Национальный исследовательский ядерный университет
«МИФИ»

На правах рукописи

Каннер Андрей Михайлович

Модель и алгоритмы обеспечения безопасности управления доступом в операционных системах семейства Linux

Специальность 2.3.6 —
Методы и системы защиты информации, информационная безопасность

ДИССЕРТАЦИЯ
на соискание ученой степени
кандидата технических наук

Автор:

Научный руководитель:
кандидат технических наук, доцент
Епишкина Анна Васильевна

Москва – 2023

Оглавление

ВВЕДЕНИЕ

Актуальность темы исследования. Операционная система (ОС) GNU/Linux является наиболее известным представителем свободного программного обеспечения (ПО). После принятия государственных программ GNU/Linux широко применяется в учебных, государственных и иных учреждениях Российской Федерации – ПФР, ФНС и других. В связи с этим все больше конфиденциальной информации и персональных данных хранится и обрабатывается в среде GNU/Linux, и, значит, возрастает необходимость в обеспечении безопасности таких систем.

Операционные системы общего назначения, которой является GNU/Linux, изначально проектировались без учета необходимости обеспечения защиты данных. Из-за этого средства разграничения доступа не являются неотъемлемой частью ядра Linux – в случае их отключения система будет продолжать функционировать. В соответствии с этим любое средство разграничения доступа GNU/Linux имеет уязвимые периоды: когда ОС уже начала работать, но функции защиты информации еще не активизированы; когда функции защиты активизированы, но были случайно или преднамеренно отключены. Важным преимуществом GNU/Linux перед другими распространенными ОС с этой точки зрения является открытость исходных кодов встроенных средств защиты, что упрощает возможность анализа и исправления их недостатков.

Существуют известные уязвимости компонент ОС, которые позволяют внутреннему нарушителю[1] исключить активацию средств разграничения доступа GNU/Linux и других систем на начальном этапе загрузки средства вычислительной техники (СВТ) еще до запуска их ядра[2]. Также известны уязвимости, позволяющие несанкционированно воздействовать на средство разграничения доступа ОС GNU/Linux с целью изменения порядка его работы или отключения функций защиты информации[3]. Приведенные уязвимости позволяют осуществлять несанкционированный доступ (НСД) к защищаемым данным ОС. При этом главной причиной возникновения этих уязвимостей является то, что встроенные средства разграничения доступа GNU/Linux и ассоциированные с ними объекты (списки разрешенных субъектов и их права доступа) не рассматриваются как полноценные объекты системы, которые необходимо защищать как и данные.

В связи с этим для обеспечения безопасности управления доступом в среде GNU/Linux необходимо обеспечивать активацию и непрерывность работы функций защиты, что подразумевает невозможность загрузки ОС без средств защиты информации и невозможность изменения конфигурации этих средств на всем протяжении ее функционирования – от загрузки и до завершения работы.

Кроме того, в ОС GNU/Linux одновременно функционирует сразу несколько встроенных средств разграничения доступа, но внедряемые с помощью них правила не всегда соответствуют формальным моделям безопасности (Харрисона-Руззо-Ульмана, Белла-ЛаПадула и другим), из-за чего нельзя теоретически гарантировать защищенность данных от НСД или выполнение заданной политики управления доступом. Более того, в известных и широко используемых фор-

[1] 3й тип (внутренний нарушитель) в соответствии с руководящими документами ФСТЭК.
[2] уязвимости CVE-2009-4128 и CVE-2015-8370.
[3] уязвимости CVE-2018-1063, CVE-2021-23240 и CVE-2021-4034.

мальных моделях безопасности средство разграничения доступа и связанные с ним объекты не участвуют в моделируемом субъектно-объектном взаимодействии системы. В большинстве моделей безопасности на уровне аксиом подразумевается присутствие средства разграничения доступа и обязательное его участие в любых типах взаимодействия субъектов и объектов доступа, а вопрос безопасности самого этого средства защиты не рассматривается.

Таким образом, в настоящее время существует потребность в использовании GNU/Linux и в защите информации в среде этих ОС, но применения для этого встроенных средств недостаточно для защиты данных от несанкционированного доступа. Более того, устранение перечисленных недостатков для всех существующих встроенных средств разграничения доступа в GNU/Linux – практически нереализуемая задача. В соответствии с этим требуется формирование модели и алгоритмов для обеспечения безопасности управления доступом в среде GNU/Linux, а также их практическая реализация, и тема диссертационной работы является актуальной.

Степень разработанности темы исследования. В разное время ощутимый вклад в развитие теории и практики основ компьютерной безопасности сложных информационных систем (ИС), в том числе в части формальных моделей безопасности, защиты технических, программных и информационных ресурсов внесли такие отечественные и зарубежные ученые, как В. А. Герасименко, В. П. Лось, В. А. Конявский, М. Харрисон (M. Harrison), В. Руззо (W. Ruzzo), Дж. Ульман (J. Ullman), Д. Белл (D. Bell), Л. ЛаПадула (L. LaPadula), Р. Сандху (R. Sandhu), Д. Феррайоло (D. Ferraiolo), Р. Кун (R. Kuhn), Д. Деннинг (D. Denning), Дж. МакЛин (J. McLean), П. Н. Девянин, Д. П. Зегжда и А. Ю. Щербаков. Вопросам возможности применения полученных научных результатов для совершенствования систем управления доступом в существующих реальных компьютерных системах (КС), в том числе в ОС на основе ядра Linux, посвящены труды П. Н. Девянина, Д. П. Зегжды, А. Ю. Щербакова и В. А. Конявского, в которых описаны результаты практического использования моделей безопасности компьютерных систем.

Средства разграничения доступа в ОС должны соответствовать одной или нескольким согласованным формальным моделям безопасности КС, предложенным в работах вышеперечисленных авторов, только в этом случае при применении таких средств можно обеспечить защищенность данных от НСД. Однако даже при использовании известных формальных моделей безопасности не удается устранить все недостатки существующих средств разграничения доступа ОС, в частности в GNU/Linux. Средство разграничения доступа должно активироваться на начальном этапе работы ОС, исключая любую возможность повлиять на процесс своей загрузки и дальнейшего функционирования, для последующего предотвращения НСД к информации. При этом должна также существовать четкая процедура для санкционированного изменения правил управления доступом, препятствующая их модификации недоверенными субъектами доступа – процедура безопасного администрирования. Поэтому формальные модели безопасности должны учитывать вопросы построения и администрирования средств разграничения доступа. Работы А. Ю. Щербакова и В. А. Конявского частично учитывают эту необходимость, однако, использования созданных на основе результатов этих исследований средств доверенной загрузки не всегда достаточно. В исследованиях остальных авторов средство разграничения доступа не является сущностью системы, которую также необходимо защищать. Они направлены на вы-

полнение другой важной задачи – обеспечение безопасности данных за счет принятой политики управления доступом при условии, что все доступы проходят через абстрактное средство разграничения доступа.

С другой стороны, результаты работ указанных авторов являются научно обоснованными и вместе с разработанными моделями безопасности позволяют обеспечить теоретические гарантии защищенности систем, в которых они применяются. В связи с этим целесообразно провести исследование существующих средств разграничения доступа с учетом выделенных аспектов и с использованием результатов известных работ разработать и обосновать корректность новых подходов к защите информации, устраняющих выявленные недостатки таких средств в ОС GNU/Linux. Отсюда вытекает научная задача диссертации, которая заключается в формировании модели и алгоритмов обеспечения безопасности управления доступом, позволяющих в условиях потребности использования ОС GNU/Linux соблюдать установленный порядок предоставления доступа к хранящимся и обрабатываемым данным.

Целью диссертационной работы является разработка научно-обоснованных алгоритмов обеспечения безопасности управления доступом, исключающих возможность обхода действующих правил доступа в ОС GNU/Linux.

Задачи работы. Для достижения поставленной цели в работе решались следующие задачи:

1. Анализ существующих средств разграничения доступа к данным в ОС GNU/Linux и используемых в них способов защиты информации.

2. Разработка научно-обоснованных алгоритмов обеспечения безопасности управления доступом, основанных на использовании модели изолированной программной среды субъектов, исключающей возможность обхода действующих правил управления доступом.

3. Разработка средства разграничения доступа для ОС GNU/Linux, реализующего предложенные алгоритмы и модель безопасности.

4. Апробация и анализ результатов применения разработанных алгоритмов и средства разграничения доступа в ОС GNU/Linux.

Положения, выносимые на защиту. В диссертационной работе получены и выносятся на защиту следующие положения и научные результаты:

1. Стандартные средства разграничения доступа ОС GNU/Linux имеют уязвимые фазы работы на этапе загрузки системы и в процессе ее функционирования, что обеспечивает возможность обхода нарушителем применяемой политики управления доступом.

2. Предложенный алгоритм доверенной загрузки загрузчика и ОС GNU/Linux, устраняет возможность блокировки процесса активации средства разграничения доступа при загрузке системы на различных аппаратных платформах.

3. Предложенный алгоритм встраивания функций защиты от НСД на начальном этапе загрузки ОС GNU/Linux, обеспечивает их активацию и блокирует возможность отключения функций безопасности, а также осуществляет принудительную остановку системы, обусловленную нарушением точек встраивания средства разграничения доступа.

4. Разработанное средство разграничения доступа для ОС GNU/Linux, в котором реализованы предложенные алгоритмы безопасности управления доступом, обеспечивает выполнение заданной политики безопасности на протяжении всего периода работы ОС.

Научная новизна работы. Новизна полученных научных результатов работы заключается в следующем:

– Предложена научно-обоснованная модель безопасности для ОС GNU/Linux и средства разграничения доступа, основанная на положениях известной субъектно-ориентированной модели изолированной программной среды (СО-модели ИПС), которая обеспечивает реализацию заданной политики управления доступом различных пользователей и системных субъектов и одновременный контроль непрерывного выполнения функций защиты для всех действующих в системе правил доступа (п. 15).

– Предложен алгоритм обеспечения доверенной загрузки загрузчика и ОС GNU/Linux, в котором, в отличие от известных алгоритмов, используются процедуры ограничения и контроля загрузчиков, что, в свою очередь, обеспечивает невозможность нарушения целостности компонент системы и позволяет устранить возможность блокировки процесса активации средства разграничения доступа (п. 2).

– Предложен алгоритм встраивания функций защиты от НСД на начальном этапе загрузки ОС GNU/Linux, в котором, в отличие от известных алгоритмов, используются процедуры обеспечения активации функций безопасности и блокировка их отключения, что исключает нарушение функционирования средства разграничения доступа (п. 18).

Теоретическая значимость работы. Теоретическая значимость научных результатов исследования заключается в обосновании необходимых и достаточных условий достижения безопасного состояния на начальной фазе работы ОС и условий последующего непрерывного выполнения заданных правил доступа, а также в разработке соответствующих им алгоритмов обеспечения безопасности управления доступом.

Практическая ценность работы. Практическая значимость результатов исследования заключается в том, что на базе полученных научных результатов обоснованы требования и разработано удовлетворяющее им средство разграничения доступа для ОС GNU/Linux, реализующее предложенные алгоритмы обеспечения безопасности. Самостоятельное практическое значение имеют следующие результаты работы:

– Разработанная модель безопасности может быть использована не только в рамках ОС Linux, но также и для других ОС и систем, в которых реализуется субъектно-объектное взаимодействие (системы управления базами данных, системы виртуализации, межсетевое взаимодействие и др.).

– Система с внедренным средством разграничения доступа, соответствующая предложенной модели безопасности и реализующая необходимые алгоритмы обеспечения безопасности управления доступом, смоделирована с использованием темпоральной логики действий и верифицирована на соответствие инвариантам безопасности.

– Предложенные автором алгоритмы обеспечения безопасности, а также разработанное средство разграничения доступа имеют широкое назначение, могут быть применены не только

в отношении GNU/Linux, но и для других POSIX/SUS[4]-совместимых ОС, а также на СВТ с архитектурой x86_64, arm64, s390x (мейнфреймы IBM System Z) и других.

Методология и методы исследования. В диссертационной работе используются методы системного анализа, а также теории моделирования, математической логики, автоматов и множеств.

Соответствие специальности научных работников. Содержание диссертационной работы и полученные научные результаты соответствуют п. 2, 15 и 18 Паспорта специальности 2.3.6 Методы и системы защиты информации, информационная безопасность.

Достоверность научных положений и выводов обеспечена корректным использованием теорий моделирования и автоматов, математической логики и теоретических основ компьютерной безопасности, а также положительными результатами использования разработанного средства разграничения доступа в реальных проектах и совпадением ожидаемых результатов от использования предложенных алгоритмов обеспечения безопасности с полученными при экспериментальных исследованиях.

Внедрение результатов работы. Результаты диссертационной работы используются ГНИИ ПТЗИ ФСТЭК России в рамках исследований уязвимостей системного ПО для национального банка данных угроз безопасности информации и для верификации функциональных требований к средствам защиты информации от НСД на раннем этапе загрузки ОС. Также результаты работы применяются ЗАО «ОКБ САПР» в программно-аппаратном комплексе средств защиты информации от НСД «Аккорд-X» (свидетельство о государственной регистрации № 2015612555). Результаты исследования также внедрены в учебный процесс НИЯУ МИФИ по дисциплине «Программно-аппаратные средства защиты информации» (лабораторные работы). Соответствующие документы, подтверждающие практическое использование и внедрение результатов исследования, приведены в приложении к тексту диссертационной работы.

Апробация работы. Основные положения и результаты работы представлялись и обсуждались на следующих конференциях: XVII, XVIII, XIX, XXIII, XXIV, XXV и XXVI Международные конференции «Комплексная защита информации», г. Суздаль (2012, 2018), г. Брест (РБ, 2013), г. Псков (2014), г. Витебск (РБ, 2019), МО (2020), Минск (РБ, 2021); XIII Международная конференция «Информационная безопасность», г. Таганрог (2013); Международная конференция «Обеспечение безопасности инфокоммуникационных и цифровых технологий», г. Воронеж (2015 и 2016); VII Международная конференция «Engineering & Telecommunication», г. Долгопрудный (2020); Международная конференция «Ural Symposium on Biomedical Engineering, Radioelectronics and Information Technology», г. Екатеринбург (2021 и 2022).

Публикации. Основные результаты по теме диссертации изложены в 24 печатных работах общим объемом 9.52 п.л., в которых автору принадлежит 8.30 п.л. Из них 18 печатных работ изданы в рецензируемых научных изданиях, определенных Высшей аттестационной комиссией при Министерстве науки и высшего образования Российской Федерации и Аттестационным

[4] **p**ortable **o**perating **s**ystem **i**nterface for Uni**x** (POSIX), **S**ingle **U**nix **S**pecification (SUS) – набор стандартов, описывающих интерфейсы между ОС и прикладным программным обеспечением.

советом УрФУ, и изданиях, приравненных к ним, в том числе 4 – в журналах, индексируемых международной системой научного цитирования Scopus, а также 6 — другие публикации.

Личный вклад. Содержание диссертации и основные положения, выносимые на защиту, отражают персональный вклад автора в работу. Все основные представленные в диссертации результаты получены автором самостоятельно. В работах, опубликованных в соавторстве, лично автору принадлежат: схема взаимодействия субъектов и объектов в GNU/Linux, способы встраивания средства разграничения доступа в ОС и их практическая реализация; анализ средств разграничения доступа к информации в GNU/Linux и путей их совершенствования, программно-технические решения по созданию средств разграничения доступа; реализация очистки динамически выделяемой памяти в ОС GNU/Linux, оценка ее влияния на производительность системы; результаты тестирования и верификации средств защиты информации.

1. Анализ состояния предметной области. Постановка задач исследования

Операционные системы Linux являются наиболее известным представителем свободного ПО и на сегодняшний день повсеместно используются при создании различных информационных систем и в других областях информационных технологий. Linux – это обобщенное название Unix-подобных операционных систем, в основе которых лежит одноименное ядро Linux. Ядро Linux создается и распространяется в соответствии с моделью разработки свободного и открытого ПО. При этом общее название «Linux» не подразумевает какой-либо официальной комплектации ОС, большое количество комплектаций часто распространяются бесплатно в виде различных дистрибутивов, имеющих свой набор прикладных программ и уже настроенных под конкретные нужды пользователей. В качестве основной части прикладных программ (системных программ или пользовательского «окружения») часто используется свободное программное обеспечение проекта GNU, поэтому большинство дистрибутивов Linux, а также и все семейство таких операционных систем, принято называть GNU/Linux.

С тех пор, как ядро Linux было создано для x86-совместимых СВТ, оно было портировано на множество программных и аппаратных платформ, в том числе x86_64, powerpc, arm, sparc, s390x и другие, включая портирование всего прикладного ПО. Семейство GNU/Linux изначально получило широкое распространение в качестве серверных ОС, но на сегодняшний день может применяться на различных устройствах – от встраиваемых (англ. embedded) систем и портативных устройств, которые каждый день использует современный человек (смартфоны, планшеты, одноплатные компьютеры, разнообразные мультимедийные и «носимые» электронные устройства) до крупных серверных комплексов и мейнфреймов различных производителей. Сейчас наиболее популярные дистрибутивы GNU/Linux стали удобными, надежными и, наконец, простыми в освоении, в том числе и неподготовленным пользователем. Так или иначе, GNU/Linux постепенно занимает свое место в жизни практически каждого человека. Это подтверждается результатами последних статистических исследований, приведенных на Рисунке 1.1 и в Таблице 1.1, а также многочисленными случаями внедрения и использования свободного ПО (в том числе и GNU/Linux) в учебные, государственные и иные учреждения Российской Федерации и других стран.

Таблица 1.1 – Статистические данные по использованию GNU/Linux в качестве ОС для различных классов СВТ (на конец 2020 г.)

Категория СВТ	**Источник данных**	**ОС на основе Linux kernel**	**ОС на основе BSD и др. Unix**	**Windows**
1. Суперкомпьютеры	TOP500	100%	–	–
2. Смартфоны, планшеты и пр.	StatCounter Global Stats	70.80%	28.79%	0.07%
3. Серверы (web)	W3Techs	37%	29.5%	29%
4. Встраиваемые устройства	UBM Electronics	38.42%	2.82%	10.73%
5. Мейнфреймы	Gartner	28%	72%	–
6. Настольные ПК, ноутбуки	Net Applications	2.87%	9.75%	86.92%

Продолжение таблицы 1.1

Категория СВТ	Источник данных	ОС на основе Linux kernel	ОС на основе BSD и др. Unix	Windows

– наименьший процент использования среди других ОС в классе СВТ

– наибольший процент использования среди других ОС в классе СВТ

– процент использования среди других ОС в классе СВТ ниже наибольшего и выше наименьшего

Однако в связи с постоянно растущей популярностью GNU/Linux на основе этих ОС появляется все больше систем, обрабатывающих и хранящих конфиденциальную информацию и персональные данные. При этом с каждым годом объем этих данных только увеличивается. В связи с этим актуальной задачей является обеспечение безопасности хранящихся и обрабатываемых в GNU/Linux данных с использованием различных средств защиты информации. При этом средства разграничения и логического управления доступом к информации совместно со средствами идентификации и аутентификации являются базовыми для любой системы защиты данных, без них использование всех прочих средств защиты информации невозможно.

Кроме того, например, в Российской Федерации существует ряд нормативных документов[1], которые предъявляют требования к системам разграничения доступа, а для некоторых категорий данных допускают использование только сертифицированных средств защиты информации. В этих нормативных документах нет явного ограничения на производство таких средств только на территории Российской Федерации, но для зарубежных компаний является серьезным барьером, например, предоставление исходных кодов своих продуктов третьим лицам (сертификационным лабораториям и органам по сертификации). Поэтому количество импортных средств защиты информации в России мало, а сертифицированные средства защиты в ОС GNU/Linux в основном представлены только отечественными производителями. Следует отметить, что на сегодняшний день стандартные средства защиты информации в GNU/Linux в общем случае не сертифицированы по существующим требованиям РФ[2].

Также сфера информационных технологий и защиты информации (в том числе и в ОС GNU/Linux) вошли в государственную программу по импортозамещению[3] и должны стать ключевым фактором повышения устойчивости страны к внешним воздействиям и обеспечения ее безопасности. Одной из задач в данной программе является импортозамещение в области информационных технологий и информационной безопасности. Возможно, использование только отечественных средств защиты информации для уже применяемых (импортных) ОС и другого прикладного ПО будет первым этапом планового замещения импортных технологий в РФ.

Таким образом, в настоящий момент существует потребность в использовании ОС Linux и в защите информации в среде этих ОС от несанкционированного доступа, в том числе в

[1] Руководящие документы Гостехкомиссии России для автоматизированных систем и средств вычислительной техники, приказы ФСТЭК России № 17 и № 21, а также другие нормативные документы.

[2] Существуют сертифицированные по требованиям безопасности информации РФ дистрибутивы ОС GNU/Linux со встроенными средствами защиты. Сами механизмы защиты ядра Linux не сертифицированы.

[3] По поручению Президента РФ распоряжением Правительства Российской Федерации от 1 ноября 2013 года № 2036–р утверждена Стратегия развития отрасли информационных технологий в Российской Федерации на 2014–2020 годы и на перспективу до 2025 года.

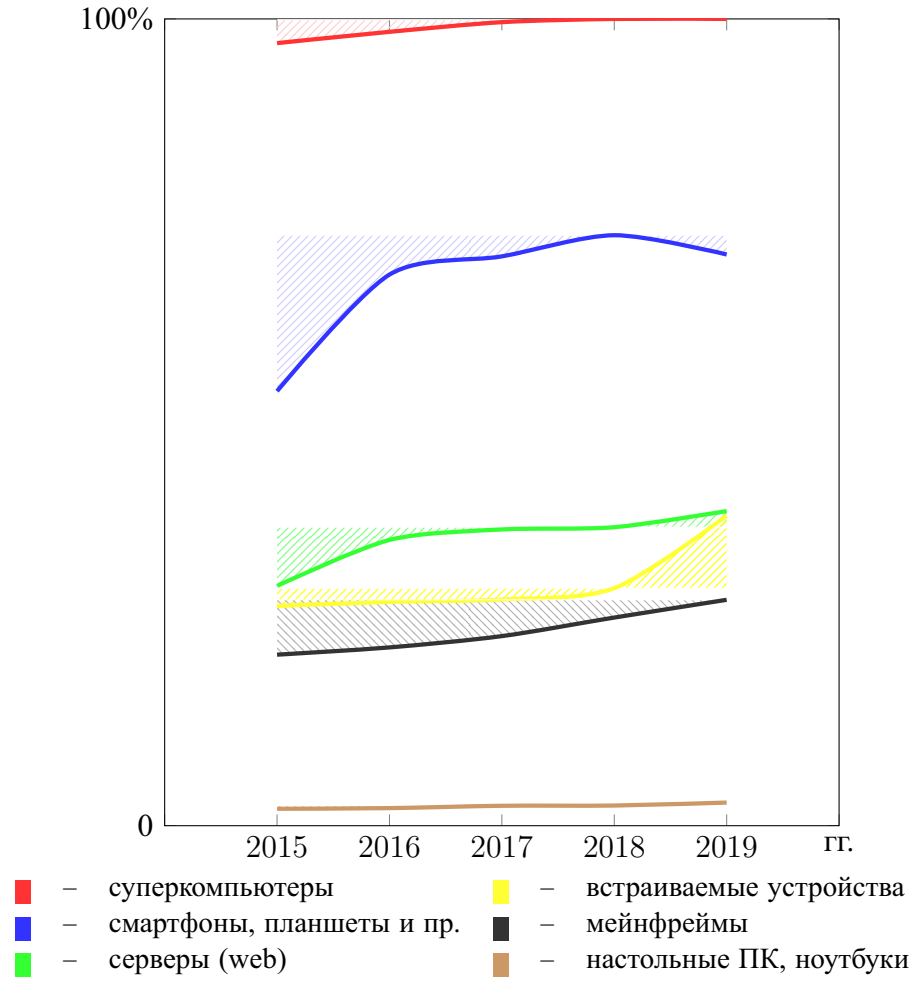

Рисунок 1.1 – Изменение процентного соотношения использования GNU/Linux в качестве ОС для различных классов СВТ за 2016 – 2020 гг.

целях импортозамещения и выполнения требований государственных регуляторов РФ. Для определения возможности удовлетворения этой потребности необходимо вначале рассмотреть описательную модель управления доступом к информации в GNU/Linux, проанализировать существующие средства и механизмы защиты, а также требования нормативных документов РФ, предъявляемые к ним. Отдельно необходимо исследовать результаты известных работ отечественных и зарубежных ученых по совершенствованию механизмов управления доступом в ОС и моделям безопасности компьютерных систем[4].

1.1. Анализ существующих средств управления доступом в ОС GNU/Linux и используемых в них механизмов защиты информации

Средства управления доступом (разграничения или контроля доступа) позволяют специфицировать и контролировать действия, которые субъекты (пользователи, процессы и так далее) некоторой системы (операционной или автоматизированной системы, системы управления база-

[4]Формальные модели безопасности принято использовать для анализа свойств средств защиты от НСД и защищенности компьютерных систем. Реализация определенной модели в системе позволяет получить теоретические гарантии ее защищенности.

ми данных и так далее) могут выполнять над объектами (данными и другими ресурсами). Далее в работе будет рассматриваться именно логическое управление доступом, которое в отличие от физического, реализуется программными или программно-аппаратными средствами.

Логическое управление доступом – это основной механизм многопользовательских систем (а на данный момент современные системы всегда являются многопользовательскими), призванный обеспечить конфиденциальность и целостность объектов, а также их доступность в части невозможности несанкционированного удаления[5]. Задача логического управления доступом заключается в определении легитимных субъектов, а также для каждой пары субъект/объект (сущностей системы) – множества допустимых операций и в контроле выполнения таких ограничений в процессе работы системы с целью предупреждения несанкционированного использования объектов. При этом логическое управление доступом подразумевает предварительную реализацию следующих процедур: идентификации и аутентификации субъектов, авторизации.

При принятии решения о возможности осуществления определенного типа доступа обычно анализируется следующая информация: атрибуты объекта доступа, идентификатор субъекта и его права доступа, тип и дополнительные параметры запроса на доступ (время и место проведения, другая необходимая информация).

Понятия субъектов и объектов, а также типы возможных операций между ними могут меняться от системы к системе, а также необходимого уровня детализации этих сущностей. Таким образом, перед рассмотрением логического управления доступом, например в GNU/Linux, необходимо вначале определить состав субъектов и объектов доступа, а также все возможные способы их взаимодействия.

1.1.1. Анализ состава субъектов и объектов доступа, используемых в ОС GNU/Linux, и механизмов их взаимодействия

Архитектура GNU/Linux базируется на принципах, заложенных в ОС Unix [38]. Крылатое выражение «In Unix everything is a File, unless it isn’t» описывает одну из определяющих черт ОС Unix и ОС, производных от нее, в том числе и GNU/Linux, – за небольшим исключением (например, таких сущностей как сокеты и некоторых других), любая сущность (объект) GNU/Linux представляется на уровне ОС в виде файла в дереве каталогов файловой системы.

Любой файл в файловой системе (ФС), поддерживаемой GNU/Linux, кроме непосредственно данных, содержит метаданные файлового объекта, среди которых можно выделить атрибуты доступа для владельца файла, группы владельца и всех остальных субъектов (вектор «rwxrwxrwx» – атрибуты чтения, записи или выполнения для перечисленных субъектов соответственно).

В качестве субъектов доступа в ОС GNU/Linux в общем случае выступают либо системные процессы ОС, либо процессы, запущенные от имени того или иного пользователя. Возможность доступа определенного процесса (субъекта) ОС GNU/Linux к объекту (файлу в ФС) определяется в зависимости от текущих прав субъекта доступа (процесса) и атрибутов доступа объекта.

[5]Рассмотрение угроз доступности в общем случае выходит за рамки данной работы, далее основное внимание будет уделено вопросам обеспечения конфиденциальности и целостности данных.

1.1.1.1. Анализ состава объектов доступа в ОС GNU/Linux

Определим, что конкретно представляют собой объекты доступа в GNU/Linux и какие данные необходимо защищать в ОС. В GNU/Linux почти любой объект доступа представляется в виде файла в ФС, который на низком уровне представляет собой набор битов без какого-либо признака формата. Таким образом, в системах GNU/Linux файлами являются следующие объекты:

– физические диски – это так называемые файлы блочных устройств (из каталога /dev), которые могут быть использованы для передачи данных какому-либо устройству с помощью записи в них данных (система сама отошлет эти данные драйверу устройства);

– различные порты ввода/вывода;

– пользовательские или системные данные, которые, в отличие от описанных выше объектов, имеют нефиксированный размер.

Преимущество такого подхода к построению системы состоит в том, что операции с любой сущностью (объектом) ОС проводятся одинаково, точно так же как с обычным файлом. А для использования в прикладных программах каких-либо устройств не нужно разрабатывать дополнительный интерфейс взаимодействия. Другими словами, в ОС GNU/Linux имеется некоторый механизм абстракции представления объектов в ОС от механизмов взаимодействия с этими объектами. При этом каждому объекту соответствует некоторый файл в ФС, идентифицируемый по абсолютному пути.

Однако в GNU/Linux существует следующая особенность – данные могут содержаться в различных ФС, при этом одни ФС могут монтироваться «поверх» других. Например, часто вместо одной ФС со всеми системными и пользовательскими файлами при установке GNU/Linux создается несколько разделов, в каждый из которых записываются либо пользовательские данные (/home), либо файлы с данными для загрузки ОС (/boot), либо все остальные данные (/). В такой схеме разметки носителя информации при загрузке ОС GNU/Linux сначала монтируется основной раздел в точку монтирования / (так называемый корень файловой системы – англ. root), затем разделы с файлами данных для загрузки ОС или с пользовательскими данными – в соответствующие каталоги, располагающиеся на этом корневом разделе (в /boot и /home соответственно).

Имея достаточные привилегии в ОС GNU/Linux, можно монтировать устройства, файлы и каталоги вручную в любые точки монтирования (каталоги). В связи с этим такой идентификационный признак для любого файла, как абсолютный путь в ФС, зависит от текущей схемы монтирования файловых систем в ОС. Таким образом, в общем случае нельзя однозначно идентифицировать объекты доступа в ОС GNU/Linux с помощью их абсолютных путей.

Кроме понимания того, что является объектом в ОС GNU/Linux, необходимо определиться с тем, какие именно данные необходимо защищать и как. При классификации угроз выделяют три основных свойства безопасности информации в КС [67] – конфиденциальность, целостность и доступность. При этом средства разграничения доступа и защиты информации от НСД, рассматриваемые в рамках данной диссертационной работы, должны в первую очередь обеспечивать конфиденциальность и целостность данных, а свойство доступности должно полноцен-

но обеспечиваться другими средствами. С точки зрения конфиденциальности и целостности в GNU/Linux необходимо рассмотреть такие объекты ОС, как обычные пользовательские данные, исполняемые бинарные файлы, библиотеки, модули ядра Linux.

В соответствии со стандартом унификации местонахождения файлов и директорий с общим назначением в файловой системе Unix [87], который также используется и в GNU/Linux, перечисленные объекты ОС могут располагаться по следующим абсолютным путям:

– обычные пользовательские данные: /root/ и все домашние каталоги пользователей /home/USERNAME/, где USERNAME – имена субъектов доступа;

– исполняемые бинарные файлы: /bin/, /opt/, /sbin/, /usr/bin/, /usr/sbin/, /usr/local/bin/, /usr/local/sbin/, а также домашние каталоги пользователей;

– библиотеки: /lib[6]/, /usr/lib[6]/, /usr/local/lib[6]/, а также все домашние каталоги пользователей;

– модули ядра Linux: /lib[6]/modules/$(uname -r)/*, где $(uname -r) – текущая версия ядра Linux.

Обеспечение конфиденциальности и целостности пользовательских данных должно осуществляться непосредственно механизмами контроля доступа и целостности. Для статически скомпонованных бинарных файлов, в которые на этапе компиляции вносятся все данные из используемых статических библиотек, также достаточно механизмов контроля доступа и целостности самих объектов. Однако большинство бинарных файлов в ОС GNU/Linux в своей работе используют динамически подключаемые или разделяемые библиотеки (англ. shared libraries), поэтому для контроля доступа и целостности таких файлов недостаточно механизмов защиты только самих объектов, необходимо контролировать все подключаемые на этапе выполнения ядром Linux разделяемые библиотеки.

Кроме того, в ОС GNU/Linux существует возможность динамически дополнять функциональность ядра Linux (новые возможности будут выполняться на уровне ядра ОС – в kernel mode) с помощью загружаемых модулей ядра (англ. loadable kernel modules, LKM). Модули ядра GNU/Linux являются чрезвычайно важными объектами ОС, поэтому для них необходимо обеспечивать конфиденциальность и целостность, начиная с самого раннего этапа загрузки ОС (ведь именно в процессе загрузки ОС загружается большая часть модулей ядра).

1.1.1.2. Анализ состава субъектов доступа в ОС GNU/Linux

При осуществлении любого доступа в ОС GNU/Linux необходимо однозначно определять, какой именно субъект в данный момент совершает этот доступ [26]. С одной стороны, эта задача не выглядит сложной, однако существуют определенные нюансы.

Каждому пользователю системы на уровне ядра Linux соответствует так называемый идентификатор пользователя (англ. user identifier, user ID, UID) и идентификатор основной группы, в которую он входит (англ. group identifier, group ID, GID). Суперпользователю root соответствует UID и GID со значениями 0. UID может соответствовать не только реальному пользователю СВТ, но и всем системным учетным записям, от имени которых выполняются какие-либо процессы в ОС GNU/Linux [54].

[6] lib64 для 64-разрядных ОС GNU/Linux

Как уже было сказано ранее, в качестве субъектов доступа в ОС GNU/Linux в общем случае выступают процессы ОС. У каждого процесса ОС на уровне ядра в описывающей его структуре данных task_struct [3, 45] существует как минимум четыре параметра, идентифицирующие пользователя, от имени которого выполняется в данный момент процесс: эффективный и реальный идентификаторы пользователя и группы пользователей (англ. effective and real user/group ID). Для версий ядра Linux $\geqslant 2.6.29$ эти значения перенесены в структуру cred внутри структуры task_struct (для безопасности эти значения нельзя свободно изменять [3]). Эффективные идентификаторы пользователя и группы характеризуют текущие привилегии процесса, тогда как реальные идентификаторы соответствуют UID или GID субъекта, запустившего процесс. Таким образом, в общем случае присутствие определенного пользователя в ОС (реального пользователя, прошедшего процедуры идентификации и аутентификации, имеющего незавершенные задачи) означает наличие хотя бы одного процесса с реальным или эффективным UID, соответствующим UID этого пользователя.

Процессы всех реальных пользователей, кроме системных процессов с реальным и эффективным UID 0, могут появляться в ОС GNU/Linux только после встроенных процедур идентификации и аутентификации:

– первоначально проводится процедура идентификации: определяется корректность идентифицирующей информации пользователя (введенный логин), существование субъекта доступа с соответствующим идентификатором (UID);

– пользователь проходит аутентификацию: проверку в ОС введенных значений логин/пароль со значениями, заданными в процессе регистрации (иногда этот шаг совмещен с шагом идентификации);

– достоверность идентификации полностью определяется уровнем достоверности выполненной аутентификации.

В ОС GNU/Linux все процедуры идентификации и аутентификации вынесены в такой компонент ОС, как подключаемые модули аутентификации (англ. Pluggable Authentication Modules, PAM). При этом непосредственно за идентификацию и аутентификацию отвечает, как правило, всего один модуль – pam_unix.so (pam_unix2.so или pam_tcb.so в зависимости от используемой версии PAM) [51, 81]. После успешных процедур идентификации и аутентификации пользователя, у процесса, запросившего эту процедуру, изменяются UID на значения, соответствующие логину идентифицированного пользователя (UID пользователя), а сам процесс запускает так называемый командный интерпретатор (командную оболочку, англ. shell) – текстовый или графический пользовательский интерфейс ОС.

Все процессы, возникающие в дальнейшем, например, при запуске различных программ, будут наследовать реальный и эффективный UID этого процесса командного интерпретатора пользователя. Некоторые программы, такие как su (сокр. от англ. «substitute user») или sudo (сокр. от англ. «substitute user do»), для смены текущего пользователя без завершения текущего сеанса работы или для выполнения одной команды от имени другого пользователя в ходе своей работы могут использовать PAM и менять реальный или эффективный UID процесса. При этом процессы от имени другого пользователя могут появляться «каскадно» без завершения текущей

сессии пользователя. В общем случае количество таких «каскадных» сессий разных пользователей не ограничено.

При завершении работы пользователя в определенной сессии либо завершаются все процессы ОС, созданные из первоначального процесса командного интерпретатора пользователя (а затем и сам процесс командной оболочки завершает свою работу), либо процесс с командной оболочкой может принудительно завершить все созданные в рамках текущей сессии процессы. При завершении процесса-родителя по соответствующему сигналу ОС завершаются все его процессы-потомки. При этом любой процесс-потомок сообщает родителю о своем завершении, процесс командного интерпретатора сообщает об этом процессу-родителю, который (в случае отсутствия каскада сессий) в дальнейшем перезапустит процедуры идентификации и аутентификации пользователя. При создании множества каскадных сессий действует аналогичное правило – при завершении определенного процесса все его процессы-потомки завершаются, в том числе и все процессы, образующие (каскадные) сессии других пользователей [44, 54].

1.1.1.3. Анализ механизмов взаимодействия субъектов и объектов в ОС GNU/Linux и управления доступом

Для реализации доступа субъектов (пользователей или процессов / приложений) к объектам в GNU/Linux необходимо обратиться к данным на уровне ядра Linux. Ядро в данном случае позволяет абстрагировать все аппаратные компоненты любого СВТ непосредственно от субъектов доступа. При этом все процессы изначально выполняются в пространстве пользователя или, как иногда говорят, на пользовательском уровне (англ. user space или user mode), а для осуществления какого-либо доступа к объекту им необходимо выполнить системный вызов (англ. system call), то есть воспользоваться некоторыми доступными возможностями в пространстве ядра или, как принято говорить, на уровне ядра ОС (англ. kernel space или kernel mode) [45, 54].

Для различных типов доступа существуют разные системные вызовы: доступ на открытие файла (запись, чтение или исполнение); доступ на создание, переименование или удаление файла; доступ на создание, переименование или удаление каталога/директории; доступ на создание/удаление жесткой или символьной ссылки; и другие.

Системные вызовы существуют не только для осуществления доступа к объектам в GNU/Linux, системный вызов – единственный способ получить из непривилегированного режима доступ к каким бы то ни было ресурсам СВТ. Кроме системных вызовов, взаимодействующих с файлами, существуют также системные вызовы, реализующие операции ввода/вывода (англ. ioctl – input/output control), специфичные для различных устройств; системные вызовы, связанные с монтированием и видимостью файловых систем (mount, umount, pivot_root, chroot); для синхронизации «грязных» блоков файловых систем; для установки и изменения системного времени; для выделения, освобождения и перераспределения оперативной памяти; для создания и завершения процессов; загрузки модулей ядра и так далее [45, 54].

Все системные вызовы описаны в специальной таблице системный вызовов (англ. system call table). Адрес таблицы системных вызовов при загрузке ядра ОС Linux можно найти в специальном файле System.map, содержащем символьную таблицу адресов используемых функций

и процедур. Таблица системных вызовов представляет собой массив, индексами которого являются номера системных вызовов, которые в символьном обозначении имеют вид __NR_xxx, где xxx – имя соответствующего системного вызова (полный список индексов системных вызовов содержится в исходных кодах ядра Linux). В каждой ячейке таблицы системных вызовов содержится указатель (адрес) функции, осуществляющей тот или иной системный вызов. Выполнить тот или иной системный вызов можно, либо напрямую осуществив такой вызов из приложения, либо с использованием функций из стандартных библиотек GNU/Linux – glibc и, возможно, других [22,36,45,48], что изображено на Рисунке 1.2. Для выполнения системного вызова достаточно указать его номер или символьное обозначение при вызове специальной команде/инструкции (int 80h или sysenter/syscall).

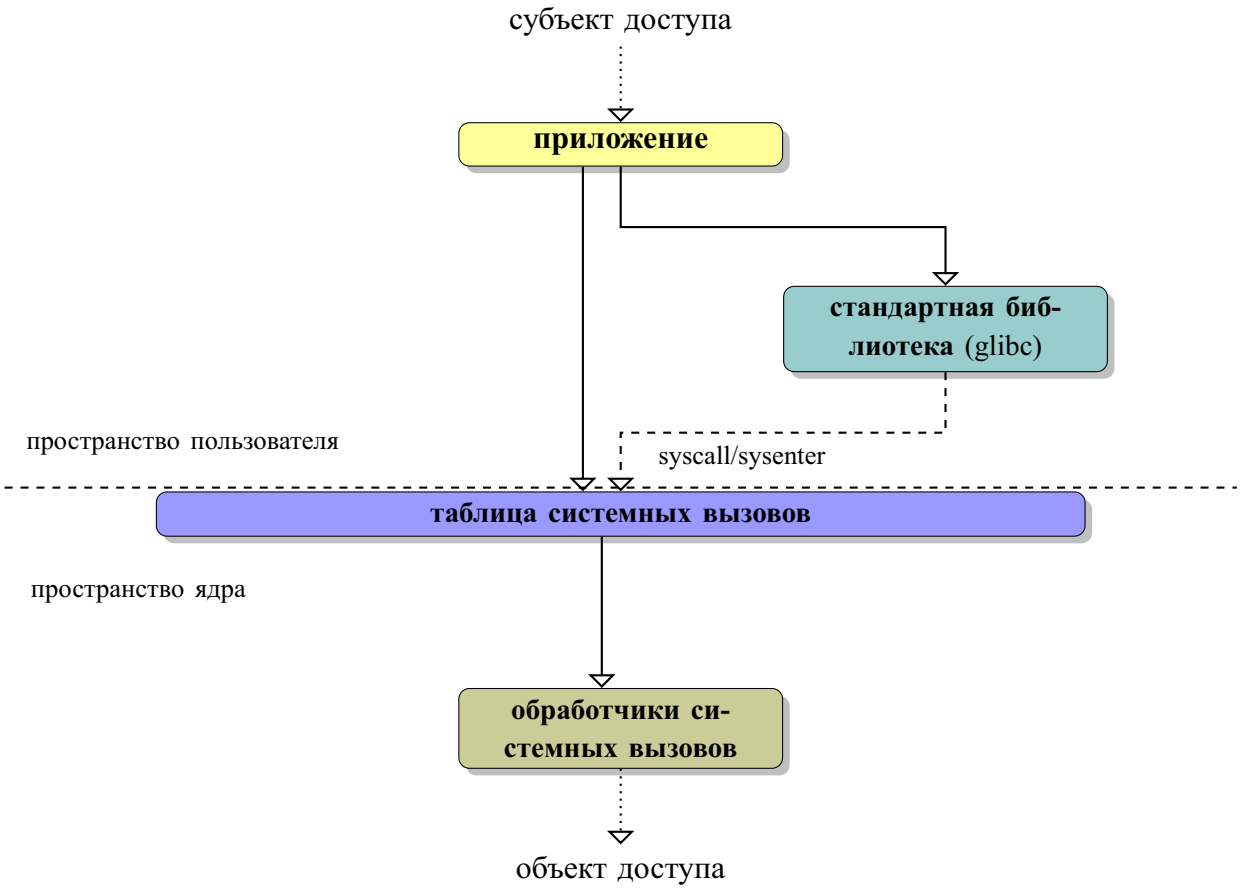

Рисунок 1.2 – Схема организации доступа субъектов к объектам и выполнения других системных вызовов в ОС GNU/Linux

Осуществляя некоторый системный вызов, любой процесс ОС GNU/Linux переходит из пользовательского режима в привилегированный режим (режим ядра). При этом за дальнейшую обработку и выполнение системного вызова отвечают обработчики системных вызовов ядра Linux. После выполнения этих обработчиков процесс возвращается обратно в пользовательский режим, получает код возврата и (в случае наличия) выходные данные.

