности файлов и каталогов. Несмотря на функциональное сходство системных вызовов sys_setxattr (предназначен для установки расширенных атрибутов) и sys_chlbl, назначение последнему отдельного события chlbl упрощает верификацию и делает структуры PARSEC-модели более понятными.

Инструмент трансляции трасс на шаге 5 преобразует системные трассы в модельные с учётом особенностей ОС Astra Linux и PARSEC-модели:

- битовые маски, соответствующие неиерархическим уровням (категориям) целостности и конфиденциальности, отображаются в их заданные с помощью множеств эквиваленты;
- специальные флаги интерпретируются как имеющие булевы значения;
- события выбираются в зависимости от типа обрабатываемой сущности – файл или каталог.

На этапе трансляции особое внимание уделяется обработке компонент из начального окружения системы. Создание этих компонент на шаге 1 не отображается в виде

последовательности системных вызовов в системной трассе, и информацию о том как они были созданы и сконфигурированы необходимо выводить из их результирующего по итогу выполнения шага состояния. Например, пусть при создании файла в каталоге в начальном окружении присутствует каталог /foo и файл /foo/bar с максимальными уровнями целостности HighI (в ОС Astra Linux ему соответствует системное значение max_ilev, по умолчанию равное 0x0000003f:0, т.е. десятичному значению 63). Также пусть на каталог /foo/bar не установлены никакие специальные флаги. При ошибочной трансляции можно предположить, что файл /foo/bar был создан с уровнем целостности HighI, наследуя его от родительского каталога. Однако такое наследование возможно только в случае, если на каталог был установлен специальный флаг IINH. Таким образом, событие создания файла /foo/bar необходимо рассматривать как последовательность двух событий open_create (системный вызов open, создающий файл) и chlbl_object (устанавливает на файл уровень целостности HighI).

Как отмечено в работах [10, 30] ProB как инструментальное средство, используемое для анимации трасс на шаге 6, имеет ряд ограничений, одно из которых – невозможность работы с множествами большой мощности (из-за проблемы «комбинаторного взрыва»). Такое ограничение не позволяет использовать в PARSEC-модели несущие множества, точно соответствующие неиерархическим уровням (категориям) целостности и конфиденциальности, поскольку в программной реализации ОС они представлены в виде масок размером 32 и 64 бита, соответственно. Поэтому на шаге 5 для дальнейшей корректной анимации модельных трасс используются урезанные несущие множества для неиерархических уровней (категорий) целостности и конфиденциальности – 8 бит.

Сами тесты для шага 4 были сформированы в соответствии с верхнеуровневой МРОСЛ ДП-моделью в математической нотации, т. к. именно в ней формализовано описание МКЦ и МРД, реализуемое в ОС Astra Linux. В основе системы тестирования используется фреймворк pytest [31], который позволяет автоматизировать ее конфигурирование, запуск, а также «очистку» после очередного теста. Каждый тест представляет собой исполняемый скрипт на языке Python и задает описание последовательности действий в ОС (создание, удаление, чтение, запись, выполнение, изменение уровней целостности и конфиденциальности файлов и каталогов). В настоящее время полнота тестирования в основном обеспечивается экспертным анализом МРОСЛ ДП-модели, но в будущем планируется создавать тесты с применением формального описания PARSEC-модели и оценки покрытия ее инвариантов и охранных условий событий. Такой подход соответствует рекомендациям по оценке покрытия формальных моделей ГОСТ Р 59453.4-2025 [11] и включает в себя создание тестов с использованием структурных элементов спецификации событий модели. Например, рассмотренное в предыдущем разделе охранное условие @grd26 события open_exists необходимо разбить на дизъюнкты и обеспечить минимальное тестовое покрытие, при котором каждый дизъюнкт хотя бы раз принимает значение истина (TRUE).

На шаге 6 данные о проведенном тестировании преобразуются в формат платформы Allure [29], для чего результат каждого теста транслируется в специальный JSON-объект.