При осуществлении доступа к данным для любого процесса на уровне обработчиков системных вызовов производится проверка прав текущего субъекта с типом осуществляемого в данный момент доступа. Например, при осуществлении доступа на запись у соответствующего

субъекта доступа (владельца, субъекта из группы владельца или другого субъекта) в метаданных объекта доступа должно быть право «w» (право на запись). В противном случае, обработчик системного вызова вернет код ошибки и не выполнит фактической записи данных в указанный объект доступа. Таким образом, возможность доступа определенного процесса (субъекта доступа) ОС GNU/Linux к объекту (файлу в ФС) определяется в зависимости от текущих прав субъекта доступа (процесса) и атрибутов доступа объекта.

Указанный механизм организации доступа субъектов к объектам свойственен не только ОС GNU/Linux, но и всем Unix-подобным ОС, которые соответствуют стандартам POSIX, SUS [91] или LSB [100]. Все такие ОС обладают одинаковым набором системных вызовов, в связи с чем их функционирование практически идентично с точки зрения использования.

Применительно к ОС GNU/Linux системные вызовы являются наиболее «стабильным» неизменяемым компонентом ОС. Для соответствия POSIX/SUS существует определенный набор обязательных системных вызовов (подробнее в [91], разд. 3 «System Interfaces»), однако в своем развитии в ядре Linux иногда появляются новые системные вызовы, при этом сохраняется обратная совместимость со старыми версиями ядра ОС.

1.1.2. Анализ встроенных средств и механизмов защиты информации от несанкционированного доступа в ОС GNU/Linux

По умолчанию контроль доступа к объектам ОС GNU/Linux может осуществляться встроенными механизмами разграничения доступа ОС, например механизмом дискреционного разграничения доступа (англ. discretionary access control, DAC) с помощью атрибутов файловых объектов; процедурами идентификации и аутентификации пользователей с помощью PAM; использованием расширенных атрибутов доступа или списков контроля доступа (англ. access control lists, ACL).

В качестве основы для всех механизмов защиты информации в GNU/Linux существует стандартный механизм идентификации и аутентификации пользователей, с помощью которого создаваемым пользовательским процессам присваивается корректное значение UID (соответствующее зарегистрированному пользователю) – PAM. Именно в результате корректной работы подсистемы идентификации и аутентификации возможно дальнейшее управление доступом пользовательских процессов к объектам ОС GNU/Linux.

PAM является основной системой идентификации и аутентификации в ОС семейства Unix (в том числе и GNU/Linux). Фактически PAM – это набор внешних модулей идентификации и аутентификации, которые можно встроить в любые приложения, при этом в рамках самих приложений нет необходимости беспокоиться об идентификации или аутентификации пользователя, достаточно использовать соответствующий PAM-модуль [20].

Для контроля доступа в ОС GNU/Linux по умолчанию используется стандартный механизм дискреционного управления доступом. Данный механизм реализуется с помощью атрибутов доступа (чтение, запись или исполнение) для объектов ОС, хранящихся в индексном дескрипторе (описателе или метаданных файла), при этом права доступа в виде атрибутов отдельно задаются

для трех типов субъектов: права владельца файла; права группы, владеющей файлом; права для остальных субъектов доступа.

В рамках работы встроенного механизма DAC необходимо выделить такое понятие как владелец объекта ОС GNU/Linux. Все объекты в ОС GNU/Linux имеют двух владельцев: пользователя и группу пользователей, при этом одной из особенностей понятия «владения» является то, что владелец-пользователь объекта, в общем случае, может не состоять в соответствующей владельце-группе пользователей. В отношении создаваемых объектов существует еще одна особенность «владения», в соответствии с которой владельцем-пользователем всегда назначается субъект доступа, создавший объект, а владельцем-группой может назначаться как первичная группа этого пользователя, так и владелец-группа каталога, в котором создается новый объект [54].

Как правило, владельца-пользователя объекта доступа может изменить только суперпользователь root, но иногда и сам владелец. Владельца-группу может изменить суперпользователь или владелец-пользователь в случае, если он сам является членом новой группы-владельца. Права доступа на объекты в GNU/Linux могут быть изменены только владельцем объекта или суперпользователем.

В отношении прав доступа для объектов ОС GNU/Linux существуют следующие особенности [54]:

– для запуска бинарного файла на выполнение (при наличии атрибута *x*) дополнительно требуется возможность чтения (атрибут *r*) для считывания команд из объекта доступа;

– для каталогов атрибуты доступа имеют несколько другой смысл, а именно:

– право чтения для каталога позволяет получать имена содержащихся в нем файлов;

– право на выполнение для каталога позволяет получать дополнительные данные о содержащихся в нем файлах;

– для перехода в каталог необходимо иметь доступ на исполнение к нему и всем каталогам на пути к этому целевому каталогу;

– для создания или удаления файлов в каталоге необходимо наличие права на запись в каталог (этого достаточно для удаления даже в случае отсутствия прав владения на удаляемые объекты);

– для символьных ссылок атрибуты доступа в целом не используются, так как используются атрибуты доступа целевого файла, а не его файла-ссылки.

При получении любого доступа к объекту в GNU/Linux последовательно выполняются следующие проверки [54]:

1. Если доступ запрашивается суперпользователем, то доступ разрешается и дополнительные проверки не производятся.

2. Если доступ запрашивается владельцем объекта, то доступ разрешается в случае наличия у объекта соответствующего атрибута для пользователя-владельца, иначе доступ запрещается.

3. Если доступ запрашивается пользователем, состоящим в группе-владельца объекта, то доступ разрешается в случае наличия у объекта соответствующего атрибута для этой группы-владельца, иначе доступ запрещается.

4. Доступ разрешается, если требуемое право доступа для прочих пользователей установлено, иначе доступ запрещается.

Необходимо отметить, что перечисленные проверки проводятся строго последовательно, значит, для пользователя-владельца права доступа определяются исключительно атрибутами доступа объекта (для пользователя) даже в случае, если этот же пользователь состоит в группе-владельце.

Кроме основных атрибутов доступа к файловым объектам в GNU/Linux также существуют дополнительные атрибуты, влияющие на механизмы разграничения доступа, а именно [54]:

– Sticky bit – защищает файлы каталога от удаления пользователем с правами на запись (к самим файлам пользователь не имеет прав на запись, не является их владельцем);

– Set UID (SUID), Set GID (SGID) – позволяют изменить права пользователя при запуске объекта на выполнение (только на время выполнения и только в отношении этого запущенного объекта и всех его процессов-потомков);

– атрибут блокировки файла.

Необходимо отметить, что установку SUID/SGID атрибутов требуется осуществлять с осторожностью и только на объекты, исключающие возможность запуска произвольных задач и соответствующие повышенным требованиям по безопасности.

Кроме дополнительных атрибутов в GNU/Linux существуют скрытые специфические атрибуты файлов, которые можно задавать глобально для всех пользователей системы в рамках файловых систем ext2, ext3, ext4 и некоторых других. С помощью данных атрибутов появляются такие возможности как защита файлов от удаления или записи любым пользователем ОС GNU/Linux (атрибут *i*, immutable), ограничение доступа к файлу только в режиме «добавления» (атрибут *a*, append only), синхронизация записи изменений файла с носителем информации (атрибут *S*, synchronous updates), сохранение данных файла при удалении для дальнейшей возможности восстановления (атрибут *u*, undeletable), затирание информации при удалении файла (атрибут *s*, secure deletion) и другие.

В дополнение к правам доступа на объекты в ОС GNU/Linux существует механизм, реализующий списки контроля доступа. Эти списки позволяют расширить традиционные права доступа, а также настраивать их «индивидуально» на каждый объект ОС, то есть при задании списков контроля доступа необходимо указывать, какой субъект доступа и каким типом доступа обладает / не обладает на конкретный объект. Существенным минусом списков доступа является невозможность их использования на всех ФС, поддерживаемых GNU/Linux, а также возможность отключения данного механизма на этапе загрузки ОС (например, через параметр ядра Linux *acl_mode* или с помощью опции *no_acl* в */etc/fstab*, учитывающийся при монтировании определенного раздела).

Таким образом, в ОС GNU/Linux существует большое количество встроенных средств и механизмов разграничения доступа к объектам. Ключевым механизмом, который также позволяет обеспечивать возможность реализации других механизмов защиты, является идентификация и аутентификация пользователей с помощью PAM-модулей аутентификации. Большое значение

в рамках разграничения доступа к объектам ОС GNU/Linux имеет право владения объектом, которое в некоторых ситуациях позволяет как изменять права доступа к объектам, так и передавать права владения ими. Использование некоторых дополнительных атрибутов доступа может привести к непредсказуемым последствиям, а для программ с такими атрибутами существуют особые требования безопасности. Для стандартных прав и атрибутов доступа в GNU/Linux характерна строгая последовательность проверки возможности доступа, в соответствии с которой права доступа для владельца, группы-владельца или остальных субъектов рассматриваются друг за другом. В случае невозможности доступа в результате одной проверки, другие права доступа не проверяются, а доступ запрещается.

1.1.3. Анализ расширенных средств защиты информации от несанкционированного доступа в ОС GNU/Linux

Кроме атрибутов и прав доступа для отдельных объектов в ОС GNU/Linux существует механизм, позволяющий выделять пользователям отдельные права на выполнение строго определенных действий в системе (часть прав суперпользователя) – «возможности» (привилегии) или capabilities (POSIX capabilities, далее привилегии). Использование привилегий позволяет, с одной стороны, отказаться от использования SUID-бита и не давать приложениям чрезмерных привилегий в системе, с другой стороны, приложениям становятся доступны те привилегированные действия в системе, которые им действительно необходимы. В качестве «возможностей», устанавливаемых через привилегии выступают очень детализированные действия в системе, например: пропуск этапа проверки атрибутов в DAC, возможность менять владельца объектов доступа, возможность выполнения *setuid()*, возможность выполнения административных действий (монтирование ФС и так далее), возможности смены корневого каталога (при выполнении chroot), загрузки модулей ядра Linux, перезагрузки СВТ и многие другие. Привилегии можно задавать как для отдельных бинарных файлов, так и ограничить системные «возможности» (например, запретить загрузку модулей ядра Linux для всех пользователей в рамках системы) [44, 45, 81].

Дополнительно к возможностям разграничения доступа к объектам в GNU/Linux существует ряд средств для ограничения процессов ОС – cgroups (control groups), пространства имен (англ. namespaces), seccomp (secure computing mode) и другие. Все механизмы реализованы в ядре Linux, при этом изначально cgroups и namespaces используются совместно, а seccomp – как независимое средство изоляции процессов [45].

Cgroups позволяют пользователю ограничивать выделенные процессам ресурсы (процессорное время, системная память, пропускная способность сети и так далее), вводить приоритезацию и учет ресурсов, организовать изоляцию на уровне пространств имен, а также динамически переконфигурировать указанные параметры во время работы ОС GNU/Linux.

Изоляция процессов на уровне пространств имен в ядре Linux (kernel namespaces) является основой виртуализации с использованием возможностей ядра. Цель каждого типа пространств имен – абстрагировать некоторый ресурс от системы, при этом в системе будет существовать набор независимых абстракций (пространств имен) того или иного ресурса. Конкретно для про-

цессов каждое пространство имен расценивается как изолированный и единственный набор процессов в ОС.

Создание изолированных окружений для различных процессов ОС GNU/Linux также возможно с помощью использования *chroot*, изменяя корневой каталог для целевого процесса.

Seccomp позволяет организовать для процесса в определенном смысле «песочницу» (англ. sandbox). Данный механизм переводит процессы в «защищенное» состояние, в котором ему будут доступны для исполнения только системные вызовы *exit* и *sigreturn* (завершения), а также *read* и *write* для уже открытых файловых объектов. При попытке использования «запрещенного» системного вызова в «защищенном» состоянии процесс будет принудительно завершен ядром ОС Linux.

Кроме рассмотренных механизмов разграничения доступа в ОС GNU/Linux существуют дополнительные (часто встраиваемые) средства, позволяющие реализовать, в том числе и механизм мандатного разграничения доступа (англ. mandatory access control – MAC), например: SELinux, AppArmor, Tomoyo и другие. Большинство таких средств входит в состав ОС GNU/Linux опционально, причем по умолчанию может использоваться только одно из них.

SELinux (Security-Enhanced Linux) представляет собой реализацию архитектуры защищенной системы FLASK (Flux Advanced Security Kernel), разработанной Агентством национальной безопасности США [68], и в данный момент применяется в большинстве дистрибутивов GNU/Linux в качестве дополнительного средства защиты информации вместе со стандартными правами доступа к объектам.

В SELinux реализована модель принудительной типизации (англ. type enforcement), которая, в том числе позволяет максимально приблизиться к реализации принципа наименьших привилегий (англ. least privilege principle) [15, 16]. При этом правила безопасности SELinux функционируют независимо от DAC и дополнительных атрибутов объектов доступа, в общем случае в SELinux не существует каких-либо особых привилегий для суперпользователя root или других субъектов.

Для корректного разграничения доступа с помощью SELinux все объекты ОС GNU/Linux помечаются контекстами безопасности. Правила SELinux позволяют разграничивать различные типы доступа как к объектам (файлам), так и к таким сущностям как сокеты для сетевых соединений, POSIX capabilities, загружаемые модули ядра Linux, разделяемая память и так далее. Для указанных сущностей SELinux позволяет настраивать мандатную политику управления доступом в рамках решетки многоуровневой безопасности (англ. Multilevel security, MLS) [13], то есть с помощью иерархических и неиерархических меток безопасности, а также политику ролевого управления доступом и так далее.

AppArmor (Application Armor) – второе по популярности расширенное средство разграничения доступа, предоставляющее возможность настройки аналогичных SELinux функций, но имеющее другой принцип встраивания и способ идентификации объектов доступа. AppArmor реализовано в виде загружаемого модуля, который, начиная с некоторой версии, включен в состав основной ветки ядра Linux. Для указания объектов доступа в AppArmor необходимо задавать абсолютные пути до объектов в рамках файловой системы ОС. С одной стороны, идентифи-

кация объектов доступа по абсолютному пути может быть неоднозначной в разные моменты времени работы ОС GNU/Linux, а также существует возможность неконтролируемого доступа к объекту по другому абсолютному пути с использованием жестких ссылок. С другой стороны, AppArmor, в отличие от SELinux, не зависит от конкретного типа файловых систем и является более простым в настройке средством защиты.

Еще одним расширенным средством разграничения доступа, использующим для идентификации объектов абсолютные пути в файловой системе, является Tomoyo. Отличительной особенностью этого средства является возможность запуска комплекса в режиме, в котором происходит «запись» цепочек запуска тех или иных процессов ОС и дальнейшее формирование на основе этой информации правил, содержащих только разрешенные цепочки для тех или иных приложений. При попытке запуска приложения, которое не будет соответствовать ни одной цепочке запусков, создание нового процесса ОС будет запрещено.

Кроме готовых средств защиты информации существуют также наборы исправлений для GNU/Linux, с помощью которых можно увеличить защищенность ядра ОС или используемого прикладного ПО (англ. hardening patches). Так, например, для ряда дистрибутивов GNU/Linux существуют исправления для следующих относительно независимых компонент:

– средства разработки с расширенными возможностями по усилению защитных свойств конечных бинарных модулей (англ. hardened toolchain) – «инструментарий», компилятор и стандартные утилиты (gcc/glibc/binutils), реализующие, например, защиту от переполнения буфера (англ. stack smashing protector, SSP) с помощью добавления специальных маркеров (англ. «canaries»), изменение которых сигнализирует о выходе за пределы текущего фрейма стека;

– ядро ОС с усиленными защитными свойствами (англ. hardened kernel), содержащее такие компоненты, как PAX (для запрета выполнения кода в страницах памяти с данными, очистки памяти при освобождении и перераспределении, рандомизации адресного пространства), grsecurity (для ограничения потенциально опасных действий, например, mount в chroot-окружении, запуск на выполнение общедоступных на запись или пользовательских бинарных файлов) и другие.

В дополнение к описанным выше средствам для современных СВТ, в состав которых входит модуль Trusted Platform Module (TPM), становится возможным применять такие технологии, как Integrity Measurement Architecture (IMA) и Linux Extended Verification Module (EVM) [31].

IMA представляет собой подсистему контроля целостности в ядре Linux, с помощью которой можно проверять неизменность объектов доступа. При этом эталонные хэш-коды могут храниться как в дополнительных файловых атрибутах в открытом виде, так и защищаться с помощью подписи с использованием TPM. Описанные технологии совместно с BIOS/UEFI для СВТ позволяют соответствовать таким относительно новым открытым стандартам, как Trusted Boot, Trusted Software Stack от Trusted Computing Group и некоторым другим. Однако на данный момент технологии IMA/EVM еще не внедрены в массовое использование.

Таким образом, в GNU/Linux дополнительно к стандартным правам и расширенным атрибутам доступа существует большое количество разнообразных средств и механизмов защиты, при применении которых становится возможным повысить уровень безопасности ОС и ее объектов доступа.

1.1.4. Анализ средств и механизмов защиты информации от несанкционированного доступа в ОС GNU/Linux с точки зрения их соответствия требованиям нормативных документов РФ

В качестве средств для GNU/Linux, которые соответствуют требованиям нормативных документов РФ, можно рассматривать как дистрибутивы GNU/Linux со встроенными сертифицированными средствами защиты информации от несанкционированного доступа (СЗИ НСД), так и отдельные, независимые от ОС сертифицированные средства защиты данных.

В качестве дистрибутивов GNU/Linux, встроенные программные средства защиты которых сертифицированы по требованиям ФСТЭК России (Министерства обороны РФ), можно выделить Astra Linux (версия Special Edition), Alt Linux (СПТ), Rosa Linux (Никель, Кобальт и Хром), МСВС/МСВСфера и некоторые другие.

Необходимо отметить, что Astra Linux представляет собой не только автономное решение в виде ОС GNU/Linux, но и целый комплекс с соответствующими средствами защиты для систем управления базами данных (СУБД), средств организации доменов LDAP, почтового сервера Exim/Dovecot, web-сервера и так далее [60]. Кроме того, в Astra Linux реализованы дискреционный и мандатный механизмы разграничения доступа для всех входящих в ОС приложений и используемых файловых систем; очистка оперативной и внешней памяти, гарантированное удаление файлов; маркировка печати для сервера печати CUPS; подсистема протоколирования, осуществляющая надежную регистрацию событий безопасности во всех приложениях ОС; контроль неизменности и подлинности исполняемых файлов в формате ELF (на основе электронной подписи по ГОСТ Р 34.10-2001/2012); контроль целостности (на основе хэширования по ГОСТ Р 34.11-94/2012); возможность применения аппаратно-программного модуля доверенной загрузки (АПМДЗ «Максим-М1»). Astra Linux имеет сертификаты ФСТЭК России (на соответствие 3 классу СВТ и 2 уровню контроля отсутствия недекларированных возможностей), ФСБ России и Министерства обороны РФ, в соответствии с которыми может использоваться для защиты как конфиденциальной информации и персональных данных, так и сведений, составляющих государственную тайну.

Практически аналогичные возможности предоставляют дистрибутивы Alt Linux (СПТ), Rosa Linux (Никель, Кобальт и Хром) и МСВС/МСВСфера, кроме контроля подлинности исполняемых файлов. Однако они являются скорее автономными решениями и не предлагают решений для защиты других систем, кроме непосредственно ОС и данных в среде этой ОС. Все решения имеют сертификаты ФСТЭК на соответствие 4 классу СВТ и 3 или 4 уровню контроля отсутствия недекларированных возможностей. МСВС также имеет сертификаты Министерства обороны РФ.

В качестве сертифицированного встраиваемого СЗИ НСД для защиты ОС GNU/Linux можно отметить Secret Net LSP, обладающее следующими возможностями [63]: усиленные процедуры идентификации и аутентификации пользователей GNU/Linux; дискреционное разграничение доступа пользователей и групп к объектам файловой системы и внешним устройствам; контроль целостности компонент СЗИ НСД и объектов файловой системы GNU/Linux; регистрация событий безопасности в журналах СЗИ НСД; уничтожение содержимого файлов при их удалении,

очистка освобождаемых областей оперативной памяти СВТ и запоминающих устройств; контроль печати с возможностью «теневого» копирования и маркировки документов, разграничения доступа к принтерам; возможность дополнительно обеспечивать доверенную загрузку ОС Linux с помощью аппаратного модуля доверенной загрузки (АМДЗ) «Соболь».

Таким образом, Secret Net LSP, являясь встраиваемым в ОС GNU/Linux средством, предоставляет сравнимые с описанными выше сертифицированными дистрибутивами Linux возможности, кроме реализации мандатного управления доступом. Secret Net LSP обладает сертификатом ФСТЭК России на соответствие 5 классу СВТ, 4 уровню контроля отсутствия недекларированных возможностей. При этом данное средство обладает таким преимуществом по сравнению со специализированными дистрибутивами как теоретическая возможность его использования на большом количестве разнообразных дистрибутивов GNU/Linux (в случае реализации поддержки производителем). Приведенные выше специализированные дистрибутивы Linux требуют перехода с уже внедренных в организации ОС на их собственные версии. При этом выпуск обновлений прикладного и системного ПО в таких ОС может идти с значительным запаздыванием, что может сказаться на защищенности систем с использованием этих дистрибутивов.

1.1.5. Оценка полноты принципов функционирования, механизмов и алгоритмов защиты информации от несанкционированного доступа, реализованных в ОС GNU/Linux

На основе проведенного ранее анализа функциональных возможностей различных стандартных и расширенных средств защиты информации от НСД в ОС GNU/Linux можно выделить некоторые их общие принципы функционирования, особенности используемых защитных механизмов и недостатки, представленные в таблице 1.2.

Таблица 1.2 – Сравнение принципов и особенностей функционирования существующих средств защиты от НСД для ОС GNU/Linux

Характеристики СЗИ НСД	**DAC/ACL, CAPS**	**SELinux**	**Tomoyo, AppArmor**	**TPM, IMA/EVM**	**Alt / Astra / Rosa Linux, МСВС**	**Secret Net LSP**
1. Тип СЗИ НСД (встроенное, встраиваемое)	встроенное	встраив-е, в составе ядра	встраив-е, в составе ядра	встраив-е, частично аппаратная реализация	встроенные подсистемы разграничения доступа	встраив-е
2. Возможности по разграничению доступа, контролю целостности	дискр. и ролевое управление доступом	мандатное управление доступом	мандатное управление доступом	только предварительный контроль целостности	дискр., мандатное (только в Astra Linux) управление доступом	дискр. управление доступом и контроль целостности при загрузке ОС

Продолжение таблицы 1.2

Характеристики СЗИ НСД	**DAC/ACL, CAPS**	**SELinux**	**Tomoyo, AppArmor**	**TPM, IMA/EVM**	**Alt / Astra / Rosa Linux, MCBC**	**Secret Net LSP**
3. Зависимость от компонент / конфигурации ОС	от типа ФС	от типа ФС	нет	нет, кроме необходимости поддержки в ядре и СВТ	от типа ФС	от типа ФС
4. Возможность ограничения всех субъектов системы	нельзя ограничить суперпользователя	в полном объеме	в полном объеме для сервисов	нет, не участвует в разграничении доступа	нельзя ограничить суперпользователя	нельзя ограничить суперпользователя
5. Способ идентификации объектов доступа	по inode	по inode и специальным метаданным	по абсолютным путям	IMA – по абсолютным путям, EVM – по inode и метаданным	по inode	по inode
6. Способ хранения прав доступа	в объекте доступа	в дополнительных метаданных	в отдельной файле	в аппаратном устройстве или в метаданных объекта доступа	в объекте доступа и отдельном файле	в объекте доступа и отдельном файле
7. Схема администрирования (распределенная, централизованная), распространение прав доступа	распред-я, распространяет владелец или администратор	распред-я, распространяет администратор	централиз-я, распространяет администратор	скорее централиз-я, нет	распред-я, распространяет владелец или администратор	централиз-я, распространяет владелец или администратор
8. Возможность отключения / исключения активизации	да, с изменением параметров загрузки или при физическом доступе	аналогично DAC	аналогично DAC	только IMA/EVM	аналогично DAC	аналогично DAC
9. Возможность сочетания с другими средствами защиты	возможно, уже сочетаются между собой	невозможно, только частные случаи	невозможно, только частные случаи	да, т.к. не участвует в разграничении доступа	в общем случае невозможно	возможно

Наиболее важными отличиями между различными встроенными и расширенными средствами защиты от НСД в GNU/Linux являются способ идентификации объектов доступа (объектов файловой системы) и способ хранения прав доступа [23]. Так, различные атрибуты хранятся непосредственно в метаданных файловых объектов, таким образом, для работы стандартного

дискреционного механизма разграничения доступа не требуется искать правила для объекта, к которому осуществляется доступ, – эти правила содержатся непосредственно в самом объекте доступа и фактически являются «неотделимыми» от защищаемых данных. Такая «привязка» атрибутов доступа непосредственно к защищаемым данным является логичной и правильной – защищаемые данные однозначно идентифицированы, а атрибуты доступа, содержащиеся в метаданных, соответствуют вполне определенным блокам с данными на носителе информации.

Расширенные средства разграничения доступа, такие как SELinux, AppArmor и Tomoyo в этом контексте работают несколько иначе. SELinux для хранения атрибутов объектов доступа использует расширенные атрибуты в нативных файловых системах GNU/Linux. Данное решение, аналогично стандартным атрибутам доступа, позволяет отказаться от задачи поиска соответствующих прав для объекта, к которому реализуется доступ, однако при этом накладывается ограничение на возможность использования механизмов защиты только для файловых систем, поддерживающих создание необходимых расширенных атрибутов.

Такие средства как AppArmor и Tomoyo для описания прав доступа используют некоторую таблицу (матрицу), в которой указываются объекты (по абсолютному пути) и права доступа существующих субъектов доступа. При реализации определенного доступа необходимо найти в таблице соответствующий объект доступа и проверить права доступа к нему. Недостаток такого решения описан ранее в разделе 1.1.1 и заключается в том, что в общем случае нельзя однозначно идентифицировать объекты доступа GNU/Linux по абсолютному пути. AppArmor и Tomoyo нельзя назвать системами, в которых субъектам назначаются минимальные привилегии, но зато они позволяют настраивать права доступа в централизованной схеме администрирования [72] и более простые в настройке.

Для стандартных механизмов защиты ОС GNU/Linux характерна распределенная схема администрирования, в которой права доступа для объектов могут назначаться самими пользователями (владельцами файловых объектов). С точки зрения защиты, например, конфиденциальной информации, право владения, его наследование при создании файловых объектов и передача другим пользователям, а также возможность изменения пользователями прав доступа могут быть не совсем корректны (владельцем обычно является юридическое лицо) [72]. В связи с этим желательно исключить возможность владения и передачи прав доступа пользователями, а также использовать централизованную схему администрирования, в которой права доступа задаются ответственным представителем организации (администратором безопасности).

Некоторые встроенные механизмы защиты ОС GNU/Linux не позволяют использовать правила разграничения доступа в отношении всех пользователей системы (как правило, суперпользователь root имеет неограниченные права в ОС), что может быть угрозой безопасности информации при использовании нарушителем. Для минимизации ущерба в случае компрометации привилегированных пользователей ОС, необходимо принудительно ограничивать права абсолютно всех субъектов системы, реализуя для них принцип наименьших привилегий.

Принципиально существует два подхода реализации СЗИ НСД: реализация в виде встроенного и встраиваемого средства. Каждый из этих подходов имеет свои сильные и слабые стороны, определяющие границы его применимости. Встроенные средства характеризуются изначальным

присутствием в системе, начиная со старта ОС. Встраиваемые СЗИ НСД необходимо внедрять в защищаемую систему в процессе ее работы (желательно на достаточно раннем этапе работы). При этом множество встроенных СЗИ НСД имеют ограничения по использованию только с определенными файловыми системами или версиями ядра Linux, в то же время встраиваемые СЗИ НСД разработчики изначально проектируют для встраивания в большое количество различных версий систем.

Общим недостатком для всех рассмотренных средств разграничения доступа является то, что при наличии физического доступа к СВТ или носителю информации, на котором размещаются защищаемые данные, такие средства достаточно просто либо отключить на раннем этапе загрузки ОС GNU/Linux, либо исключить их активизацию. Даже при безопасной настройке загрузчиков Linux известны актуальные даже на сегодняшний день уязвимости[7,8], которые позволяют несанкционированным образом вмешаться в процесс загрузки и, например, загрузиться с другого носителя информации, либо изменить параметры загрузки ядра ОС [38], в обоих случаях минуя установленные средства защиты информации. Другим типом известных уязвимостей является возможность влияния на средства защиты и связанные с ними объекты в процессе работы. Так, например, для SELinux известны уязвимости[9,10], которые позволяют непривилегированному пользователю несанкционированно изменить атрибуты объекта доступа с возможностью дальнейшего НСД к данным. Кроме того, известны уязвимости[11] в Policykit/Polkit, позволяющие локально повышать привилегии в системе.

Также многие средства защиты уделяют внимание только вопросам разграничения доступа, при этом частично опуская вопросы контроля целостности (целостность обеспечивается только с помощью разграничения доступа, а не с использованием периодических процедур контроля).

Необходимо отметить еще один важный аспект: все рассмотренные средства защиты информации от НСД в GNU/Linux применяются в некоторой совокупности, не существует возможности использовать только одно из них. Так, например, дискреционный механизм разграничения доступа на основе файловых атрибутов является обязательным и обычно сочетается с capabilities, расширенными файловыми атрибутами и, возможно, другими СЗИ НСД. При этом возникают сложности в администрировании такого сочетания СЗИ НСД и в анализе защищенности системы в целом. Однако при этом некоторые средства защиты нельзя использовать совместно, так как они полностью или частично дублируют свои функции относительно друг друга (например, SELinux и AppArmor), что вносит еще большую путаницу. Таким образом, при использовании большого количества существующих СЗИ НСД отсутствует понимание того, какой состав таких средств необходим для защиты информации от НСД в ОС GNU/Linux (в том числе с точки зрения контроля всех возможных типов доступа), а также насколько корректно реализованные механизмы защиты будут взаимодействовать между собой.

[7] https://nvd.nist.gov/vuln/detail/CVE-2009-4128.
[8] https://nvd.nist.gov/vuln/detail/CVE-2015-8370.
[9] https://nvd.nist.gov/vuln/detail/CVE-2018-1063.
[10] https://nvd.nist.gov/vuln/detail/CVE-2021-23240.
[11] https://nvd.nist.gov/vuln/detail/CVE-2021-4034

1.2. Анализ требований действующих нормативно-правовых документов к средствам защиты информации от несанкционированного доступа

В Российской Федерации существует ряд нормативных документов [56, 57], регламентирующих процессы разработки, внедрения и использования средств защиты информации или программно-аппаратных комплексов средств защиты информации (ПАК СЗИ), в том числе и СЗИ от НСД (ПАК СЗИ НСД) для различных категорий данных. В соответствии с этими нормативными документами контроль за разработкой, внедрением и использованием СЗИ возложен на две основные государственные структуры РФ – ФСТЭК и ФСБ России. ФСБ ответственна в основном за контроль средств криптографической защиты информации, в то время как ФСТЭК России – за контроль всех остальных СЗИ. Однако существует ряд закрытых требований ФСБ и к средствам защиты информации от НСД, например, требования к аппаратно-программным модулям доверенной загрузки СВТ и требования к защищенности автоматизированных систем (АС) с учетом различных типов нарушителей.

Разработка средств защиты информации для конфиденциальных или персональных данных, как правило, сопровождается сертификацией этих средств в государственной системе сертификации СЗИ. Одной из главных целей сертификации является удостоверение соответствия третьей (независимой) стороной заявленных защитных механизмов, реализуемых в СЗИ, действующим требованиям в области защиты информации.

Для СЗИ НСД существует ряд регламентирующих документов ФСТЭК России, в соответствии с которыми эти средства принято классифицировать в зависимости от степени конфиденциальности информации, которую можно с помощью них защищать (и, например, в каком классе АС можно использовать СЗИ НСД). При этом существуют отдельные требования для защиты СВТ (ОС, СУБД и так далее) [8] и требования для защиты АС [6], которые неформально зависимы по соответствующим классам защиты.

Основные нормативные документы РФ в области защиты информации приведены в таблице 1.3 и будут подробно разобраны ниже.

Таблица 1.3 – Основные нормативные документы РФ в области защиты информации и их краткая характеристика применительно к защите данных от НСД

Наименование документа	**Тип защищаемых данных**	**Требования к компонентам СЗИ НСД**	**Классы защиты**
1. РД ФСТЭК. АС. Защита от несанкционированного доступа к информации. Классификация АС и требования по защите информации	конф. информация, гос. тайна	к подсистемам управления доступом, идентификации и аутентификации, регистрации и учета, очистки оперативной памяти и остаточной информации, контроля печати, обеспечения целостности	для многопользовательских систем с разными правами доступа субъектов – классы защищенности АС 1Д–1А по мере усиления требований

Продолжение таблицы 1.3

Наименование документа	**Тип защищаемых данных**	**Требования к компонентам СЗИ НСД**	**Классы защиты**
2. РД ФСТЭК. СВТ. Защита от несанкционированного доступа к информации. Показатели защищенности от несанкционированного доступа к информации	конф. информация, гос. тайна	к подсистемам управления доступом (дискреционная и мандатная политики), идентификации и аутентификации, очистки памяти, изоляции модулей, маркировки документов, защиты ввода-вывода на внешние носители данных, регистрации и учета, контроля целостности	классы защищенности СВТ 6–1 по мере усиления требований (частично повторяют требования классов защищенности АС 1Д–1А)
3. Приказ ФСТЭК России № 17	конфиденциальные данные в государственных ИС	к идентификации и аутентификации (ИАФ), управлению доступом (УПД), ограничению программной среды (ОПС), защите машинных носителей информации (ЗНИ), регистрации событий безопасности (РСБ), обеспечению целостности (ОЦЛ) и доступности (ОДТ), защите среды виртуализации (ЗСВ) и др.	классы защищенности государственных ИС 3–1 по мере усиления требований
4. Приказ ФСТЭК России № 21	персональные данные	аналогичные Приказу № 17 группы требований, а также требования к анализу защищенности персональных данных (АНЗ), выявлению инцидентов и реагированию на них (ИНЦ), управлению конфигурацией системы защиты (УКФ)	уровни защищенности персональных данных 4–1 по мере усиления требований
5. Методические документы ФСТЭК России. Профили защиты операционных систем (типа «А», «Б», «В»)	гос. тайна, данные государственных ИС и систем персональных данных различных классов	аналогичные Приказу № 17 группы требований, а также требования к изоляции процессов и защите памяти, доверенной загрузке и др.; отдельные требования для защиты компонент ОС и модулей ядра, ограничения установки ПО из недоверенных источников, разработки формальной модели безопасности для высоких классов, к функциям ОС реального времени; часть требований унаследована из [4]	классы защиты ОС 6–1 по мере усиления требований для различных типов систем (А – ОС общего назначения, Б – встроенные ОС, В – ОС реального времени)

В Руководящем документе (РД) для АС [6] выделяют 3 класса систем: однопользовательская, в которой пользователь имеет доступ ко всей информации (классы 3Б–3А); многопользовательская, в которой пользователи имеют одинаковый уровень доступа к информации (классы 2Б–2А); многопользовательская, в которой не все пользователи имеют доступ к информации то-

го или иного уровня конфиденциальности (классы 1Д–1А – большинство современных систем относится к этим классам).

Кроме Руководящих документов, в которых требования сформулированы для защиты конфиденциальной информации и сведений, составляющих государственную тайну, начиная с 26 января 2007 года после вступления в силу Федерального закона № 152-ФЗ «О персональных данных» в РФ определена необходимость защиты персональных данных (ПДн), для которых в дальнейшем были утверждены отдельные требования защиты. Нормативные акты в области защиты ПДн достаточно часто претерпевают изменения, на момент написания диссертационной работы актуальными являются следующие документы:

– Постановление Правительства РФ № 1119 от 1 ноября 2012 г. «Об утверждении требований к защите персональных данных при их обработке в информационных системах персональных данных»;

– Приказ ФСТЭК России № 21 от 18 февраля 2013 г. «Об утверждении состава и содержания организационных и технических мер по обеспечению безопасности персональных данных при их обработке в информационных системах персональных данных» [58];

– Приказ ФСТЭК России № 17 от 11 февраля 2013 г. «Об утверждении требований по защите информации, не составляющей государственную тайну, содержащейся в государственных информационных системах» [59];

– Дополнительные нормативные документы по построению модели угроз безопасности ПДн, рекомендациям по использованию криптосредств для защиты ПДн и другие.

Приказы ФСТЭК России № 17 и 21 более подробно описывают требования по защите для разных классов информационных систем ПДн (ИСПДн).

Также с 1 июня 2017 г. приказом ФСТЭК России № 119 от 19 августа 2016 г.[12] введены «Требования безопасности информации к операционным системам» [49], в которых выделены 3 типа и 6 классов ОС каждого типа. Данные требования актуальны для ОС со встроенными средствами защиты информации и не применяются по отношению к встраиваемым средствам.

Несмотря на то, что формально нормативные документы в области защиты информации четко разделяются для систем (АС, ИСПДн) и СВТ (ОС СВТ), сформулированные в них требования часто дублируются, а на настоящий момент целесообразнее даже для отдельно взятой ОС (СЗИ НСД, используемого в ОС) учитывать в том числе и требования к системам. Связано это с тем, что современные ОС можно рассматривать как часть или целую АС (в зависимости от уровня абстракции), в них реализован многопользовательский режим работы (что определено только в РД АС), могут функционировать локальные компоненты АС (СУБД, web/файловые сервера и так далее), а отдельно взятое СВТ может участвовать в построении АС. В этом плане нормативные документы РФ являются в некоторой степени устаревшими, их требования часто сформулированы до появления современных ИС (ОС) и не учитывают природу функционирующих в них субъектов и обрабатываемых объектов доступа.

Не вдаваясь в подробности по классификации ПДн, ИСПДн или защищаемой конфиденциальной информации в АС (СВТ, ОС), разработку моделей нарушителя и проектирование систе-

[12]Информационное сообщение ФСТЭК России от 18 октября 2016 г. № 240/24/4893.

мы защиты, опишем обобщенные требованиях к АС (СВТ, ОС) и ИСПДн в соответствии с РД ФСТЭК [6, 8], требованиями нормативных документов по защите ПДн и ОС [49], выделив при этом следующие группы механизмов защиты:

– механизмы управления доступом, а именно:

– идентификация и аутентификация всех субъектов доступа с ограничением неуспешных попыток прохождения этих процедур;

– контроль и управление доступом субъектов к объектам (с помощью дискреционной, мандатной или другой политики управления доступом для необходимых типов доступа);

– назначение минимально необходимых прав и привилегий субъектам доступа (принцип минимальных привилегий, описанный ранее в [109]);

– ограничение числа параллельных сеансов доступа субъекта (УПД.9, Приказ № 21);

– управление потоками информации с помощью меток или атрибутов безопасности (например, для классов АС 2А, 1В, 1Б и 1А) с поддержкой и сохранением этих меток в процессе работы (УПД.12);

– ограничение программной среды (ограничение возможности установки новых компонент ПО, запрет / разрешение / перенаправление записи во временные файлы, удаление временных файлов);

– изоляция процессов.

– механизмы регистрации и учета (регистрация событий безопасности и защита информации об этих событиях);

– механизмы криптографической защиты;

– механизмы контроля целостности (ПО, СЗИ НСД и защищаемой информации, в том числе с помощью обеспечения доверенной загрузки СВТ);

– дополнительные механизмы и меры защиты (очистка памяти и остаточной информации, маркировка документов, антивирусная защита, обнаружение вторжений, анализ защищенности АС, защита отчуждаемых носителей информации, защита среды виртуализации, защита сетевого взаимодействия, защита технических средств, обеспечение доступности информации).

Причем механизмы управления доступом в приведенном списке являются основополагающими для остальных механизмов защиты, так как именно они отвечают за противодействие НСД (то есть являются средствами превентивной защиты). Остальные группы механизмов в основном реализуются для дополнительной защиты в случае осуществления НСД, пропущенного механизмами управления доступом.

Отдельно стоит выделить такие требования по защите информации, как регистрация и учет выдачи печатных (графических) документов (то есть, контроль и маркировка печати) и очистка памяти [21, 37], поскольку они специфичны для РФ и часто не настолько подробно учтены в требованиях по защите информации других государств [84, 109].

Требование по маркировке печати обязательно для выполнения в АС классов 3А, 2А, 1Г–1А, а также в ИСПДн в соответствии с нормативными документами ФСТЭК [58, 59]. При этом модуль контроля печати должен входить в состав подсистемы управления доступом (в которой реализованы прочие защитные функции) с целью принципиальной возможности реализации такого

модуля (для определения соответствия уровня конфиденциальности документа) и возможности интеграции всех событий безопасности в единую систему регистрации и учета.

Также для классов АС 3А, 2А, 1Г–1А (и для классов СВТ 1–4) и некоторых уровней защищенности ПДн должна осуществляться очистка (обнуление, обезличивание) освобождаемых областей оперативной памяти и внешних накопителей (с двукратной произвольной записью в освобождаемую область памяти). При этом выделяются отдельно требования по очистке остаточной информации при удалении данных с носителей информации и требования по очистке оперативной памяти при ее освобождении или перераспределении. В ОС GNU/Linux вследствие изолированности адресного пространства каждого процесса требование по очистке оперативной памяти носит скорее дополнительный характер, так как получить доступ к памяти другого процесса ОС в общем случае затруднительно, даже обладая административными привилегиями. Требование же по очистке данных с носителей информации при удалении объектов доступа в достаточной степени обоснованно, так как при удалении объекта в любой файловой системе данные остаются в неизменном виде в выделенных этому объекту секторах (кластерах ФС) и будут затерты только при записи новых объектов ОС (до этого момента потенциально возможен несанкционированный доступ к остаточной информации).

В современных ОС (в том числе в ОС GNU/Linux) не выполняются все требования даже для низких классов защищенности АС, СВТ, ПДн из нормативных документов РФ. Чаще всего не реализованы мандатный механизм управления доступом, очистка памяти, контроль печати и некоторые другие. Дискреционная модель управления доступом в ОС GNU/Linux реализована не в полном объеме (невозможно ограничить права суперпользователя). Контроль целостности в ОС GNU/Linux возможен при использовании дополнительных средств, как правило доступно выполнение контроля целостности по расписанию и в основном для конфигурационных файлов (хотя требуется его запуск непосредственно перед осуществлением доступа, в том числе и для пользовательских данных). Ряд классов защищенности АС (3А, 2А, 1В-1А) требует использования для защиты информации только сертифицированных СЗИ НСД, стандартные средства ОС при этом не сертифицированы. Также в большинстве современных систем не может быть выполнено требование для ОС классов защиты 1–3 о необходимости разработки и предоставления формальной модели политики безопасности (текущие реализации управления доступом в ОС и требования нормативных документов не соответствуют ни одной формальной модели безопасности).

Таким образом, для средств защиты информации (в том числе для СЗИ НСД) существуют необходимые, формализованные меры и требования безопасности, регламентированные требованиями нормативных актов РФ в области защиты информации. Применительно к ОС GNU/Linux (так как эти ОС являются многопользовательскими с различными правами доступа) минимальными необходимыми требованиями для защиты конфиденциальной информации являются требования класса АС 1Г (5 класс СВТ, 4 класс ОС), для гостайны – 1В (3 класс СВТ и ОС). Описанные меры и требования носят достаточно общий характер, в них не в полной мере предусматривается классификация объектов защиты [72], они едины для всех семейств ОС и множества различных информационных систем, в них не дается рекомендаций по способам построения и

администрирования систем защиты. В результате в формализованных требованиях по защите информации приводятся лишь основополагающие, а не конкретизированные механизмы защиты информации.

Так, например, в требованиях не описано то, каким образом необходимо встраивать механизмы защиты в АС или СВТ, а также как при этом учитываются встроенные механизмы защиты ОС, СУБД и других систем, в том числе при использовании встроенных механизмов в дополнительных СЗИ НСД. Также в требованиях не указано каким образом необходимо защищать атрибуты доступа, ассоциированные с объектами, и не указаны необходимые и достаточные атрибуты для защиты всех возможных типов доступа к этим объектам. Кроме того, достаточно абстрактно определены сами понятия субъекта и объекта доступа, способы их идентификации (однозначного определения, что доступ осуществляет конкретный субъект и именно к конкретному объекту), требования четко не конкретизируют то, к чему должен разграничиваться доступ (только к защищаемым данным или к данным всей АС). Если доступ требуется разграничивать только к защищаемым данным, то не складывается понимания того, как определить, что в том или ином объекте, например, ОС содержится защищаемая информация. При этом, например, требования по обеспечению доверенной загрузки СВТ описаны в достаточной степени конкретно. Для СЗИ НСД, реализующих функции доверенной загрузки, с 1 января 2014 года введены отдельные требования, утвержденные приказом ФСТЭК России от 27 сентября 2013 г. № 119. Требования описаны в виде профилей защиты средств доверенной загрузки уровня загрузочной записи, платы расширения и базовой системы ввода-вывода для классов защиты 1–6.

Таким образом, могут быть сделаны выводы о том, что существующие требования нормативных документов РФ в области защиты информации не уделяют достаточного внимания особенностям современных СВТ и систем, в том числе и ОС GNU/Linux, а также не формируют полноценного представления о способах построения подсистемы управления доступом. В соответствии с этим существует необходимость формирования уточненных и конкретизированных требований для средств защиты информации от НСД в среде ОС GNU/Linux, которые бы устраняли все приведенные недостатки, для чего требуется провести дополнительные исследования.

1.3. Анализ результатов исследований, посвященных совершенствованию алгоритмов разграничения доступа в ОС GNU/Linux, и обоснование необходимости их модернизации

В разное время ощутимый вклад в развитие теории и практики безопасности сложных ИС, защиты технических, программных и информационных ресурсов внесли такие отечественные ученые как В. А. Герасименко, В. П. Лось и В. А. Конявский.

В. А. Герасименко и В. П. Лось в своих работах подробно освещают вопросы теории и практики обеспечения безопасности ИС, средств их защиты, а также требования к защите информации в соответствии с нормативными документами РФ. Большой вклад в развитие средств технической защиты информации, в том числе в системах электронного документооборота, а также в развитие теории доверенных вычислительных систем и безопасного взаимодействия в них внес В. А. Конявский.

Развитию же теоретических основ компьютерной безопасности в части формальных моделей безопасности компьютерных систем и их практического использования посвящены работы следующих ученых: М. Харрисон, В. Руззо, Дж. Ульман, Д. Белл, Л. ЛаПадула, Р. Сандху, Д. Феррайоло, Р. Кун, Д. Деннинг, Дж. МакЛин, П. Н. Девянин, Д. П. Зегжда и А. Ю. Щербаков.

М. Харрисон, В. Руззо, Дж. Ульман, Д. Белл, Л. ЛаПадула, Р. Сандху, Д. Феррайоло, Р. Кун, Д. Деннинг и Дж. МакЛин являются основоположниками классических моделей безопасности.

В трудах П. Н. Девянина и Д. П. Зегжды проводится анализ классических моделей безопасности компьютерных систем, в небольшой степени описаны результаты их практического применения. Д. П. Зегжда сформировал подход и методологию построения защищенных систем обработки информации, на основе которых разработана собственная защищенная ОС и обоснована ее корректность в соответствии с предложенной моделью безопасности. П. Н. Девянин предложил собственные модели безопасности компьютерных систем (ДП-модели) и обосновал необходимость их использования в ОС для исключения возможности возникновения запрещенных информационных потоков не только по памяти, но и по времени.

В работах А. Ю. Щербакова рассмотрены теоретические и практические основы компьютерной безопасности и представлена модель, гарантирующая невозможность изменения действующей в компьютерной системе политики безопасности. В целом работы А. Ю. Щербакова и В. А. Конявского посвящены развитию теории доверенных вычислительных систем и защищенного взаимодействия в них с использованием принципа пошагового контроля целостности и формальной модели изолированной программной среды.

Далее остановимся подробнее на исследовании наиболее актуальных результатов работ перечисленных специалистов применительно к защите информации в реальных компьютерных системах, особенно в ОС GNU/Linux. При этом основное внимание уделим формальным моделям безопасности, которые на сегодняшний день являются основными научно обоснованными достижениями существующих исследований в области совершенствования механизмов управления доступом, и их применимости к современным ОС.

Формальные модели безопасности компьютерных систем, особенно модели безопасности управления доступом и информационными потоками, могут быть использованы непосредственно для анализа защищенности существующих систем – ОС, систем управления базами данных, распределенных или любых других систем, а также позволяют получить теоретические гарантии защищенности существующей или новой КС [4, 84]. Однако не все классические модели безопасности в неизменном виде могут быть применимы для современных систем, в основном они применяются для формального анализа свойств механизмов защиты уже известных КС.

Для описания моделей безопасности КС используют ряд следующих понятий [13]: «объект» (англ. object) или «контейнер» (англ. container – объект, обладающий внутренней структурой), «субъект» (англ. subject) и «доступ» (англ. access). Объект или контейнер – это сущность КС, которая содержит в себе некоторую информацию (данные, другие объекты или контейнеры) и над которой субъекты выполняют определенные операции (запись или чтение и другие). Субъект – это сущность КС, которая может инициировать выполнение определенных операций над объектами. Для выполнения этих операций субъект осуществляет доступ к объекту, при этом в

классических моделях безопасности, как правило, рассматриваются следующие основные (базовые, фундаментальные) типы доступов [13]: чтение (англ. read) – доступ на чтение из объекта доступа; запись (англ. write) и «конкатенация» (англ. append) – доступы на запись и дополнение объекта доступа; выполнение (англ. execute) – доступ на активизацию (выполнение) субъекта доступа из объекта. Прочие типы доступа субъектов к объектам КС обычно могут быть реализованы с помощью сочетания перечисленных базовых типов.

Учитывая такой тип доступа как выполнение (активизацию субъекта доступа из объекта), в некоторых моделях безопасности под субъектом доступа подразумевается объект, который обладает свойством активности. При этом, в общем случае, такие понятия как пользователь (физическое лицо, управляющее субъектами доступа через средства управления КС) и субъект доступа выделяют как разные сущности. Субъекты КС могут изменять состояния объектов доступа, а такой внешний фактор, как пользователь, не оказывает влияния на свойства этих сущностей.

Важную роль при исследовании безопасности КС играет анализ информационных потоков (англ. information flow), возникающих в результате реализации доступов субъектов к объектам КС [78, 86]. Информационный поток – это преобразование данных в некоторой сущности (объекте КС), реализуемое субъектами доступа и зависящее от данных, содержащихся в сущности, от которой этот информационный поток возникает [13].

В современной теории компьютерной безопасности для формального моделирования КС и ее безопасности чаще всего используется представление исследуемой КС в виде абстрактной системы, каждое состояние которой полностью описывается доступами субъектов к объектам [109]. При этом на уровне аксиом предлагается принять предположение о том, что все вопросы безопасности в КС описываются доступами субъектов к объектам, а все действия (доступ к объектам, порождение информационных потоков и новых субъектов доступа, изменение параметров и настроек системы защиты КС) могут быть инициированы только субъектами КС с помощью осуществления доступа к ее сущностям. Другими словами, функционирование КС моделируется с помощью последовательности доступов субъектов к объектам, при этом практически игнорируются ее состав, структура и различия объектов доступа. В актуальных на данный момент моделях безопасности защищенность объектов доступа (данных) рассматривается с позиции защищенности системы в целом, без защиты отдельно взятых объектов (электронных документов, файлов и так далее). Все чаще возникает необходимость в специализации моделей безопасности, например, с целью учета качественного различия свойств субъектов и объектов КС, а также предъявляемых к ним требований [42]. Возможно, со временем другие подходы, ориентированные, например, на формальное описание политики безопасности [83], сетевых протоколов [47, 71], правил фильтрации пакетов сетевого информационного обмена и так далее, при должном обосновании и повсеместном распространении также смогут быть применены для моделирования безопасности КС.

Доступ всех субъектов к объектам КС разрешается системой управления доступом (подсистемой разграничения доступа КС) только при наличии у субъектов соответствующих прав доступа. Способ задания разрешенных прав доступа для субъектов регламентируется политикой управления доступом и информационными потоками, то есть составной частью политики

безопасности КС [4]. Под политикой безопасности понимается совокупность правил, регулирующих управление ресурсами КС, их защиту и распределение в пределах КС [13]. Выполнение этих правил обеспечивает защиту от угроз безопасности, влияющих на основные свойства безопасности информации (см. разд. 1.1.1), и является необходимым (а иногда и достаточным) условием безопасности всей системы. Формальное выражение политики безопасности называют моделью политики безопасности.

Основная цель описания политики безопасности КС в виде формальной модели – это выработка и доказательство критерия безопасности (условий, которым должно соответствовать поведение КС) и проведение формального доказательства соответствия КС этому критерию при соблюдении определенных условий и ограничений [13]. Соответственно, определенное состояние КС может быть либо «безопасным», либо «небезопасным». Считается, что КС «безопасна», если субъекты доступа в ней не имеют возможности нарушить правила политики безопасности (в любом состоянии КС). На практике соответствие КС критерию безопасности означает, что в любой момент времени только определенные субъекты могут получить доступ к объектам, а тип доступа субъектов к объектам должен быть санкционированным.

Для практического применения теоретических выкладок по отношению к безопасности различных КС либо необходим механизм анализа последовательностей их состояний, что на практике является достаточно труднореализуемой задачей (так как количество состояний любой КС бесконечно велико) [17], либо на возможные переходы КС из одного состояния в другое должны распространяться ограничения, при которых гарантируется безопасность КС.

Существуют следующие основные виды политик управления доступом (а конкретные модели безопасности таких политик называются классическими), определяющие способ задания разрешенных прав доступа:

– дискреционная политика управления доступом (англ. discretionary access control) [90], частная реализация которой рассмотрена ранее в ОС GNU/Linux;

– мандатная политика управления доступом (англ. mandatory access control) [82];

– политика ролевого управления доступом (англ. role-based access control) [104, 106].

Дискреционное управление доступом требует выполнения ряда условий:

1. Должна быть задана матрица доступов, каждая ячейка которой содержит список доступных прав доступа конкретного субъекта к объекту КС.

2. Субъект обладает правом доступа к сущности КС только в том случае, если в соответствующей ячейке матрицы доступов содержится данное право доступа.

В качестве моделей, реализующих дискреционную политику управления доступом, можно выделить модель матрицы доступов Харрисона-Руззо-Ульмана (ХРУ) [67, 90], модель типизированной матрицы доступов [18, 103], модель распространения прав доступа Take-Grant [67, 89] и дискреционные ДП-модели [10].

При этом в указанных моделях отличаются как объект анализа, так и набор рассматриваемых прав доступа, а для некоторых моделей существуют относительно эффективные [13, 101] алгоритмы проверки безопасности КС, основанные на построении замыкания графа доступов,

на условиях проверки истинности специально определенных предикатов [13] или на поиске компонент связанности [1].

Мандатное управление доступом требует выполнения следующих условий:

1. Должна быть задана решетка [13] уровней конфиденциальности информации (и соответствующих им уровней доступа).
2. Каждой сущности (объекту КС) должен быть присвоен уровень конфиденциальности, задающий ограничения на доступ к этой сущности.
3. Каждому субъекту КС должен быть присвоен уровень доступа (или уровень полномочий субъекта в КС) – то есть множество субъектов и объектов упорядочено в соответствии с их уровнем доступа и конфиденциальности соответственно.
4. Субъект может получить доступ только в том случае, если его уровень доступа позволяет это сделать по отношению к сущности с заданным уровнем конфиденциальности (на основе отношения в решетке уровней конфиденциальности).
5. Реализация доступа не должна приводить к возникновению информационных потоков от сущностей с высоким уровнем конфиденциальности к сущностям с низким уровнем конфиденциальности.

В качестве моделей, реализующих мандатную политику управления доступом можно выделить: классическую модель Белла-ЛаПадулы [69, 82], модель мандатной политики целостности информации Биба [85] (а также – политику low-watermark [9, 85]), модель систем военных сообщений (англ. Military Message Systems) [69, 98] и мандатную ДП-модель [10].

Как правило, все мандатные политики безопасности описывают в терминах классической модели Белла-ЛаПадулы, которая формально записана в терминах теории отношений. В модели Белла-ЛаПадулы рассматриваются условия, при выполнении которых в КС невозможно возникновение информационных потоков от объектов с большим уровнем конфиденциальности к объектам с меньшим уровнем конфиденциальности. В качестве прав доступа в данной модели рассматриваются read, write, append и execute. При этом к изменению состояния системы может привести добавление или удаление возможности доступа, изменение уровня доступа субъекта или уровня конфиденциальности объекта, добавление или удаление права доступа.

Безопасность КС в классической модели Белла-ЛаПадулы определяется с помощью трех свойств, выполнение которых для каждого состояния и реализации системы обязательно, а именно: ss-, *- и ds-свойство. Упрощенно ss- и *-свойства можно интерпретировать как «не читать сверху» (англ. «No Read Up», NRU) и «не писать вниз» (англ. «No Write Down», NWD), данные свойства позволяют исключить запрещенные информационные потоки «сверху вниз». ds-свойство требует наличия прав доступа к соответствующим объектам в матрице доступов (аналогично дискреционной политике управления доступом). В модели дополнительно определено множество доверенных субъектов доступа (множество субъектов, не подчиняющихся *-свойству), для которых действуют более мягкие ограничения на доступ.

Для классической модели Белла-ЛаПадулы проверка безопасности КС по определениям указанных выше свойств является в общем случае неразрешимой задачей [46] (так как проверку свойств необходимо проводить для всех реализаций и состояний системы, а их, как правило,

бесконечно много). В связи с этим для обеспечения безопасности КС в модели Белла-ЛаПадулы вводятся ограничения для множества действий системы [13, 19], то есть на основе ss-, *- и ds-свойств формируются условия для переходов системы из состояния в состояние с сохранением безопасности КС (из некоторого безопасного начального состояния).

В политике low-watermark для классической модели Белла-ЛаПадула субъекту всегда разрешена запись в объект, но уровень конфиденциальности объекта понижается до уровня доступа субъекта (если такое понижение реально происходит, то вся информация в объекте стирается).

Модель мандатной политики целостности информации Биба ориентирована на обеспечение целостности объектов доступа. Данная модель является противоположностью классической модели Белла-ЛаПадулы, так как информационные потоки последней могут представлять угрозу целостности данных (но не конфиденциальности).

Модель систем военных сообщений ориентирована, в первую очередь, на системы приема, передачи и обработки почтовых сообщений, и основана на следующих основных понятиях [69, 98]: авторизация, иерархия уровней конфиденциальности (для объектов-контейнеров), безопасный просмотр и перенос информации (информация всегда наследует уровень конфиденциальности объекта, в том числе при перемещении или прочтении на устройстве вывода).

Мандатная ДП-модель является развитием базовой (дискреционной) ДП-модели и ориентирована на анализ информационных потоков по памяти, позволяющих субъекту повысить свой уровень доступа. Для мандатной ДП-модели существуют относительно эффективные алгоритмы проверки безопасности КС, основанные на условиях проверки истинности специально определенных предикатов [13, 19].

Ролевое управление доступом требует выполнения следующих условий:

1. Должно быть задано множество ролей, каждой роли соответствует подмножество множества прав доступа.

2. Каждому субъекту доступа разрешено использовать некоторое множество ролей (авторизованные для субъекта роли).

3. Субъект может получить доступ к сущности только в том случае, если он обладает ролью, в числе прав доступа которой существует право доступа к данной сущности.

В качестве моделей, реализующих политику ролевого управления доступом, можно выделить: базовую модель ролевого управления доступом (англ. Role-Based Access Control) [18, 19, 104, 105], модель администрирования ролевого управления доступом, модель мандатного ролевого управления доступом [105], базовую ролевую ДП-модель [11, 12, 19].

Ролевое управление доступом является развитием политики дискреционного управления доступом – права доступа группируются в роли. Правила ролевого управления доступом задают порядок предоставления прав доступа субъектам КС в зависимости от сессии их работы и имеющихся в каждый момент времени ролей. На основе ролевого управления доступом могут быть реализованы дискреционное и мандатное управление доступом. Дополнительно к базовой ролевой модели существуют модели администрирования ролевого управления доступом, модель мандатного ролевого управления доступом (со свойствами мандатных моделей), базовая ролевая ДП-модель (на основе базовой ролевой модели, модели администрирования ролевого управления

доступом и ДП-моделей с дискреционным или мандатным управлением доступом). Формальное описание перечисленных моделей безопасности ролевого управления доступом приведено в [13, 19].

Кроме перечисленных основных видов политик управления доступом и соответствующих моделей безопасности, существуют также следующие модели безопасности: модель безопасности информационных потоков (автоматная модель [9], программная модель контроля информационных потоков [85], вероятностная модель [102]), ДП-модели безопасности информационных потоков по времени, модели изолированной программной среды (субъектно-ориентированная модель или СО-модель ИПС [67, 78–80] и ее расширение).

Модели безопасности информационных потоков в своей основе разделяют все возможные информационные потоки КС между сущностями на непересекающиеся множества – множество разрешенных и запрещенных информационных потоков. При этом основная цель состоит в обеспечении невозможности возникновения в КС запрещенных информационных потоков. В ряде моделей это сделано с помощью определения информационной невыводимости или информационного невлияния (высокоуровневые и низкоуровневые объекты КС изолируются друг от друга), что является слишком жестким требованием и сложновыполнимо в современных КС. Данные модели могут задавать различные виды политик безопасности, определяющих не только правила управление доступом, но и порядок взаимодействия между собой компонент КС.

В ДП-моделях безопасности информационных потоков, в том числе рассматриваются системы, в которых недоверенные субъекты не создают (или, наоборот, создают) сущности или другие субъекты, в которых порожденные субъекты отождествляются, в которых реализуется политика строгого мандатного контроля (*-свойство для доверенных субъектов) и другие.

Цель модели изолированной программной среды (ИПС) – задание порядка безопасного взаимодействия субъектов КС, обеспечивающего невозможность воздействия на систему защиты или модификацию ее параметров. Данное условие реализуется путем изоляции субъектов КС друг от друга и путем контроля порождения новых субъектов, чтобы в системе могли активизироваться только субъекты из предопределенного списка. При этом должна контролироваться целостность сущностей, влияющих на активизируемые субъекты.

В рамках СО-модели ИПС определяются: механизм порождения новых субъектов доступа; монитор порождения субъектов и монитор безопасности субъектов, разрешающий порождение субъектов только для фиксированного множества пар активизирующих субъектов и объектов-источников; порождение субъекта с контролем неизменности объекта-источника; монитор обращений и монитор безопасности объектов, позволяющий осуществить только разрешенные информационные потоки; понятия корректности и абсолютной корректности субъектов относительно друг друга, то есть невозможности изменения состояний субъектов.

В модели предполагается, что КС обладает замкнутой программной средой (замкнута по порождению субъектов), когда в ней действует монитор безопасности субъектов. Программная среда называется изолированной (абсолютно изолированной), когда она является замкнутой и порождаемые субъекты корректны (абсолютно корректны) относительно друг друга, монитора безопасности субъектов и монитора безопасности объектов. При этом, если с некоторого началь-

ного момента времени в абсолютной ИПС действует только порождение с контролем неизменности, а все порождаемые субъекты абсолютно корректны относительно друг друга и других субъектов, то в любой момент времени программная среда остается абсолютной изолированной программной средой.

Важную роль для создания ИПС в СО-модели играет поэтапная активизация субъектов КС. Практическая реализация всех КС позволяет выделить две фазы их работы: активизацию субъектов с ростом уровня представления объектов (фаза загрузки, начальная фаза) и фазу стационарного состояния системы (когда уровень представления объектов не увеличивается). В таком случае реализация ИПС может состоять из двух этапов, изображенных на Рисунке 1.3: предопределенного выполнения начальной фазы, включая момент активизации мониторов безопасности субъектов и объектов (ступенчатая загрузка: включения питания СВТ в $t = 0$, начало активизации субъектов с t_0); работы в стационарной фазе в режиме ИПС с контролем неизменности объектов-источников (стационарная фаза функционирования КС с t_1, начало действия ИПС с t_2, завершение активизации всех компонент ИС с t_3).

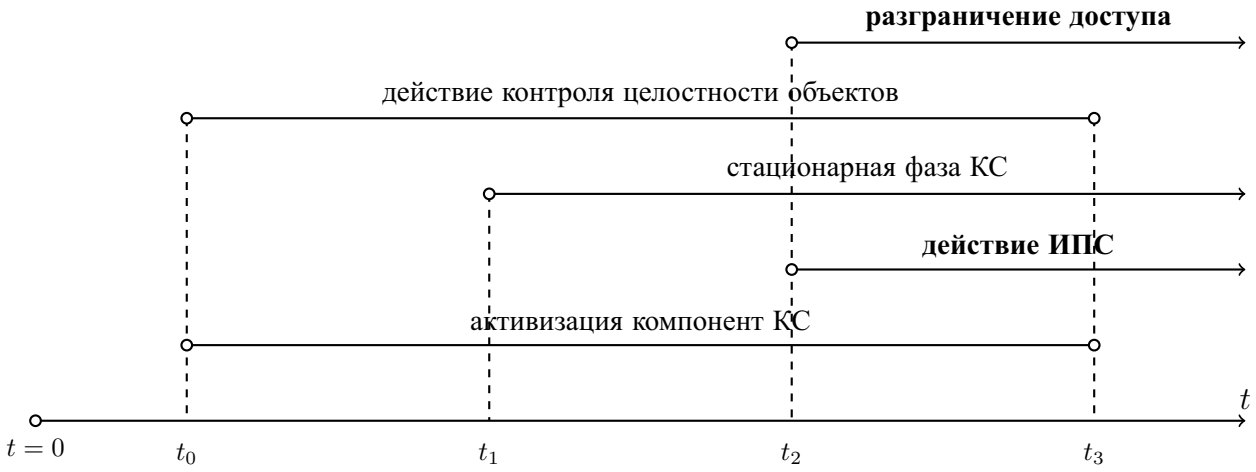

Рисунок 1.3 – Схематичное представление этапов функционирования КС в СО-модели ИПС

Субъект контроля неизменности объектов должен быть активен уже на этапе работы субъектов аппаратно-программного уровня (имеется ввиду этап загрузки КС, например, после включения питания). В СО-модели постулируется, что объект-источник такого субъекта контроля технически не может быть проверен на неизменность (постулируется его изначальная неизменность). При этом, в отличие от других моделей, в СО-модели описывается метод достижения «безопасного» начального состояния системы. Кроме того, СО-модель инвариантна относительно любой принятой в КС политики безопасности или управления доступом (не противоречащей требованиям СО-модели).

В качестве расширения СО-модели в [41, 42] была предложена идея создания доверенной вычислительной среды (ДВС), в основе которой лежит защита начального этапа работы КС в СО-модели. В соответствии с мультипликативностью защитных свойств [41] необходимо обеспечивать защиту на всех этапах работы КС, а не только на этапах, имеющих прямое отношение к получению доступа и обработке защищаемых данных (как это следует из требований

нормативных документов по защите информации из раздела 1.2). В соответствии с этим было предложено [41, 42]:

– требование неизменности программных средств модели ИПС заменить требованием целостности программных и аппаратных средств КС;

– для контроля среды функционирования КС встраивать в ее состав программно-аппаратный резидентный компонент безопасности, целостность которого обеспечивается технологически [41].

С помощью внедрения описанного компонента безопасности и осуществления пошагового контроля целостности формируется ДВС – фрагмент среды, в которой установлена и поддерживается целостность необходимых объектов системы (в том числе и объектов-источников для субъекта контроля неизменности объектов).

Таким образом, в некоторых моделях безопасности существует ряд принципиальных недостатков и ограничений, которые могут стать ключевым фактором в вопросе выбора необходимой и достаточной модели защиты КС. Рассмотрим подробнее некоторые из них.

Во всех рассмотренных в разделе моделях для реализации политики безопасности должны существовать, как минимум, следующие сущности (в некоторых моделях такие сущности и требования к ним только подразумеваются):

– активный субъект, осуществляющий контроль всех операций субъектов над объектами;

– объект, содержащий информацию о разрешенных и запрещенных операциях субъектов над объектами КС. При этом за данными в этом объекте КС (содержимым матрицы доступов в дискреционной политике управления доступом, уровнями доступа и конфиденциальности субъектов и объектов в мандатной политике управления доступом, множествами прав доступа ролей и разрешенных ролей в ролевой политике управления доступом и так далее) должен осуществляться контроль и обеспечение целостности.

В ряде моделей (например, в СО-модели) для данных сущностей формируются достаточно абстрактные требования, при выполнении которых гарантирована невозможность несанкционированного изменения действующей в КС политики безопасности. В других требования предъявляются к взаимодействию с данными сущностями (так называемые политики безопасного администрирования и абсолютного разделения административных и пользовательских полномочий).

Кроме активного субъекта, осуществляющего контроль всех доступов в КС должен существовать специальный субъект, который имеет право в случае необходимости санкционировано изменять политику безопасности (политику безопасности недостаточно один раз настроить перед эксплуатацией). В некоторых из приведенных моделей безопасности не уделяется внимания вопросам администрирования, в других в качестве субъекта, способного изменять политику безопасности КС, рассматривается некоторый абстрактный системный офицер безопасности.

В ряде моделей (Белла-ЛаПадулы, ДП-модели) вводится понятие доверенных субъектов, на которые не распространяются некоторые требования политики безопасности, однако не формируется понимания того, с какой целью эти субъекты введены и какую роль они играют в модели безопасности (могут ли изменять политику безопасности или просто считается, что их дей-

ствия не могут негативно сказаться на безопасности КС). В мандатной ДП-модели доверенными могут являться субъекты с низким уровнем доступа (например, системные процессы, функциональность которых не требует предоставления им возможности доступа к объектам с высоким уровнем конфиденциальности), недоверенные же субъекты могут иметь разный уровень доступа.

Одним из базовых условий в классических политиках управления доступом в КС является фиксация множеств субъектов и объектов доступа (постулируется факт, что данные множества должны быть идентифицированы). В реальных КС это требование является практически невыполнимым – при переходе системы в новые состояния могут появляться как новые объекты доступа (сетевые устройства, файлы, съемные устройства и так далее), так и субъекты доступа (хотя в отдельно взятом состоянии все сущности, действительно, могут быть идентифицированы). Такое ограничение выглядит достаточно искусственным и мешает использовать классические модели безопасности для реальных КС. В ряде моделей рассмотрены процессы порождения субъекта другим субъектом доступа или порождение субъекта из объекта (при выполнении доступа *exec*), вводится понятие ассоциированности объекта и субъекта доступа и другие.

При введении такого понятия, как порождение субъектами доступа других субъектов КС, возникают вопросы – например, «что порождает самый первый субъект доступа?» и «с какого момента начинается активизация субъектов?». На данные вопросы в ряде моделей не дается четкого ответа, возникает неопределенность, как в проблеме «курицы и яйца», не складывается понимания, что является причиной появления субъектов: другой субъект или активизирующийся перво-объект? В ролевой политике управления доступом предполагается отсутствие механизмов, позволяющих одной сессии активизировать другую – все сессии активизируются только пользователями (однако то, как и из чего происходит активизация, также приведено поверхностно). Дилемма первичности субъектов или объектов в некоторой степени разрешается только в СО-модели ИПС.

При рассмотрении политик мандатного управления доступом классической модели Белла-ЛаПадулы и модели целостности Биба была продемонстрирована проблема сочетания защитных механизмов от угроз конфиденциальности и целостности объектов доступа КС. В случае, если необходимо построить систему, предотвращающую одновременно и угрозы целостности и угрозы конфиденциальности, то использование правил моделей Белла-ЛаПадулы и Биба может привести к ситуации, когда уровни безопасности и целостности будут использоваться противоположными способами [53, 76]. Некоторые модели безопасности (например, модель мандатного ролевого управления доступом) по аналогии с указанными моделями Белла-ЛаПадулы и Биба разделяют требования обеспечения конфиденциальности и целостности, однако в реальных КС необходимо выполнять их одновременно.

Ряд моделей безопасности рассматривает невозможность только некоторых типов запрещенных информационных потоков. Так, классической модели Белла-ЛаПадулы достаточно для предотвращения реализации запрещенных информационных потоков по памяти между сущностями с большим и меньшим уровнем конфиденциальности (но только в случае «отсутствия памяти» при временном понижении уровня доступа субъекта). Однако при этом в полной мере

защищать КС от запрещенных информационных потоков по времени с помощью данной модели невозможно. Обеспечить защиту от всех типов информационных потоков (особенно по времени) возможно, если все информационные потоки проходят через систему защиты (предположение из вероятностной модели информационных потоков и СО-модели), что на практике сложно реализовать, особенно для потоков по времени, или используя мандатные ДП-модели.

Таким образом, практически для всех моделей безопасности КС остаются открытыми рассмотренные выше вопросы, в том числе характерные и для защиты ОС GNU/Linux [93]. Однозначно ответить на вопрос о том, какая формальная модель безопасности КС лучше применительно к GNU/Linux, в общем случае невозможно. У всех моделей есть специфичные особенности, из-за которых их применение в GNU/Linux может быть затруднено. Из описанных в данном разделе моделей безопасности наибольший интерес представляют дискреционные и мандатные механизмы разграничения доступа (в силу их наибольшего распространения в различных системах), СО-модель и ее расширение до ДВС, рассматривающие не только непосредственно доступы субъектов к объектам КС, но и среду, в которой они совершаются, а также ДП-модели. Однако вопрос выбора той или иной модели безопасности для защиты определенной КС зависит скорее от предпочтений, а также непосредственно от реализации КС и конкретных задач по защите тех или иных объектов доступа. Поэтому выбор формальной модели безопасности должен быть согласован с условиями функционирования и требованиями безопасности КС. Также необходимо удостовериться, что выбранная (или выбранные) формальные модели позволяют полностью выполнить требования к безопасности.

Наиболее подходящей моделью для защиты СВТ с ОС GNU/Linux в рамках цели данного исследования можно считать СО-модель ИПС в сочетании с одной или несколькими политиками управления доступом (например, дискреционной и мандатной). В случае решения всех перечисленных выше открытых для моделей безопасности вопросов с помощью такого сочетания возможно обеспечить необходимый уровень безопасности как для данных (объектов доступа) и пользовательской среды, в которой непосредственно происходит обработка информации в GNU/Linux, так и для самой подсистемы управления доступом.

1.4. Постановка задач исследования

Проведя анализ существующих средств разграничения доступа, требований нормативных документов РФ в области защиты информации, а также результатов известных исследований по совершенствованию механизмов управления доступом можно выделить факторы, которые препятствуют их применению по отношению к современным ИС и ОС GNU/Linux в частности.

Для рассмотренных существующих средств разграничения доступа в GNU/Linux характерны следующие недостатки:

– неоднозначность способов идентификации субъектов или объектов доступа;

– ограниченная работоспособность в зависимости от конфигурации ОС (например, в зависимости от типа используемых файловых систем и так далее);

– использование распределенной схемы администрирования (в большинстве средств), что приводит к сложностям в управлении и анализе безопасности системы;

– невозможность следовать принципу наименьших привилегий;

– наличие прав «владения» объектом, что создает сложности в контроле за защищенностью информации в системе, а также неприменимо при защите некоторых категорий данных;

– невозможность применять правила разграничения доступа в отношении всех субъектов системы (обычно связано с наличием привилегированных пользователей);

– некорректность встраивания и активизации защитных механизмов (существование возможности отключить или исключить активизацию СЗИ НСД);

– обеспечение преимущественно конфиденциальности защищаемых данных, без обеспечения целостности (дополнительные средства контроля целостности в данный момент сильно ограничены в своих возможностях, другие средства обеспечивают целостность только за счет разграничения доступа);

– невозможность использования одного средства защиты в рамках подсистемы управления доступом в ОС (только сочетания нескольких средств), что приводит к сложностям в администрировании и анализе защищенности (одновременный учет DAC, расширенных атрибутов доступа, порядка проверки полномочий для разных типов субъектов, расширенных средств защиты совместно с DAC и так далее).

Применения существующих стандартных или расширенных средств разграничения доступа в GNU/Linux не всегда достаточно, так как в них, например, не учитывается возможность неправомерного доступа к СВТ или самому носителю данных, в том числе и потенциальным внутренним нарушителем. Кроме того, вследствие существования большого разнообразия таких средств и того факта, что их механизмы защиты всегда работают совместно (хотя бы для некоторых из них) возникает еще несколько негативных с точки зрения защиты информации особенностей:

1. Отсутствует понимание того, какой состав существующих средств в подсистеме управления доступом ОС необходим для защиты информации от несанкционированного доступа.

2. Внедряемые в GNU/Linux со средствами разграничения доступа правила могут не всегда соответствовать их формальным моделям (или быть корректными).

3. Нельзя гарантировать, что сочетание нескольких внедренных средств разграничения доступа не нарушает всех правил соответствующих им формальных моделей.

Проанализированные требования нормативных документов РФ в области защиты информации подразумевают, что правила подсистемы управления доступом должны применяться в отношении защищаемых данных определенной категории (но при этом изменение критически важных данных ОС, например, ядра, может стать потенциальным каналом для НСД к защищаемым данным, что в требованиях в явном виде не учитывается). Также в требованиях не дается рекомендаций по способам построения, встраивания и безопасного администрирования систем защиты, которые бы могли помочь исключить возможность воздействия на подсистему управления доступом до загрузки ОС или во время дальнейшей работы. Однако большинство требований нормативных документов не могут быть в полной мере выполнены с использованием существующих СЗИ НСД для GNU/Linux даже для относительно низких классов защищенности АС, СВТ и ОС (1Г, 5 СВТ, 4 ОС и выше).

Результаты же известных исследования по совершенствованию механизмов управления доступом, а также формальные модели безопасности компьютерных систем тоже не лишены недостатков. Большинство моделей безопасности не могут быть в полной мере применимы в современных системах как в силу особенностей своей реализации (например, требования информационного невлияния или невыводимости), так и в силу динамической природы состава сущностей в современных системах. Практически для всех моделей безопасности КС остаются открытыми следующие вопросы, в том числе характерные и для защиты ОС GNU/Linux:

1. Корректность реализации подсистемы разграничения доступа, где и каким образом хранить правила управления доступом, вопрос безопасности администрирования.
2. Определение необходимого уровня встраивания функций защиты и контроля их корректного функционирования (в том числе отход от понятия безопасного начального состояния).
3. Корректность способов идентификации субъекта и объекта (вследствие абстрактности этих понятий в моделях безопасности КС) при осуществлении определенного доступа.
4. Корректность политики безопасности при изменении состава субъектов и объектов.
5. Необходимость и достаточность атрибутов, полнота контролируемых типов доступа, кроме рассматриваемых в моделях базовых типов.
6. Применение одновременно как механизмов обеспечения конфиденциальности, так и контроля целостности объектов доступа.

На основе проведенного анализа определим основные направления совершенствования существующих средств защиты информации от НСД, а также применяемых в них способов защиты данных.

При рассмотрении вопросов логического управления доступом в ОС (в том числе и в GNU/Linux) на некотором СВТ, необходимо учитывать тип нарушителя, который потенциально (преднамеренно или непреднамеренно) может осуществлять несанкционированный доступ к информации [7,64]. Так как защита от НСД должна обеспечиваться вне зависимости от полномочий пользователей, далее будет предполагаться возможность присутствия именно внутреннего нарушителя (а не только внешнего), имеющего доступ к СВТ, обладающего некоторой учетной записью в ОС и имеющего следующие возможности: запуск программ из фиксированного набора; создание и запуск собственных программ с новыми функциями по обработке информации; воздействие на базовое ПО, состав и конфигурацию аппаратных компонент СВТ. При этом перечисленные возможности могут быть ограничены как техническими (корректной настройкой BIOS/UEFI для исключения загрузки с внешних носителей, непосредственно с помощью СЗИ НСД и так далее), так и организационными мерами (опечатывание корпуса СВТ, разделение полномочий привилегированных пользователей, внедрение систем контроля и управления доступом в помещение и так далее).

Так или иначе, логическое управление доступом должно предусматривать возможность контроля действий абсолютно всех пользователей системы, включая привилегированные учетные записи (администраторы системы, root), реализуя принцип наименьших привилегий. Как было показано в разделе 1.1.5, существующие средства управления доступом в ОС GNU/Linux не

всегда позволяют ограничивать привилегированные учетные записи. Кроме того, в некоторых средствах отдельно учитывается право «владения» объектом доступа, которое позволяет передавать права доступа другим пользователям системы без ведома администратора. Применительно к конфиденциальной информации, персональным данным и другим видам защищаемой информации необходимо исключить такую возможность, права доступа на объекты должны выдаваться и изменяться централизованно администратором системы.

В основе любой подсистемы логического управления доступом должны существовать механизмы идентификации и аутентификации пользователей системы, корректная работа которых необходима для осуществления последующих функций защиты, таких как разграничение доступа. Кроме непосредственно процедур идентификации и аутентификации, также со всеми процессами ОС GNU/Linux, запускаемыми в дальнейшем от имени зарегистрированного пользователя, должны быть ассоциированы соответствующие атрибуты с результатами процедуры входа пользователя в систему (например, UID пользователя в атрибутах процессов ОС GNU/Linux). На основе этих атрибутов в дальнейшем при обращении к объектам ОС станет возможным однозначно ассоциировать (идентифицировать) конкретного пользователя системы с субъектами (процессами), которые от его имени осуществляют этот доступ. Однако в существующих средствах управления доступом в ОС GNU/Linux подобная идентификация пользователей часто бывает недостаточно точной, так как после создания пользовательских процессов и смены их идентификаторов дополнительно происходит множество обращений к системным объектам и конфигурационным файлам, необходимым для работы системы (особенно при использовании различных графических подсистем и сред), но к которым сам пользователь в дальнейшем не должен иметь доступа. Кроме того, в ОС GNU/Linux множество системных процессов изначально в ходе своей работы имеют $UID = 0$, однако какой-либо реальный пользователь системы (root, администратор ОС) не имеет к ним никакого отношения. В связи с вышесказанным, для построения подсистемы логического управления доступом, в которой бы пользователям можно было назначить только необходимые им для работы права доступа в соответствии с принципом наименьших привилегий, необходимо разработать способ однозначной идентификации субъектов доступа в ОС GNU/Linux.

Дополнительно в отношении идентификации субъектов необходимо провести исследование того, какое влияние могут оказывать на защищенность системы такие возможности GNU/Linux как создание каскадных сессий нескольких разных субъектов или создание множества сессий одного и того же субъекта доступа.

Другой важной составляющей описательной модели логического управления доступом являются объекты доступа и их атрибуты (права доступа субъектов к объектам). В качестве объектов доступа в ОС, как правило, выступают файлы в рамках файловых систем, которые, с учетом определенных возможностей GNU/Linux, в общем случае нельзя однозначно идентифицировать по их абсолютным путям (см. раздел 1.1.1). Однако некоторые существующие средства управления доступом в ОС GNU/Linux, тем не менее, используют абсолютные пути для описания объектов доступа в своих правилах. Другие средства управления доступом используют другой подход и хранят атрибуты объектов доступа непосредственно в их метаданных (задавая тем самым од-

нозначное соответствие), но накладывая при этом ограничения на применимость механизмов защиты в зависимости от конфигурации ОС. Таким образом, все существующие способы идентификации объектов доступа обладают определенными недостатками: для некоторых способов пропадает точность самой идентификации объектов доступа, а для других подсистему управления доступом невозможно применять по отношению к любым существующим и появляющимся объектам. В связи с этим необходимо провести исследование, определить все возможные пути корректной идентификации объектов доступа в GNU/Linux и целесообразность их применения в составе подсистемы управления доступом ОС.

Также многие средства защиты информации от НСД рассматривают только вопросы конфиденциальности информации, а такое важное свойство безопасности информации, как целостность (см. разд. 1.1.1), обеспечивается преимущественно с помощью контроля доступа. Однако необходимость наличия механизмов контроля целостности и их совмещения с механизмами контроля доступа обоснована во многих известных исследованиях и регламентирована требованиями по защите информации от НСД, в том числе и в РФ (см. разделы 1.2 и 1.3). Применение существующих средств контроля целостности в ОС GNU/Linux ограничивается конфигурационными файлами, в то время как для других объектов доступа целостность не контролируется даже во время периодических проверок.

В [25] приведен показательный пример того, чем может быть опасно нарушение целостности загружаемых модулей ядра Linux и других POSIX/SUS-совместимых ОС. Ранее в разделе 1.1.1 были выделены такие файловые объекты как разделяемые библиотеки и модули ядра, целостность которых является критической для ОС GNU/Linux. Разделяемые библиотеки могут использоваться многими исполняемыми бинарными файлами при запуске последних, а модули ядра Linux являются важными компонентами ОС и исполняются с максимальными привилегиями. В примере из [25] продемонстрирована возможность вследствие нарушения целостности одного из стандартных модулей ядра Linux в дальнейшем осуществлять НСД к системе с максимальными привилегиями и, возможно, в обход любой подсистемы управления доступом. Кроме того, в скомпрометированной описанным образом системе возможно повторное возникновение НСД даже после перезагрузки вследствие того, что многие модули ядра в GNU/Linux активируются автоматически в процессе загрузки или при подключении соответствующего оборудования в уже работающей ОС. Указанные возможности предполагают, что в системе должны быть привилегированные права доступа, однако в любом случае необходимо каким-либо образом реагировать на нарушение целостности таких важных компонент ОС.

В первую очередь, желательно контролировать целостность тех модулей ядра Linux, которые автоматически загружаются при старте ОС. Размер всех модулей на современных дистрибутивах GNU/Linux изначально может быть равен 90-100Мб и более, поэтому простой статический контроль целостности (единовременная проверка целостности всех перечисленных объектов при наступлении определенного события) [38] в данном случае будет выполняться непозволительно долго. Можно использовать динамический контроль целостности (проверку целостности при открытии файлов на исполнение) [38] и проводить его не только для бинарных файлов, но и для любых файлов вне зависимости от выполняемого доступа. Данный вариант дополнительно

предоставляет возможность провести контроль целостности не только самих бинарных файлов, но и всех используемых ими разделяемых библиотек (см. разд. 1.1.1). Во-вторых, можно проверять легитимность загрузки модулей ядра (при выполнении системного вызова *sys_init_module*). Так или иначе, целью внедрения механизмов контроля целостности должна являться не превентивная защита системы (как в случае с контролем доступа), а фиксация уже осуществленного НСД, блокирование любых попыток его дальнейшего проявления и сигнализация о возникновении данного инцидента.