При анализе результатов тестирования на шаге 7 наибольший интерес представляют тесты, завершившиеся неудачей. Для его осуществления отчет о каждом таком тесте должен в явном виде указывать на событие и охранные условия (guards), которые в нем были нарушены. Для упрощения работы с PARSEC-моделью большинство охранных условий имеют ссылку на соответствующие условия в МРОСЛ ДП-модели в математической нотации. Таким образом, для тестов, завершившихся неудачей, в отчете Allure указываются следующие данные: системная и модельная трассы; имя события в PARSEC-модели; идентификатор нарушенного охранного условия; предикат, соответствующий охранному условию; комментарий, указывающий на то, какое условие МРОСЛ ДП-модели было нарушено. Это позволяет упростить анализ ошибок, т.к. он не требует глубокого «погружения» в детали описания всей PARSEC-модели.

4. Применение технологии eBPF для распараллеливания тестирования

В основе системы тестирования, рассмотренной в предыдущем разделе, лежит модуль monitor ядра ОС, осуществляющий мониторинг тестируемых процессов с применением механизмов Kprobes и Kretprobes. Он перехватывает выполнение функций ядра ОС, что позволяет собирать данные об этих функциях, включая аргументы, с которыми они были вызваны. Для взаимодействия с пользовательским пространством ОС этот модуль ядра использует Netlink-сокеты.

Такой подход к реализации системы тестирования имеет ряд сложностей. Во-первых, компиляция и работоспособность модуля будут отличаться при разных версиях ядра ОС. Во-вторых, при текущей реализации системы тестирования один модуль ядра способен выполнять мониторинг только одного процесса. В-третьих, ошибки, допущенные при написании кода модуля ядра, могут негативно влиять на работоспособность ОС в целом.

Поэтому в качестве альтернативы модулю ядра в системе тестирования авторами был разработан монитор системных вызовов на основе технологии eBPF [22]. Эта технология позволяет запускать специализированные программы в привилегированном контексте, то есть на уровне ядра ОС. Данные программы активизируются при возникновении некоторых событий. Например, в контексте сбора трасс системных вызовов источниками таких событий могут выступать механизмы Kprobes, Kretprobes или Kernel Tracepoints.

Технология eBPF представляет собой виртуальную машину, имеющую 10 регистров: R0-R9, и отдельный указатель на список фреймов стека R10, доступный только для чтения программами eBPF. При этом каждая программа eBPF имеет свой независимый стек в 512 байт. Реализация виртуальной машины eBPF в ядре ОС включает в себя как интерпретатор, так и JIT-компилятор специальных машинных инструкций eBPF в инструкции для выполнения непосредственно на процессоре. Все машинные инструкции eBPF проходят проверку в его верификаторе, представляющем собой встроенный в ядро ОС статический анализатор кода, который предотвращает исполнение программ с небезопасными операциями, такими как небезопасные обращения к памяти или циклы с неограниченным числом итераций. Архитектура среды исполнения eBPF представлена на рис. 2.

Для коммуникации между программами eBPF и для передачи их данных в пользовательское пространство используются так называемые карты eBPF. Они представляют собой общие для пространства ядра и пользовательского пространства ОС реализации структур данных. С использованием этих карт, передаваемых в пользовательское пространство в виде файловых дескрипторов, может осуществляться сохранение данных программ eBPF. Например, массив произвольного размера (карта BPF_MAP_TYPE_ARRAY) позволяет обходить ограничения по размеру стека в 512 байт с целью хранения длинных строк, а карта BPF_MAP_TYPE_RINGBUF – использовать для передачи данных в пользовательское пространство кольцевой буфер. Также может быть использована хэш-таблица (карта BPF_MAP_TYPE_HASH), например, в качестве промежуточного буфера для хранения аргументов системных вызовов конкретных процессов или для осуществления фильтрации процессов по их идентификаторам.

Для управления программами eBPF в ядре ОС предусмотрен специальный системный вызов bpf(), позволяющий взаимодействовать (создавать, удалять, находить элементы) с соответствующими картами и загружать программы. При этом сами программы разрабатываются на значительно ограниченном из-за верификатора варианте языка Си. Так они не могут вызывать произвольные функции ядра ОС. Вместо этого предусмотрен ограниченный состав доступных «вспомогательных» функций [32]. Помимо невозможности выполнения произвольных функций ядра и ограничений по размеру стека верификатором eBPF дополнительно проверяются следующие ограничения: во-первых, все вызываемые в программе eBPF функции должны быть встраиваемыми (inline), во-вторых, возможно использование только циклов с ограниченным числом итераций.