Необходимо также понимать, что использование механизмов контроля целостности или доступа, целостность которых никак не подтверждена, не имеет смысла [41]. Поэтому целесообразным видится использование в составе подсистемы управления доступом в GNU/Linux известного механизма пошагового контроля целостности и доверенной загрузки ОС для обеспечения целостности вначале аппаратных средств СВТ в начальный момент его загрузки, программных механизмов контроля целостности и доступа критичных компонент системы, а затем, с помощью этих механизмов самих защищаемых данных. При этом механизмы контроля целостности должны активизироваться с самого раннего этапа загрузки ОС GNU/Linux, обеспечивая тем самым «самозащиту» используемых компонент подсистемы управления доступом.

Некоторые существующие средства управления доступом в GNU/Linux для реализации механизма пошагового контроля целостности и доверенной загрузки ОС применяют аппаратные модули доверенной загрузки [41]. Однако данные средства были спроектированы достаточно давно и применимы для систем, в которых существовали только простейшие BIOS и системные загрузчики ОС. Сейчас в СВТ, на которых используются современные системы GNU/Linux, не всегда возможно использовать стандартные АМДЗ в неизменном виде (например, может отсутствовать стандартный BIOS, поддержка PCI или расширений BIOS для конкретной архитектуры СВТ). Кроме того, даже в случае использования IBM-совместимого СВТ в GNU/Linux по умолчанию применяются более современные загрузчики, которые позволяют в динамике изменить процесс загрузки ОС. Таким образом, на данный момент использовать существующие АМДЗ для доверенной загрузки ОС и гарантированной активизации ее подсистемы управления доступом невозможно. В связи с этим необходимо провести исследование и скорректировать существующий способ доверенной загрузки ОС для устранения перечисленных недостатков.

Для подавляющего большинства существующих средств управления доступом в GNU/Linux невозможно гарантировать активизацию и непрерывность функционирования в процессе работы ОС. Таким образом, дополнительно к доверенной загрузке необходимо исследовать и предложить корректный алгоритм встраивания подсистемы управления доступом в ОС. По отношению к такому встраиванию естественным образом возникает ряд требований: необходимо гарантировать загрузку подсистемы управления доступом и целостность связанных с ней объектов; встраивание должно происходить на самом раннем этапе загрузки ОС (до возникновения реальных субъектов или защищаемых объектов); необходимо обеспечить непрерывность работы механизмов защиты подсистемы управления доступом; в случае невыполнения перечисленных требований – приостанавливать работу ОС или сигнализировать о возникших событиях безопасности.

При этом в противоположность требованиям действующих нормативных документов РФ в области защиты информации при активизации правила подсистемы управления доступом должны применяться в отношении всех субъектов, а главное – объектов доступа (а не только в отношении конфиденциальных или других защищаемых данных) уже начиная с раннего этапа загрузки ОС.

Также необходимо отметить, что в ОС GNU/Linux всегда существует сочетание нескольких средств разграничения доступа. Такое одновременное сосуществование нескольких средств защиты обусловлено как исторически, так и вследствие необходимости обратной совместимости с более ранними версиями ОС. По существующим стандартам, таким как POSIX, SUS [91] и LSB [100] в GNU/Linux не может быть отключена, например, существующая дискреционная политика управления доступом, использующая стандартные права доступа файловых объектов. Любое другое средство разграничения доступа в любом случае должно будет сочетаться и взаимодействовать с такими встроенными механизмами защиты. Таким образом, полноту используемых защитных механизмов в GNU/Linux необходимо рассматривать с учетом существующих стандартных механизмов защиты ОС в сочетании с внедряемыми дополнительными средствами управления доступом. Для повышения защищенности ОС GNU/Linux и обрабатываемых в ней данных необходимо компенсировать недостатки существующих средств управления доступом и реализованных в них способов защиты информации (описанных в разделе 1.4), но при этом обеспечить корректное сочетание и непротиворечивое взаимодействие с такими средствами. При этом сама подсистема управления доступом и реализованные в ней способы защиты информации не должны зависеть от конфигурации и настроек ОС.

Таким образом, в настоящий момент существует потребность в использовании Linux и в защите информации в среде этих ОС от несанкционированного доступа, однако при этом встроенных средств защиты в GNU/Linux недостаточно для надежной защиты пользовательских данных. Более того, эти средства не могут в полной мере удовлетворять требованиям существующих нормативных документов РФ в области защиты информации [6,8,49,58,59]. В большинстве средств реализованы не все возможности по разграничению доступа и контролю целостности (например, не реализовано мандатное управление доступом, отсутствует динамический контроль целостности), не существует возможности реализовать принцип наименьших привилегий для всех субъектов системы (например, для суперпользователя root), а самое главное – существует возможность исключить активизацию этих средств на раннем этапе загрузки системы.

Кроме того, существуют сложности в применимости моделей безопасности и известных исследований по совершенствованию механизмов управления доступом в отношении современных информационных систем. В связи со всем упомянутым возникает необходимость в проведении исследования и, возможно, разработке необходимых алгоритмов обеспечения безопасности, призванных решить возникшие сложности, а также в обосновании их корректности. Это обуславливает актуальность диссертационной работы, а также важность решения сформулированной научной задачи.

Для достижения поставленной цели необходимо использовать выделенные ключевые и наиболее значимые недостатки как существующих средств защиты от НСД, так и известных исследований в области совершенствования механизмов управления доступом в ОС, а для их устранения – разработать необходимые алгоритмы или функции защиты информации.

Логично использовать результаты известных исследований в области защиты информации при разработке таких новых алгоритмов или функций защиты, а их корректность (то есть безопасность системы, использующей их) обосновывать на основе утверждений для уже существующей и подходящей для этого модели безопасности (при наличии последней). При этом вначале предстоит формально описать систему (ОС GNU/Linux), например, с использованием существующего аппарата теорий моделирования и автоматов, математической логики и теоретических основ компьютерной безопасности. Далее требуется сформировать условия, при выполнении которых в системе обеспечивается выполнение некоторых отсутствующих ранее свойств (например, гарантированное отсутствие запрещенных информационных потоков, невозможность отключить выполнение функций защиты от НСД), и на их основе предложить и обосновать необходимость использования новых алгоритмов защиты информации, которые бы позволили выполнить предложенные требования для безопасности системы. Попробуем определить модель, которая больше других соответствует системе в виде СВТ с установленной ОС GNU/Linux и которая бы могла стать основой для обоснования новых алгоритмов или функций обеспечения безопасности управления доступом.

В известном исследовании в области безопасности КС вводится обязательное требование по защите информации на всех этапах работы ОС GNU/Linux, а не только во время непосредственной работы с защищаемыми данными. При этом выбираемая модель безопасности должна рассматривать безопасность защищаемых объектов доступа (данных, конфиденциальной и иной информации) с позиции защищенности системы в целом, а не с позиции защиты отдельно взятых объектов (электронных документов, важных файлов и так далее). Кроме того, модель должна учитывать еще и тот факт, что средства защиты КС (ОС GNU/Linux) также представляют собой сущности системы, активизируются на некотором этапе ее запуска и принимают участие в доступе субъектов к объектам (а к связанным с ними объектам и конфигурационным данным могут получать доступ определенные субъекты в целях администрирования).

Из рассмотренных в разделе 1.3 моделей перечисленным требованиям более других соответствует СО-модель ИПС, которая изначально устраняет часть наиболее значимых недостатков существующих средств защиты GNU/Linux и требований, описанных ранее в данном разделе (описан порядок активизации механизмов защиты системы и субъектов доступа, рассматриваются невозможность несанкционированного выключения или изменения действующей в системе политики безопасности и вопросы безопасного администрирования). Однако при этом реализовать абсолютную ИПС в понятиях СО-модели безопасности на практике практически невозможно из-за присутствия в современных системах объектов, которые в любой момент времени могут быть ассоциированы сразу с несколькими субъектами доступа (переменные окружения, разделяемые библиотеки и др.). Так или иначе, СО-модель ИПС можно взять за основу, для которой уже доказаны ключевые положения в рамках безопасности компьютерных систем. Учитывая все

общие открытые вопросы для формальных моделей безопасности и прочие необходимые нюансы, можно использовать известные положения СО-модели ИПС в новой модели безопасности ОС и подсистемы управления доступом, для которой в дальнейшем предложить конкретные алгоритмы достижения безопасного состояния КС.

В случае успешного формирования и обоснования новой модели безопасности и необходимости использования для нее предложенных алгоритмов защиты информации появится возможность разработки соответствующей подсистемы управления доступом, которая будет лишена недостатков существующих средств защиты GNU/Linux, является корректной в соответствии с формальными моделями безопасности и учитывает все другие описанные в данном разделе особенности и недостатки. Для создания такой подсистемы управления доступом следует сформировать необходимые требования, которые бы актуализировали и уточняли существующие нормативные документы РФ в области защиты информации (например, взяв за основу минимально необходимый класс защищенности АС 1Г, 5 класс СВТ и 4 класс ОС, см. раздел 1.2). При этом необходимо описать эти требования как можно более обобщенно для возможности их применения не только в рамках GNU/Linux, а также и для других систем.

Для созданной на основе сформированных требований подсистемы управления доступом в GNU/Linux необходимо обосновать корректность взаимодействия и сочетания с другими применяемыми в ОС средствами защиты (и, возможно, формальными моделями безопасности). В качестве основы для такого обоснования, например, можно использовать не формальное описание системы (ОС GNU/Linux) из модели безопасности, а использовать основы теории алгоритмов и автоматов. При этом каждое средство защиты ОС GNU/Linux можно представить в виде некоторого автомата, который, принимая на вход запрос о доступе (атрибуты объекта и субъекта, тип доступа), будет либо запрещать, либо разрешать этот доступ. Соответственно сочетание нескольких СЗИ НСД будет представляться в виде соединений таких автоматов.

Также для созданной подсистемы управления доступом в GNU/Linux необходимо будет оценить ожидаемый эффект от ее применения и провести ряд экспериментальных исследований: исследовать особенности встраивания и невозможности отключения или несанкционированного изменения правил разграничения доступа, исследовать полноту защитных функций, оценить накладываемые расходы от использования и, возможно, иные.

1.5. Выводы к главе 1

1. Проведен анализ существующих средств защиты информации от НСД в GNU/Linux и предъявляемых к ним требований нормативных документов РФ, а также результатов известных исследований по совершенствованию способов управления доступом в ОС, который выявил ряд серьезных недостатков, приводящих к возможности возникновения НСД к данным.

2. Выявлено, что стандартные средства разграничения доступа ОС GNU/Linux имеют уязвимые фазы работы, на которых существует возможность обхода нарушителем применяемой политики управления доступом: на этапе загрузки системы, когда функции защиты еще не активированы; на этапе дальнейшей работы системы, когда функции защиты были активированы, но могут быть случайно или преднамеренно отключены.

3. Обосновано, что существующие средства разграничения доступа GNU/Linux обладают следующими недостатками в выявленных уязвимых фазах работы: неоднозначностью способов идентификации субъектов и объектов, ограниченной работоспособностью при некоторых конфигурациях ОС, невозможностью следовать принципу наименьших привилегий, распределенной схемой администрирования, возможностью отключить или исключить активизацию функций защиты от НСД, обеспечением целостности данных преимущественно с помощью контроля доступа. Кроме того, часто отсутствует обоснование корректности реализованных в средствах защиты информации от НСД моделей безопасности и их сочетания при взаимодействии друг с другом, а сами средства защиты не могут выполнить требований нормативных документов РФ даже для относительно низких классов защищенности.

4. Показано, что для защиты информации от НСД в современных системах (в том числе ОС GNU/Linux) требуется актуализировать существующие нормативные документы РФ в следующих аспектах: учет состава сущностей системы, корректности способов идентификации субъектов и объектов доступа, а также способов их возможного взаимодействия; учет необходимости защиты системы в целом, включая критически важные компоненты ОС и подсистему управления доступом, а не только данных определенной категории; рекомендации по способам построения, встраивания и безопасного администрирования подсистемы управления доступом.

5. Обосновано, что использование результатов известных исследований по совершенствованию способов управления доступом и моделей безопасности в современных системах требует дополнительных исследований и решения выявленных неучтенных вопросов, характерных и для защиты ОС GNU/Linux: корректность реализации подсистемы управления доступом (хранение правил, безопасное администрирование, встраивание функций защиты от НСД и контроль их функционирования, полнота контролируемых типов доступа), корректность политики безопасности при изменении состава субъектов и объектов, необходимость применения одновременно функций обеспечения конфиденциальности и целостности.

2. Разработка научно-обоснованных алгоритмов обеспечения безопасности управления доступом ОС GNU/Linux, исключающих возможность обхода действующих правил доступа

В ходе проведенного анализа существующих способов и средств защиты информации от НСД в GNU/Linux, а также и предъявляемых к ним требований нормативных документов РФ и результатов известных исследований по совершенствованию способов управления доступом в ОС показана необходимость в проведении исследования и разработке алгоритмов обеспечения безопасности управления доступом, устраняющих недостатки существующих средств в GNU/Linux. Целью создания данных алгоритмов является противодействие неучтенным возможностям возникновения НСД к данным в ОС с помощью обхода правил действующей политики безопасности при воздействии на подсистему управления доступом или при исключении ее активизации. Для формирования таких алгоритмов обеспечения безопасности управления доступом необходимо первоначально формально описать систему (ОС с функциями защиты от НСД, субъекты и объекты доступа и др.) с использованием существующего аппарата теорий моделирования и автоматов, математической логики, а также теоретических основ компьютерной безопасности, и предложить условия, при выполнении которых описанные выше возможности несанкционированного доступа будет нельзя реализовать. Формальное описание и условия невозможности нарушения действующих правил доступа сформируют модель безопасности рассматриваемой системы, на основе которой в дальнейшем будут предложены алгоритмы обеспечения безопасности.

2.1. Разработка модели изолированной программной среды субъектов ОС GNU/Linux, исключающей возможность обхода действующих правил управления доступом

Для противодействия выявленным в Главе 1 неучтенным возможностям возникновения НСД к данным в ОС необходимо вначале формально описать систему (далее КС, ОС с подсистемой управления доступом), все существующие в ней сущности и возможные операции, выполняемые с участием этих сущностей. Далее необходимо сформировать ограничения на эти операции и другие условия, при выполнении которых в КС будет невозможно возникновение доступа субъектов к объектам в обход действующей политики безопасности. В качестве основы, наиболее подходящей для формального описания рассматриваемой КС, будем использовать СО-модель ИПС (см. разделы 1.3–1.4 и неформальное описание из [13, 67, 78–80]), идеи которой будем использовать для формирования новой модели безопасности ОС с подсистемой управления доступом [93].

Далее все вводимые обозначения с индексом so будут относиться только к СО-модели ИПС, остальные обозначения будут относиться к новой модели безопасности для произвольной ОС с подсистемой управления доступом.

В рамках СО-модели ИПС вводится декомпозиция сущностей системы на субъекты, объекты и сущности, выполняющие функции защиты системы. Так, множество субъектов в СО-модели состоит из подмножества разрешенных ($S_l{}^{so}$) и запрещенных ($S_n{}^{so}$) субъектов досту-

па системы (зарегистрированные, незарегистрированные или заблокированные субъекты доступа соответственно), а также монитора безопасности субъектов (${s_{srm}}^{so}$) и монитора безопасности объектов (${s_{orm}}^{so}$), которые осуществляют соответственно контроль возникновения субъектов системы и разграничение их доступа. Таким образом, множество субъектов СО-модели $S^{so} = {S_l}^{so} \cup {S_n}^{so} \cup \{{s_{srm}}^{so}, {s_{orm}}^{so}\}$.

В рамках новой модели безопасности произвольной ОС с подсистемой управления доступом обозначим с помощью $E = S \cup O$ – абстрактное множество всевозможных сущностей системы, где S – множество субъектов, а O – множество объектов.

В соответствии с разделом 1.1.1 в роли фактических субъектов в GNU/Linux, а также в любой другой ОС, и подсистеме управления доступом выступают пользователи, «псевдопользователи» и процессы, выполняемые от их имени. Однако с точки зрения возникновения информационных потоков, в отличие от СО-модели ИПС, необходимо рассматривать в качестве субъектов доступа следующие (иногда абстрактные) сущности:

– реальные пользователи ОС, интерактивно взаимодействующие с системой (далее с помощью S_{users} будем обозначать множество таких пользователей, зарегистрированных в системе);

– системные демоны и сервисы, неинтерактивно функционирующие в ОС в соответствии с заданным алгоритмом работы (далее с помощью S_{system} будем обозначать таких «псевдопользователей», зарегистрированных в системе).

Изоляцию среды имеет смысл рассматривать для различных субъектов всей системы, а не только для процессов-субъектов, запущенных в рамках сессии определенного пользователя в ОС (как в СО-модели ИПС). В связи с этим далее будет подразумеваться, что субъект в рамках новой модели безопасности ОС с подсистемой управления доступом – некоторая абстрактная сущность, представляющая собой множество всех процессов, запущенных от имени этого субъекта, а существование того или иного субъекта в момент времени $t \geqslant 0$ характеризуется наличием в этот момент времени t выполняющихся объектов-процессов.

Также в подсистеме управления доступом для GNU/Linux или другой ОС в общем случае существует субъект, выполняющий одновременно роль монитора безопасности субъектов и объектов СО-модели ИПС (обозначим его s_{sorm} – ядро защиты подсистемы управления доступом). Данный субъект в рамках новой модели безопасности ОС и подсистемы управления доступом не будет разделяться на два независимых субъекта системы, выполняющих разные функции защиты от НСД, из-за того, что требования к мониторам безопасности субъектов и объектов в СО-модели ИПС идентичны.

Таким образом, определим состав субъектов для новой модели безопасности ОС и подсистемы управления доступом. Множество субъектов ОС с подсистемой управления доступом представим в виде $S = S_l \cup S_n$, где:

– $S_l = S_{users} \cup S_{system} \cup \{s_{sorm}\}$ – множество разрешенных (зарегистрированных) в системе субъектов доступа ОС, где S_{users} – множество разрешенных пользователей, S_{system} – множество разрешенных системных сервисов и демонов, s_{sorm} – субъект, представляющий ядро защиты подсистемы управления доступом;

– S_n – множество запрещенных (заблокированных, незарегистрированных) в системе субъектов доступа;

Необходимо отметить, что состав активных субъектов в системе изменяется во времени, так как запускаются новые или перезапускаются существующие сервисы, входят и начинают работать субъекты-пользователи. Однако множества S_l и S_n фиксированы $\forall t \in \mathbb{N}_0$, то есть S описывает все возможные субъекты системы за все время ее работы.

Для описания активных субъектов системы (существования активной сессии субъектов доступа) в некоторый момент времени $t \in \mathbb{N}_0$ введем множество $S^t_{active} \subseteq S_l$, которое по мере возникновения новых субъектов доступа должно пополняться элементами $s \in S_l$, а при выходе или завершении работы субъекта – соответствующий элемент множества должен исключаться из S^t_{active}. В соответствии с разделом 1.1.1 в ходе инициализации и загрузки ядра ОС в системе должен активироваться системный субъект (обозначим его $s_0 \in S_{system}$), который в дальнейшем постоянно присутствует в системе, является предком для всех процессов ОС и участвует в порождении других субъектов доступа. Кроме того, очевидно, что s_{sorm} также должен присутствовать в системе с самого начала. В соответствии с этим в начальный момент времени $t = 0$ в системе присутствует два субъекта, то есть $S^0_{active} = \{s_0, s_{sorm}\}$. Множество возможных множеств активных субъектов обозначим как $S_{active} = \{S'_l \subseteq S_l\}$.

Множество объектов доступа в СО-модели ИПС состоит из двух непересекающихся подмножеств: объекты, ассоциированные (функционально ассоциированные или ассоциированные в качестве объектов-данных, ${O_a}^{so}$) [13] с произвольным субъектом доступа и не ассоциированные ни с одним субъектом доступа (${O_{na}}^{so}$). Для мониторов безопасности субъектов и объектов вводятся отдельные ассоциированные с ними объекты – ${o_{srm}}^{so}$, ${o_{orm}}^{so}$. Таким образом, множество объектов СО-модели $O^{so} = {O_a}^{so} \cup {O_{na}}^{so} \cup \{{o_{srm}}^{so}, {o_{orm}}^{so}\}$.

В рамках СО-модели ИПС подразумевается, что функционально ассоциированные объекты для некоторого субъекта – это бинарные файлы, активируемые для порождения этого процесса-субъекта в системе. В рамках же новой модели безопасности ОС с подсистемой управления доступом в качестве функционально ассоциированных с субъектом в некоторый момент времени объектов выступают сами процессы, выполняемые от имени субъектов-пользователей или системных сервисов. Также множество ассоциированных с субъектом доступа объектов в момент времени $t \in \mathbb{N}_0$ будет содержать все ассоциированные с его процессами объекты в момент времени t – ассоциированные объекты-данные (бинарные объектные файлы, разделяемые библиотеки, конфигурационные файлы и переменные окружения). Для субъекта, представляющего ядро защиты подсистемы управления доступом также существует множество ассоциированных с ним в момент времени $t \in \mathbb{N}_0$ объектов доступа – процессы утилит администрирования, файлы конфигурации ядра защиты, правила доступа, правила порождения субъектов, идентификационная и аутентификационная информация и так далее). В роли остальных объектов доступа системы, не ассоциированных в некоторый момент времени с субъектами доступа, выступают любые объекты файловой системы ОС (см. разд. 1.1.1).

Таким образом, определим более подробно состав абстрактного множества всевозможных объектов O (всевозможные состояния процессов и файловых объектов) в некоторый момент

времени $t \in \mathbb{N}_0$ для новой модели безопасности ОС и подсистемы управления доступом. Обозначим:

– $O^t \subseteq O$ – множество объектов в момент времени t.

– $O^t_{na} \subseteq O^t$ – множество неассоциированных с субъектами доступа объектов в момент времени t, в начальный момент времени $t = 0$: $O^0_{na} = O^0$. $O_{na} = \{O' \subseteq O\}$ – множество возможных множеств неассоциированных с субъектами доступа объектов.

– $O^t_{func} \subseteq S^t_{active} \times O^t$ – множество функционально ассоциированных с субъектами доступа объектов в момент времени t (процессы субъектов доступа, включая процессы ядра защиты s_{sorm}). В начальный момент времени $t = 0$: $O^0_{func} = \{(s_0, o_0),(s_{sorm}, o'_{sorm})\}$, где $o_0 \in O^0$ – процесс первоначального системного субъекта $s_0 \in S_{system}$ (для GNU/Linux этим процессом является *init* с $PID = 0$), $o'_{sorm} \in O^0$ – процесс ядра защиты s_{sorm}. Для каждого $o \in O^t$ существует не более одного $s \in S^t_{active}$: $(s, o) \in O^t_{func}$ (то есть объект не может быть функционально ассоциирован сразу с несколькими субъектами доступа). $O_{func} = \{S'_l \times O' \subseteq S_l \times O\}$ – множество возможных множеств функционально ассоциированных с субъектами доступа объектов.

– $O^t_{data} \subseteq S^t_{active} \times O^t$ – множество ассоциированных с субъектами доступа объектов-данных в момент времени t (бинарные объектные файлы, разделяемые библиотеки, конфигурационные файлы и переменные окружения, в том числе и для ядра защиты s_{sorm}). В начальный момент времени $t = 0$: $O^0_{data} = \varnothing$. В отличие от элементов множества O^t_{func}, могут существовать элементы $s, s' \in S^t_{active}$: $(s,o) \in O^t_{data}$, $(s',o) \in O^t_{data}$ и $s \neq s'$ (то есть объект может быть ассоциирован как данные с несколькими субъектами одновременно). $O_{data} = \{S'_l \times O' \subseteq S_l \times O\}$– множество возможных множеств ассоциированных с субъектами доступа объектов-данных.

Содержимое элементов $o \in O$ имеет свойство изменяться в процессе работы системы, в соответствии с этим обозначим o_t – состояние объекта в некоторый момент работы системы t. В общем случае содержимое любого объекта может быть описано в виде последовательности некоторых символов (символы какого-либо алфавита, двоичный формат представления данных и др.). Соответственно понятие состояние будем использовать для краткого обозначения состава объекта, а обозначение $o_t = \varnothing$ означает, что содержимое объекта пусто. При этом, при равенстве $o_t = o_l$ будем говорить о тождественности объекта o в моменты времени $t,l \in \mathbb{N}_0$, $t \neq l$ или об обеспечении его целостности в эти моменты времени.

Необходимо отметить, что состав множеств O^t, O^t_{func}, O^t_{data} и O^t_{na} не фиксирован во времени. Могут появляться или исчезать новые элементы $o \in O^t$, при этом они одновременно будут появляться или исчезать во множестве O^t_{na}. Элементы $o \in O^t$: $o \in O^t_{na}$ могут быть исключены из множества O^t_{na}, при этом одновременно появится элемент $(s, o) \in O^t_{data}$, $s \in S$ (при выполнении доступов или запросов в системе, которые будут рассмотрены далее), однако для фиксированного t и $o \in O^t_{na}$ не существует $s \in S$: $(s, o) \in O^t_{data}$ и наоборот. Элементы O^t_{func} добавляются в это множество в результате запуска новых процессов на выполнение (также является доступом или запросом в системе) и удаляются из множества при завершении работы, при этом влияние на множества O^t_{data} и O^t_{na} не оказывается.

Обозначим с помощью $V = S_{active} \times O_{func} \times O_{data} \times O_{na}$ множество всевозможных состояний системы, $v_t = (S^t_{active}, O^t_{func}, O^t_{data}, O^t_{na}) \in V$ – состояние в момент времени $t \in \mathbb{N}_0$.

Далее опишем основные операции, которые могут выполняться над субъектами и объектами системы. В СО-модели ИПС введены две операции: операция порождения субъектов (процессов) из объектов (бинарных файлов) и операция возникновения потока информации от одного объекта к другому при активизирующем воздействии некоторого субъекта. Рассмотрим последовательно эти операции применительно к новой модели безопасности ОС и подсистемы управления доступом.

Обозначим $Q = S \times O \times E$ – множество возможных запросов к системе, при этом для $(s,o,e) \in Q$: s является активизирующим запрос субъектом, o является объектом-источником, а e – сущностью-приемником, над которой выполняется некоторая операция. Все возможные запросы в ОС с подсистемой управления доступом $q \in Q$ приведены в Таблицах 2.1 и 2.2. Прочие запросы к системе, например, более сложные информационные потоки субъектов к объектам, могут быть представлены в виде композиции приведенных в таблицах элементарных запросов.

Запросы из Q, приведенные в Таблице 2.1, относятся к субъектам и их функционально ассоциированным объектам, то есть соответствуют операции порождения субъектов СО-модели ИПС. В качестве операции порождения субъектов в GNU/Linux и других ОС используется сочетание нескольких системных вызовов ядра и операций ОС: *fork()* – порождение дочернего процесса ($create\_process(s,o,o')$ из Таблицы 2.1), *exec()* – исполнение бинарного файла ($exec(s,o,o')$ из Таблицы 2.1), *login()* и/или *setuid()* – идентификация/аутентификация и/или смена идентификатора пользователя ($create\_user(s,o,s')$ или $create\_shadow(s,o,s')$ из Таблицы 2.1 соответственно). При этом для различных субъектов доступа порождение выполняется по следующему алгоритму:

1. Выполняются одна или несколько операций ОС *fork()* и *exec()*;
2. Если порождается субъект $s' \in S_n$, то порождение невозможно.
3. Если порождается субъект-пользователь ($s' \in S_{users} \subseteq S_l$), то предварительно выполняется операция ОС *login()*, состоящая из двух этапов:

– идентификация пользователя, то есть проверка значения предоставленной пользователем идентификационной информации в соответствии с данными из подсистемы управления доступом $o_{sorm} \in O^t$: $(s_{sorm}, o_{sorm}) \in O^t_{data}$ в момент выполнения порождения $t \in \mathbb{N}_0$;

– аутентификации пользователя, то есть проверка значения предоставленной пользователем аутентификационной информации в зависимости от значения идентификационной информации в соответствии с данными из подсистемы управления доступом $o_{sorm} \in O^t$: $(s_{sorm}, o_{sorm}) \in O^t_{data}$ в момент выполнения порождения t.

При этом в рамках описанных запросов к системе из Таблицы 2.1 эта операция ОС входит в проверку принадлежности $s' \in S_{users}$ для запроса $create\_user(s,o,s')$.

4. Если порождается субъект, соответствующий системному процессу ($s' \in S_{system} \cup \{s_{sorm}\}$), то порождающим субъектом должен быть субъект $s = s_0 \in S_{system}$.
5. Выполняется операция ОС непосредственного порождения *setuid()*, при которой функционально ассоциированный объект порождающего субъекта становится функционально ассоциированным объектом порожденного субъекта.

Таблица 2.1 – Типы запросов к системе по изменению субъектов и функционально ассоциированных объектов с пред- и постусловиями в рамках модели безопасности, $t \in \mathbb{N}_0$ – время выполнения запроса

Тип запроса $q \in Q$	**Описание**	**Предусловия**	**Постусловия**
$create\_process(s,o,o')$	создание функционально ассоциированного объекта o' с помощью объекта o при активизирующем воздействии s	$s \in S^t_{active}$; $(s,o) \in O^t_{func}$; $o' \in O$, $o' \notin O^t$, $o'_t = \varnothing$	$O^{t+1} = O^t \cup \{o'\}$; $O^{t+1}_{func} = O^t_{func} \cup \{(s,o')\}$; $o'_{t+1} = o_t$ до выполнения следующего запроса *exec* с участием o'
$delete\_process(s,o,o')$	уничтожение функционально ассоциированного объекта o' объектом o при активизирующем воздействии s	$s \in S^t_{active}$; $(s,o) \in O^t_{func}$; $o' = o$, $\lvert\{o'' : (s,o'') \in O^t_{func}\}\rvert > 1$	$O^{t+1} = O^t \setminus \{o'\}$; $O^{t+1}_{func} = O^t_{func} \setminus \{(s,o')\}$; $o'_{t+1} = \varnothing$
$create\_user(s,o,s')$	порождения субъекта-пользователя s' из объекта o при активизирующем воздействии субъекта s	$s \in S^t_{active}$, $s' \in S_{users}$; $s = s_0 \in S_{system}$ или $s \in S_{users}$; $(s,o) \in O^t_{func}$; $\lvert\{o' : (s,o') \in O^t_{func}\}\rvert > 1$	$S^{t+1}_{active} = S^t_{active} \cup \{s'\}$; $O^{t+1}_{func} = O^t_{func} \setminus \{(s,o)\} \cup \{(s',o)\}$; если $s' \in S^t_{active}$ – субъект инициировал множество сессий
$create\_shadow(s,o,s')$	порождения системного субъекта s' из объекта o при активизирующем воздействии субъекта s	$s \in S^t_{active}$, $s = s_0 \in S_{system}$; $(s,o) \in O^t_{func}$, $s' \in S_{system} \cup \{s_{sorm}\}$; $s' \notin S^t_{active}$, $\lvert\{o' : (s,o') \in O^t_{func}\}\rvert > 1$	$S^{t+1}_{active} = S^t_{active} \cup \{s'\}$; $O^{t+1}_{func} = O^t_{func} \setminus \{(s,o)\} \cup \{(s',o)\}$
$delete\_subject(s,o,s')$	уничтожение с помощью функционально ассоциированного объекта o всех функционально ассоциированных объектов s' при воздействии s	$s \in S^t_{active} \setminus \{s_0, s_{sorm}\}$; $(s,o) \in O^t_{func}$; $s' = s$	$\forall o' \in O^t$: $(s',o') \in O^t_{func}$ выполнить *delete_process(s',o',o')*; $S^{t+1}_{active} = S^t_{active} \setminus \{s'\}$; $\forall o'' : O^{t+1}_{data} = O^t_{data} \setminus \{(s',o'')\}$
$exec(s,o,o')$	загрузка бинарного образа для выполнения из o' в o при активизирующем воздействии субъекта s	$s \in S^t_{active}$; $(s,o) \in O^t_{func}$, $(s,o') \in O^t_{data}$; $exec(s,o,o')$ разрешен в соответствии с правилами из $o_{sorm} \in O^t$: $(s_{sorm}, o_{sorm}) \in O^t_{data}$	$o_{t+1} = o'_t$; $O^{t+1}_{data} = O^t_{data} \setminus \{(s,o')\}$; $O^{t+1}_{na} = O^t_{na} \cup \{o'\}$ если $\lvert\{s' : (s',o') \in O^{t+1}_{data}\}\rvert = 0$

6. Выполняется операция *exec()*, запускающая на выполнение первый процесс нового субъекта доступа.

В соответствии с этим в Таблице 2.1 для новой модели безопасности ОС и подсистемы управления доступом введено две операции порождения: для субъектов-пользователей $create\_user(s,o,s')$ и системных процессов $create\_shadow(s,o,s')$. При этом из описания процесса порождения субъектов видно, что для возникновения в системе новых субъектов доступа необходимо изначальное присутствие субъекта s_0, а также s_{sorm}. Роль s_0 для операции порождения состоит в том, чтобы ограничить возможность беспрепятственного возникновения других системных субъектов, которым, в отличие от субъектов-пользователей, не требуется выполнять процедуры идентификации и аутентификации, а роль s_{sorm} состоит в контроле порождения субъектов. Возникновение функционально ассоциированных с субъектами объектов (процессов) происходит при осуществлении запросов типа $create\_process(s,o,o')$ уже порожденными субъектами доступа.

Запросы из Q, приведенные в Таблице 2.2 относятся к ассоциированным объектам-данным и неассоциированным объектам, то есть соответствуют операции возникновения информационного потока СО-модели ИПС. Результатом реализации информационного потока при активизирующем воздействии субъекта s является преобразование информации в объекте $o' \in O$: $(s,o') \in O^t_{data}$ или $o' \in O^t_{na}$, либо в объекте $o \in O^t$: $(s,o) \in O^t_{func}$ в зависимости от запроса к системе из Таблицы 2.2. Таким образом, в качестве информационных потоков в GNU/Linux или другой ОС выступают любые обращения субъектов на доступ к объектам – чтение, запись, создание/удаление объектов и так далее.

В результате выполнения любого запроса $q = (s,o,e) \in Q$ система переходит из одного состояния $v_t = (S^t_{active}, O^t_{func}, O^t_{data}, O^t_{na}) \in V$ в момент времени $t \in \mathbb{N}_0$ в состояние $v_{t+1} = (S^{t+1}_{active}, O^{t+1}_{func}, O^{t+1}_{data}, O^{t+1}_{na}) \in V$ в следующий момент времени $t+1$ в соответствии с постусловиями, приведенными в Таблицах 2.1 и 2.2. Обозначим $T = Q \times V \times V$ – множество переходов системы по запросам из одного состояния в другое.

Определение 1. Системой для новой модели безопасности ОС с подсистемой управления доступом называется детерминированный автомат без выходов (V, v_0, Q, f), для которого:

– $V = S_{active} \times O_{func} \times O_{data} \times O_{na}$ – множество состояний автомата, соответствующее состояниям системы;

– $v_0 = (S^0_{active}, O^0_{func}, O^0_{data}, O^0_{na}) \in V$ – начальное состояние автомата, такое что:
  – $S^0_{active} = \{s_0, s_{sorm}\}$;
  – $O^0_{func} = \{(s_0, o_0), (s_{sorm}, o'_{sorm})\}$;
  – $O^0_{data} = \varnothing$;
  – $O^0_{na} = O^0 \subseteq O$;

– $Q = S \times O \times E$ – множество входов автомата, соответствующих введенным ранее запросам к системе из Таблиц 2.1 и 2.2.

– $f : Q \times V \to V$ – функция переходов системы по запросам.

X – множество функций x: $\mathbb{N}_0 \to Q$, задающее все последовательности запросов к системе.

Таблица 2.2 – Типы запросов к системе по изменению ассоциированных объектов-данных или неассоциированных объектов с пред- и постусловиями в рамках модели безопасности, $t \in \mathbb{N}_0$ – время выполнения запроса

Тип запроса $q \in Q$	**Описание**	**Предусловия**	**Постусловия**
$read(s,o,o')$	реализация информационного потока на чтение из объекта o' с помощью ассоциированного объекта o при активизирующем воздействии s	$s \in S^t_{active}$; $(s,o) \in O^t_{func}$; $(s,o') \in O^t_{data}$ или $o' \in O^t_{na}$, $o'_t \neq \varnothing$; $read(s,o,o')$ разрешен в правилах из $o_{sorm} \in O^t$: $(s_{sorm}, o_{sorm}) \in O^t_{data}$	$o_{t+1} \neq o_t$; $O^{t+1}_{data} = O^t_{data} \cup (s,o')$ и $O^{t+1}_{na} = O^t_{na} \setminus \{o'\}$, то есть o' может оказывать влияние на дальнейшее функционирование s посредством o (за счет считанных данных)
$write(s,o,o')$	реализация информационного потока на запись в объект o' с помощью ассоциированного объекта o при активизирующем воздействии s	$s \in S^t_{active}$; $(s,o) \in O^t_{func}$; $(s,o') \in O^t_{data}$ или $o' \in O^t_{na}$; $write(s,o,o')$ разрешен в правилах из $o_{sorm} \in O^t$: $(s_{sorm}, o_{sorm}) \in O^t_{data}$	$o'_{t+1} \neq o'_t$; $(s,o') \in O^{t+1}_{data}$ или $o' \in O^{t+1}_{na}$ (в соответствии с предусловием)
$create(s,o,o')$	реализация информационного потока на создание объекта o' с помощью ассоциированного объекта o при активизирующем воздействии s	$s \in S^t_{active}$; $(s,o) \in O^t_{func}$; $o' \in O$, $o' \notin O^t$, $o'_t = \varnothing$; $create(s,o,o')$ разрешен в правилах из $o_{sorm} \in O^t$: $(s_{sorm}, o_{sorm}) \in O^t_{data}$	$O^{t+1} = O^t \cup \{o'\}$; $O^{t+1}_{na} = O^t_{na} \cup \{o'\}$; $o'_{t+1} = \varnothing$
$delete(s,o,o')$	реализация информационного потока на удаление объекта o' с помощью ассоциированного объекта o при активизирующем воздействии s	$s \in S^t_{active}$; $(s,o) \in O^t_{func}$; $(s,o') \in O^t_{data}$ или $o' \in O^t_{na}$; $delete(s,o,o')$ разрешен в правилах из $o_{sorm} \in O^t$: $(s_{sorm}, o_{sorm}) \in O^t_{data}$	$O^{t+1} = O^t \setminus \{o'\}$; $\forall s' \in S^t_{active} : O^{t+1}_{data} = O^t_{data} \setminus \{(s',o')\}$ или $O^{t+1}_{na} = O^t_{na} \setminus \{o'\}$ (в соответствии с предусловием); $o'_{t+1} = \varnothing$

Y – множество функций y: $\mathbb{N}_0 \to V$, задающее все последовательности состояний системы, $\forall y \in Y$: $y(0) = v_0$, но с различным составом множества $O^0 \subseteq O$.

Для элементов введенного ранее множества всевозможных переходов системы $\forall(q, v_t, v_{t+1}) \in T$ выполняется $f(q,v_t) = v_{t+1}$.

Определение 2. Реализацией системы (V, v_0, Q, f) называется элемент $(x,y) \in X \times Y$, такой, что выполняется условие: $\forall t \in \mathbb{N}_0$, $f(x(t), y(t)) = y(t+1)$, то есть $(x(t),y(t),y(t+1)) \in T$.

Все реализации системы (V, v_0, Q, f) описывают любые возможные начальные состояния v_0 в части состава множества O^0, последовательности запросов из Q и последовательности состояний из V для ОС с подсистемой управления доступом.

Из описанных в Таблицах 2.1 и 2.2 предусловий для запросов к системе очевидно, что критически важная с точки зрения выполнения установленной политики безопасности информация для любой реализации системы (V, v_0, Q, f) в любой момент времени $t \in \mathbb{N}_0$ содержится в объектах $o_{sorm} \in O^t$: $(s_{sorm}, o_{sorm}) \in O^t_{data}$. Так в ассоциированных с ядром защиты объектах доступа содержатся: информация о разрешенных и запрещенных субъектах доступа (состав множеств S_l, S_n, S_{users}, S_{system}, S^t_{active}), эталонные значения идентифицирующей и аутентифицирующей информации для субъектов-пользователей, текущее состояние множеств ассоциированных и неассоциированных объектов (O^t_{func}, O^t_{data} и O^t_{na}), а также правила для разрешенных информационных потоков (запросов к системе $q \in Q$). Таким образом, при изменении ассоциированных с ядром защиты объектов могут измениться его свойства, то есть в дальнейшем станет возможным возникновение запрещенных информационных потоков.

Для исключения такой возможности в СО-модели ИПС вводятся понятия корректности и абсолютной корректности субъектов доступа. Субъекты системы называются корректными относительно друг друга в соответствии с СО-моделью ИПС, когда в любой момент времени отсутствует поток между любыми двумя объектами, ассоциированными соответственно с этими субъектами доступа. Учитывая введенные ранее обозначения и определения, сформулируем более жесткие требования к корректности субъектов доступа системы.

Определение 3. В реализации $(x,y) \in X \times Y$ системы (V, v_0, Q, f) при переходе $(q, v_t, v_{t+1}) \in T$ субъект $s \in S^t_{active}$ корректен относительно субъекта $s' \in S^{t+1}_{active}$, $s \neq s'$ в моменты времени работы системы $t \in \mathbb{N}_0$ и $t+1$, если для запроса $q \in Q$ выполняется одно из условий:

- $\forall o, o' \in O$: $q \neq (s,o',o)$, то есть запрос выполняется без участия s.
- Если для $o, o' \in O$: $q = (s,o',o)$ и $(s,o') \in O^t_{func}$, тогда в постусловии запроса q:
  - Либо $o_t = o_{t+1}$, то есть состояние объекта o не изменилось.
  - Либо $o_t \neq o_{t+1}$ и $(s',o) \notin O^{t+1}_{func}$ (то есть не существует возможности любого межпроцессного взаимодействия между разными субъектами доступа, функционально ассоциированные объекты-процессы выполняются в индивидуальных адресных пространствах и не могут влиять друг на друга).

– Либо $o_t \neq o_{t+1}$ и $(s',o) \notin O_{data}^{t+1}$ (то есть не существует возможности записи/удаления «чужих» ассоциированных объектов-данных).

Субъекты $s \neq s'$: $s \in S_{active}^{t}$ и $s' \in S_{active}^{t+1}$ называются корректными в моменты времени $t \in \mathbb{N}_0$ и $t+1$ если s корректен относительно s', а s' корректен относительно s.

Субъекты называются абсолютно корректными относительно друг друга в соответствии с CO-моделью ИПС, если они корректны, а множества ассоциированных с ними объектов не пересекаются. В CO-модели ИПС постулируется, что абсолютная корректность достижима, если в системе для всех субъектов-процессов используются собственные виртуальные адресные пространства [80] (что уже было введено выше в условиях для корректных субъектов), а доступы к одним и тем же (ассоциированным) объектам не могут выполняться одновременно. В этом и заключается основной недостаток этой модели применительно к современным системам, в которых, например, один и тот же бинарный файл или библиотека могут одновременно быть ассоциированы с несколькими субъектами доступа. В соответствии с этим в подобных системах обеспечение абсолютной корректности субъектов в соответствии с CO-моделью на практике невозможно.

Учитывая изложенные выше положения, необходимо либо исключить из системы объекты, которые могут одновременно или в смежные промежутки времени быть ассоциированы с несколькими субъектами доступа (в новой модели безопасности это объекты, входящие в O_{data}, которые представляют собой общие для субъектов бинарные файлы, разделяемые библиотеки, конфигурационные файлы и переменные окружения, или объекты-процессы O_{func}), либо ограничить информационные потоки или запросы от всех субъектов, которые изменяют данные объекты (например, запросы $write(s,o,o')$ и $delete(s,o,o')$), контролируя при этом их целостность (состояние).

Определение 4. В реализации $(x,y) \in X \times Y$ системы (V, v_0, Q, f) при переходе $(q, v_t, v_{t+1}) \in T$ субъект $s \in S_{active}^{t}$ абсолютно корректен относительно субъекта $s' \in S_{active}^{t+1}$, $s \neq s'$ в моменты времени работы системы $t \in \mathbb{N}_0$ и $t+1$, если для запроса $q \in Q$ выполняется одно из условий:

– s и s' корректны в моменты времени t и $t+1$ в соответствии с Определением 3.

– $\forall o, o' \in O$: $q \neq (s,o',o)$, то есть запрос выполняется без участия s.

– Если для $o, o' \in O$: $q = (s,o',o)$ и $(s,o') \in O_{func}^{t}$, тогда в постусловии запроса q:

– Либо $o_t = o_{t+1}$, то есть состояние объекта o не изменилось.

– Либо $o_t \neq o_{t+1}$ и для реализации (x,y): $\nexists\, l > t+1$ и $v_l = y(l)$: $(s',o) \in O_{data}^{l}$ (то есть нельзя выполнять информационный поток, изменяющий состояние объекта доступа, который в дальнейшем будет являться объектом-данными другого субъекта доступа).

– Либо $o_t \neq o_{t+1}$ и при $s \neq s_0$ для реализации (x,y): $\nexists\, l > t+1$ и $v_l = y(l)$: $(s',o) \in O_{func}^{l}$.

Субъекты $s \neq s'$: $s \in S_{active}^{t}$ и $s' \in S_{active}^{t+1}$ называются абсолютно корректными в моменты времени $t \in \mathbb{N}_0$ и $t+1$ если s абсолютно корректен относительно s', а s' абсолютно корректен относительно s.

Для дальнейшего формирования условий невозможности возникновения запрещенных информационных потоков в системе сформулируем еще несколько ключевых определений.

Определение 5. Переход $(q, v_t, v_{t+1}) \in T$ системы (V, v_0, Q, f) обладает свойством корректности (абсолютной корректности) субъектов доступа, если все $s \in S^t_{active}$ и $s' \in S^{t+1}_{active}$ корректны относительно друг друга в моменты времени t и $t+1$.

Определение 6. Реализация $(x,y) \in X \times Y$ системы (V, v_0, Q, f) обладает свойством корректности (абсолютной корректности) субъектов доступа, если $\forall t \in \mathbb{N}_0$ все переходы $(x(t), y(t), y(t+1)) \in T$ обладают свойством корректности (абсолютной корректности) субъектов доступа.

Определение 7. Система (V, v_0, Q, f) обладает свойством корректности (абсолютной корректности) субъектов доступа, если каждая ее реализация обладает свойством корректности (абсолютной корректности) субъектов доступа.

Система, обладающая свойством абсолютной корректности субъектов доступа, называется **изолированной программной средой субъектов** (ИПСС).

Теперь для окончательного формирования модели безопасности ОС и подсистемы управления доступом сформулируем и докажем утверждение, в соответствии с которым в системе обеспечивается выполнение заданной политики безопасности, то есть гарантируется возможность возникновения только разрешенных и невозможность возникновения запрещенных информационных потоков (по аналогии с так называемыми критериями безопасности других моделей безопасности [13]).

Утверждение (*Базовая теорема ИПСС*). *В системе (V, v_0, Q, f) невозможно нарушение действующей политики управления доступом тогда и только тогда, когда она является ИПСС.*

Доказательство. Докажем вначале необходимость, для этого обозначим с помощью $t \in \mathbb{N}_0$ – произвольный момент работы некоторой реализации системы (V, v_0, Q, f).

Из описания запросов из Таблиц 2.1 и 2.2 следует, что в системе множество функционально ассоциированных с любым субъектом доступа объектов изменяется только при реализации запросов типа $read(s,o,o')$, $exec(s,o,o')$, $create\_process(s,o,o')$, $create\_user(s,o,s')$ и $create\_shadow(s,o,s')$, изменение же объектов-данных может происходить только при реализации запросов типа $write(s,o,o')$, $create(s,o,o')$ и $delete(s,o,o')$. Запросы $delete\_process(s,o,o')$ и $delete\_subject(s,o,s')$ изменяют ассоциированные объекты, которые уже не могут быть переданы другим субъектам доступа за счет особенности изменения множества O^t_{func} во времени и поэтому далее рассматриваться не будут.

В соответствии с описанием запросов из Таблиц 2.1 и 2.2 нарушение действующей политики управления доступом в системе может возникнуть если произошло:

1. Нарушение предусловий или постусловий запросов на порождение субъектов доступа $create\_user(s,o,s')$ или $create\_shadow(s,o,s')$ (например, $s' \notin S_{users}$, идентификационная или аутентификационная информация не совпадает со значением в $o_{sorm} \in O^t$: $(s_{sorm}, o_{sorm}) \in O^t_{data}$,

$s' \notin S_{system} \cup \{s_{sorm}\}$), в результате чего возник несанкционированный субъект, либо несанкционированно возник санкционированный субъект, либо санкционированный субъект доступа не может появиться.

2. Реализован информационный поток типа $exec(s,o,o')$, $read(s,o,o')$, $write(s,o,o')$, $create(s,o,o')$ или $delete(s,o,o')$, запрещенный текущему субъекту доступа, от имени другого субъекта доступа.

3. Нарушение предусловий или постусловий запросов по реализации информационных потоков $exec(s,o,o')$, $read(s,o,o')$, $write(s,o,o')$, $create(s,o,o')$ или $delete(s,o,o')$ (запрос разрешен из-за изменения правил из $o_{sorm} \in O^t$: $(s_{sorm}, o_{sorm}) \in O^t_{data}$), в результате чего был выполнен запрос, который должен быть запрещен.

Предположим, что в обозначенное время работы некоторой реализации системы t невозможно нарушение действующей политики управления доступом, значит в текущем и предыдущих состояниях системы одновременно невозможны все нарушения, описанные выше. Рассмотрим каждое из них подробнее.

В первом и втором случаях нарушения действующей политики управления доступом в некоторый момент времени $l \leqslant t$ была изменена процедура контроля порождения субъектов доступа или были модифицированы соответствующие данные из $o_{sorm} \in O^l$: $(s_{sorm}, o_{sorm}) \in O^l_{data}$. Порождение субъектов в любой ОС происходит с помощью процессов из нескольких фиксированных объектов, например, в GNU/Linux это объекты */bin/login*, */bin/su*, */usr/bin/sudo* и некоторые другие (для $s' \in S_{users}$), бинарные файлы системных процессов ($s' \in S_{system} \cup \{s_{sorm}\}$). При этом в любом случае ассоциированными объектами в процессе порождения новых субъектов задействуются другие объекты (посредством реализации запросов $read(s,o,o')$ и последующих $exec(s,o,o')$): PAM-модули и разделяемые библиотеки, которые также на некоторый промежуток времени становятся ассоциированными с порождающим субъектом доступа (см. разд. 1.4), то есть влияют на них функционально. Во втором случае нарушение контроля порождения субъекта также может быть осуществлено с помощью изменения функционально ассоциированного объекта запросами типа $create\_process(s,o,o')$, $read(s,o,o')$ и $exec(s,o,o')$ с последующим $create\_user(s,o,s')$ или $create\_shadow(s,o,s')$, но только в случае если $create\_user(s,o,s')$ или $create\_shadow(s,o,s')$ выполняются для $s \neq s_0$.

В соответствии с этим изменение процедуры контроля порождения субъектов возможно если в момент времени l был выполнен запрос $write(s,o,o')$, $create(s,o,o')$ или $delete(s,o,o')$ в указанные объекты или в объект $o_{sorm} \in O^l$: $(s_{sorm}, o_{sorm}) \in O^l_{data}$, либо выполнены запросы $create\_process(s,o,o')$, $read(s,o,o')$ и $exec(s,o,o')$ с последующими $create\_user(s,o,s')$ или $create\_shadow(s,o,s')$ для $s \neq s_0$. Таким образом изменение процедуры контроля порождения субъектов в произвольный момент t невозможно, если в системе, начиная с момента 0 во все последующие моменты $l \leqslant t$ невозможно изменение состояния перечисленных объектов, для которых при произвольном $s \in S$ может выполняться $(s,o) \in O^l_{data}$ или $(s,o) \in O^l_{func}$, что соответствует условиям Определений 3–4 и говорит об абсолютной корректности субъектов до момента t.

В третьем случае нарушения действующей политики управления доступом в некоторый момент времени $l \leqslant t$ была изменена процедура фильтрации информационных потоков или модифицированы данные из $o_{sorm} \in O^l$: $(s_{sorm}, o_{sorm}) \in O^l_{data}$. Изменение объекта ядра защиты o_{sorm} или другого общедоступного объекта является влиянием посредством функционально ассоциированных объектов (влияние на функционально ассоциированный объект-процесс) или объектов-данных, то есть это аналогично первому и второму случаю и с помощью аналогичных рассуждений сводится к тому же результату.

В соответствии с произвольным выбором момента времени t и реализации системы из вышесказанного следует, что:

– свойство абсолютной корректности всех субъектов выполняется для переходов системы в текущей реализации по Определению 5;

– свойство абсолютной корректности выполняется для всей реализации системы по Определению 6;

– свойство абсолютной корректности выполняется для всей системы (V, v_0, Q, f), то есть по Определению 7 система является ИПСС.

Достаточность утверждения следует напрямую из Определений 4–7. Если система является ИПСС, то каждая ее реализация по Определению 7 обладает свойством абсолютной корректности субъектов доступа. По Определению 6 это означает, что все переходы каждой реализации обладают свойством абсолютной корректности субъектов доступа. То есть с учетом Определения 5 при каждом переходе каждой реализации системы все субъекты абсолютно корректны относительно друг друга. По Определению 4 это значит, что ни в одной реализации системы не мог выполниться запрос, изменяющий состояние объекта доступа, который в дальнейшем будет являться функционально ассоциированным объектом или объектом-данными другого субъекта доступа, что противоречит описанным возможностям нарушения действующей политики управления доступом для системы.

□

В общем случае для достижения ИПСС необходимо соответствующим образом реализовать ядро защиты s_{sorm} в части правил на ограничение запросов и убедиться в попарной абсолютной корректности субъектов системы относительно друг друга. С другой стороны организация ИПСС может состоять из двух этапов (см. разд. 1.3): предопределенного выполнения начальной фазы с активизацией ядра защиты и работы в режиме ИПСС. В СО-модели ИПС постулируется возможность достижения ИПС в случае неизменности определенных субъектов/объектов до начальной фазы работы системы технологически (то есть активизация ядра защиты и абсолютная корректность субъектов предполагается по умолчанию). Однако в предлагаемой модели безопасности ОС и подсистемы управления доступом возможно описать процесс достижения ИПСС. Необходимо отметить, что абсолютная корректность субъектов подразумевает, что информационные потоки на запись (изменение объектов) нельзя совершать в объекты, которые в какой-то момент времени будут функционально ассоциированными объектами или объектами-данными с некоторым (другим) субъектом доступа (например, в общедоступные бинарные объекты доступа или файлы конфигурации). Также существует особенность, связанная с тем, что до порождения

s_{sorm} ограничения на запросы к системе фактически не могут быть реализованы, что может в дальнейшем повлечь возникновение запрещенных информационных потоков. Поэтому для того, чтобы обеспечить достижимость ИПСС необходимо сформулировать следующее определение.

Определение 8. В реализации $(x,y) \in X \times Y$ системы (V, v_0, Q, f) субъект $s \in S^t_{active}$ абсолютно корректен в обратном смысле относительно субъекта $s' \in S^t_{active}$, $s \neq s'$ в момент времени работы системы $t \in \mathbb{N}_0$, если выполняется:

– Либо $\forall o, o' \in O$: $(s',o) \in O^t_{data}$ в реализации (x,y): $\nexists\, l < t-1$ и состояний $v_l = y(l)$, $v_{l+1} = y(l+1)$, для которых в постусловиях запроса $x(l) = (s,o',o)$: $o_l \neq o_{l+1}$ (то есть объектами-данными не могут становиться объекты, ранее измененные другим субъектом доступа).

– Либо при $s' \neq s_0$ для $\forall o, o' \in O$: $(s',o) \in O^t_{func}$ в реализации (x,y): $\nexists\, l < t-1$ и состояний $v_l = y(l)$, $v_{l+1} = y(l+1)$, для которых в постусловиях запроса $x(l) = (s,o',o)$: $o_l \neq o_{l+1}$.

Субъекты $s \neq s'$: $s, s' \in S^t_{active}$ называются абсолютно корректными в обратном смысле в момент времени работы системы $t \in \mathbb{N}_0$, если s абсолютно корректен в обратном смысле относительно s' и s' абсолютно корректен в обратном смысле относительно s.

Определение 9. Переход $(q, v_t, v_{t+1}) \in T$ системы (V, v_0, Q, f) обладает свойством абсолютной корректности субъектов доступа в обратном смысле, если все субъекты $s \in S^t_{active}$ в начальном состоянии v_t и все субъекты $s' \in S^{t+1}_{active}$ в конечном состоянии v_{t+1} абсолютно корректны в обратном смысле относительно друг друга в моменты времени t и $t+1$ соответственно.

Следствие (*о достижении безопасного начального состояния*)**.** В системе (V, v'_0, Q, f), в которой, в отличие от Определения 1 системы (V, v_0, Q, f), начальное состояние имеет вид $v'_0 = (\{s_0\}, \{(s_0, o_0)\}, O^0_{data}, O^0_{na}) \in V$ невозможно нарушение действующей политики управления доступом тогда и только тогда, когда $\forall (x,y) \in X \times Y$: $\exists t \in \mathbb{N}$ и выполняются условия:

1. $\forall\, l < t-1$: выполняется $S^l_{active} = \{s_0\}$.
2. $x(t-2) = (s_0, o'_{sorm}, s_{sorm})$, в постусловиях которого $s_{sorm} \in S^{t-1}_{active}$, $(s_{sorm}, o'_{sorm}) \in O^{t-1}_{func}$, и переход $(x(t-2), y(t-2), y(t-1))$ обладает свойством абсолютной корректности субъектов в обратном смысле.
3. $x(t-1) = (s_{sorm}, o'_{sorm}, o_{sorm})$, в постусловиях которого $(s_{sorm}, o_{sorm}) \in O^t_{data}$ и переход $(x(t-1), y(t-1), y(t))$ обладает свойством абсолютной корректности субъектов в обратном смысле.
4. $\forall\, l > t$: $(x(l), y(l), y(l+1))$ обладает свойством абсолютной корректности субъектов в обратном смысле.

В случае нарушения условий 1–3 дальнейшая работа КС должна прерываться, то есть последующие запросы не выполняются. В случае нарушения условия 4 соответствующий запрос $x(l)$ должен блокироваться за счет правил доступа из $o_{sorm} \in O^l$: $(s_{sorm}, o_{sorm}) \in O^l_{data}$ либо работа системы должна прерываться.

В условии 2 выполняется запрос ОС типа $create\_shadow(s_0, o'_{sorm}, s_{sorm})$ – загрузка средства разграничения доступа. В условии 3 выполняется запрос ОС типа $read(s_{sorm}, o'_{sorm}, o_{sorm})$ – чтение/загрузка правил управления доступом.

Доказательство. Справедливость следствия следует напрямую из Базовой теоремы ИПСС и введенного определения абсолютной корректности в обратном смысле.

Обозначим момент t из условия как момент достижения ИПСС – t_0.

По условию 1 до момента $(t_0 - 1)$ существует только один субъект доступа – s_0, это условие используется далее в условии 2.

После порождения s_{sorm} возможны предыдущие запросы к системе, изменяющие состояние функционально ассоциированных с ним объектов или объектов-данных, поэтому для невозможности нарушения действующей политики управления доступом необходимо выполнить условия 2 и 3. В результате будет гарантировано отсутствие нарушения абсолютной корректности s_{sorm} и s_0 (в прямом смысле) до момента t_0 и обеспечена невозможность дальнейшей работы системы без s_{sorm}.

Условие 4 обеспечивает выполнение соответствующих запросов к системе во все последующие моменты времени. За счет выполнения свойства абсолютной корректности в обратном смысле у субъектов доступа, непосредственно с момента времени t_0 не могут появиться функционально ассоциированные объекты или объекты данные, которые были изменены другими субъектами доступа. Поэтому во всех состояниях системы (V, v'_0, Q, f) не будет нарушаться и свойство абсолютной корректности (в прямом смысле), а в соответствии с Базовой теоремой ИПСС в системе (V, v'_0, Q, f) невозможно нарушение действующей политики управления доступом. □

Необходимо отметить, что ассоциированные с ядром защиты объекты o_{sorm} играют ключевую роль в работе ИПСС, так как при их изменении возможно «размыкание» программной среды и влияние некоторых субъектов доступа на другие субъекты. В этих условиях возникает вопрос возможности администрирования, то есть возможности в некоторый момент $t \in \mathbb{N}_0$ для некоторых $s \in S_l$, $o \in O^t$: $(s,o) \in O^t_{func}$ реализовать запрос $write(s,o,o_{sorm})$. Так, например, в политике абсолютного разделения административных и пользовательских полномочий [13] описываются требования для защиты от несанкционированного использования нарушителем особых полномочий в КС, которые необходимы доверенным пользователям для выполнения административных действий. Однако требование абсолютной корректности всех субъектов (V, v_0, Q, f) не позволяет выполнять в системе следующие необходимые административные действия: создание и изменение объектов, доступных на чтение и выполнение всем субъектам доступа (например, бинарные объектные файлы, разделяемые библиотеки, конфигурационные файлы и переменные окружения); изменение правил политики управления доступом; изменение состава разрешенных и запрещенных субъектов; изменение идентифицирующей и аутентифицирующей информации субъектов-пользователей и другие.

Для сохранения возможности администрирования ИПСС в соответствии с политикой абсолютного разделения административных и пользовательских полномочий требуется выделить отдельный субъект доступа (администратор, $s_{admin} \in S_l$), которому будут доступны исключительные права на модификацию описанных выше объектов и фиксацию их эталонного состояния при изменении (для дальнейшего контроля неизменности). Однако даже если обеспечить отсутствие сессий других субъектов-пользователей и системных процессов в момент админи-

стрирования $t \in \mathbb{N}_0$ ($S^t_{active} = \{s_0, s_{sorm}, s_{admin}\}$), то данный административный субъект доступа некорректен (абсолютно в прямом или обратном смысле) относительно s_0 и s_{sorm}.

Таким образом, единственно возможным вариантом администрирования остается выключение (V, v_0, Q, f) и ее краткосрочный перевод в специальное состояние для администрирования, в котором будет присутствовать только субъект s_{admin} (вместо s_0), а ядро защиты s_{sorm} будет выключено (например, single user mode или recovery mode в GNU/Linux).

Необходимо также уточнить, что на практике представляет собой выполнение требований корректности и абсолютной корректности (в прямом или обратном смысле) для всех субъектов доступа. По сути эти свойства накладывают следующие ограничения:

– ассоциированные с субъектами объекты-процессы выполняются в изолированных виртуальных адресных пространствах, а также отсутствует возможность прямого межпроцессного взаимодействия между разными субъектами доступа;

– введен запрет изменения всеми субъектами общих бинарных объектных файлов, разделяемых библиотек, конфигурационных файлов, переменных окружения и других объектов (с помощью правил политики управления доступом для выполнения свойства абсолютной корректности);

– при каждом доступе к перечисленным выше объектам выполняется контроль целостности, в случае нарушения целостности доступ запрещается (с помощью правил политики управления доступом для выполнения свойства абсолютной корректности в обратном смысле).

Необходимо также отметить, что в Определениях 3, 4 и 8 учитывается только случай $s \neq s'$, то есть субъекты системы должны быть разными. Однако в ряде ОС существует возможность создания множества одновременных сессий одного и того же субъекта-пользователя или каскадных сессий разных субъектов доступа. Одновременные сессии можно считать тождественными или дополняющими друг друга (на основании влияния одной сессии на другую через общие конфигурационные файлы и другие объекты) при неизменности всех объектов, связанных с их объектами-источниками в моменты порождения этих сессий. Однако в общем случае и одновременные сессии, и каскадные сессии не изолированы, свойства корректности и абсолютной корректности для них не могут быть обеспечены. Это также является актуальным при различных полномочиях разных сессий одного субъекта-пользователя (например, различном текущем уровне доступа в рамках мандатной политики управления доступом). В соответствии с этим при запросах к системе типа $create\_user(s,o,s')$ необходимо ограничить возможность одновременных сессий и каскадных сессий в ОС средствами подсистемы управления доступом, в том числе разрешить выполнение таких запросов только субъекту s_0.

Главными условиями Следствия к Базовой теореме ИПСС, которых необходимо достичь на практике, являются обеспечение активизации s_{sorm} в качестве первого субъекта системы после s_0 в обозначенный безопасный начальный момент работы системы t_0 и дальнейшее функционирование правил доступа из ассоциированных объектов o_{sorm}. Данный аспект будет исследован далее в данной главе.

Необходимо отметить, что ИПСС можно рассматривать как ИПС (для субъектов-процессов) внутри ИПС (для субъектов-пользователей и системных субъектов). Понятия корректности и аб-

солютной корректности (в прямом или обратном смысле) ИПСС значительно шире аналогичных понятий ИПС: в системе ограничивается порождение как субъектов доступа, так и функционально ассоциированных с ними объектов (процессов) за счет конечности набора правил для разрешенных информационных потоков в o_{sorm} для любого $t \in \mathbb{N}_0$. Более того, за счет свойств абсолютной корректности (в прямом или обратном смысле) обеспечивается не только выполнение заданной политики управления доступом, но и изоляция пользовательских сред субъектов доступа. Таким образом, например, появление вредоносного ПО в рамках сессии одного субъекта никак не повлияет на других субъектов доступа.

Приведенные Теорема и Следствие несут важный практический смысл безопасного субъектно-объектного взаимодействия в системе. При выполнении сформулированных выше условий в системе при любых ее реализациях во все последующие моменты времени гарантируется выполнение заданной политики безопасности. При этом вне зависимости от сущности правил управления доступом будет невозможно возникновение запрещенных информационных потоков, что и является конечной целью создания любого средства разграничения доступа. Важным также является, то, что ИПСС является инвариантной относительно действующей в КС политики управления доступом, если ее правила не нарушают положений для ИПСС. Это говорит о том, что ИПСС может корректно сочетаться и взаимодействовать с любой другой моделью политик управления доступом.

Таким образом, приведенное выше формальное описание системы вместе с введенной Базовой теоремой ИПСС и Следствием формируют модель безопасности ОС с подсистемой управления доступом, для которой обоснована теоретическая невозможность возникновения неразрешенных информационных потоков в обход действующих правил управления доступом. Все используемые при формировании данной модели положения в достаточной степени общие, в соответствии с чем, сама модель безопасности имеет широкое назначение и может быть использована как для ОС GNU/Linux, так и для множества других систем.

2.2. Идентификация субъектов и объектов доступа при их взаимодействии в системе

В рамках рассмотренной в разделе 2.1 модели безопасности уже учтены такие направления совершенствования существующих средств разграничения доступа как централизованная схема администрирования с исключением прав владения для передачи прав доступа на объекты (за счет возможности администрирования подсистемы управления доступом только в специальные промежутки времени и только специальным субъектом доступа) и применение правил доступа для всех объектов системы (а не только для защищаемых данных).

В соответствии с разделом 1.4 еще одним направлением совершенствования существующих средств разграничения доступа является идентификация субъектов и объектов при их взаимодействии для последующего применения принципа наименьших привилегий. При этом в рамках предложенной модели безопасности вопрос идентификации объектов не рассматривается совсем, а для субъектов доступа введены запросы из Таблицы 2.1, позволяющие ограничить их порождение, а также первоначальный субъект ОС (s_0, суперпользователь на раннем этапе загрузки), существование которого в системе предполагается априорно. Однако в предложенной

модели безопасности не предлагается конкретного способа идентификации субъектов (то есть определения принадлежности определенных процессов тому или иному субъекту), особенно на раннем этапе загрузки системы, когда операции порождения еще не реализованы. В соответствии с этим предложим способы идентификации субъектов и объектов, которые бы устранили соответствующие пробелы в рассмотренной модели безопасности.

2.2.1. Идентификация субъектов доступа системы и контроль их порождения

Как было показано в разделах 1.1.1 и 2.1, субъекты доступа в GNU/Linux представлены группами процессов ОС, запущенных от имени реальных пользователей системы или системных субъектов. Рассмотрим подробнее жизненный цикл процессов ОС GNU/Linux на основе [26, 65, 66], способы их возникновения и выполнения от имени определенного пользователя или субъекта системы.

На самом раннем этапе загрузки ОС GNU/Linux, а именно, в момент загрузки ядра создается самый первый процесс – *swapper* или *sched* (процесс с идентификатором PID 0), выполняющийся фактически как функция ядра Linux и отвечающий за организацию управления памятью и другие возможности. В ходе дальнейшей загрузки ядра Linux запускается первый процесс пользовательского режима *init* (PID 1), который является родителем всех запускаемых в ОС процессов, в том числе и пользовательских. По умолчанию перечисленные выше процессы выполняются от имени суперпользователя с UID 0 (s_0 в предложенной модели безопасности).

При выполнении каждый процесс может создавать новые дочерние процессы (англ. child process), по отношению к которым он будет родительским процессом (англ. parent process), что изображено на Рисунке 2.1. Такие новые процессы могут запускать на выполнение другие задачи (операции *fork()* и *exec()*) или выполняться дальше (операция *fork()*), с возможностью порождения других процессов в обоих случаях.

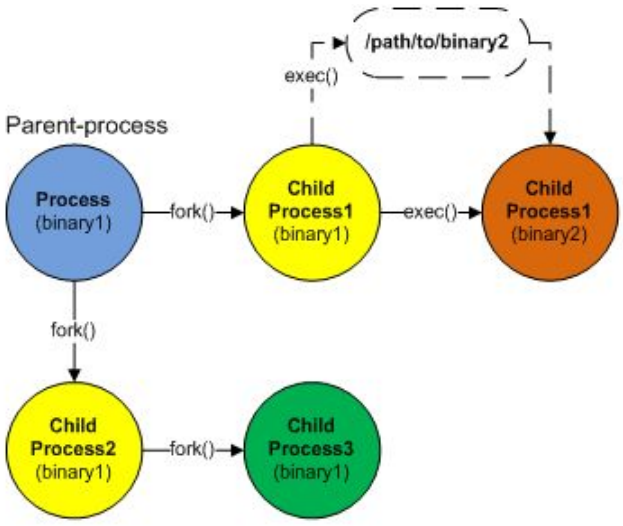

Рисунок 2.1 – Схематичное представление операции создания новых процессов ОС GNU/Linux

Таким образом, все процессы ОС GNU/Linux в определенный момент времени можно представить в виде дерева, корнем которого является процесс *init*. Необходимо отметить, что при со-

здании дочерних процессов последние наследуют большинство параметров от своего родительского процесса, в том числе и UID. Но не все процессы даже во время загрузки ОС GNU/Linux будут постоянно иметь UID 0, наследуя его от процесса init. Существует возможность изменить текущий UID процесса с помощью системного вызова *setuid()*. Так, в ходе загрузки ОС GNU/Linux, некоторые процессы будут иметь UID, соответствующий определенным системным сервисам, для которых существуют учетные записи (например, apache, mysql и другие, у которых обычно не существует возможности прохождения идентификации и аутентификации). В ходе стандартной процедуры идентификации и аутентификации в ОС GNU/Linux UID некоторого процесса, который в дальнейшем запустит сессию и командный интерпретатор пользователя, будет заменен на UID реального пользователя.

В связи с описанным выше подсистема логического управления доступом в ОС GNU/Linux должна в любой момент времени однозначно определять, какой субъект осуществляет тот или иной доступ, то есть реализовывать корректный способ идентификации субъектов доступа [65]. В соответствии с этим можно предложить способ контроля порождения субъектов доступа и идентификации субъектов, от имени которых соответствующие процессы ОС GNU/Linux взаимодействуют с объектами системы, заключающийся в выполнении следующей последовательности действий [26]:

1. Пометить специальным дескриптором безопасности процессы специального субъекта доступа shadow root (от имени которого будут выполняться большинство процессов во время загрузки, включая init) все процессы ОС, уже существующие на момент загрузки подсистемы управления доступом (процессы s_0 из модели безопасности).

2. Начиная с момента загрузки подсистемы управления доступом, помечать все вновь создаваемые процессы дескрипторами безопасности (при выполнении операций *fork()* и *exec()*), соответствующими дескриптору родительского процесса.

3. Во время выполнения санкционированной операции *setuid()* (после прохождения процедуры идентификации и аутентификации пользователя или при порождении системных субъектов по запросам $create\_user(s,o,s')$ и $create\_shadow(s,o,s')$ из модели безопасности) соответствующим образом изменять дескриптор безопасности процесса [15].

Описанный выше субъект доступа shadow root представляет собой «псевдопользователя» ОС GNU/Linux, имеющего максимальные привилегии в системе и осуществляющего все операции по загрузке ОС, запуску необходимых сервисов, активизации механизмов идентификации и аутентификации пользователей и так далее[1]. Так как во время запуска различных сервисов в ОС GNU/Linux могут возникать процессы с UID, отличным от 0, то необходимо предусмотреть отдельный тип субъектов «псевдопользователей» (будем обозначать такой тип *shadow*, как в предложенной модели безопасности). Для субъектов этого типа должны применяться правила разграничения доступа, как и для реальных пользователей ОС GNU/Linux (далее тип субъектов *user*), но возникать они могут без прохождения идентификации и аутентификации при выполнении операции *setuid()* (обычно от имени shadow root, например, при запуске сервисов).

[1]Этот субъект должен отличаться от реального пользователя root, также имеющего $UID = 0$, что позволит разделить реальных пользователей системы от «псевдопользователей».

В приведенном способе идентификации субъектов в каждый момент времени в ОС GNU/Linux невозможно существование процессов, которые не помечены каким-либо дескриптором безопасности [26, 74, 75]. При этом для гарантии корректности идентификации субъектов доступа необходимо осуществлять только санкционированные операции *setuid()* в соответствии со следующими правилами:

1. Изменять субъект доступа типа *user* на другой субъект доступа типа *user* можно только в случае успешного прохождения процедур идентификации и аутентификации ($create\_user(s,o,s')$) одним из процессов-предков текущего процесса и при условии, что UID в *setuid()* совпадает с UID из процедуры идентификации. Данную возможность каскадных сессий, однако, требуется заблокировать, если требуется обеспечить соответствие модели безопасности из раздела 2.1.

2. Изменять субъект доступа типа *shadow* необходимо:

– либо на субъект доступа типа *user*, если проведена идентификация и аутентификация и UID в *setuid()* совпадает с UID из этой процедуры (это характерно для стандартного входа пользователя в ОС GNU/Linux, как в $create\_user(s,o,s')$), а если требуется обеспечить соответствие модели безопасности из раздела 2.1 должна также отсутствовать другая активная сессия субъекта-пользователя;

– либо на другой субъект доступа типа *shadow*, если текущий и заменяющий UID оба равны 0 и UID в *setuid()* не совпадает с UID из процедуры идентификации и аутентификации или если новый субъект *shadow* существует и у текущего субъекта есть разрешение делать *setuid()* (обычно требуется только shadow root в соответствии с $create\_shadow(s,o,s')$).

В предложенном способе идентификации субъектов не будет нарушаться абсолютная корректность субъектов системы из раздела 2.1, так как в описании нет противоречия с определениями запросов $create\_user(s,o,s')$ и $create\_shadow(s,o,s')$. Положения предложенной модели безопасности могут нарушаться в случае реализации множества сессий одного пользователя, а также каскадных сессий разных субъектов доступа (порождение субъектом-пользователем сессии нового субъекта-пользователя), в соответствии с этим такие возможности были исключены в соответствующих пунктах предложенного способа идентификации субъектов.

Для корректной работы идентификации субъектов в подсистеме управления доступом необходимо создать пользователей типа *shadow* (соответствующие необходимым субъектам для правильной работы всех сервисов). Для создания таких пользователей можно предусмотреть режим работы подсистемы логического управления доступом, в котором все операции *setuid()* будут фиксироваться и заноситься в журнал событий. В дальнейшем из журнала событий можно автоматически выделить требуемые субъекты типа shadow и занести их в список пользователей подсистемы управления доступом. Также для всех субъектов типа shadow должны существовать параметры с разрешением или запрещением производить от их имени операции *setuid()*, предназначенные для исключения возможности эскалации привилегий (по умолчанию должны запрещать *setuid()*). Кроме того, в подсистеме управления доступом необходимо использовать специальный PAM-модуль идентификации и аутентификации, одной из задач которого является

«привязка» результатов идентификации и аутентификации к процессам ОС до последующего выполнения *setuid()* [20].

Таким образом, разработанный автором и описанный выше способ контроля порождения субъектов доступа и идентификации субъектов, от имени которых соответствующие процессы ОС GNU/Linux взаимодействуют с объектами системы, в отличие от применяемых в существующих средствах разграничения доступа в GNU/Linux способов контроля за субъектами (см. раздел 1.4) позволяет:

– в точности описать реальное поведение процессов ОС;

– осуществлять более точный контроль действий реальных пользователей системы посредством разделения всех субъектов на типы shadow и user (с собственными особенностями);

– исключить возможность создания множества сессий одного субъекта и каскадных сессий различных субъектов доступа;

– избавиться от избыточных (лишних) прав доступа для субъектов ОС (доступ к хэшированным паролям и списку учетных записей в /etc/passwd и /etc/shadow, доступ к активизирующимся при старте графической среды сервисам и другим), позволяя, тем самым, в дальнейшем использовать принцип наименьших привилегий для субъектов в рамках подсистемы управления доступом;

– непосредственно после активизации подсистемы управления доступом реализовать запросы $create\_user(s,o,s')$ и $create\_shadow(s,o,s')$, а также сформировать начальный состав множеств системных субъектов (S_{system}) и объектов-процессов, ассоциированных с субъектами доступа (O_{func}), для последующего выполнения других запросов из предложенной модели безопасности.

2.2.2. Идентификация объектов при взаимодействии с субъектами доступа

Существующие средства защиты информации от НСД для ОС GNU/Linux реализуют различные способы идентификации объектов доступа (см. разд. 1.4), обладающие своими недостатками и ограничениями по использованию [23]. В рамках предложенной в разделе 2.1 модели безопасности ОС и подсистемы управления доступом предполагается, что все объекты системы однозначно идентифицированы, то есть при возникновении любого информационного потока к объекту проверяются права доступа, заданные для этого самого объекта в некоторый предыдущий момент времени. Однако при применении некоторых способов идентификации объектов в разные моменты времени используемый идентификатор (например, абсолютный путь в ФС) может соответствовать другому объекту доступа. Далее предложим способы идентификации объектов при их взаимодействии с субъектами доступа, которые устраняют выявленные недостатки. Кроме того, предложим другие перспективные способы идентификации объектов доступа, после чего оценим целесообразность использования каждого из рассмотренных способов для применения в составе подсистемы управления доступом ОС GNU/Linux в дополнение к модели безопасности из раздела 2.1.

Наиболее естественным способом идентификации объектов доступа является указание их абсолютного пути в рамках файловой системы ОС GNU/Linux. При всей простоте данный способ обладает следующими существенными недостатками:

– неоднозначность абсолютного пути до защищаемого объекта при изменении правил монтирования файловых систем, монтировании каталогов или смене корневого каталога в результате *chroot* (объект будет доступен по другому абсолютному пути);

– несвязанность абсолютного пути с объектом при переименовании или в случае если объект доступа является символьной или жесткой ссылкой.

Перечисленные недостатки приводят к тому, что в подсистеме управления доступом, использующей такой способ идентификации объектов, кроме непосредственно механизмов разграничения доступа, должны присутствовать следующие механизмы [23]:

– контроль точек монтирования – то есть контроль монтирования разделов (каталогов) в различные точки ФС в соответствии с заранее составленным списком контроля монтирования;

– контроль создания жестких ссылок на объекты доступа;

– «пополнение» или изменение прав доступа при создании новых, переименовании существующих объектов доступа и в иных необходимых случаях.

При этом в GNU/Linux не всегда на уровне ядра ОС существует возможность определить для объекта точку монтирования соответствующего раздела, кроме того, для раздела может одновременно существовать множество точек монтирования. В данном случае целесообразно перебирать всевозможные точки монтирования и запрещать доступ, если он запрещен для объекта хотя бы в одной точке монтирования.

Другим способом идентификации объектов в ОС GNU/Linux является использование метаданных файловых объектов для хранения прав доступа совместно с защищаемыми данными (например, в некотором дополнительном атрибуте файлового объекта). Используя такой способ хранения прав доступа, удается избежать основных недостатков идентификации объектов доступа по абсолютному пути [23]:

– в случае изменения порядка монтирования ФС в ОС GNU/Linux, а также при монтировании каталога с защищаемыми объектами в другую точку монтирования и при смене корневого каталога, «связанность» прав доступа с защищаемыми данными не нарушается;

– в случае переименования объекта доступа или при обращении к этому объекту с использованием ссылок права доступа остаются ассоциированными с объектом.

Кроме того, даже в случае переноса носителя информации на другое СВТ, можно возобновить работу подсистемы управления доступом, так как атрибуты будут переноситься вместе с объектами. При использовании такого способа идентификации объектов и хранения прав доступа отсутствует необходимость в контроле точек монтирования, однако должен присутствовать механизм, назначающий соответствующие права доступа при создании новых объектов. Недостатком хранения прав доступа в метаданных файловых объектов является фактическое изменение самих данных (на уровне метаданных), а также зависимость от типа используемых файловых систем.

В качестве дополнения к рассмотренному способу идентификации объектов по абсолютным путям можно использовать дополнительную идентификацию с помощью внутренних структур в рамках виртуальной файловой системы (англ. Virtual File System, VFS) ядра Linux, например, *inode*, *dentry* [3, 23, 44, 45]. Поскольку такие объекты формируются при загрузке ядра ОС и монтировании конкретной файловой системы, то изначально все равно необходимо использовать абсолютные пути до объектов доступа. Однако после монтирования всех необходимых ФС возможно сопоставить эти абсолютные пути конкретным объектам *inode* и *dentry* на уровне ядра ОС, контролируя любые попытки доступа уже к ним без использования абсолютных путей. Такой подход позволяет:

– частично избавиться от неоднозначности при изменении порядка монтирования ФС (*inode* и *dentry* будут оставаться неизмененными, может изменяться только структура, определяющая их точку монтирования – *vfsmount*);

– исключить неточность идентификации при переименовании объектов доступа (в рамках текущего интервала работы ОС GNU/Linux) и обращении по ссылкам.

Контроль точек монтирования для использования данного механизма необходимо реализовать только в самом начале работы для корректного сопоставления абсолютных путей с *inode* и *dentry*. В случае отсутствия объекта с указанным абсолютным путем (например, если объект будет доступен позже после монтирования дополнительных ФС) необходимо предусмотреть механизм, который будет периодически перестраивать соответствие *inode* и *dentry* файловым объектам. Механизм же «пополнения» прав доступа должен быть реализован для новых создаваемых объектов.

В качестве еще одного способа идентификации объектов в ОС GNU/Linux можно использовать задание соответствия прав доступа такой уникальной характеристике, как хэш-код или контрольная сумма от защищаемых данных [55]. У данного подхода существуют следующие преимущества:

– одновременно сочетаются механизмы контроля целостности и доступа к данным;

– отсутствует неточность идентификации при перемещения объектов доступа или изменения порядка монтирования файловых систем;

– для хранения прав доступа возможно применение хэш-таблиц с использованием всех преимуществ этой структуры данных (быстрый поиск, добавление и удаление элементов) [1].

Недостатками способа идентификации объектов по контрольным суммам или хэш-кодам являются: значительное увеличение затрат на вычисление хэш-функций (особенно для часто загружаемых в память объектов, разделяемых библиотек), связанная с этим неприменимость для идентификации больших по объему объектов доступа, а также возможность возникновения коллизий (избежать которых можно с помощью метода цепочек или открытой адресации [1]). С учетом описанных недостатков способа идентификации объектов по контрольным суммам или хэш-кодам его целесообразно использовать в сочетании с другими описанными способами.

Вопрос выбора подходящего для идентификации объектов доступа способа граничит с его «доступностью» для понимания пользователем ОС или администратором подсистемы управления доступом. В связи с этим многие существующие средства разграничения доступа в

GNU/Linux для упрощения идентифицируют объекты по абсолютным путям, не вводя при этом каких-либо дополнительных механизмов защиты, описанных выше.

Автором предложены способы идентификации объектов при взаимодействии с субъектами доступа, которые устраняют соответствующие недостатки существующих средств разграничения доступа в GNU/Linux (см. раздел 1.4), а именно:

– идентификация по абсолютным путям с контролем точек монтирования и создания жестких ссылок, а также «пополнением» или изменением прав доступа при создании новых или переименовании существующих объектов доступа;

– идентификация на основе метаданных объекта с назначением прав доступа при создании новых объектов;

– идентификация на основе абсолютных путей и внутренних структур ядра ОС Linux с перестроением соответствия этих структур файловым объектам и пополнением прав доступа при создании новых объектов;

– на основе хэш-кодов или контрольных сумм от данных с пополнением прав доступа при создании новых объектов.

При этом выбор конкретного способа идентификации объектов доступа необходимо осуществлять в соответствии с требованиями, предъявляемыми к подсистеме управления доступом с обязательным использованием соответствующих дополнительных механизмов защиты (контроль точек монтирования и другие, описанные в данном разделе). Предложенные способы идентификации объектов позволяют однозначно идентифицировать сущности из множеств O^t_{data} и O^t_{na} в любой момент времени $t \geqslant 0$ для дальнейшей корректной реализации запросов в рамках модели безопасности из раздела 2.1.

Подтверждение ожидаемых свойств от использования предложенных в данном разделе способов идентификации субъектов и объектов доступа будет проведено в соответствующих разделах последующих глав.

2.3. Алгоритм доверенной загрузки загрузчика и ОС GNU/Linux при пошаговом контроле целостности

В соответствии со Следствием из Базовой теоремы ИПСС из раздела 2.1 для исключения возможности обхода действующих правил доступа необходимо обеспечить активизацию подсистемы управления доступом (s_{sorm}) и выполнение свойства абсолютной корректности в обратном смысле относительно существующего системного субъекта s_0. При этом абсолютная корректность в обратном смысле подразумевает запрет на выполнение некоторых информационных потоков (на чтение и выполнение) в случае, если состояние (целостность) объектов, требуемых подсистеме управления доступом или ОС, каким-то образом изменилось.