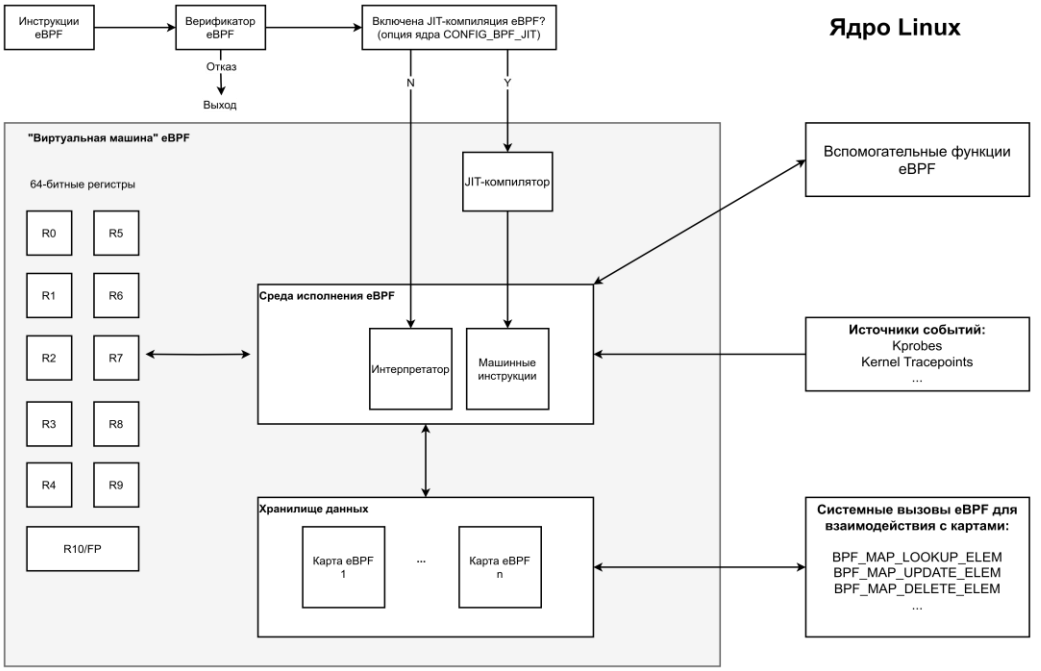

Рис. 2. Архитектура среды исполнения eBPF.
Fig. 2. eBPF runtime architecture.

Одним из ключевых преимуществ технологии eBPF является возможность создавать программы, не требующие компиляции для работы с разными версиями ядра ОС (это делается в соответствии с концепцией Compile Once – Run Everywhere [33]). Такой результат достигается за счет не «прямого» использования внутренних структур ядра ОС, а путем применения специального файла с метаинформацией формата BTF (BPF Type Format), в котором определены сигнатуры вызываемых функций, а также имена и данные о смещении полей структур. Однако в свою очередь для этого требуется ядро ОС с соответствующими активными параметрами компиляции: CONFIG_DEBUG_INFO_BTF и CONFIG_DEBUG_INFO_BTF_MODULES. Эти параметры установлены в отладочных ядрах семейства debug ОС Astra Linux, начиная с релиза 2024 г. (1.8.1), и на стандартных ее ядрах generic, начиная с релиза 2025 года (1.8.2).