Поэтому для гарантии выполнения заданных правил доступа и корректности функционирования средств разграничения доступа необходимо обеспечить целостность критически важных составляющих ОС и СВТ. С этой целью обычно применяют аппаратные модули доверенной загрузки, которые реализуют *механизм пошагового контроля целостности и доверенной загрузки ОС СВТ*, в том числе и в ОС GNU/Linux (подробнее см. раздел 1.4 и [41, 42]). Однако примене-

ние АМДЗ для ОС GNU/Linux в общем случае не гарантирует доверенной загрузки, что связано с особенностями загрузчиков, используемых в процессе запуска этих ОС [24]. В связи с этим, необходимо определенным образом скорректировать механизм доверенной загрузки для использования в ОС GNU/Linux и гарантии результатов его применения. С этой целью рассмотрим подробнее процесс загрузки типового СВТ с установленным в нем АМДЗ и ОС GNU/Linux.

Загрузку СВТ с АМДЗ упрощенно можно представить в виде следующей последовательности выполняемых друг за другом шагов:

1. Управление передается BIOS (UEFI) СВТ, инициализируется оборудование.
2. Управление перехватывает АМДЗ (сразу после процедуры RomScan), проводятся процедуры идентификации и аутентификации, контроля аппаратных и программных компонент СВТ.
3. В случае успешного прохождения проверочных процедур управление передается загрузчику ОС.
4. Загрузчик загружает ядро Linux, начинается ранний этап работы ОС, инициализируются соответствующие драйверы, монтируются файловые системы и так далее.
5. Запускаются необходимые системные процессы и сервисы, завершается этап загрузки GNU/Linux, запускаются соответствующие процессы для стандартных процедур идентификации и аутентификации в ОС.

В описанных выше пунктах 1-3 при правильной настройке АМДЗ и соблюдении определенных организационных мер [52] невозможно каким-либо образом несанкционированно повлиять на процесс загрузки СВТ. В пунктах 4-5 на основании того, что с помощью АМДЗ можно контролировать целостность всех компонент ОС (ядра, драйверов, конфигурационных файлов и так далее) все дальнейшие механизмы защиты GNU/Linux фактически могут выполняться «доверенно». Однако после контрольных процедур АМДЗ управление передается загрузчику, а не ядру ОС Linux, именно он обеспечивает загрузку в память и дальнейшую передачу управления ядру ОС. Таким образом, в отношении АМДЗ для ОС GNU/Linux уместнее говорить о реализации механизма доверенной загрузки загрузчика ОС.

В ОС семейства Windows при обеспечении доверенной загрузки стандартного загрузчика (NTLDR, Windows Boot Manager) фактически обеспечивается доверенная загрузка ОС вследствие того, что не существует возможностей несанкционированно изменить сценарии загрузки. Однако, при использовании на СВТ таких загрузчиков, как grub и lilo (являющихся стандартными для ОС GNU/Linux), доверенная загрузка этих загрузчиков может не гарантировать доверенной загрузки ОС (например, могут игнорироваться приоритеты загрузки с CD или другого носителя, выставленные в BIOS [24]).

Рассмотрим подробнее загрузчики GNU/Linux на примере grub (GRand Unified Bootloader, GRUB). Данный загрузчик: соответствует стандарту Multiboot Specification [88], то есть позволяет загружать любую поддерживающую этот стандарт ОС; позволяет передать управление «по цепочке» другому загрузчику; позволяет в динамике модифицировать и задавать произвольные параметры при загрузке (параметры ядра Linux, путь к ядру ОС или образу начальной загрузки initrd / initramfs, альтернативный загрузочный носитель информации) без модификации кода загрузчика или его конфигурационных файлов.

С точки зрения защиты информации в ОС GNU/Linux, в первую очередь, интересны следующие динамические изменения grub:

– загрузка ОС GNU/Linux в однопользовательском режиме (англ. single user mode) вместо стандартного сценария загрузки;

– передача различных параметров ядру ОС Linux, например, для отключение механизмов и средств защиты информации.

В первом случае пользователь получает привилегии суперпользователя ОС GNU/Linux (root). Однопользовательский режим обычно используется в административных целях, но, как правило, изначально он не запрещен и не ограничен в большинстве дистрибутивов GNU/Linux. В однопользовательском режиме можно беспрепятственно получить доступ ко всем данным ОС GNU/Linux (также можно отключить встраиваемые или стандартные СЗИ НСД или изменить их конфигурационные файлы). В качестве же примера несанкционированной передачи параметра ядру ОС Linux можно привести отключение такой подсистемы безопасности как SELinux с помощью передачи параметра «selinux=0».

Кроме модификации параметров или выбора какого-либо старого сценария загрузки (с предыдущей версией ядра Linux), в загрузчике grub существует возможность смены загрузочного носителя информации (хотя эта возможность и исключается АМДЗ до передачи управления загрузчику). Для реализации смены текущего загрузочного диска достаточно воспользоваться интерактивной командной строкой grub, в которой с помощью определенных команд можно изменить текущий загрузочный раздел, что изображено на Рисунках 2.2 и 2.3.

Рисунок 2.2 – Сценарий передачи управления загрузчику на внешнем устройстве

Рисунок 2.3 – Сценарий «прямой» загрузки ядра и образа initramfs с внешнего устройства

Таким образом, в grub существуют, как минимум, следующие возможности, способные повлиять на процесс загрузки ОС GNU/Linux: возможность динамического изменения параметров уже существующих сценариев загрузки, возможность входа в интерактивную командную строку с изменением как загрузочного носителя информации, так и с созданием нового сценария загрузки. В отношении них необходимо принять соответствующие меры для гарантии обеспечения доверенной загрузки ОС после доверенной загрузки ее загрузчика. Следует, во-первых, исключить возможность загрузки ОС GNU/Linux в однопользовательском режиме или принуди-

тельно запрашивать пароль суперпользователя. Для последнего в большинстве дистрибутивов необходимо дополнить файл /etc/inittab следующей строкой: «~:S:wait:/sbin/sulogin».

Для защиты от изменения параметров существующих сценариев загрузки можно выставить минимальное время ожидания (англ. timeout) в меню загрузчика (GRUB_TIMEOUT). Однако значение тайм-аута 0 (секунд) все равно оставляет возможность войти в меню загрузчика со сценариями загрузки. В grub также имеются другие параметры для тайм-аутов: GRUB_HIDDEN_TIMEOUT, GRUB_HIDDEN_TIMEOUT_QUIET (или просто hiddenmenu), однако, их использование также не отменяет возможности входа в меню grub. Возможным решением является блокировка существующих вариантов загрузки с помощью пароля grub, что приведено на рисунках 2.4 и 2.5, при установке которого также исключается возможность входа в интерактивную командную строку и изменение процесса загрузки. Однако, как было отмечено в разделе 1.1.5, известны уязвимости, приводящие к возможности обхода пароля grub.

```
(09:51) root@gentooxa://26|136Kb # grub-md5-crypt
Password:
Retype password:
$1$W/XJ8$hBOSG46IkZ14g9pwDb5or0
(09:51) root@gentooxa://26|136Kb #
```

Рисунок 2.4 – Создание пароля для доступа к расширенным возможностям загрузчика

```
default 0
timeout 1
splashimage=(hd0,0)/boot/grub/splash.xpm.gz

password --md5 $1$W/XJ8$hBOSG46IkZ14g9pwDb5or0

title live-usb startup
root (hd2)
chainloader +1
```

Рисунок 2.5 – Настройка блокировки изменения существующих сценариев загрузки

Поскольку загрузчики в GNU/Linux, имеют в своем составе собственные реализации разнообразных модулей для поддержки оборудования (USB-стек и так далее), то желательно ограничить эти модули-расширения только теми, которые реально используются на данном СВТ, а остальные удалить (например, оставить модуль файловой системы ext2 для раздела /boot, удалить модули для поддержки CD/DVD-дисков, USB-устройств, fat, uhci и так далее). В разделе /boot желательно не оставлять предыдущих версий ядра Linux и образов начальной загрузки.

Основной целью ввода защитных мер для загрузчика является ограничение любого пользователя в выборе доступных сценариев загрузки. Процесс каждой загрузки ОС GNU/Linux должен быть строго одинаков, не должно быть способа, при котором изменить процесс загрузки может кто-либо, кроме администратора безопасности, ответственного за работу СВТ и всех средств защиты информации.

Таким образом, стандартные средства доверенной загрузки ОС (АМДЗ) применительно к ОС GNU/Linux позволяют обеспечивать только доверенную загрузку загрузчика ОС. Для обеспечения доверенной загрузки ОС GNU/Linux необходимо наряду с организационными мерами

при использовании АМДЗ соответствующим образом защищать загрузчик ОС GNU/Linux – разграничивать доступ к его возможностям, ограничивать и удалять лишние модули расширений для неиспользуемых компонент, а также сокращать доступные сценарии загрузки ОС.

Также необходимо учитывать, что аппаратные модули доверенной загрузки не всегда применимы по отношению к системам, на которых может использоваться GNU/Linux (аналогичное верно и для других современных ОС). АМДЗ были спроектированы для IBM-совместимых СВТ (то есть, в основном, для архитектур x86, x86_64), в которых присутствует BIOS, поддержка PCI и расширений BIOS, а также используются простейшие загрузчики ОС. При этом предложенный механизм доверенной загрузки загрузчика и ОС позволяет учесть только наличие в составе GNU/Linux более современных загрузчиков, но не может быть использован на СВТ, не поддерживающих работу АМДЗ (без поддержки PCI или не со стандартным BIOS). Кроме того, многие АМДЗ сильно зависят от конкретной используемой аппаратной платформы (даже в случае IBM-совместимого СВТ), с чем могут быть связаны различные проблемы совместимости [62]. Таким образом, на любом СВТ в данный момент использовать существующие АМДЗ для доверенной загрузки ОС и гарантированной активизации ее подсистемы управления доступом невозможно.

В связи с этим предложенный способ доверенной загрузки загрузчика и ОС необходимо скорректировать для случая, когда применение стандартных АМДЗ невозможно (например, на архитектуре arm или s390x). При этом вместо контроля целостности критически важных компонент системы (в том числе участвующих в процессе загрузки СВТ) достаточно обеспечить технологическую невозможность нарушения их целостности (то есть, технологическую неизменность) с использованием специальных носителей информации с памятью, в обычном режиме доступной только для чтения. Подобный способ создания доверенной вычислительной среды был впервые предложен В. А. Конявским в [43] в виде концепции доверенного сеанса связи с использованием специальных носителей, обладающих памятью с управляемым доступом. Однако данную концепцию можно применить и по отношению к СВТ, на которых невозможно использовать стандартные АМДЗ. Для этого необходимо использовать носители информации, доступ к памяти которых на физическом уровне можно ограничить (на чтение, чтение и запись). Соответственно все объекты, которые ранее требовалось ставить на контроль целостности в АМДЗ, в начальный момент времени должны размещаться в областях памяти, доступной только на чтение. При этом, для возможности санкционированного изменения некоторых таких объектов необходимо иметь возможность перевода доступной только для чтения памяти в режим чтения и записи. Описанный способ доверенной загрузки позволяет полноценно заменить всю функциональность АМДЗ на неподдерживающих их архитектурах СВТ при соблюдении аналогичных организационных мер защиты информации.

Таким образом, в разделе предлагается способ и алгоритм обеспечения доверенной загрузки загрузчика и ОС, который разработан лично автором и основан на известном механизме пошагового контроля целостности и доверенной загрузки ОС с помощью АМДЗ [41, 42], а также на концепции доверенных сеансов связи [43]. Блок-схема алгоритма обеспечения доверенной загрузки загрузчика и ОС приведена на Рисунке 2.6, при этом сам алгоритм предполагает следующие действия:

1. В случае совместимости СВТ со стандартными АМДЗ (как правило, для архитектур x86, x86_64) необходимо:

– использовать АМДЗ для контроля целостности аппаратных и программных средств СВТ с учетом организационных мер защиты информации, например, описанных в [52] (в данный пункт входит *известный механизм пошагового контроля целостности и доверенной загрузки ОС*);

– контролировать целостность кода загрузчика и всех его файлов конфигурации средствами АМДЗ до загрузки ОС;

2. В случае невозможности использования на СВТ стандартных АМДЗ (на архитектурах s390x, arm и других) необходимо:

– ограничить носители информации СВТ, с которых может инициироваться загрузка ОС (на большинстве таких СВТ это требование выполняется автоматически вследствие наличия одного такого носителя);

– для критически важных компонент системы (загрузчик, файлы конфигурации, ядро ОС) вместо контроля целостности использовать носители информации, память которых изначально доступна только для чтения, обеспечивая целостность технологически;

3. Необходимо ограничить возможности современных загрузчиков по изменению сценариев загрузки ОС в динамике (в том числе, удалить лишние модули расширений, сократить доступные сценарии загрузки).

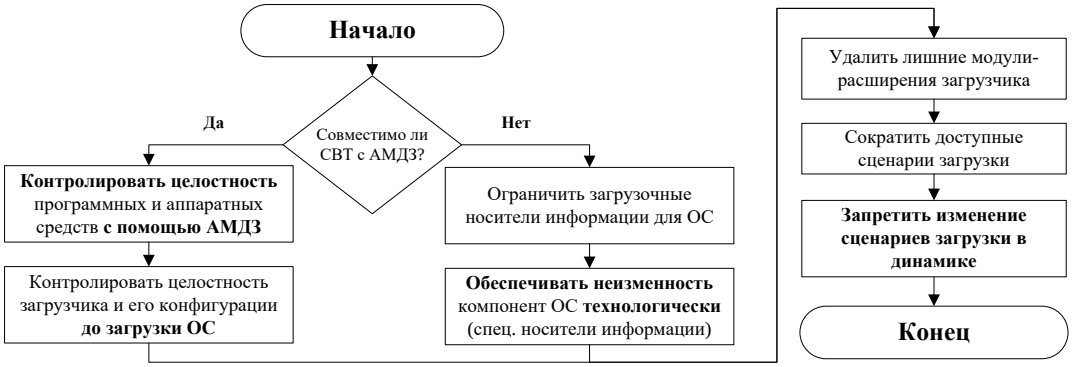

Рисунок 2.6 – Блок-схема алгоритма обеспечения доверенной загрузки загрузчика и ОС

Данный алгоритм, в отличие от известных, применим на широком спектре СВТ и обеспечивает не только доверенную загрузку загрузчика GNU/Linux, но и дальнейшую доверенную загрузку самой ОС посредством использования новых процедур по ограничению и контролю загрузчиков, обеспечению технологической невозможности нарушения целостности критически важных компонент системы.

Применительно к предложенной в разделе 2.1 модели безопасности ОС и подсистемы управления доступом предложенный алгоритм доверенной загрузки загрузчика и ОС позволяет обеспечить активизацию ядра защиты s_{sorm} и начало выполнения соответствующих запросов к системе из Таблиц 2.1 и 2.2. При этом за счет аппаратного контроля целостности (технологической неизменности) объектов, которые будут ассоциированы с s_{sorm} обеспечивается свойство абсолютной корректности в обратном смысле s_{sorm} относительно существующих в системе

субъектов (при нарушении целостности этих объектов дальнейшая загрузка ОС не будет производиться).

Кроме того, сам принцип пошагового контроля целостности в дальнейшем активно используется в предложенной модели безопасности, а его область применения расширена на общедоступные для субъектов объекты доступа посредством введения динамического и статического контроля целостности в процессе работы системы. Таким образом, помимо основной цели применения принципа пошагового контроля целостности для доверенной загрузки ОС с помощью выполнения взаимосвязанных последовательных шагов по контролю целостности обеспечивается «самозащита» подсистемы управления доступом, ее активизация и непрерывность работы, а также защита критически важных и общедоступных компонент системы с дальнейшей организацией изолированной программной среды для субъектов. При этом механизмы аппаратного контроля целостности (технологической неизменности), выполняемые на первом этапе активизации системы, обеспечивают неизменность и активизацию программных механизмов статического/-динамического контроля целостности и разграничения доступа, а те, в свою очередь, позволяют организовать ИПСС и неизменность компонент самой подсистемы управления доступом.

Подтверждение ожидаемых свойств от использования предложенного в данном разделе алгоритма обеспечения доверенной загрузки загрузчика и ОС при пошаговом контроле целостности будет проведено в соответствующих разделах последующих глав.

2.4. Алгоритм встраивания функций защиты от несанкционированного доступа на раннем этапе загрузки ОС GNU/Linux

Обеспечить активизацию подсистемы управления доступом и непрерывность ее функционирования в процессе работы ОС можно посредством предложенного в разделе 2.3 механизма пошагового контроля целостности с доверенной загрузкой загрузчика и ОС, но только частично. Посредством этого же механизма можно обеспечить целостность компонент подсистемы управления доступом в ОС.

Для выполнения оставшихся условий Следствия из Базовой теоремы ИПСС из раздела 2.1 дополнительно необходимо корректным образом встроить и активировать функции контроля доступа субъектов к объектам и порождения субъектов ОС – условия 2-4 Следствия к Базовой теореме ИПСС), учитывая возможность сочетания с другими средствами защиты GNU/Linux. Также естественным требованием для встраивания является обеспечение ранней активизации подсистемы управления доступом (на раннем этапе загрузки ОС, до возникновения реальных субъектов или защищаемых объектов – t_0 из Следствия к Базовой теореме ИПСС). Кроме того, в случае невозможности выполнить перечисленные выше требования подсистема управления доступом должна приостановить работу ОС или сигнализировать о возникших ошибках (условия 1-4 Следствия).

Корректность встраивания подсистемы управления доступом, кроме этого, подразумевает вызов соответствующих механизмов защиты при осуществлении любого типа доступа и в отношении всех субъектов, а главное – объектов системы (а не только в отношении конфиденциальных или других защищаемых данных, см. раздел 1.2). Также работоспособность подсистемы

управления доступом не должна зависеть от каких-либо компонент ОС или ее конфигурации (например, типа ФС).

В разделе 1.1.1 была описана схема организации доступа субъектов к объектам в ОС GNU/Linux, в соответствии с которой любое обращение к объекту системы возможно только при выполнении определенного системного вызова. При этом системный вызов может быть инициирован как напрямую, так и с использованием стандартной библиотеки GNU/Linux[2] (glibc или аналогичных). Так как инициирование системного вызова происходит на пользовательском (непривилегированном) уровне и может осуществляться достаточно большим количеством способов [48] – встраивать таким образом механизмы защиты подсистемы управления доступом небезопасно. Также в данном случае нельзя гарантировать, что встроенные механизмы защиты будут задействованы вне зависимости от конфигурации ОС, версии ядра и других факторов. В связи с этим защитные механизмы подсистемы управления доступом для гарантии надежности необходимо внедрять в ядро ОС Linux, а не на уровне стандартных библиотек или прочих компонент ОС в пользовательском режиме. В соответствии с этим рассмотрим вопросы встраивания и сочетания механизмов защиты СЗИ НСД на уровне ядра ОС GNU/Linux, а также их гарантированный вызов при осуществлении доступа к объектам и непрерывность работы.

Подсистему управления доступом, в случае ее встраивания на уровне ядра ОС, проще всего реализовать в виде загружаемого модуля Linux (см. разд. 1.1.1 и [22, 36]), который должен корректным образом изменить порядок работы ядра ОС. Это позволяет сохранить статическую неизменность ядра GNU/Linux (нарушать которую в ряде случаев просто невозможно), реализовав при этом не только дополнительные способы защиты информации, но и сохранив в неизменности работу других средств разграничения доступа (и соответствие существующим стандартам типа POSIX/SUS и LSB).

В соответствии с описанным выше, а также на основе анализа из раздела 1.1.1 и результатов исследования из раздела 2.1 предложим способ и алгоритм встраивания подсистемы управления доступом в ОС. Блок-схема алгоритма встраивания подсистемы управления доступом на раннем этапе загрузки ОС приведена на Рисунке 2.7, при этом сам алгоритм предполагает следующие последовательные действия:

1. Встраивание должно происходить на раннем этапе загрузки ОС (до возникновения реальных субъектов или защищаемых объектов, до загрузки стандартных модулей ядра, до появления системных сервисов и процессов).

2. В случае невозможности встраивания подсистема управления доступом должна прервать дальнейшую загрузку ОС.

3. Функции защиты от НСД должны встраиваться либо в таблицу системных вызовов, либо в обработчики вызовов на уровне ядра Linux, что изображено на Рисунке 2.8. При этом встраивание должно происходить по отношению ко всем защищаемым и возможным типам доступа.

[2]Стандартные библиотеки GNU/Linux в своей основе также используют системные вызовы, предоставляя только интерфейс доступа к ним.

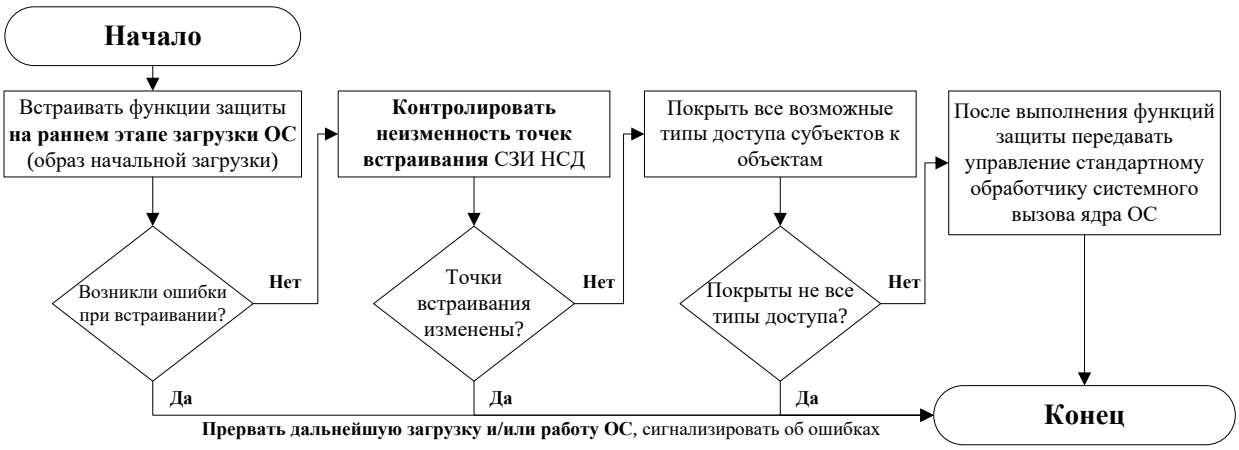

Рисунок 2.7 – Блок-схема алгоритма встраивания функций защиты на раннем этапе загрузки ОС

4. Подсистема управления доступом должна контролировать неизменность точек встраивания своих защитных механизмов, а в случае их изменения – приостанавливать работу ОС или сигнализировать о возникших ошибках.

5. При осуществлении любого типа доступа (при выполнении системного вызова) управление должно передаваться обработчику соответствующего системного вызова в подсистеме управления доступом.

6. В случае если доступ является санкционированным в соответствии с правилами подсистемы управления доступом, управление должно передаваться далее стандартному обработчику системного вызова ядра GNU/Linux, иначе необходимо прервать выполнение системного вызова.

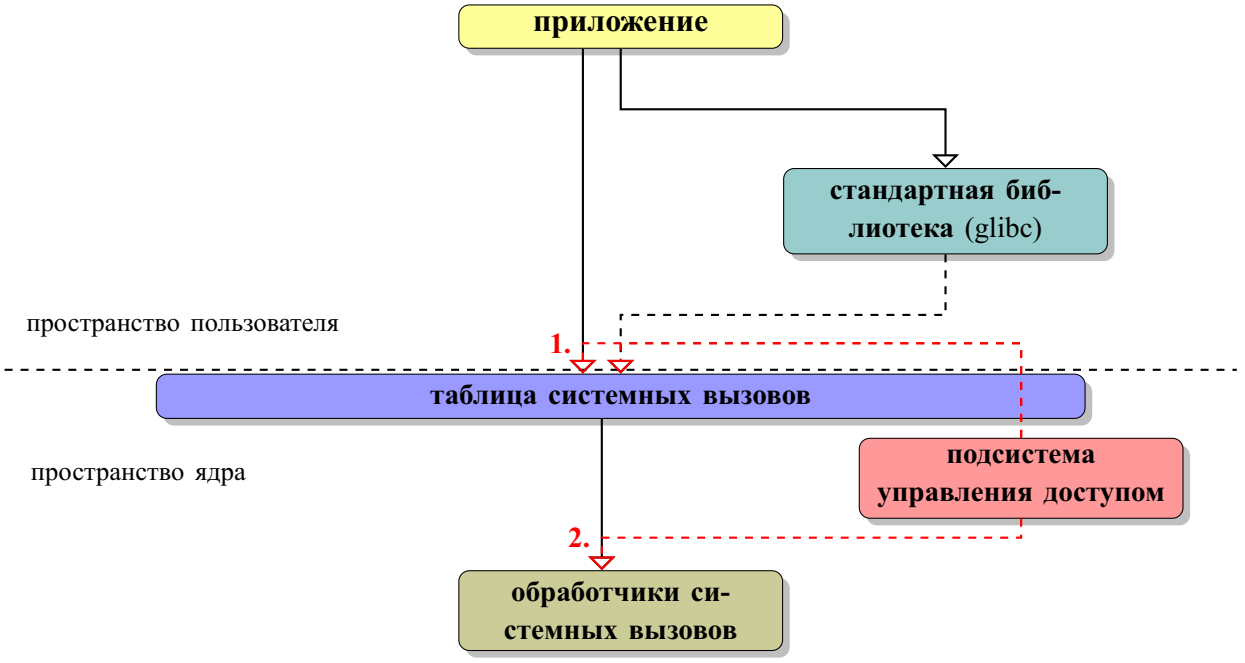

Рисунок 2.8 – Встраивание механизмов защиты подсистемы управления доступом в ОС GNU/Linux: *1.* – в элементы таблицы системных вызовов; *2.* – в обработчики системных вызовов

При этом для подсистемы управления доступом, реализующей описанный алгоритм встраивания в ОС GNU/Linux, будет гарантироваться активизация, вызов соответствующих механизмов при осуществлении любого защищаемого типа доступа субъектов к объектам и непрерывность работы в случае использования предложенного в разделе 2.3 механизма пошагового контроля целостности с доверенной загрузкой загрузчика и ОС (в том числе для «самозащиты» всех компонент этой подсистемы). Кроме того, в случае сохранения оригинальной цепочки вызовов и обработчиков ядра Linux также при таком встраивании будет обеспечиваться вызов механизмов защиты других средств разграничения доступа в GNU/Linux и не будет нарушаться соответствие существующим стандартам.

Непосредственно для встраивания могут использоваться различные техники [48], с учетом которых различия в двух приведенных вариантах (см. пункт 3 выше) могут заключаться в том, что:

– встраивание может быть универсальным и иметь широкое назначение (для любых POSIX/SUS-совместимых ОС), либо ограничиваться только ОС семейства GNU/Linux;

– встраивание может быть применимо только по отношению к определенной архитектуре СВТ, либо может не зависеть от нее;

– встраиваемые обработчики подсистемы управления доступом могут вызываться до или после отработки других механизмов защиты GNU/Linux.

Так или иначе, используя описанный выше алгоритм встраивания при реализации подсистемы управления доступом, а также соответствующие механизмы идентификации объектов доступа из раздела 2.2.2 можно добиться независимости от используемых файловых систем и прочих компонент, связанных с объектами ОС GNU/Linux.

Для выполнения требования по активизации на раннем этапе загрузки ОС подсистема управления доступом в виде LKM должна загружаться в ядро Linux на одном из последних этапов работы *init* [2, 20], но до монтирования корневой файловой системы на запись (в системе активен только субъект s_0). После завершения процесса init подсистема управления доступом будет находиться в состоянии ожидания каких-либо внешних воздействий, а для корректной работы механизмов защиты дополнительно нужно реализовать идентификацию субъектов доступа из раздела 2.2.1 ($create\_user(s,o,s')$ и $create\_shadow(s,o,s')$ из Таблиц 2.1 и 2.2).

В подсистеме управления доступом должны существовать обработчики системных вызовов, с помощью которых осуществляется доступ к объектам ОС GNU/Linux (чтение, запись, исполнение и всевозможные другие), а также вызовы, с помощью которых возможно создание и смена субъектов доступа (*setuid*, *fork*, *exec* и связанные с ними, см. разд. 2.2.1). При описании атрибутов доступа для объектов необходимо учитывать, что для исполнения бинарного файла, кроме стандартного файлового атрибута *x* (*exec*), требуется доступ на чтение (атрибут *r*, *read*) и некоторые другие особенности (см. раздел 1.1.2 и [54]) При этом атрибуты СЗИ НСД должны строиться на основе базовых атрибутов *rwx* ОС GNU/Linux – могут соответствовать или быть шире их.

Таким образом, разработанный автором алгоритм встраивания функций защиты на раннем этапе загрузки операционной системы устраняет соответствующие недостатки существующих

средств разграничения доступа в GNU/Linux (см. раздел 1.4), а также позволяет совместно с другими предложенными ранее способами и алгоритмами обеспечения безопасности:

– гарантировать активизацию подсистемы управления доступом на раннем этапе работы ОС GNU/Linux и непрерывность функционирования (условия из Следствия к Базовой теореме ИПСС с момента t_0);

– применять правила разграничения доступа в отношении всех субъектов, объектов (а не только в отношении конфиденциальных или других защищаемых данных) и типов доступа;

– исключить зависимость подсистемы управления доступом от конфигурации ОС;

– корректным образом сочетать несколько средств разграничения доступа.

Подтверждение ожидаемых свойств от использования предложенного в данном разделе алгоритма встраивания функций защиты на раннем этапе загрузки ОС будет проведено в соответствующих разделах последующих глав.

Необходимо отметить ключевую роль предложенных в разделах 2.2–2.4 способов и алгоритмов обеспечения безопасности управления доступом в организации ИПСС:

– без пошагового контроля целостности с доверенной загрузкой загрузчика и ОС и корректного встраивания подсистемы управления доступом невозможно достижение ИПСС в процессе загрузки КС и дальнейшее ее поддержание;

– без предложенного способа идентификации субъектов невозможно реализовать запросы $create\_user(s,o,s')$ и $create\_shadow(s,o,s')$;

– без идентификации субъектов и объектов невозможно реализовать другие запросы в рамках модели безопасности из Таблиц 2.1 и 2.2.

Таким образом, применительно к сформированной модели безопасности предложенные способы идентификации субъектов и объектов при их взаимодействии, алгоритм обеспечения доверенной загрузки загрузчика и ОС при пошаговом контроле целостности, а также алгоритм встраивания функций защиты на раннем этапе загрузки ОС позволяют обеспечить контролируемую активизацию ядра защиты системы с контролем целостности всех связанных с ним объектов до запуска КС (подразумеваемое «предопределенное выполнение начальной фазы функционирования КС» [13] из СО-модели). Предложенные в главе способы и алгоритмы позволяют выполнить все условия Следствия к Базовой теореме ИПСС, то есть в системе с применением полученных в данной главе результатов обеспечивается невозможность исключения активизации подсистемы управления доступом или обход действующих правил политики безопасности.

2.5. Выводы к главе 2

1. Предложена научно-обоснованная модель безопасности для ОС GNU/Linuxи средства разграничения доступа, обеспечивающая невозможность нарушения заданной политики безопасности и возможность возникновения в системе только разрешенных информационных потоков. Данная модель безопасности основана на положениях известной СО-модели изолированной программной среды, и в ней, в отличие от последней, проводится декомпозиция системы на субъекты (пользователи и системные сервисы), функционально ассоциированные с ними объекты

(процессы) и объекты-данные с возможностью динамического изменения их состава во времени; реализация заданной политики управление доступом обосновывается в отношении различных субъектов доступа, а не между процессами одного субъекта; задается порядок достижения начального состояния системы с контролем непрерывного выполнения функций защиты от НСД. При этом предложенную модель безопасности, вследствие ее инвариантности относительно применяемой политики управления доступом, возможно использовать для анализа свойств других способов и средств защиты от НСД, в GNU/Linux и других POSIX/SUS-совместимых ОС.

2. Предложен способ контроля порождения субъектов доступа и идентификации субъектов, от имени которых соответствующие процессы ОС GNU/Linux взаимодействуют с объектами системы, который, в отличие от применяемых в существующих средствах разграничения доступа в GNU/Linux способов контроля за субъектами, позволяет исключить изменение состава субъектов в ходе работы системы, осуществлять более точный контроль различных субъектов посредством их разделения на соответствующие типы (пользователи, системные субъекты).

3. Предложенные способы идентификации объектов при взаимодействии с субъектами доступа устраняют соответствующие недостатки существующих средств разграничения доступа в GNU/Linux. В отличие от аналогов, разработанные способы позволяют исключить неточность идентификации объектов в подсистеме управления доступом, учесть особенности ОС GNU/Linux и решить вопрос изменения состава объектов в ходе работы системы (например, при появлении новых объектов доступа).

4. Предложен алгоритм обеспечения доверенной загрузки загрузчика и ОС GNU/Linux при пошаговом контроле целостности, устраняющий возможность исключения активизации подсистемы управления доступом до или на раннем этапе загрузки системы на различных аппаратных платформах посредством введения новых процедур по ограничению возможностей загрузчиков, контролю или обеспечению технологической невозможности нарушения целостности критически важных компонент системы. В отличие от аналогов, данный алгоритм применим для широкого спектра СВТ и обеспечивает не только доверенную загрузку загрузчика GNU/Linux, но и дальнейшую доверенную загрузку самой ОС.

5. Алгоритм встраивания функций защиты от НСД, в отличие от аналогов, учитывает необходимость обеспечения активизации и целостности подсистемы управления доступом на раннем этапе работы ОС, а также непрерывности дальнейшего функционирования с принудительной блокировкой системы в случае нарушения точек встраивания; применения правил разграничения доступа в отношении всех субъектов, объектов и типов доступа.

6. Комплексное использование перечисленных способов и алгоритмов обеспечения безопасности управления доступом позволяет выполнить необходимые и достаточные условия для реализации изолированной программной среды субъектов доступа в соответствии с предложенной моделью безопасности. Кроме того, данные способы и алгоритмы имеют широкое назначение и могут применяться как в ОС, отличных от GNU/Linux, так и по аналогии в системах управления доступом других систем.

3. Разработка средства разграничения доступа для ОС GNU/Linux, реализующего предложенные алгоритмы и модель безопасности

В данной главе на основе результатов проведенного исследования необходимо обосновать состоятельность предложенных способов и алгоритмов обеспечения безопасности, дать рекомендации по их практическому применению и сформировать необходимые требования для реализующей эти способы и алгоритмы защиты подсистемы управления доступом в ОС Linux.

3.1. Обоснование требований для подсистемы управления доступом с гарантией активизации и непрерывности функционирования в ОС GNU/Linux

Используя результаты исследования из главы 2 и [17, 32], сформулируем уточненные и конкретизирующие требования (по отношению к требованиям нормативных документов РФ в области защиты информации) для подсистемы управления доступом в ОС GNU/Linux с учетом возможного наличия в системе описанного в разделе 1.4 типа нарушителя. Требования к подсистеме управления доступом в ОС должны актуализировать существующие нормативные документы РФ в области защиты информации в части исключения возможности обхода действующей политики управления доступом. При этом данные требования необходимо описать как можно более обобщенно для возможности их применения не только в рамках GNU/Linux, но и для аналогичных систем.

1. *Подсистема управления доступом должна корректным образом встраиваться в ядро ОС, гарантируя при этом его статическую неизменность.*

Для формирования требований к подсистеме управления доступом в GNU/Linux необходимо изначально определить, каким образом проводить встраивание защитных механизмов в ОС. При разработке СЗИ НСД этот вопрос может влиять на возможность реализации некоторых функций защиты (например, хранение прав доступа в атрибутах файловых объектов), а для конечного пользователя тип встраивания может серьезно влиять на удобство или на принципиальную возможность использования средства защиты. Как было обосновано в разделе 2.4 подсистему управления доступом необходимо встраивать в ядро ОС Linux, при этом в некоторых организациях существуют требования по невозможности изменения такого важного компонента, как ядро Linux. Это может быть связано как с ограничениями технической поддержки производителя того или иного дистрибутива ОС, так и с внутренними требованиями конкретной организации (например, банка). Кроме того, в организациях часто существует «привязанность» к использованию вполне определенных дистрибутивов GNU/Linux, поэтому реализация СЗИ НСД в виде встроенного в новую ОС компонента (см. раздел 1.1.2) в общем случае не может быть универсальным решением. Таким образом, для встраивания наиболее целесообразным видится два возможных сценария: включить подсистему управления доступом в состав основной ветки разработки ядра Linux; реализовать ее в виде встраиваемого средства.

Первый сценарий может потребовать значительного времени, а подсистему управления доступом можно будет использовать только начиная с версии ядра, в которое был добавлен соответствующий код по поддержке и, соответственно, только в дистрибутивах GNU/Linux с соответствующей версией ядра. Кроме того, в данном случае решение должно поставляться с открытым исходным кодом и со свободной лицензией по использованию (например, GNU GPL). Второй сценарий не требует дополнительных временных издержек, кроме времени разработки и позволяет использовать подсистему управления доступом для множества уже существующих дистрибутивов ОС GNU/Linux, но с некоторыми ограничениями.

В связи с перечисленным наиболее целесообразным способом реализации подсистемы управления доступом для защиты уже существующих данных, хранящихся и обрабатываемых в ОС GNU/Linux, является реализация в виде встраиваемого средства. Только в этом случае можно обеспечить возможность использования СЗИ НСД на уже применяемых в организациях ОС GNU/Linux и исключить временные издержки на его включение в ядро или обновление конкретных дистрибутивов.

Необходимо также отметить, что встраивание подсистемы управления доступом в ядро ОС Linux необходимо проводить доступным для конечного пользователя образом (не в виде исправлений к ядру с необходимостью его пересборки). Таким образом, целесообразным видится реализация подсистемы управления доступом в виде загружаемого модуля ядра Linux – LKM. Необходимость же данного требования напрямую вытекает из необходимости применения разработанного автором в разделе 2.4 алгоритма встраивания функций защиты на раннем этапе загрузки операционной системы. Однако предложенный алгоритм должен быть дополнен требованием сохранения статической неизменности ядра ОС при встраивании.

2. *Должна гарантироваться активизация встраиваемой подсистемы управления доступом при включении СВТ, начиная с раннего этапа загрузки, и непрерывность ее работы в процессе функционирования ОС*.

Подсистема управления доступом должна обеспечивать невозможность своего отключения или удаления несанкционированным образом. СЗИ НСД должно быть реализовано так, чтобы его нельзя было полностью выгрузить, при этом активизация должна гарантироваться на самом раннем этапе загрузки ОС (до возникновения реальных субъектов или защищаемых объектов, до загрузки стандартных модулей ядра, до появления системных сервисов и процессов), в ином случае загрузка ОС должна быть прекращена. В том числе должна обеспечиваться невозможность несанкционированной загрузки СВТ с другого носителя информации, чтобы не противоречить указанному требованию. Кроме того, подсистема управления доступом должна контролировать неизменность своих точек встраивания в ОС, а в случае их изменения – приостанавливать работу системы или сигнализировать о возникших ошибках. Выполнение этого требования обеспечивает принципиальную невозможность отключения защитных механизмов, что является необходимым для корректной реализации любого СЗИ НСД.

Обычно в качестве средств, с помощью которых становится возможным выполнять данное требование принято использовать [25,38,41] АМДЗ, которые должны «обезопасить» подсистему

управления доступом от отключения или изменения настроек на ранних этапах загрузки ОС. Однако в соответствии с разделом 2.3 в отношении GNU/Linux, а также некоторых аппаратных архитектур для выполнения данного требования недостаточно применения существующих АМДЗ. Данное требование выполняется полностью только при реализации в подсистеме управления доступом алгоритмов встраивания функций защиты на раннем этапе загрузки операционной системы из раздела 2.4 и доверенной загрузки загрузчика и ОС при пошаговом контроле целостности из раздела 2.3.

3. *Встраиваемая подсистема управления доступом должна корректно сочетаться с другими используемыми в ОС средствами защиты информации.*

Как было показано в разделах 1.1.2 и 1.4, в силу особенностей GNU/Linux любое новое СЗИ НСД в любом случае будет сочетаться с другими существующими механизмами защиты информации ОС. В связи с этим необходимо обеспечить корректность их совместного функционирования и непротиворечивость «накладываемых» друг на друга правил доступа (всех средств разграничения доступа системы). Встраиваемая подсистема управления доступом должна обеспечивать комплексирование и усиление защитных свойств и механизмов встроенных СЗИ НСД в ОС GNU/Linux, а также контроль корректности функционирования таких средств и их резервирование, обеспечивая избыточность и оптимальность защитных механизмов [72].

Данное требование выполняется при реализации в подсистеме управления доступом описанного в разделе 2.4 алгоритма встраивания функций защиты на раннем этапе загрузки операционной системы.

4. *Необходимо ограничить возникновение субъектов доступа системы, осуществляя их идентификацию и аутентификацию, а также идентифицировать все объекты при взаимодействии с субъектами с гарантией их соответствия защищаемым данным. Необходимо обеспечить надежную связь всех процессов ОС и объектов с атрибутами, назначенными в подсистеме управления доступом (результатами идентификации и аутентификации, правами доступа соответственно).*

Для корректной работы подсистемы управления доступом необходимо ограничивать возникновение субъектов системы, осуществляя процедуры их идентификации и аутентификации (в отношении системных субъектов используется только идентификация и иные процедуры контроля). Также должна обеспечиваться надежная связь результатов этих процедур со всеми процессами, запускаемыми от имени субъектов (вместо дополнительной идентификации каждого отдельного процесса при его появлении и запуске). Кроме этого, должна осуществляться корректная идентификация объектов доступа, гарантирующая соответствие конкретного объекта той сущности, установка прав доступа для которой проводилась в процессе настройки подсистемы управления доступом.

В результате в ОС GNU/Linux не должно существовать процессов, не имеющих связи с определенным субъектом доступа, и объектов, не учтенных в подсистеме управления доступом (кроме возникающих новых объектов, по отношению к которым должны применяться осо-

бые правила). Также подсистема управления доступом должна корректным образом обрабатывать изменение состава субъектов или объектов, автоматически создавая новые правила доступа (например, при создании новых, неучтенных объектов доступа; перемещении существующих объектов и так далее). Дополнительно для объектов-источников, используемых при порождении субъектов доступа, должна контролироваться целостность в процессе работы системы (для обеспечения неизменности механизма порождения субъектов системы).

Данное требование выполняется при реализации в подсистеме управления доступом описанных в разделах 2.2.1 и 2.2.2 способа контроля порождения субъектов доступа и идентификации субъектов, от имени которых соответствующие процессы ОС GNU/Linux взаимодействуют с объектами системы, и одного из способов идентификации объектов при взаимодействии с субъектами доступа.

5. *Необходимо гарантировать вызов соответствующих функций защиты от НСД встраиваемой подсистемы управления доступом при осуществлении любого защищаемого типа доступа субъектов к объектам ОС.*

В процессе работы ОС GNU/Linux порядок функционирования механизмов защиты различных средств разграничения доступа при условии выполнения приведенного ранее требования их корректного сочетания в общем случае не принципиален. Однако необходимо гарантировать вызов соответствующих встраиваемых механизмов защиты при осуществлении любого типа доступа субъектов к объектам ОС (на уровне обработчиков или таблицы системных вызовов ядра), а в дальнейшем продолжать выполнение соответствующих вызовов или возвращать код ошибки.