В системе тестирования при загрузке программ с помощью библиотеки libbpf, в свою очередь осуществляющей системный вызов bpf(), верификатором проводится их статический анализ, и в случае успеха осуществляется назначение им обработчиков в ядре ОС, то есть инициализация соответствующего механизма перехвата событий. Далее при возникновении события перехвата происходит вызов программы eBPF, который сопровождается выполнением последовательности действий, представленных диаграммой на рис. 3. При запуске самих тестов в системе тестирования для получения информации о системных вызовах используется механизм точек трассировки ядра (Kernel Tracepoints): raw_syscalls/sys_enter – для трассировки аргументов системных вызовов, raw_syscalls/sys_exit – для получения их кодов возврата и передачи данных в кольцевой буфер для чтения программой пользовательского пространства. При этом для получения данных о значениях аргументов функций подсистемы безопасности PARSEC, таких как уровни целостности и конфиденциальности (доступа) процессов, файлов и каталогов, а также для получения их фактических имен, используются механизмы Kprobes и Kretprobes. Для сохранения данных между программами eBPF используется промежуточный буфер, представляющий собой хэш-таблицу. Таким образом, разработанный монитор системных вызовов имеет возможность записывать трассу всех системных вызовов теста.

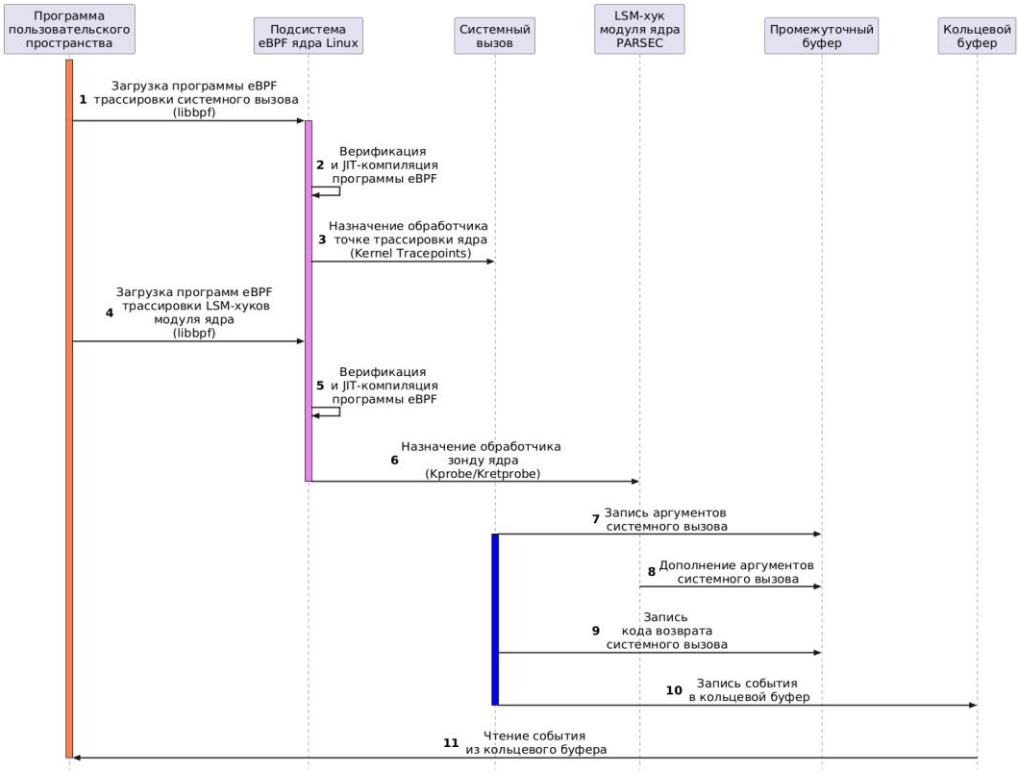

Рис. 3. Диаграмма работы монитора системных вызовов на основе eBPF.
Fig. 3. Sequence diagram of the eBPF-based syscall monitor.

В итоге применение технологии eBPF позволяет расширить функциональные возможности системы тестирования, в том числе за счет распараллеливания тестов.

5. Заключение

В настоящей статье рассмотрены итоги соответствующих рекомендациям ГОСТ Р 59453.4-2025 исследований и разработок по реализации системы тестирование подсистемы безопасности PARSEC ОС Astra Linux на основе нижнеуровневого представления МРОСЛ ДП-модели – PARSEC-модели. Для этого авторами было осуществлено описание PARSEC-модели на языке формального метода Event-B и ее верификация с применением инструментального средства Rodin. Далее на основе результатов [12-13] была разработана адаптированная к подсистеме безопасности PARSEC система тестирования, ориентированная в первую очередь на реализуемые в ОС Astra Linux механизмы МКЦ и МРД. Для повышения эффективности работы этой системы тестирования в статье предложен подход к использованию для ее распараллеливания технологии eBPF.

В дальнейшем планируется продолжить доработку PARSEC-модели с учетом изменений, вносимых как в верхнеуровневое представление МРОСЛ ДП-модели, так и в программный код подсистемы безопасности PARSEC ОС Astra Linux. Для системы тестирования будут разрабатываться новые тесты с целью расширения охвата ими функций механизмов МКЦ и МРД. Кроме того, предполагается исследовать применимость технологии eBPF для разработки системы мониторинга в реальном времени (то есть в штатно работающей ОС по аналогии с технологией Run-Time Verification [34]) подсистемы безопасности PARSEC на соответствие PARSEC-модели.

Список литературы / References

[1]. ГОСТ Р 56939-2024 «Защита информации. Разработка безопасного программного обеспечения. Общие требования». М.: Стандартинформ. 36 с. / GOST R 56939-2024 «Information protection. Secure Software Development. General requirements», 2024. (in Russian).

[2]. ГОСТ Р 59453.1-2021 «Защита информации. Формальная модель управления доступом. Часть 1. Общие положения». М.: Стандартинформ. 16 с. / GOST R 59453.1-2021 «Information protection. Formal access control model. Part 1. General principles», 2021 (in Russian).

[3]. Девянин П.Н. Модели безопасности компьютерных систем. Управление доступом и информационными потоками. Учебное пособие для вузов. 3-е изд., перераб. и доп. М.: Горячая линия – Телеком, 2020. 352 с.: ил. / P.N. Devyanin. Security models of computer systems. Control for access and information flows. Hotline-Telecom, 2020, 352 p. (in Russian).

[4]. Выписка из Требований по безопасности информации, утвержденных приказом ФСТЭК России от 2 июня 2020 г. N 76. Доступно по ссылке: https://fstec.ru/dokumenty/vse-dokumenty/spetsialnye-normativnye-dokumenty/trebovaniya-po-bezopasnosti-informatsii-utverzhdeny-prikazom-fstek-rossii-ot-2-iyunya-2020-g-n-76, 05.06.2025 / Excerpts from Requirements for information security approved by FSTEK Russia order #76 of 2nd June 2020. Available at https://fstec.ru/dokumenty/vse-dokumenty/spetsialnye-normativnye-dokumenty/trebovaniya-po-bezopasnosti-informatsii-utverzhdeny-prikazom-fstek-rossii-ot-2-iyunya-2020-g-n-76, accessed 05.06.2025. (in Russian).

[5]. Операционная система специального назначения Astra Linux Special Edition. Доступно по ссылке: https://astra.ru/software-services/os/, 05.06.2025. / Astra Linux Special Edition operating system. Available at https://astra.ru/software-services/os/, accessed 05.06.2025.

[6]. Девянин П.Н., Тележников В.Ю., Третьяков С.В. Основы безопасности операционной системы Astra Linux Special Edition. Управление доступом. Учебное пособие. М., Горячая линия – Телеком, 2022, 148 с. / Devyanin P.N., Telezhnikov V.Y., Tret'yakov S.V. Astra Linux Special Edition security basics. Access control. Hotline-Telecom, 2022, 148 p. (in Russian).

[7]. Bishop M. Computer Security: Art and Science, 2nd edition. Pearson Education Inc., 2018, 1440 p.

[8]. Abrial J.-R., Hallerstede S. Refinement, decomposition, and instantiation of discrete models: Application to Event-B // Fundamenta Informaticae, Volume 77, Issue 1-2, 2007, pp. 1-28.