Данное требование выполняется при использовании в подсистеме управления доступом алгоритмов встраивания функций защиты на раннем этапе загрузки операционной системы из раздела 2.4 и доверенной загрузки загрузчика и ОС при пошаговом контроле целостности из раздела 2.3. При этом в GNU/Linux в качестве контролируемых типов доступа должны выступать: чтение, запись, исполнение, создание / удаление / переименование объектов и каталогов, операции с жесткими ссылками и переход в объекты-каталоги. Необходимость и достаточность контроля приведенных типов доступа регламентируется возможными способами взаимодействия субъектов и объектов в соответствии со стандартами POSIX, SUS [91] и LSB [100]. При этом механизмы контроля указанных типов доступа должны действовать в отношении любых объектов ОС, а не только в отношении защищаемых данных.

6. *Подсистема управления доступом должна реализовывать принцип наименьших привилегий для всех субъектов ОС, запрещая любые типы доступа, не описанные в ее правилах.*

С помощью подсистемы управления доступом необходимо иметь возможность настройки принудительного управления доступом с обеспечением минимальных привилегий на осуществление субъектами доступа (в том числе, привилегированными) только необходимых им действий. Для корректного задания необходимых полномочий в подсистеме управления доступом должен существовать специальный режим, позволяющий создавать правила в соответствии с «типичным поведением» того или иного субъекта доступа ОС GNU/Linux.

Данное требование выполняется при реализации в подсистеме управления доступом алгоритма встраивания функций защиты на раннем этапе загрузки операционной системы из раздела 2.4 (с контролем описанных типов доступа), а также способа контроля порождения субъектов доступа и идентификации субъектов, от имени которых соответствующие процессы ОС GNU/Linux взаимодействуют с объектами системы (см. раздел 2.2.1), и одного из способов идентификации объектов при взаимодействии с субъектами доступа (см. раздел 2.2.2).

7. *Подсистема управления доступом не должна зависеть от типа используемой файловой системы, носителя информации, прочих специфичных компонент и конфигурации ОС.*

В соответствии с разделом 1.4 многие существующие средства разграничения доступа зависят от типа используемой файловой системы или других настроек ОС. Однако для обеспечения непрерывности работы СЗИ НСД в отношении всех объектов доступа необходимо обеспечивать его работоспособность вне зависимости от этих факторов.

Данное требование выполняется при использования в подсистеме управления доступом способов и алгоритмов защиты из разделов 2.2–2.4 вследствие их независимости как от конфигурации ОС, так и от аппаратной платформы, на которой применяется GNU/Linux.

8. *В подсистеме управления доступом должна быть предусмотрена процедура безопасного централизованного администрирования.*

Для корректной защиты данных необходимо обеспечивать возможность администрирования встраиваемой подсистемы управления доступом только санкционированным способом – выделенным субъектом доступа (администратором безопасности). Вследствие особенностей, перечисленных в разделе 1.4, схема администрирования должна быть централизованной.

Данное требование выполняется при использовании абсолютного разделения административных и пользовательских полномочий и выполнении остальных условий, приведенных после доказательства Следствия из Базовой теоремы ИПСС из раздела 2.1.

9. *Требование «самозащиты»: для всех компонент подсистемы управления доступом и связанных с ней объектов должна обеспечиваться конфиденциальность и целостность.*

В соответствии с требованиями гарантии из [109] КС должна содержать аппаратные и/или программные механизмы, которые могут независимо определять обеспечивается ли исполнение системой указанных требований и корректность работы. В связи с этим для самой подсистемы управления доступом и связанных с ней объектов необходимо также, как и для защищаемых данных, обеспечивать конфиденциальность и целостность.

Выполнение данного требования при реализации в подсистеме управления доступом способов и алгоритмов защиты из разделов 2.2–2.4 обеспечивается за счет Базовой теоремы ИПСС и ее Следствия, представленных в разделе 2.1, в соответствии с которыми изменять объекты подсистемы управления доступом может только она сама или выделенный субъект-администратор в специальном режиме работы при отсутствии других субъектов.

10. *Подсистема управления доступом должна выполнять требования нормативных документов РФ по защите информации в зависимости от категории хранящихся и обрабатываемых данных (управление доступом, регистрация и учет, контроль целостности, очистка памяти и остаточной информации, маркировка печатаемых документов), не влияя на выполнение других перечисленных требований*.

Для защиты различных категорий данных по требованиям РФ используются различные требования, предъявляемые к подсистеме управления доступом. Основным является требование по реализации в ОС, как минимум, одной из моделей политик управления доступом. При этом, реализация той или иной политики или других функций защиты не должны влиять на выполнение других перечисленных в данном разделе требований, что исключает необходимость повторного исследования корректности всех реализованных функций защиты. Выполнение данного требования обеспечивается вследствие инвариантности построенной в разделе 2.1 модели безопасности относительно применяемой политики управления доступом (а также других способов и алгоритмов защиты информации). Таким образом, становится возможным выполнить требования нормативных документов РФ в области защиты информации, не нарушая при этом положений предложенной модели безопасности, и, следовательно, обеспечивая корректность используемых защитных механизмов.

Полнота сформированных требований основывается на охвате всех возможных типов доступа субъектов к объектам в соответствии с существующими стандартами (POSIX, SUS и LSB), на полноте реализуемых защитных механизмов в соответствии с требованиями нормативных документов РФ по защите информации, а также на результатах исследования из раздела 2.1.

Таким образом, в разделе сформированы требования к подсистеме управления доступом, которые расширяют требования нормативных документов РФ в области защиты информации и устраняют выявленные в главе 1 недостатки. Так, для подсистемы управления доступом, выполняющей указанные требования, и ОС обеспечиваются [32]:

– корректность способов идентификации субъектов и объектов доступа;

– контроль всех возможных способов взаимодействия субъектов и объектов;

– применимость функций безопасности не только в отношении защищаемых данных, но и других критически важных объектов;

– возможность использования принципа наименьших привилегий для всех субъектов системы (в том числе, для привилегированных пользователей и системных процессов);

– корректность внедрения и безопасного администрирования подсистемы управления доступом, а также ее сочетания с другими средствами защиты ОС;

– отсутствие других выявленных недостатков, характерных для существующих средств разграничения доступа в GNU/Linux.

Кроме того в ОС и подсистеме управления доступом при выполнении сформированных требований обеспечивается гарантия исполнения целевых функций по защите информации от НСД в среде GNU/Linux при сохранении работоспособности ОС посредством гарантии активизации, непрерывности работы, невозможности изменения действующей политики безопасности и

других процедур (если не рассматривать возможность физического извлечения носителя информации, которая должна исключаться организационными мерами или с использованием средств криптографической защиты) [24]. В результате ОС с подсистемой управления доступом соответствует модели безопасности, разработанной в разделе 2.1, в которой вследствие одновременной работы механизмов обеспечения конфиденциальности и целостности информации теоретически предотвращается возможность возникновения запрещенных информационных потоков.

Необходимо отметить, что разработанные в данном разделе требования к подсистеме управления доступом при использовании способов и алгоритмов защиты из разделов 2.2-2.4, а также результатов исследования из раздела 2.1, имеют широкое назначение и применимы как для различных аппаратных платформ, поддерживающих GNU/Linux, так и для других программных платформ.

В соответствии со сформированными требованиями, а также на основе проведенного в разделах 1.1 и 1.4 анализа, рассмотрим возможности и характеристики основных существующих средств разграничения доступа для GNU/Linux, приведенные в Таблице 3.1.

Как можно видеть, часть требований уже в той или иной степени выполняется некоторыми средствами разграничения доступа по отдельности, однако все перечисленные требования целиком не выполняются ни одним конкретным средством (кроме того, этого нельзя достичь, в том числе при сочетании нескольких средств – часть требований невозможно выполнить таким образом). Для выполнения всех сформированных требований необходимо разработать реализующее их средство (подсистему) управления доступом для ОС GNU/Linux. Применение этой подсистемы управления доступом, в том числе, позволит обеспечить комплексирование и усиление защитных свойств встроенных средств защиты ОС GNU/Linux.

Таблица 3.1 – Сравнение возможностей и характеристик средств защиты от НСД в Linux в соответствии с предложенными требованиями к подсистеме управления доступом ОС

Возможности (характеристики) СЗИ НСД	**DAC/ACL CAPS**	**SELinux**	**Tomoyo AppArmor**	**TPM IMA/EVM**	**Alt/Astra/ Rosa Linux, MCBC**	**Secret Net LSP**
1. Корректность встраивания, статическая неизменность ядра ОС		+	+	+		~
2. Гарантия активизации и непрерывности работы	~	–	–	+	~	~
3. Возможность сочетания с другими средствами защиты	+	~	~	+	–	+
4. Контроль порождения субъектов, корректность идентификации субъектов и объектов, надежная связь с атрибутами доступа	+	+	–		~	+
5. Гарантия вызова функций защиты для любого возможного типа доступа	+	+	~		~	+
6. Ограничение всех субъектов системы (соответствие принципу наименьших привилегий)	–	+	~		–	–

Продолжение таблицы 3.1

Возможности (характеристики) СЗИ НСД	**DAC/ACL CAPS**	**SELinux**	**Tomoyo AppArmor**	**TPM IMA/EVM**	**Alt/Astra/ Rosa Linux, MCBC**	**Secret Net LSP**
7. Независимость от компонент / конфигурации ОС	–	–	+	+	–	–
8. Возможность безопасного централизованного администрирования	–	~	+		~	~
9. Выполнение требования «самозащиты»	–	–	–	+	–	–
10. Сертифицированность / соответствие требованиям защиты информации РФ	–	–	–	–	+	+

– возможность не реализована / характеристика отсутствует
+ – возможность реализована / характеристика присутствует
~ – возможно при определенных условиях
– не предусмотрено для выполнения

Необходимо выделить ключевую особенность приведенных требований: первые 9 из них обязательны к выполнению вне зависимости от вида защищаемой информации, что учтено только в последнем требовании. Если не удовлетворяются 9 первых (обязательных) требований – СЗИ НСД не может выполнять возложенные на него функции безопасности даже для низких классов защищенности АС/СВТ/ОС и иных.

3.2. Обоснование рекомендаций по использованию дополнительных функций защиты от несанкционированного доступа в ОС GNU/Linux, обеспечивающих выполнение требований действующих нормативно-правовых документов

Как было показано ранее, для выполнения большинства требований, сформированных в разделе 3.1, в рамках подсистемы управления доступом в ОС необходимо реализовать предложенные в разделах 2.2–2.4 механизмы защиты и произвольную (не нарушающую выполнение других требований) политику управления доступом, а также использовать результаты исследования из раздела 2.1. Однако для определенных категорий данных по требованиям нормативных документов РФ в области защиты информации в подсистеме управления доступом необходимо также реализовать и ряд специфичных механизмов защиты – например, очистку (обнуление, обезличивание) освобождаемых областей оперативной памяти СВТ и внешних носителей информации, подключаемых к нему, а также контроль и маркировку печати (графических) документов.

Также необходимо отметить, что некоторые требования нормативных документов в области защиты информации в РФ могут быть избыточными для ОС GNU/Linux (и других современных ОС). Так, например, такое требование, как изоляция процессов ОС («изоляция модулей») в определенной степени избыточно для современных систем вследствие повсеместной реализации концепции виртуального адресного пространства для каждого процесса. В такой концепции адресные пространства процессов, в том числе одного и того же пользователя ОС, изначально изолированы друг от друга с помощью механизма защиты страниц оперативной памяти. В связи с этим применительно к ОС GNU/Linux требование изоляции процессов выполняется всегда

(автоматически), однако не с использованием какого-либо СЗИ НСД, а с помощью изначальной реализации подсистемы управления памятью в ядре Linux, то есть посредством архитектурной особенности системы. Однако в данном случае не рассматривается возможность привилегированного доступа к страницам памяти на уровне ядра ОС.

Необходимо отметить, что указанные требования предъявляются, в том числе, к достаточно низким классам защищенности АС, СВТ и ОС, например, к классам 1Г, 5 и 4 соответственно (или немного выше), которые в соответствии с разделом 1.2 являются минимально необходимыми классами защищенности для современных систем. Существующие средства разграничения доступа в GNU/Linux, как правило, выполняют большинство необходимых требований безопасности информации из нормативных документов РФ, однако перечисленные выше специфические требования часто в них не учитываются. С учетом этого в данном разделе будет рассмотрена техническая возможность по выполнению таких специфических требований (контроль и маркировка печати, очистка оперативной памяти и памяти внешних носителей информации, изоляция модулей), а также пути реализации соответствующих механизмов защиты в рамках разрабатываемой подсистемы управления доступом.

Контроль, регистрация и маркировка печати документов

В ОС GNU/Linux печать организована с помощью специального сервера печати (на большинстве систем используется сервер печати CUPS, разработанный Easy Software Products, ныне – Apple Inc.). Таким образом, вывод данных на печать не связан с каким-либо системным вызовом, данные с помощью CUPS API передаются сервису cupsd, который с помощью специальных фильтров преобразует их в формат, допустимый для вывода на печать (в GNU/Linux это, как правило, PostScript). Предварительно данные в изначальном формате с помощью планировщика записываются в очередь на сервере печати (для локальной системы это обычно /var/spool/cups, локальный спулер) и заносятся в очередь для выбранного принтера.

В соответствии с требованиями нормативных документов РФ по защите информации, любой выводимый на печать документ необходимо сопровождать автоматической маркировкой каждого листа рядом атрибутов, таких как [6, 8]: последовательный номер документа, наименование или краткое описание; дата выдачи документа; уровень конфиденциальности документа; фамилия лица, распечатавшего документ (идентификатор субъекта доступа); учетные реквизиты АС; общее количество страниц документа и копий (в том числе при неполной распечатке документа); логическое/сетевое имя устройства печати.

С учетом этих требований при разработке модуля контроля печати в рамках подсистемы управления доступом ОС GNU/Linux необходимо учитывать следующие особенности [21]:

– печать необходимо контролировать из всевозможных приложений, в том числе и из командного интерпретатора, который использует для печати утилиты из набора *lpr*;

– в общем случае невозможно определить пользователя, от имени которого осуществляется печать, также имя можно легко изменить в задании на печать;

– в общем случае невозможно получить абсолютный путь до печатаемого документа (объекта файловой системы) или определить его атрибуты, так как печать происходит уже тех данных, которые были составлены из эталонного документа и были записаны в спулер;

– необходимо учитывать возможность вывода на печать данных, которые физически не записывались в объекты ОС или записываются во временные файлы (печать из браузера и/или редактируемых файлов);

– необходимо контролировать вывод на печать на большинстве доступных моделей принтеров, при этом существуют определенные сложности в идентификации принтеров, а также в работе с «не PostScript»-принтерами.

В соответствии с первой особенностью модуль контроля печати должен быть реализован в виде фильтра обработки данных, который будет отрабатывать в конце процедуры преобразования данных в формат PostScript. При этом контроль печати будет работать на уровне подсистемы CUPS. Данный способ построения модуля контроля печати, кроме всего прочего, более универсален чем модификация данных в протоколах взаимодействия конкретного принтера и ядра ОС.

Вторая особенность связана с тем, что на печать отправляются копии документов из спулера с именем пользователя в виде параметра задания на печать. При этом такой параметр может задаваться различными приложениями произвольным образом (либо отсутствовать), а также его легко можно изменить. Задача точного определения субъекта (пользователя), от имени которого осуществляется печать, не может быть эффективно решена для многопользовательских систем без значительной модификации CUPS [21]. Но вследствие того, что печать в целом не обязательно реализовывать для нескольких пользователей одновременно, в качестве более простого решения такой задачи (без модификации CUPS) можно ввести ограничение печати только при одновременной работе одного пользователя ОС GNU/Linux (на момент вывода документа на печать).

Третья и четвертая особенности связаны с тем, что любое приложение, выводящее данные на печать через CUPS-сервер, может совершить это четырьмя различными способами[1]:

– используя функции cupsPrintFile(), cupsPrintFiles() и указывая файлы для печати;

– используя cupsCreateJob() для создания пустого задания на печать и в дальнейшем добавляя туда данные (буфер с данными) с помощью cupsWriteRequestData();

– используя возможности протокола IPP (Internet Printing Protocol) по выводу на печать конкретного файла с помощью функции DoFileRequest();

– используя возможности протокола IPP по выводу на печать некоторого буфера с помощью функции DoIORequest().

В описанных случаях определить печатаемый объект доступа или его атрибуты возможно только в случаях вывода на печать конкретных объектов-файлов. В случае использования для печати других функций API решить задачу по идентификации объекта печати в общем случае сложно, а в отдельных (в зависимости от особенностей приложения, вызывающего печать) – невозможно [21]. Таким образом, в качестве метки конфиденциальности при печати данных целесообразным видится использование уровня доступа субъекта, выводящего данные на печать,

[1]На момент проведения данного исследования.

так как этот уровень, в соответствии с мандатной политикой управления доступом (см. разд. 1.3), не должен быть ниже уровня конфиденциальности объекта (то есть уровень конфиденциальности объекта может только повыситься). При этом приведенное решение применимо и для случая, когда на печать выводятся данные, которые физически не содержатся в определенном объекте ОС GNU/Linux (например, печать данных, измененных с помощью прикладного ПО, но еще не записанных в файл; печать данных из нескольких временных файлов и тому подобное).

Для пятой особенности характерно то, что:

– для идентификации принтера в модуле контроля печати можно использовать его краткое обозначение (параметр «short name») или название PPD-файла, который соответствует конкретному принтеру, однако данные значение не могут являться уникальным идентифицирующим признаком принтера;

– работа с «не PostScript»-принтерами на сервере контроля печати CUPS реализована иначе, то есть те фильтры для преобразования данных, которые применяются в случае с PostScript-принтерами не отрабатывают.

В соответствии с этим модуль контроля печати должен, во-первых, идентифицировать принтеры с помощью других характеристик (например, Vendor ID или IP для сетевого принтера), а, во-вторых, для «не PostScript»-принтеров необходимо либо разрабатывать другие фильтры (отличающиеся от фильтров для PostScript-принтеров и, возможно, уникальные для разных моделей принтеров, поддерживаемых CUPS), либо вообще запрещать отправку на печать.

Также для подсистемы управления доступом при использовании модуля контроля печати, разработанного на основе изложенных рекомендаций, может существовать еще одна особенность – CUPS-сервер в некоторых ситуациях должен обладать максимальными правами доступа в системе, чтобы иметь возможность читать объекты с любым уровнем конфиденциальности и создавать для них копии (задания для дальнейшей печати на принтере). Однако реализованный описанным в данном разделе образом модуль контроля печати будет представлять собой кросс-платформенное решение (как и сам сервер печати CUPS), которое можно использовать на широком спектре различных систем.

В результате, систематизируя приведенные выше рекомендации, можно сформулировать требования для реализации контроля, регистрации и маркировки печати документов в ОС с подсистемой печати CUPS:

1. Модуль контроля и маркировки печати необходимо реализовывать в виде фильтра обработки данных, активизирующегося в конце преобразования данных в формат PostScript. Для «не PostScript»-принтеров любые задания на печать для стандартного сервера CUPS нужно запретить.

2. Для корректной идентификации субъекта доступа, который запросил вывод документа на печать, требуется ввести ограничение на наличие в системе (на момент печати) только одного пользователя ОС. Печать при одновременной работе нескольких пользователей для стандартного сервера печати CUPS нужно запретить.

3. В качестве метки конфиденциальности документа необходимо использоваться уровень доступа субъекта, запросившего вывод на печать.

4. Для идентификации принтеров, на которые осуществляется вывод соответствующих документов на печать, вместо имени или краткого обозначения необходимо использовать Vendor ID или IP-адрес (последнее для сетевого принтера).

5. Субъекту доступа, соответствующему CUPS-серверу печати (cupsd), необходимо назначать максимально возможную метку в рамках мандатной политики управления доступом.

Очистка освобождаемых областей оперативной памяти и остаточной информации носителей данных, изоляция модулей

Требования по очистке памяти в соответствии с нормативными документами РФ в области защиты информации предъявляются по отношению к нескольким сущностям – для оперативной памяти СВТ и для остаточной информации на носителях данных (в том числе внешних, подключаемых к СВТ).

В оперативную память в процессе работы любого СВТ и любой ОС загружается обрабатываемая информация, в том числе и критически важная, такая как: ключи шифрования или подписи, пароли пользователей, конфиденциальные и другие защищаемые данные. В ОС GNU/Linux, обладая необходимыми привилегиями, существует возможность получить доступ на чтение к оперативной памяти (/dev/mem, /dev/kmem), и, следовательно, получить доступ к информации, которая не была удалена. В связи с этим необходимо очищать память при освобождении или перераспределении, записывая в соответствующие области последовательность из нулей или произвольных значений.

При реализации модуля очистки оперативной памяти необходимо [37]: определить что и в какой момент нужно очищать, не вызвав при этом нарушений в работе приложений или ядра Linux; корректно определить адреса начала и конца подлежащих очистке данных; использовать методы очистки, не оказывающие сильного влияния на производительность системы.

В GNU/Linux все области виртуальной памяти, которые выделяет ядро ОС, описываются структурой *struct vm_area_struct* (определяется с помощью функции find_vma() по виртуальному адресу в памяти) [3, 45]. В этой структуре содержатся адреса начала и конца области памяти, права доступа к памяти и другие флаги (например, VM_SHARED, который обозначает разделяемую область памяти). Вся память, выделенная процессу ОС GNU/Linux расположена в областях памяти, описываемых структурой *vm_area_struct*. С другой стороны, существует более детальное деление процесса ОС на секции и сегменты, такие как: куча (англ. heap), стек, секция инициализированных данных (.data), секция кода (.text), секция неинициализированных данных (.bss), секции инициализированных и неинициализированных данных для каждой загруженной в память разделяемой библиотеки, что представлено на Рисунке 3.1. Каждая секция или сегмент расположены в одной или нескольких областях *vm_area_struct*. Для очистки оперативной памяти, в первую очередь, интересны секции и сегменты, непосредственно хранящие какие-либо данные. Такими секциями и сегментами являются куча, отображаемые в оперативную память файлы, секции данных приложения и библиотек, стек приложения, стеки порождаемых приложением потоков.

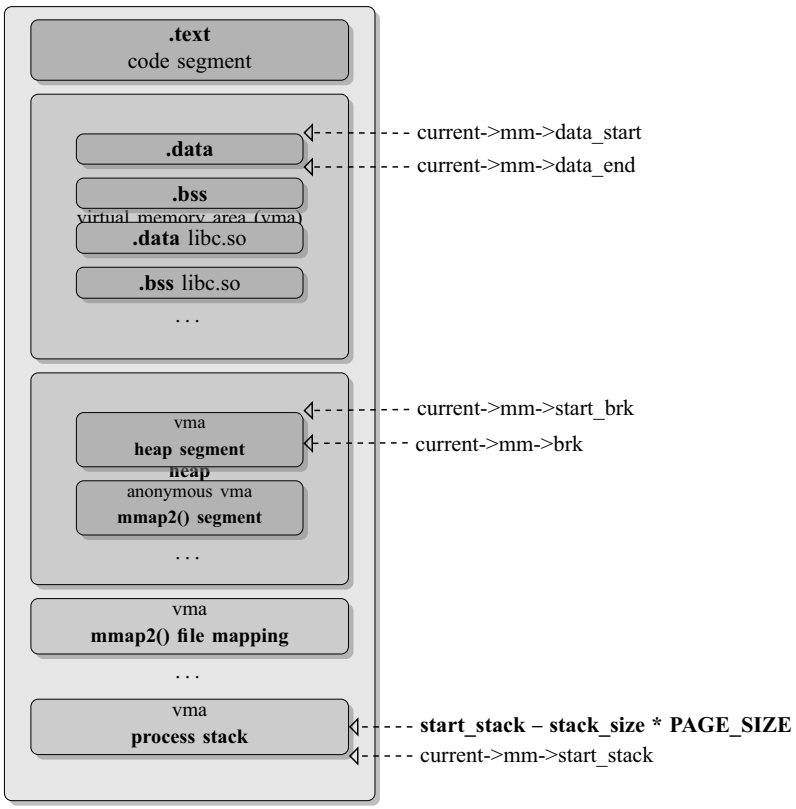

Рисунок 3.1 – Схематичное представление подлежащих очистке областей памяти, используемых текущим процессом (current)

Куча представляет собой область памяти, за выделение нового диапазона адресов которой отвечают функции malloc() (выделение памяти), realloc() (изменение размера выделенной памяти) и free() (освобождение памяти) из стандартной библиотеки glibc. При относительно маленьких размерах памяти (в зависимости от значения параметра M_MMAP_THRESHOLD), память выделяется в сегменте кучи приложения, начальный и конечный (текущий) адреса можно узнать из структуры процесса *task_struct*. При этом все операции с кучей выполняются с помощью системного вызова brk() или sbrk(). При вызове функции free(), brk может не вызываться для уменьшения размера кучи (например, при освобождении нескольких байт вызов brk будет отложен, а память зарезервирована для последующего выделения). При работе с памятью больших размеров, функция malloc() использует системный вызов mmap2 с флагом MAP_ANONYMOUS (выделяется еще одна область памяти, а ее структура *vm_area_struct* добавляется в список областей памяти процесса), realloc() – mremap, а free() – munmap. Таким образом, для очистки кучи необходимо встраивать механизмы защиты в функции brk, mmap2, mremap, munmap.

При вызове brk необходимо очищать память между передаваемым значением адреса и до текущего конца кучи, в случае если параметр с адресом меньше текущего конца кучи (current->mm) и больше адреса начала. При вызовах mremap и munmap необходимо очищать память соответствующей структуры vm_area_struct (в случае если память не VM_SHARED и есть права на запись) [37].

При завершении приложения, то есть при выполнении системного вызова exit_group, необходимо очищать секции данных приложения и всех подключенных к нему разделяемых библио-

тек. Из структуры процесса *task_struct* можно определить адрес начала и конца секции данных приложения. Секции данных для разделяемых библиотек расположены в одной области памяти, для определения дескриптора которой необходимо использовать функцию find_vma(), далее из структуры *vm_area_struct* определяются начальный и конечный адреса памяти. Очищая память в этом диапазоне адресов, сразу очищаются и все указанные секции разделяемых библиотек.

Для стека приложения необходимо во время вызова exit_group из структуры *task_struct* текущего процесса (current) определить адрес начала и размер стека в страницах и произвести очистку. При очистке стека необходимо помнить о том, что адрес начала (current->mm->stack_start – current->mm->stack_size * PAGE_SIZE) бывает больше или меньше адреса конца стека в зависимости от используемой архитектуры и программной платформы [54] – стек может увеличиваться «вверх» или «вниз».

Поток (англ. thread) представляет собой «облегченный» процесс ОС GNU/Linux, который может наследовать от главного процесса различные ресурсы (в зависимости от флагов наследования при создании потоков – CLONE_VM, CLONE_FS, CLONE_FILES, CLONE_SIGHAND). Обычно все потоки имеют собственную «облегченную» структуру *task_struct*, свой уникальный PID, но имеют один и тот же идентификатор группы TID (Thread ID – PID главного процесса). В связи с этим способ очистки стека потока не будет отличаться от очистки стека приложения (только при вызове exit).

Вызовы mmap, mremap и munmap могут использоваться для отображения файлов в оперативную память. Описанный способ очистки памяти для кучи в этом случае необходимо дополнить проверкой существования страницы для адреса в памяти, очистку проводить для существующих страниц целиком (по умолчанию 4096 байт) [37].

Если к моменту завершения программы (системный вызов exit_group) не освобождены какие-либо области памяти отображенных файлов, или просто анонимные области памяти (не принадлежащие файлу, выделенные с помощью mmap с флагом MAP_ANONYMOUS), их необходимо очистить в случае, если они не принадлежат файловым отображениям, в них разрешена запись и они не являются разделяемой памятью.

Таким образом, используя предложенные в данном разделе рекомендации становится возможным реализовать модуль очистки оперативной памяти для подсистемы управления доступом, который удовлетворяет предъявляемым к нему требованиям и позволяет полностью очищать области оперативной памяти при ее освобождении или перераспределении. При реализации такого модуля очистки памяти экспериментально была зафиксирована некоторая потеря производительности приложений и ОС GNU/Linux, в соответствии с чем для оценки его негативного влияния в дальнейшем необходимо провести экспериментальные исследования.

Как было отмечено ранее, в современных системах повсеместно используется концепция виртуальных адресных пространств, в которой для каждого процесса ОС создается иллюзия доступности всего объема оперативной памяти СВТ, а пространства разных процессов (в том числе одного пользователя) изначально изолированы друг от друга. За реализацию виртуальных адресных пространств отвечает механизм защиты страниц оперативной памяти ядра Linux

(подсистема управления памятью). Однако на уровне ядра ОС существует возможность получить доступ к определенным страницам памяти, а также изменять их атрибуты доступа – можно построить новое (временное) отображение виртуальных адресов в физические адреса страниц или сегментов памяти и назначить ему другие атрибуты доступа. Таким образом, в подсистеме управления доступом должен существовать модуль, контролирующий механизм защиты страниц или сегментов оперативной памяти ядра ОС в части соответствия атрибутов доступа, назначенных физическим блокам памяти по всем построенным виртуальным адресам.

Для очистки остаточной информации при удалении файловых объектов в ОС GNU/Linux необходимо контролировать системный вызов unlink. При этом нужно учитывать, что вследствие особенностей ОС GNU/Linux для каждого inode (указателя на данные файлового объекта), может существовать множество имен в файловой системе (см. разд. 2.2.2). Соответственно реальное удаление файлов проводится только тогда, когда количество жестких ссылок (имен в файловой системе, соответствующих данному inode) становится равным нулю, то есть равно единице при вызове unlink. Непосредственно очистку остаточной информации можно проводить с помощью различных методов уничтожения или стандартов, однако при их реализации с использованием даже двух проходов перезаписи существует вероятность понизить производительность операции удаления в несколько раз. В соответствии с этим в отношении очистки остаточной информации в дальнейшем необходимо провести экспериментальные исследования и, возможно, определить целесообразность использования этого механизма только при удаления защищаемых объектов доступа [77], а не в масштабах всей системы.

В результате, систематизируя приведенные выше рекомендации, можно сформулировать требования для очистки освобождаемых областей оперативной памяти и остаточной информации носителей данных, а также изоляции модулей:

1. При очистке областей оперативной памяти:
   – очищать память кучи, отображаемых в память файлов, секции данных и библиотек, стек приложения, стеки порождаемых потоков;
   – для очистки кучи и отображаемых файлов встраивать обработчики в системные вызовы brk/sbrk, mmap2, mremap и munmap, затирая память между передаваемыми значениями адресов и до текущего конца кучи либо диапазон адресов всей структуры *vm_area_struct* (если память не разделяемая, страница в памяти существует и к ней есть права на запись);
   – для очистки секций данных приложения и разделяемых библиотек встраивать обработчики в системный вызов exit_group, затирая память по адресам из структуры процесса или из соответствующего дескриптора области памяти;
   – для очистки стека приложения и порождаемых потоков встраивать обработчики в системный вызов exit_group или exit, затирая память от начала стека (current->mm->stack_start – current->mm->stack_size * PAGE_SIZE) и до конца (в зависимости от аппаратной архитектуры).
2. При очистке остаточной информации носителей данных:
   – встраивать обработчик в системный вызов unlink;

– очищать остаточную информацию при вызове unlink только, когда количество жестких ссылок на объект равно 1 (до обработки системного вызова);

– по необходимости очищать остаточную информацию только при удалении защищаемых объектов доступа, а не в для любых объектов всей системы.

3. Для изоляции модулей необходимо контролировать механизм защиты страниц или сегментов оперативной памяти ядра ОС в части соответствия атрибутов доступа, назначенных физическим блокам памяти по всем построенным виртуальным адресам.

3.3. Обоснование рекомендаций по практическому использованию разработанных алгоритмов обеспечения безопасности управления доступом в ОС GNU/Linux

Для описания всех программно-технических решений по созданию подсистемы управления доступом в GNU/Linux, соответствующей сформированным в разделе 3.1 требованиям, рассмотрим постепенно различные ее компоненты.

Встраивание в ядро операционной системы с гарантией его статической неизменности, корректность сочетания с другими средствами защиты информации

Для выполнения требований 1 и 3 из раздела 3.1 необходимо реализовать для подсистемы управления доступом алгоритм встраивания функций защиты на раннем этапе загрузки операционной системы. Как было отмечено в разделе 2.4 реализовать такое встраивание подсистемы управления доступом в ОС можно несколькими способами [22, 36, 48]: с помощью перехвата функций ядра Linux, системных вызовов, или обработчиков исключения, а также с использованием возможностей GNU/Linux (LSM).

Встраивание при помощи перехвата системных вызовов Linux ((1) на Рисунке 3.2) является наиболее универсальным способом для различных POSIX/SUS-совместимых ОС. Перехват системных вызовов можно осуществлять как на уровне записей таблицы системных вызовов (заменяя адреса вызовов в соответствующих полях таблицы), так и на уровне перехвата самих обработчиков системных вызовов как функций ядра Linux (см. разделы 1.1.1 и 2.4). С помощью такого способа встраивания предоставляется возможность активировать защитные механизмы подсистемы управления доступом до отработки стандартных механизмов ОС GNU/Linux.

При загрузке подсистемы управления доступом в виде LKM необходимо при инициализации заменять адреса системных вызовов на адреса переопределяющих их функций, а при выгрузке модуля совершать обратное действие. Однако реализовать описанный подход можно только в случае, когда известен адрес таблицы системных вызовов в памяти ядра Linux (с версии ядра 2.5.41 этот адрес перестал экспортироваться), при этом замена адресов системных вызовов является потенциально опасным действиям и может приводить к различным ошибкам в функционировании ядра (в ядре версии 2.6 и выше область памяти ядра с таблицей системных вызовов защищена от записи).

Адрес таблицы системных вызовов можно найти либо в файле /boot/System.map (см. разд. 1.1.1), либо более универсально – напрямую определяя его в памяти ядра Linux, например, с

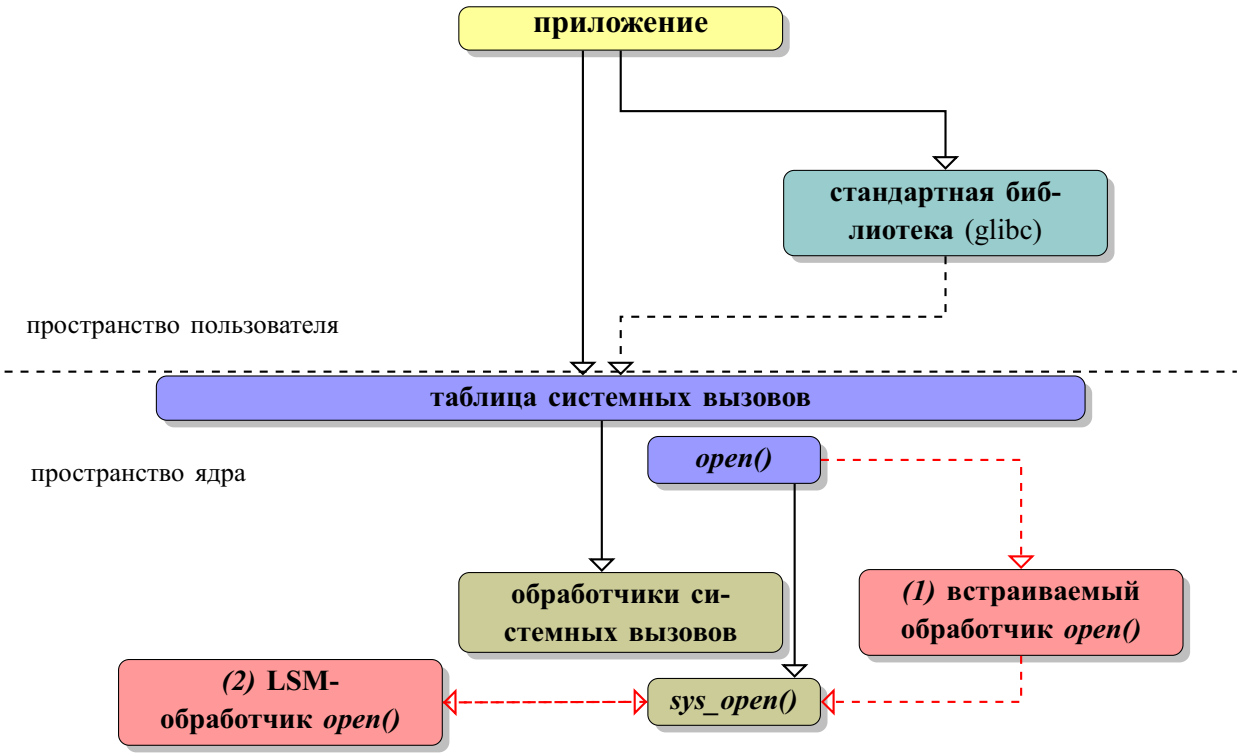

Рисунок 3.2 – Встраивание обработчиков подсистемы управления доступом с помощью перехвата системного вызова или с использованием технологии LSM

помощью разбора адресов соседних структур данных или подбора смещения для элементов таблицы системных вызовов путем сравнения их значения с эталонным по известному порядку записи (см. Листинг Б.1) [36].

По умолчанию возможности изменения sys_call_table в ядре версии 2.6 и выше не существует, так как эта таблица располагается в области памяти «read-only», тем самым обеспечивается ее защита от намеренного или непреднамеренного изменения или порчи. В связи с этим, можно либо временно (на время внесения изменений в таблицу) переключать режим доступа к памяти, либо строить новое страничное отображение к памяти с доступом на запись (см. Листинг Б.2). При этом первый способ возможно реализовать отключением блокировок типа GPF (General Protection Fault), для архитектуры x86 этого можно добиться установкой бита WP (Write Protect) в регистре CR0 (Control Register 0) в 0 (для включения – в 1, см. Листинг Б.3) [92]. Однако данный способ временно отключает защиту от записи для всех страниц памяти ядра Linux, в связи с чем потенциально небезопасен. Также любые изменения таблицы системных вызовов необходимо совершать атомарно для избежания проблем в многопроцессорных системах с консистентностью данных и синхронизацией процессов, осуществляющих чтение (запись) из этой таблицы.

Другим распространенным способом встраивания является использование возможностей ядра Linux, а точнее технологии LSM (Linux Security Modules) [99], которая позволяет встраивать собственные обработчики после стандартный механизмов защиты ОС GNU/Linux ((2) на Рисунке 3.2) без необходимости самому подменять функции ядра, адреса системных вызовов или разбираться со структурой и последовательностью использования функций и ресурсов ядра.

LSM – это набор предустановленных в ядре ОС перехватчиков (хуков), которые предоставляют API (Application Programming Interface) для внедрения собственных обработчиков непосредственно перед выполнением определенного системного вызова (после встроенных механизмов защиты ОС). Тем самым дополнительно ядро Linux защищается от преднамеренной или непреднамеренной порчи таблицы системных вызовов. Пример LSM-модуля ядра Linux, дополняющий стандартный вызов mkdir для создания каталога своим обработчиком, приведен в Листинге Б.4. Однако в целях безопасности начиная с версии ядра 2.6.24 и выше механизм LSM (функция, позволяющая использовать данный механизм) перестал экспортироваться ядром Linux, в связи с чем для реализации встраивания этим способом необходимо искать и использовать указатели из структуры security_ops или security_hook_heads вручную (сами же обработчики LSM присутствуют в ядре ОС, поэтому дополнительно к поиску и замене указателей никаких действий не требуется).

Таким образом для встраивания подсистемы управления доступом в ОС можно сформулировать следующие рекомендации:

1. При встраивании в таблицу системных выводов или в их системные обработчики:
   – при загрузке подсистемы управления доступом заменять адреса соответствующих функций на адреса обработчиков, сохраняя оригинальные адреса до выгрузки;
   – адрес таблицы системных вызовов или системных обработчиков искать в памяти с помощью разбора адресов соседних структур;
   – перед заменой соответствующих функций переключать режим доступа к памяти или строить новое страничное отображение (на запись, в атомарном контексте);
   – после замены соответствующих функций переключать режим доступа к памяти обратно (на чтение) или удалить построенное страничное отображение;
   – при выгрузке подсистемы управления доступом заменять адреса обработчиков на оригинальные адреса функций по аналогии с пунктами выше.
2. При встраивании обработчиков с помощью LSM:
   – при загрузке подсистемы управления доступом регистрировать ее обработчики в LSM для соответствующих функций;
   – адреса указателей на security_ops, security_hook_heads или другие необходимые для выполнения предыдущего пункта структуры данных искать в памяти ядра ОС;
   – при выгрузке подсистемы управления доступом отменить регистрацию ее обработчиков в LSM для соответствующих функций.

Предложенные рекомендации по встраиванию подсистемы управления доступом позволяют частично реализовать алгоритм встраивания функций защиты на раннем этапе загрузки операционной системы, предложенный в разделе 2.4, для определенных системных вызовов ядра Linux. При этом, в обоих случаях статическая неизменность ядра ОС обеспечивается тем, что встраивание реализовано с помощью загружаемого модуля, то есть целостность самого образа ядра Linux не нарушается, но должна дополнительно контролироваться (см. следующие пункты с рекомендациями). В соответствии с разделом 2.4 при встраивании подсистемы управления доступом с использованием предложенных рекомендаций будет обеспечиваться корректность при

сочетании с другими механизмами защиты ОС GNU/Linux (при использовании технологии LSM – автоматически, в других случаях – при вызове оригинальных обработчиков вместо положительного кода возврата, как в Листинге Б.3), однако в отношении этого необходимо провести экспериментальные исследования.

Обеспечение вызова соответствующих функций безопасности при осуществлении защищаемых типов доступа

В соответствии с разделом 2.4 разработанные выше рекомендации по встраиванию обработчиков подсистемы управления доступом в ОС необходимо использовать по отношению ко всем защищаемым и возможным типам доступа. При этом перечень необходимых для контроля типов доступа должен содержать: чтение, запись, исполнение, создание / удаление / переименование объектов и каталогов, а также операции с ссылками и переход в файловые объекты. В таблице системных вызовов приведенные типы доступа соответствуют следующим парам номеров ячеек и соответствующих обработчиков: (__NR_open, *sys_open*)[2], (__NR_mknod, *sys_mknod*), (__NR_mkdir, *sys_mkdir*), (__NR_unlink, *sys_unlink)*, (__NR_rmdir, *sys_rmdir*), (__NR_rename, *sys_rename*), (__NR_link, *sys_link*), (__NR_symlink, *sys_symlink*), а также некоторым другим (список может незначительно отличаться для различных версий ядра).

В случае же внедрения подсистемы управления доступом с использованием технологии LSM необходимо регистрировать следующие LSM-обработчики: *file_open()* (для версии ядра Linux $>= 3.5.0$) или *dentry_open()* ($>= 2.6.24$ и $< 3.5.0$) или *inode_permission()* ($< 2.6.24$); *path_mknod()*, *path_mkdir()*, *path_unlink()*, *path_rmdir()*, *path_rename()*, *path_link()*, *path_symlink()* (при конфигурации ядра с CONFIG_SECURITY_PATH) или *inode_create()*, *inode_mknod()*, *inode_mkdir()*, *inode_unlink()*, *inode_rmdir()*, *inode_rename()*, *inode_link()*, *inode_symlink()* (иначе); возможно дополнительно *path_truncate()*, *file_fcntl()*, *file_ioctl()* и некоторые другие (список может отличаться для различных версий ядра).