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

[10]. Девянин П.Н., Леонова М.А. Приемы по доработке описания модели управления доступом ОССН Astra Linux Special Edition на формализованном языке метода Event-B для обеспечения ее автоматизированной верификации с применением инструментов Rodin и ProB // Прикладная дискретная математика. 2021. № 52. С. 83-96. / P. N. Devyanin, M. A. Leonova, "The techniques of formalization of OS Astra Linux Special Edition access control model using Event-B formal method for verification using Rodin and ProB", Prikl. Diskr. Mat., 2021, no. 52, pp. 83–96 (In Russian).

[11]. ГОСТ Р 59453.4-2025 «Защита информации. Формальная модель управления доступом. Часть 4. Рекомендации по верификации средства защиты информации, реализующего политики управления доступом, на основе формализованных описаний модели управления доступом». М.: Стандартинформ. 20 с. / GOST R 59453.4-2025 «Information protection. Formal model of access control. Part 4. Recommendations on verification of information protection tool implementing access control policies based on formalized descriptions of access control model», 2025 (in Russian).

[12]. 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, С. 15-26 / 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).

[13]. Петренко А.К., Девянин П.Н., Ефремов Д.В., Карнов А.А., Корныхин Е.В., Кулямин В.В., Хорошилов А.В. Методы динамической верификации промышленных средств защиты информации на основе формальных моделей управления доступом // Труды ИСП РАН, 2025, Т. 37, вып. 3, С. 277-290 / Petrenko A.K., Devyanin P.N., Efremov D.V., Karnov A.A., Kornykhin E.V., Kuliamin V.V., Khoroshilov A.V. Methods of runtime verification of industrial information security tools based on formal access control models. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 3, 2025. pp. 277-290 (in Russian)

[14]. Девянин П.Н., Хорошилов А.В., Тележников В.Ю. Формирование методологии разработки безопасного системного программного обеспечения на примере операционных систем // Труды ИСП РАН, том 33, вып. 5, 2021, С. 25-40 / Devyanin P.N., Telezhnikov V.Y., Khoroshilov V.V. Building a methodology for secure system software development on the example of operating systems. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 5, 2021, pp. 25-40 (in Russian).

[15]. ГОСТ Р 59453.3-2025 «Защита информации. Формальная модель управления доступом. Часть 3. Рекомендации по разработке». М.: Стандартинформ. 20 с. / GOST R 59453.3-2025 «Information protection. Formal access control model. Part 3. Recommendations on development», 2025 (in Russian).

[16]. Девянин П.Н. О разработке проекта национального стандарта ГОСТ Р «Защита информации. Формальная модель управления доступом. Часть 3. Рекомендации по разработке» // Труды ИСП РАН, том 36, вып. 3, 2024, С. 63-82 / Devyanin P.N. On the development of the draft standard GOST R “Information protection. Formal access control model. Trudy ISP RAN/Proc. ISP RAS, vol. 36, issue 3, 2024, pp. 63-82 (in Russian).

[17]. Девянин П.Н. Результаты переработки уровней ролевого управления доступом и мандатного контроля целостности формальной модели управления доступом ОС Astra Linux // Труды ИСП РАН, том 35, вып. 5, 2023, С. 7-22 / Devyanin P.N. The Results of Reworking the Levels of Role-Based Access Control and Mandatory Integrity Control of the Formal Model of Access Control in Astra Linux. Trudy ISP RAN/Proc. ISP RAS, vol. 35, issue 5, 2023, pp. 7-22 (in Russian).

[18]. ГОСТ Р 59453.2-2021 «Защита информации. Формальная модель управления доступом. Часть 2. Рекомендации по верификация формальной модели управления доступом». М.: Стандартинформ. 12 с. / GOST R 59453.2-2021 «Information protection. Formal access control model. Part 2. Recommendations on verification of formal access control model», 2021 (in Russian).

[19]. Abrial J.-R., Butler M. 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.

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

[21]. Ковалев М.Г. Трассировка сетевых пакетов в ядре Linux с использованием eBPF // Труды ИСП РАН, том 32, вып. 3, 2020, С. 71-78 (на английском языке) / Kovalev M.G. Tracing Network Packets in the Linux Kernel using eBPF. Trudy ISP RAN/Proc.ISP RAS, vol. 32, issue 3, 2020. pp. 71-78.

[22]. Gregg B. BPF Performance Tools. Addison-Wesley Professional, 2019, 880 p.

[23]. Tetragon. Overview. Available at https://tetragon.io/docs/overview/, accessed 05.06.2025.

[24]. The Falco Project. Developer Guide. Available at https://falco.org/docs/developer-guide/, accessed 05.06.2025.

[25]. Aqua Security. Aqua Tracee. Available at https://www.aquasec.com/products/tracee/, accessed 05.06.2025.

[26]. Wright C., Cowan C., Smalley S., Morris J., Kroah-Hartman G. Linux Security Modules: General Security Support for the Linux Kernel. Proc. of the 11-th USENIX Security Symposium, pp. 17–31, 2002.

[27]. Девянин П.Н., Старостин А.А., Панов Д.С., Усачев С.В. Проектирование и развитие механизма мандатного контроля целостности в операционной системе Astra Linux // Труды ИСП РАН, том 37, вып. 2, 2025, С. 61-78 / Devyanin P.N., Starostin A.A., Panov D.S., Usachev S.V. Design and Development of a Mandatory Integrity Control Mechanism in the Astra Linux Operating System. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 2, 2025, pp. 61–78 (in Russian).

[28]. D. Efremov and I. Shchepetkov. Runtime Verification of Linux Kernel Security Module. Proc. Of International Workshop on Formal Methods, LNCS 12233:185-199, Springer, 2020.

[29]. Allure Report Documentation. Available at https://allurereport.org/docs/, accessed 05.06.2025.

[30]. Карнов А.А. Проблема неопределенности в анализе трасс на основе высокоуровневых моделей в контексте динамической верификации // Труды ИСП РАН, том 36, вып. 4, 2024, С. 169-182 / Karnov A.A. Uncertainty Problem in High-Level Model-Based Trace Analysis as Part of Runtime Verification. Trudy ISP RAN/Proc. ISP RAS, vol. 36, issue 4, 2024, pp. 169-182 (In Russian).

[31]. pytest: helps you write better programs. Available at https://docs.pytest.org/en/stable/, accessed 05.06.2025.

[32]. bpf-helpers(7) – Linux manual pages. Available at https://man7.org/linux/man-pages/man7/bpf-helpers.7.html, accessed 05.06.2025.

[33]. BPF CO-RE, eBPF Docs. Available at https://docs.ebpf.io/concepts/core/, accessed: 05.06.2025.

[34]. Linux 6.0 Adding Run-Time Verification For Running On Safety Critical Systems. Available at https://www.phoronix.com/news/Linux-6.0-Runtime-Verification, accessed 05.06.2025.

Информация об авторах / Information about authors

Петр Николаевич ДЕВЯНИН – член-корреспондент Академии криптографии России, доктор технических наук, профессор, научный руководитель ООО «РусБИТех-Астра» («Группа Астра»). Область интересов: теория информационной безопасности, формальные модели безопасности компьютерных систем, разработка безопасного программного обеспечения, операционные системы семейства Linux.

Petr Nikolaevich DEVYANIN – Doctor of Technical Sciences, corresponding member of Russian Academy of Cryptography, professor, scientific director in RusBITech-Astra (Astra Linux). Field of Interest: information security theory, formal security models of computer systems, secure software development, operating systems of Linux family.

Сергей Сергеевич ЖИЛЯКОВ – инженер ООО «РусБИТех-Астра» («Группа Астра»). Область интересов: формальные модели безопасности компьютерных систем, операционные системы семейства Linux.

Sergey Sergeevich ZHILIAKOV – engineer in RusBITech-Astra (Astra Linux). Field of Interest: formal security models of computer systems, operating systems of Linux family.

Александр Игоревич СМИРНОВ – инженер ООО «РусБИТех-Астра» («Группа Астра»). Область интересов: формальные модели безопасности компьютерных систем, операционные системы семейства Linux, разработка безопасного программного обеспечения.

Alexander Igorevich SMIRNOV – engineer in RusBITech-Astra (Astra Linux). Field of Interest: formal security models of computer systems, operating systems of Linux family, secure software development.