При этом посредством выполнения остальных требований раздела 3.1 использование предложенных в данном разделе рекомендаций по встраиванию будет автоматически обеспечивать передачу управления обработчику соответствующего системного вызова подсистемы управления доступом при осуществлении любого из контролируемых типов доступа.

Обеспечение активизации и непрерывности работы функций защиты от несанкционированного доступа

Для выполнения требования по обеспечению активизации подсистемы управления доступом при включении СВТ, начиная с раннего этапа загрузки, и непрерывности ее работы в процессе функционирования в ОС необходимо использовать алгоритм обеспечения доверенной загрузки загрузчика и ОС при пошаговом контроле целостности из раздела 2.3 и, частично, встраивание функций защиты от НСД на раннем этапе загрузки операционной системы из раздела 2.4. При этом в зависимости от используемой программной и/или аппаратной платформы для

[2]С различными атрибутами системный вызов *open* соответствует чтению, записи и/или исполнению, переходу в объекты-каталоги

доверенной загрузки загрузчика и ОС требуется использовать как существующие АМДЗ (архитектуры x86, x86_64), так и средства, позволяющие обеспечить технологическую невозможность нарушения целостности критически важных данных (например, на архитектуре s390x и arm, см. раздел 2.3).

При использовании АМДЗ применительно к ОС Linux, кроме выполнения необходимых организационных мер защиты [52], требуется контролировать целостность аппаратных средств СВТ, а также целостность программных компонент, перечисленных в Приложении А [25]. При использовании средств, обеспечивающих технологическую невозможность нарушения целостности хранящихся на них объектов, необходимо разместить все критически важные объекты ОС (раздел /boot и некоторые другие), в том числе участвующие в процессе загрузки СВТ, на специальных разделах с контролируемым на физическом уровне доступом к памяти, изначально доступных только для чтения [43] (см. раздел 2.3). Соответственно, в отношении всех объектов, для которых требовалось контролировать целостность с помощью АМДЗ (см. Приложение А), обеспечивается технологическая неизменность, а для некоторых из них (см. комментарии в таблице А.1) целостность можно контролировать в процессе работы СВТ и механизмов защиты подсистемы управления доступом. Дополнительно при использовании всех перечисленных средств необходимо также ограничить возможности загрузчика ОС по изменению сценариев загрузки в динамике в соответствии с рекомендациями из раздела 2.3.

Еще одним шагом в обеспечении активизации подсистемы управления доступом является ее встраивание на раннем этапе загрузки ОС GNU/Linux (до возникновения реальных субъектов или защищаемых объектов, до загрузки стандартных модулей ядра, до появления системных сервисов и процессов). Реализовать такое встраивание можно при использовании в процессе загрузки ОС образа начальной загрузки initrd/initramfs (на практике используется практически всегда) и активируя подсистему управления доступом, как минимум, непосредственно до монтирования основных файловых систем на запись (либо сразу после загрузки ядра ОС). В случае невозможности встраивания (активизации) подсистема управления доступом должна прерывать дальнейшую загрузку ОС, например вызывая панику ядра (функция *panic()*). Для дальнейшего контроля непрерывности работы защитных механизмов необходимо использовать в подсистеме управления доступом предложенный в разделе 3.2 модуль контроля механизма защиты страниц или сегментов оперативной памяти ядра Linux в части соответствия атрибутов доступа, назначенных определенным физическим блокам памяти по всем построенным виртуальным адресам (в случае нарушения соответствия атрибутов – приостанавливать работу ОС и сигнализировать об этом инциденте). Данного механизма будет достаточно при размещении загружаемой подсистемы управления доступом в памяти, недоступной на запись (для LKM выполняется всегда с параметрами CONFIG_DEBUG_RODATA / CONFIG_STRICT_KERNEL_RWX и CONFIG_DEBUG_SET_MODULE_RONX / CONFIG_STRICT_MODULE_RWX), в связи с чем дополнительного контроля целостности точек встраивания проводить не требуется.

Таким образом, систематизируя приведенные выше рекомендации, можно сформулировать требования для обеспечение активизации и непрерывности работы функций защиты от НСД:

1. В зависимости от архитектуры СВТ использовать АМДЗ или специальные средства, обеспечивающие неизменность хранящихся на них данных технологически.

2. При использовании АМДЗ выполнять необходимые организационные меры защиты (опечатывание и ограничение физического доступа к СВТ), контролировать целостность аппаратных средств СВТ и объектов из приведенного в Приложении А списка программных средств ОС.

3. В случае использования специальных средств обеспечения технологической неизменности данных размещать на них критически важные объекты ОС, перечисленные в таблице А.1 Приложения А, и установить режим доступа к памяти носителя только на чтение. Для оставшихся компонент ОС и подсистемы управления доступом из таблицы А.1 контролировать целостность динамически при осуществлении к ним доступа.

4. Ограничить возможность изменения сценариев загрузки ОС в динамике в соответствии с рекомендациями из раздела 2.3.

5. Встраивать подсистему управления доступом в образ начальной загрузки до монтирования всех ФС на запись.

6. При невозможности встраивания подсистемы управления доступом – прерывать дальнейшую работу ОС.

7. Для дальнейшего контроля непрерывности работы подсистемы управления доступом использовать предложенный в разделе 3.2 модуль контроля механизма защиты страниц или сегментов оперативной памяти ядра Linux. В случае нарушения соответствия атрибутов доступа к памяти (при попытке нарушить точки встраивания подсистемы управления доступом) – приостанавливать дальнейшую работу ОС и сигнализировать об этом инциденте.

Таким образом, выполнение приведенных рекомендаций обеспечивает невозможность каким-либо образом отключить или исключить активизацию встраиваемой подсистемы управления доступом, в том числе на ранних этапах загрузки ОС (см. Следствие из Базовой теоремы ИПСС из раздела 2.1). В результате будет гарантироваться загрузка и включение защитных механизмов подсистемы управления доступом, а загрузка ОС без этого шага будет невозможна.

Ограничение возникновения субъектов в системе, корректная идентификация субъектов и объектов доступа

С целью ограничения возникновения субъектов в системе, а также для их корректной идентификации при взаимодействии с объектами необходимо реализовать в подсистеме управления доступом несколько взаимодействующих между собой модулей: PAM-модуль для идентификации и аутентификации реальных пользователей ОС и модуль контроля порождения субъектов. Оба эти механизма призваны реализовать способ контроля порождения субъектов доступа и идентификации субъектов, от имени которых соответствующие процессы ОС GNU/Linux взаимодействуют с объектами системы, предложенный в разделе 2.2.1.

Модуль контроля порождения субъектов должен в момент загрузки подсистемы управления доступом пометить специальным дескриптором безопасности все существующие в ОС процессы, как выполняющиеся от имени системного субъекта доступа (shadow root). В дальнейшем этот модуль должен помечать все вновь создаваемые процессы ОС соответствующими дескрип-

торами безопасности по правилам, приведенным в разделе 2.2.1 и в Таблицах 2.1 и 2.2 из раздела 2.1. Для этого подсистемой управления доступом дополнительно к приведенному ранее списку должны контролироваться следующие пары номеров системных вызовов ОС и соответствующих им обработчиков: (__NR_setuid, *sys_setuid*), (__NR_fork, *sys_fork*), (__NR_execve, *sys_execve*/*compat_sys_execve*) и, возможно, некоторые другие.

В случае внедрения подсистемы управления доступом с использованием технологии LSM необходимо регистрировать следующие LSM-обработчики: обработчики для аллокации дескрипторов безопасности при выполнении *fork()* или завершении процессов (*task_create()*, *cred_prepare()*, *cred_free()* для версии ядра Linux $>= 2.6.29$ и дополнительно *cred_alloc_blank()*, *cred_transfer()* если версия ядра Linux $>= 2.6.32$; иначе – *task_alloc_security()*, *task_free_security()*, *bprm_alloc_security()* и *bprm_free_security()*); обработчики для обновления дескрипторов безопасности при выполнении *exec()* (*bprm_check_security()*; *bprm_compute_creds()* для версии ядра Linux $< 2.6.6$; *bprm_apply_creds()* для версии ядра Linux $< 2.6.29$ и $> 2.6.6$; *bprm_committing_creds()* для версии ядра Linux $> 2.6.29$); обработчик для смены дескрипторов безопасности (*task_fix_setuid()* для версии ядра Linux $>= 2.6.29$; иначе – *task_setuid()*).

PAM-модуль внедряемой подсистемы управления доступом должен вызываться перед стандартной идентификацией и аутентификацией в ОС, непосредственно перед процедурой санкционированной смены субъекта доступа (реального пользователя). При этом в случае успешной идентификации и аутентификации в процессе дальнейшего выполнения *setuid()* дескриптор безопасности текущего процесса в подсистеме управления доступом с помощью модуля контроля порождения субъектов меняется на соответствующий дескриптор субъекта-пользователя (иначе выполнение прерывается). Внедрение PAM-модуля описанным образом возможно при добавлении его до стандартного PAM-модуля ОС (pam_unix.so, pam_unix2.so или pam_tcb.so, см. раздел 1.1.1) во все возможные сценарии идентификации и аутентификации в /etc/pam.d/. При добавлении этого модуля с опцией *requisite* стандартный PAM-модуль ОС и прочие связанные с ним системные сервисы также будут отрабатывать в процессе и после идентификации и аутентификации пользователей, а при наличии у стандартного PAM-модуля опции *try_first_pass* повторный запрос пароля пользователя проводиться не будет. Таким образом, PAM-модуль подсистемы управления доступом внедряется в систему без нарушения работы всех существующих сценариев идентификации и аутентификации, расширяя их функциональность.

В подсистеме управления доступом также должен существовать модуль, реализующий один из способов идентификации объектов при взаимодействии с субъектами доступа, приведенный в разделе 2.2.2. В целях удобства в качестве механизма идентификации объектов был выбран наиболее естественный – с использованием абсолютных путей до файловых объектов. Для реализации этого механизма строится база данных (БД) прав доступа существующих субъектов к объектам (по абсолютному пути), а дополнительно к модулю идентификации объектов в подсистеме управления доступом реализован модуль контроля точек монтирования, использующий аналогичные /etc/fstab правила монтирования. Для реализации контроля точек монтирования в подсистеме управления доступом дополнительно должны контролироваться следующие пары номеров системных вызовов ОС и соответствующих им обработчиков: (__NR_mount,

sys_mount/*compat_sys_mount*), (__NR_umount2, *sys_umount*), (__NR_umount, *sys_oldumount*), а также (__NR_pivot_root, *sys_pivot_root*), (__NR_chroot, *sys_chroot*) и некоторые другие.

В случае внедрения подсистемы управления доступом с использованием технологии LSM необходимо регистрировать следующие LSM-обработчики: *sb_mount()*, *sb_umount()* (в случае необходимости еще *sb_post_addmount()*), а также *sb_pivotroot()*, *path_chroot()* и некоторые другие. При этом для исключения зависимости от конфигурации ОС в обоих случаях внедрения подсистемы управления доступом целесообразно для идентификации всех объектов в конечном счете использовать внутренние структуры ядра Linux (*inode*, *dentry* и *vfsmount*) в рамках виртуальной файловой системы (VFS), универсальные для всех используемых в GNU/Linux файловых систем носителей данных, перестраивая их соответствие абсолютным путям.

Также для корректной реализации идентификации по абсолютным путям в подсистеме управления доступом в обработчики для действий по созданию новых, переименованию существующих объектов доступа и созданию жестких ссылок добавлена логика по изменению и пополнению прав доступа в БД. При этом все изменения прав доступа могут вносится в БД только в случае успешного завершения соответствующих обработчиков перечисленных системных вызовов. Таким образом в подсистеме управления доступом обеспечивается корректность идентификации объектов при осуществлении к ним доступа в соответствии с разделом 2.2.2.

Необходимо отметить, что при использовании абсолютных путей в правилах подсистемы управления доступом существует возможность использовать маски для обозначения множества (вложенных) объектов. В случае совпадения объекта с несколькими масками в правилах подсистемы управления доступом необходимо выбирать в качестве действующей маску, наиболее точно соответствующую объекту (без наследования или агрегации правил от родительских объектов-каталогов).

Реализация принципа наименьших привилегий, безопасного централизованного администрирования и требования «самозащиты»

Посредством идентификации объектов доступа по абсолютным путям и формирования БД правил подсистемы управления доступом (см. подраздел выше) становится возможным организовать централизованную схему администрирования. При этом вся настройка подсистемы управления доступом может осуществляться по централизованной схеме специальным субъектом доступа – администратором безопасности, который изначально соответствует пользователю root. Более того, в данном случае может быть реализовано абсолютное разделение административных и пользовательских полномочий в соответствии с рекомендациями после Базовой теоремы ИПСС из раздела 2.1.

Для реализации безопасного централизованного администрирования с помощью абсолютного разделения административных и пользовательских полномочий в подсистеме управления доступом необходимо, в том числе, соблюдать принцип наименьших привилегий, который обеспечивается за счет выполнения остальных приведенных в данном разделе требований и условий Базовой теоремы ИПСС из раздела 2.1.

Первоначально необходимо обеспечить целостность объектов-источников для порождения субъектов доступа (*/bin/login*, */bin/su*, */usr/bin/sudo*, */usr/sbin/sshd* и некоторых других, а также бинарных файлов объектов-источников для системных сервисов) и ассоциированных с ними объектов (объектных или конфигурационных файлов, параметров учетных записей пользователей) в ходе работы системы (при выполнении системного вызова *open*) или перед порождением из них субъектов доступа (см. разделы 1.4 и 2.1). В случае нарушения целостности перечисленных объектов необходимо завершить осуществляющий доступ процесс, запретить создание сессий пользователей и вход в систему, а в дальнейшем необходимо предусмотреть возможность входа в ОС администратора безопасности (см. ниже) для анализа возникшего инцидента, но не других пользователей системы. Кроме того, целостность должна контролироваться при обращении к объектам, связанным с самой подсистемой управления доступом (ядром защиты), а также к объектам, доступных на чтение и выполнение всем субъектам системы (см. подробный список из Приложения А). При этом любые санкционированные изменения объектов, чью целостность необходимо контролировать, должны выполняться вне существования субъектов-пользователей (кроме администраторов) и сопровождаться обновлением соответствующих контрольных сумм.

Целесообразность использования для тех или иных объектов статического или динамического контроля целостности определяется с учетом их объема, а также зависит от необходимости оперативного обнаружения нарушений. Для общедоступных объектов, занимающих большой объем памяти, лучше применять динамический контроль целостности вследствие меньшего влияния этого механизма защиты на производительность системы (целостность каждого объекта контролируется только при осуществлении доступа к нему), в то время как для объектов-источников при порождении субъектов можно применять статический контроль целостности (при порождении субъекта целостность всех объектов контролируется единовременно).

Также вне зависимости от реализованной политики управления доступом (см. подраздел далее) права создания и изменения объектов, доступных на чтение и выполнение всем субъектам доступа, и объектов, ассоциированных со всеми процессами-источниками должны предоставляться выделенному субъекту – администратору ОС GNU/Linux. В то же время права создания и изменения объектов, связанных с подсистемой управления доступом, должны предоставляться выделенному субъекту доступа – администратору безопасности (может быть одним и тем же субъектом доступа с администратором ОС GNU/Linux). Права доступа к личным объектам пользователей должны предоставляться только самим пользователям. Пример прав доступа непривилегированного субъекта-пользователя приведен в Приложении А.

Реализация политик управления доступом должна по умолчанию быть в запретительном режиме работы (в соответствии с принципом наименьших привилегий, доступ к не упомянутым в правилах объектам запрещен). За счет встраиваемых обработчиков подсистемы управления доступом в отношении защищаемых объектов в таком случае будет запрещено беспрепятственно использовать свойство «владения» и передавать другим пользователем права на объекты доступа без ведома администратора [73] для всех без исключения субъектов доступа (включая root). При этом необходимо предусмотреть специальный механизм, который позволит создать все необходимые для работы пользователей объекты доступа автоматически [55, 80] (так называемый,

«мягкий» режим работы или режим обучения). В данном режиме необходимо дополнительно к созданию необходимых прав доступа для субъектов-пользователей, автоматически создать всех субъектов, соответствующих системным сервисам и назначить им необходимые права доступа. Таким образом, данный режим должен позволять формировать БД для подсистемы управления доступом в соответствии с «типичным» поведением ОС GNU/Linux. В «мягком» режиме работы подсистемы управления доступом желательно максимально точно воспроизвести (сэмулировать) работу пользователей и системных сервисов ОС GNU/Linux, так как полнота созданных в этом режиме субъектов, объектов и атрибутов доступа напрямую зависит от информации, собранной при работе в этом режиме.

Реализовав в подсистеме управления доступом все перечисленные рекомендации, а также учитывая другие предъявляемые к ней требования, становится возможным организовать в системе как процедуру безопасного централизованного администрирования (посредством абсолютного разделения административных и пользовательских полномочий), так и принцип наименьших привилегий. При этом подсистема управления доступом с учетом своей гарантированной активизации, первоначального обеспечения целостности и непрерывности работы (см. подраздел ранее), в дальнейшем будет осуществлять «самозащиту» своих компонент с помощью сочетания механизмов контроля целостности и разграничения доступа (см. Базовую теорему ИПСС из раздела 2.1). Вследствие этого в ОС GNU/Linux невозможно каким-либо образом отключить или изменить настройки такой подсистемы управления доступом, в том числе на ранних этапах загрузки системы и в процессе работы.

Выполнение базовых требований нормативных документов РФ по защите информации

Для выполнения требований нормативных документов РФ в области защиты информации в зависимости от категории защищаемых данных в подсистеме управления доступом необходимо реализовать дискреционную и/или мандатную политику управления доступом с контролем потоков. Необходимо отметить, что все рассматриваемые ранее способы и алгоритмы защиты из разделов 2.2–2.4, результаты исследования из раздела 2.1, а также предложенные в данном разделе программно-технические решения инвариантны относительно применяемой политики управления доступом и, кроме того, не зависят от конфигурации ОС. В связи с этим при реализации дискреционной и/или мандатной политики управления доступом по требованиям нормативных документов не возникает сложностей: соответствующие правила данных политик необходимо добавить в обработчики системных вызовов подсистемы управления доступом. При этом в дискреционной политике управления доступом должны использоваться атрибуты доступа, приведенные в соответствующей таблице Приложения А, а в качестве уровней доступа и конфиденциальности мандатной политики управления доступом – числа от 1 до, например, 16 (для части этих меток можно задать соответствующие обозначения: «общедоступно», «конфиденциально», «секретно», «совершенно секретно», «особой важности» и другие). Правила дискреционной политики управления доступом для объектов должны задаваться индивидуально для каждого пользователя (группы пользователей) в БД, тогда как метки конфиденциальности объектов в мандатной политике – глобально для всей системы.

Дополнительно к механизмам защиты, описанным в подразделах выше в подсистеме управления доступом необходимо организовать регистрацию и учет всех событий безопасности. В качестве событий безопасности необходимо регистрировать все попытки НСД и, как минимум, все операции по смене субъектов доступа для процессов ОС GNU/Linux (setuid), события идентификации и аутентификации (login, logout). При этом регистрироваться должны следующие параметры: порядковый номер события, дата и точное время (unix / posix time), PID и имя осуществляющего доступ процесса и его родительского процесса, уровень детальности класса событий, класс событий (операции с объектами доступа, операции с процессами ОС), тип события (чтение, запись, исполнение и так далее), результат выполнения (разрешено, запрещено мандатной или дискреционной политикой управления доступом, ошибка контроля целостности, ошибка доступа в ОС), тип осуществляющего доступа субъекта (shadow или user) и его идентификатор, объект доступа. Для различных классов и типов событий целесообразно назначать разные уровни журналирования, в зависимости от которых тот или иной доступ будет полностью регистрироваться в журнале работы подсистемы управления доступом (а не только попытки нарушения соответствующего правила доступа). При этом, для ряда типов доступа необходимо устанавливать минимальный уровень журналирования (то есть регистрация этих событий будет включена всегда) – setuid, login, logout и все события с нарушениями правил доступа. Для событий по получению доступа к объектам на чтение и запись необходимо устанавливать наиболее высокий уровень журналирования (фиксируя только факты нарушения), так как такие доступы в современных дистрибутивах GNU/Linux осуществляются достаточно интенсивно и размер журнала, например, хода загрузки ОС может составлять десятки мегабайт, чего желательно избежать.

Требования нормативных документов РФ в области защиты информации к очистке памяти и остаточной информации, а также маркировке печатаемых документов должны выполняться с помощью соответствующих модулей подсистемы управления доступом, работа которых рассмотрена в разделе 3.2. Необходимо отметить, что для реализации модуля очистки оперативной памяти в подсистеме управления доступом дополнительно должны контролироваться следующие пары номеров системных вызовов ОС и соответствующих им обработчиков: (__NR_brk, *sys_brk*), (__NR_mmap, *sys_mmap*), (__NR_munmap, *sys_munmap*) и некоторые другие.

Таким образом, разработанные в данной работе способы и алгоритмы защиты из разделов 2.2–2.4, результаты исследования из раздела 2.1, а также предложенные в данном разделе программно-технические решения позволили создать подсистему управления доступом «Аккорд-X», функционально состоящую из следующих компонент [38]:

– монитор разграничения доступа (МРД), который должен загружаться в момент старта ОС GNU/Linux до монтирования корневой файловой системы на запись для реализации основных защитных функций (разграничение доступа субъектов к объектам, динамический контроль целостности, очистка памяти и другие);

– модуль идентификации и аутентификации субъектов-пользователей в ОС (PAM), взаимодействующий с МРД (сами эти процедуры выполняются в МРД);

– модуль статического контроля целостности, инициализируемый МРД (в том числе при взаимодействии с PAM-модулем);

– модуль журналирования, ответственной за запись событий безопасности МРД;

– модуль контроля печати на уровне подсистемы печати ОС GNU/Linux (CUPS), взаимодействующий с МРД;

– утилиты администрирования комплекса для редактирования файлов конфигурации, БД пользователей и прав доступа МРД.

Ключевым элементом подсистемы управления доступом «Аккорд-X» является МРД, который выполнен в виде модуля ядра Linux. Данный модуль фактически выполняет всю основную функциональность подсистемы управления доступом, а именно:

– в момент загрузки регистрирует собственные обработчики системных вызовов в ядре Linux, в дальнейшем реализуя с их помощью дискреционную и/или мандатную политику управления доступом в ОС;

– реализует модуль контроля порождения субъектов доступа, который начиная с раннего этапа загрузки ОС присваивает всем процессам на уровне ядра ОС определенные дескрипторы безопасности в соответствии с описанными ранее рекомендациями;

– при идентификации и аутентификации субъектов-пользователей в ОС GNU/Linux осуществляет взаимодействие с PAM-модулем аутентификации для проверки корректности введенных пользователем данных (в качестве идентификационного признака используются аппаратные идентификаторы);

– реализует модули идентификации объектов доступа и контроля точек монтирования;

– проводит статический (с модулем статического контроля целостности) и динамический (в обработчиках МРД) контроль целостности данных:

– проверяет права в соответствии с собственной БД при осуществлении любого системного вызова на получение доступа к объекту ОС GNU/Linux;

– реализует модуль очистки оперативной памяти и модуль очистки остаточной информации на носителях данных;

– реализует модуль контроля механизмов защиты страниц ядра Linux;

– фиксирует все события безопасности и передает их в модуль журналирования.

Загрузку МРД в ядро Linux необходимо обеспечивать на раннем этапе загрузки ОС, в связи с этим общий порядок работы СВТ с внедренной подсистемой управления доступом «Аккорд-X» имеет следующий вид:

1. Первоначально управление передается штатному BIOS (UEFI) СВТ, инициализируется оборудование (если установлена парольная защита — BIOS требует ввести пароль).

2. Сразу после процедуры RomScan по поиску расширений BIOS управление перехватывает АМДЗ (в случае его использования в соответствии с алгоритмом из раздела 2.3), проводятся процедуры идентификации и аутентификации пользователя, контроль целостности и наличия аппаратных и программных компонент СВТ (в том числе целостность BIOS и других компонент), в случае же использования средств, обеспечивающих технологическую невозможность нарушения целостности критически важных объектов, вместо контроля целостности обеспечивается неиз-

менность следующих компонент (см. подробнее Приложение А): загрузчик ОС GNU/Linux; образ начальной загрузки системы initrd (initramfs), в скрипт инициализации которого встраивается МРД «Аккорд-Х» до выполнения /sbin/init; ядро ОС Linux – vmlinux; БД и файлы конфигурации «Аккорд-Х» (дальнейший контроль доступа и целостности к этим компонентам осуществляется самим МРД, обеспечивая «самозащиту»).

3. В случае успеха управление передается стандартному загрузчику ОС.

4. Загрузчик загружает ядро ОС и образ initrd в память (загрузчик предварительно необходимо настроить в соответствии с разделом 2.3), передает управление ядру Linux, распаковывается образ начальной загрузки, начинается ранний этап загрузки ОС GNU/Linux.

5. В ядро ОС загружается МРД, начинают работать соответствующие механизмы защиты.

6. Инициализируются необходимые драйверы ОС, подключаются прочие файловые системы, запускается сценарий /sbin/init, корневая ФС монтируется на запись.

7. Запускаются необходимые системные процессы и сервисы, завершается этап загрузки ОС GNU/Linux, загружаются соответствующие процессы для процедур идентификации и аутентификации пользователя и старта его сессии (с помощью PAM-модуля «Аккорд-Х»).

8. PAM-модуль запрашивает идентификатор пользователя и пароль, рассчитывает хэш-функцию от этих параметров и передает значения идентификатора и хэш-функции МРД.

9. МРД проверяет корректность полученных значений, возвращает PAM-модулю код ошибки или имя пользователя, в случае успешной идентификации и аутентификации аутентифицируемый процесс ОС помечается атрибутом соответствующего пользователя для дальнейшей смены субъекта доступа (в соответствии с механизмом из разд. 2.2.1).

10. При запуске пользовательской сессии выполняется операция setuid(), в случае наличия атрибута аутентифицированного пользователя у процесса, для него изменяется субъект доступа в дескрипторе безопасности «Аккорд-Х». Все дальнейшие процессы пользователя будут наследовать этот дескриптор безопасности.

Как можно видеть, ключевым моментом в процессе загрузки СВТ с подсистемой управления доступом «Аккорд-Х» и ОС GNU/Linux является контроль целостности или обеспечение неизменности образа начальной загрузки, ядра ОС и прочих компонент системы (начальная фаза работы, достижение ИПСС). Посредством этого обеспечивается дальнейшая (доверенная) активизация дискреционного и мандатного управления доступом, статического и динамического контроля целостности, контроля порождения субъектов доступа и других механизмов (вторая фаза работы, работа в режиме ИПСС). Организовав приведенным выше образом загрузку СВТ с внедренной подсистемы управления доступом и ОС GNU/Linux, становится возможным выполнить все сформированные в разделе 3.1 требования, а также обеспечить комплексирование и усиление защитных свойств встроенных средств защиты GNU/Linux, корректность их функционирования и необходимое резервирование, которое обеспечивает избыточность и оптимальность защитных механизмов в ОС.

На основе предложенной автором подсистемы управления доступом был разработан ПАК СЗИ НСД «Аккорд-Х», производства компании ОКБ САПР (свидетельство о государственной регистрации № 2015612555 [61], см. Приложение Г). Данный программно-аппаратный комплекс

включает в себя разработанную автором подсистему управления доступом в ОС GNU/Linux, а также аппаратное устройство – аппаратный модуль доверенной загрузки «Аккорд-АМДЗ» или мобильное средство организации доверенного сеанса связи «МАРШ!», которые позволяют контролировать целостность или обеспечивать неизменность критически важных компонент системы на раннем этапе работы. Результаты диссертационного исследования и разработанная подсистема управления доступом в составе ПАК СЗИ НСД «Аккорд-X» внедрены в ряде государственных ИС, используются ГНИИ ПТЗИ ФСТЭК России в рамках исследований уязвимостей системного ПО для национального банка данных угроз безопасности информации и для верификации функциональных требований к средствам защиты информации от НСД на раннем этапе загрузки ОС.

ПАК СЗИ НСД «Аккорд-X» получил сертификат соответствия требованиям ФСТЭК России № 3079, на основе которого данный комплекс соответствует требованиям нормативных документов в области защиты информации РФ, предъявляемым к СЗИ НСД, для 3 класса защищенности СВТ и 2 уровня контроля отсутствия недекларированных возможностей. Подсистема управления доступом комплекса «Аккорд-X» получила отдельный сертификат соответствия требованиям ФСТЭК России № 4447 – для 5 класса защищенности СВТ и 4 уровня контроля отсутствия недекларированных возможностей.

3.4. Выводы к главе 3

1. На основе результатов проведенного ранее исследования сформированы требования к подсистеме управления доступом в ОС, которые дополняют требования существующих нормативных документов в области защиты информации в части необходимости обеспечения активизации и непрерывности работы функций защиты от НСД, а их выполнение позволяет устранить недостатки существующих средств и способов защиты данных в GNU/Linux. Обоснована невозможность выполнения всех разработанных требований существующими средствами разграничения доступа в GNU/Linux, а также при сочетании нескольких таких средств. Показано, что сформированные в работе требования имеют широкое назначение и применимы в отношении подсистем управления доступом ОС программных платформ, отличных от GNU/Linux.

2. На базе полученных научных результатов разработана подсистема управления доступом для ОС GNU/Linux и основанный на ней программно-аппаратный комплекс «Аккорд-X», удовлетворяющие разработанным требованиям и обладающие свойствами, не характерными для существующих средств защиты информации от несанкционированного доступа, к числу которых относятся: обеспечение активизации на раннем этапе загрузки СВТ и непрерывности дальнейшей работы, невозможность отключения или удаления несанкционированным образом, невозможность загрузки ОС без подсистемы управления доступом (гарантированное выполнение функций защиты информации от НСД); гарантия вызова функций защиты от НСД при осуществлении любого защищаемого типа доступа с возможностью реализации принципа наименьших привилегий в отношении всех субъектов ОС; обеспечение защиты от НСД как объектов, так и пользовательской среды, в которой непосредственно происходит обработка информации в GNU/Linux; обеспечение «самозащиты» компонент подсистемы управления доступом; воз-

можность реализации безопасного централизованного администрирования; соответствие требованиям нормативных документов РФ в области защиты информации, сертифицированность по 3 классу защищенности СВТ и 2 уровню контроля отсутствия недекларированных возможностей; корректность в соответствии с формальными моделями безопасности (соответствие разработанной в разделе 2.1 модели безопасности, которая является прототипом для разработанной подсистемы управления доступом) и при сочетании с существующими средствами защиты ОС GNU/Linux; независимость от конфигурации ОС и аппаратной платформы.

3. Применение ПАК СЗИ НСД «Аккорд-X» позволяет уменьшить количество критичных компонент ОС до ядра Linux и внедренных политик управления доступом (исключая привилегированные приложения и другие объекты). При этом нарушение этих политик вследствие реализации в системе ИПСС может вести только к реализации угроз безопасности для данных определенного субъекта-пользователя, но не к компрометации всей системы в целом. Таким образом, использование «Аккорд-X» позволяет как усилить защитные свойства компонент ОС GNU/Linux вследствие комплексирования, так и снизить возможность реализации угроз безопасности информации и потенциальный ущерб.

Кроме применения разработанной в данном разделе подсистемы управления доступом, для защиты ОС GNU/Linux необходимо дополнительно применять рассмотренные ранее встроенные и расширенные средства защиты (в том числе исправления для увеличения защитных свойств ядра Linux), а также следить за выходом и актуальностью обновлений различных компонент ОС и сочетать все перечисленное с организационными мерами по ограничению физического доступа к защищаемому СВТ.

4. Анализ опыта применения разработанных алгоритмов обеспечения безопасности и средства разграничения доступа в ОС GNU/Linux

4.1. Методика проведения экспериментальной апробации разработанных алгоритмов обеспечения безопасности и средства разграничения доступа в ОС GNU/Linux

В главе 3 на основе предложенных автором способов и алгоритмов защиты разработана подсистема управления доступом для ОС GNU/Linux, в отношении которой в данном разделе необходимо:

– исследовать особенности встраивания, корректности взаимодействия и сочетания с другими применяемыми в ОС средствами защиты и реализованными формальными моделями безопасности;

– оценить эффективность использования применяемых способов и алгоритмов защиты для усиления защищенности системы и снижения возможности возникновения НСД (в части соответствия реализации подсистемы управления доступом предложенной ранее формальной модели безопасности): невозможность отключения или несанкционированного изменения правил разграничения доступа, невозможность возникновения несанкционированных субъектов и неучтенных объектов доступа, невозможность «размыкания» изолированной программной среды субъектов, полнота реализованных механизмов защиты;

– оценить отрицательный эффект (накладные расходы) от применения, то есть влияние на мгновенную и общую производительность системы, с помощью микро- и макротестов с различным сочетанием активных механизмов защиты (разграничение доступа, контроль целостности, очистка оперативной памяти и остаточной информации);

– провести анализ применимости использованных теоретических результатов работы и программно-технических решений на других программных и аппаратных платформах.

Методика проведения апробации разработанных алгоритмов обеспечения безопасности и средства разграничения доступа в ОС GNU/Linux будет заключаться в проведении экспериментальных исследований с использованием темпоральной логики действий Лэмпорта (сокр. англ. TLA) для верификацией на соответствие инвариантам безопасности [14, 19, 34, 35, 39, 40, 50, 94, 95, 97, 108]. Для этого были использованы инструментальные средства TLC, позволяющие с помощью метода Model Checking в автоматическом режиме находить все возможные варианты поведения системы и проверять эти состояния на наличие нарушений требуемых свойств (так называемых инвариантов и темпоральных свойств).

В TLA+ нотации для системы, приведенной в Приложении В, определены следующие действия, соответствующие запросам к системе модели ИПСС из Таблиц 2.1 и 2.2: *CreateProcessD*, *DeleteProcessD*, *CreateUserD*, *CreateShadowD*, *DeleteSubjectD*, *ExecD*, *ReadD*, *WriteD*, *CreateD*, *DeleteD* (см. Листинг В.4). Кроме того, определено действие *SormBlockSubjectD*, которое моделирует блокировку субъекта доступа, что соответствует его принадлежности множеству S_n модели ИПСС. Каждое действие описывается с помощью пред- и постусловий выполнения действия, аналогично пред- и постусловиям запросов из Таблиц 2.1 и 2.2. Предусловиями являются

предикаты, выполнение которых необходимо для совершения действия, а постусловия определяют каким образом после выполнения действия изменяются переменные модели, то есть какое будет новое состояние системы.

Аналогично условию 1 Следствия из Базовой Теоремы ИПСС из раздела 2.1 изначально в каждой реализации системы активен только системный субъект s_0, представляющий собой ядро ОС, необходимое для дальнейшего функционирования всей системы. Начальное состояние модели описывается в Листингах B.3 и B.4, либо кратко следующим выражением:

```
Init == /\ S_active = {s_0}
        \* изначально существует только s_0 и его процесс o_0
        /\ O_func = {o_0}
        /\ O_data = {}
        /\ O_na = {o_sorm, o_2}
        \* остальные субъекты еще не активизированы
        /\ S = {s_0, s_sorm, s_2, s_3, s_4}
        /\ Q = <<q_0>>
```

Спецификация *Spec* в TLA+ нотации определяет начальное состояние системы и список возможных дальнейших действий (операций или запросов к системе), представленных в Листинге B.4 в виде предикатов:

```
Next ==
        \* Запросы к системе
        \/ CreateProcessD
        \/ ...

Spec == Init /\ [][Next]_vars /\ TemporalAssumption
```

Темпоральное предположение $TemporalAssumption == WF\_vars(Next)$ введено для исключения из модели лишних состояний и реализаций, в которых система ничего не делает – так называемых stuttering-состояний. Если в системе возможно будет выполнить какое-либо действие, оно должно будет выполниться.

Изменяемыми сущностями *vars* системы являются уже известные множества из модели ИПСС и элемент Q, представляющий собой последовательность совершенных запросов к системе:

```
vars == <<S_active, O_func, O_data, O_na, S, Q>>
```

Таких сущностей математической нотации модели ИПСС, как X и Y, а также реализаций системы, не существует напрямую в нотации TLA+, однако они возникают в процессе верификации с использованием метода Model Checking – в ходе проверки генерируются все возможные последовательности состояний и запросов системы с учетом введенных ограничений и модельных значений.

В качестве свойств безопасности системы в спецификации используются инварианты *Invariants* и темпоральные свойства *Properties*, записанные в виде предикатов в Листингах B.1 и B.4. Основное отличие инвариантов или свойств безопасности в том, что они долж-

ны выполняться во всех состояниях и для каждой реализации системы, а также могут за счет использования последовательности совершенных запросов к системе Q проверять условия, произошедшие в прошлом, например, при последнем переходе системы. Темпоральные свойства, в отличие от инвариантов, могут применять специальные темпоральные операторы TLA+ [34, 39, 94, 95, 97, 108], с помощью которых можно составлять предикаты, зависящие от времени выполнения и определенных событий в прошлом или будущем.

Первая часть инвариантов из Листинга B.4 (*TypeInv*, *ConsistencyInv*, *BlockedInv*, *OSKernelExists*) введена в TLA+ нотации для контроля в каждом возможном состоянии корректности описания операций: для контроля типов переменных модели (*TypeInv*), для контроля консистентности множеств ассоциированных и неассоциированных с субъектами объектов и уникальности сущностей системы (*ConsistencyInv*), для контроля заблокированности незарегистрированных субъектов доступа (*BlockedInv*), для контроля работоспособности системы (постоянное наличие субъекта s_0 в инварианте *OSKernelExists*).

Вторая часть инвариантов из Листинга B.4 (*SormInits*, *Correctness*, *AbsCorrectnessOpp*) введена в TLA+ нотации для проверки в каждом возможном состоянии свойств безопасности модели ИПСС из раздела 2.1: для контроля активизации подсистемы управления доступом (*SormInits* – выполнение условий 1–3 Следствия из Базовой Теоремы ИПСС), для проверки свойства корректности субъектов системы (*Correctness*), для проверки свойства абсолютной корректности субъектов в обратном смысле (*AbsCorrectnessOpp* – выполнение условий 2–4 Следствия из Базовой Теоремы ИПСС).

```
\* Свойство корректности субъектов при переходе системы
Correctness ==
LET ent == CHOOSE e \in Entities: e = SelectPrevQueryDent(Q)
    subj == CHOOSE s \in Subjects: s = SelectPrevQuerySubj(Q)
    proc == CHOOSE p \in Objects: p = SelectPrevQueryProc(Q)
    r == CHOOSE q \in QueryTypes: q = SelectPrevQueryType(Q) IN
    \* Нельзя изменять состояние ассоциированных объектов другого субъекта
    IF  r \in QueriesStateChange
    THEN
        \* "write", "delete", ...
        IF ent \in Objects /\ ent.type \in {"func", "data"}
        THEN ent.subj_assoc \subseteq {subj.sid}
        ELSE IF ent \in Subjects \* delete_subject
             THEN proc.subj_assoc \subseteq {ent.sid}
             ELSE TRUE
    ELSE IF (r \in QueriesAssocChange
            /\ ent \in Subjects) \* create_user, create_shadow
    THEN    proc.state \in {ent.sid, s_0.sid}
    ELSE
        TRUE
```

```
\* Вспомогательные предикаты для свойств корректности
EntityStateChanged(subj, proc, ent) ==
    IF ent \in Objects
    \* в прошлом был "write", "delete", ... чужих объектов
    THEN ent.state \notin {subj.sid, s_0.sid} \* = StateChanged
    ELSE IF ent \in Subjects
    \* в прошлом был "create_process", ... чужих объектов
    THEN proc.state = StateChanged
    ELSE FALSE

EntityStateChanging(ent) ==
    \* "create_user", "create_shadow", "exec", "read"
    /\ SelectPrevQueryType(Q) \in QueriesAssocChange

\* Свойство абсолютной корректности субъектов в обратном смысле
AbsCorrectnessOpp ==
    LET ent == CHOOSE e \in Entities: e = SelectPrevQueryDent(Q)
        subj == CHOOSE s \in Subjects: s = SelectPrevQuerySubj(Q)
        proc == CHOOSE p \in Objects: p = SelectPrevQueryProc(Q) IN
    \* Ассоциированным объектом не может стать измененный ранее объект
    IF  EntityStateChanging(ent)
    THEN
        IF      EntityStateChanged(subj, proc, ent)
                \* была нарушена целостность
        THEN    FALSE
        ELSE    TRUE
    ELSE TRUE
```

Свойство абсолютной корректности субъектов модели ИПСС представлено в виде темпорального свойства TLA+ нотации *AbsCorrectness*, так как в нем проверяются условия, относящиеся к будущим доступам и их нельзя проверять в рамках инвариантов TLA+ или в предикатах пред- или постусловий для запросов к системе.

```
\* Свойство абсолютной корректности субъектов
AbsCorrectness ==
    LET ent == CHOOSE e \in Entities: e = SelectPrevQueryDent(Q)
        subj == CHOOSE s \in Subjects: s = SelectPrevQuerySubj(Q)
        proc == CHOOSE p \in Objects: p = SelectPrevQueryProc(Q) IN
    \* если объект изменен
    (EntityStateChanged(subj, proc, ent))
    \* объект в будущем не станет ассоциированным с другим субъектом
        ~>  <>[] (IF    SelectPrevQueryDent(Q) = ent
                 THEN  ~EntityStateChanging(ent)
```

```
                ELSE  TRUE)
```

Также в виде темпорального свойства *OSUsabilityLiveness* выражена возможность использования системы, означающая что в любой реализации системы обязательно кроме субъектов s_0 и s_{sorm} будет порождена еще одна сущность: пользователь или системный субъект.

Во все операции системы входят проверки *SormCheckSubj(o,sc,r)* и *SormCheckPerm(s,id,r)*, которые в первую очередь проверяют с помощью макроса *SormInitialized* инициализировалась ли подсистема управления доступом (условия 2–4 Следствия из Базовой Теоремы ИПСС) и только потом выполняют проверки, необходимые для выполнения свойств корректности модели ИПСС.

```
SormInitialized ==
    /\ s_sorm \in S_active
    \* прочитан ассоциированный объект o_sorm (правила доступа)
    /\ \E o \in SelectSubjData(s_sorm):
        /\ o.oid = 1
        /\ s_sorm.sid \in o.subj_assoc

SormCheckSubj(o, sc, r) ==
    /\  IF  \neg SormInitialized
        THEN \* активизируется s_sorm
             /\ sc.type = "sorm"
        ELSE \* запрос возможен только при активизированном s_sorm
             /\ SormInitialized
    \* delete_subject
    /\  IF  r \in QueriesStateChange
        THEN \* s_0, s_sorm не могут уничтожиться после активизации
             /\ sc \notin {s_0, s_sorm}
    \* create_user, create_shadow
        ELSE IF r \in QueriesAssocChange
        THEN \* субъект не должен быть заблокирован
             /\ sc.is_blocked = FALSE
    \* change_blocked
        ELSE IF r \in QueriesSystem
        THEN \* Административные действия выполняет только s_sorm
             /\ o.subj_assoc = {s_sorm.sid}
             \* Блокировать s_0 или s_sorm нельзя
             /\ sc \notin {s_0, s_sorm}
        ELSE TRUE
    \* дополнительные проверки (идентификация/аутентификация и т.д.)

SormCheckPerm(s,o,r) ==
    \* запрос разрешен s_0 и s_sorm
    /\  IF s.sid \in {s_0.sid, s_sorm.sid}
        THEN TRUE
```

```
        \* либо должен быть активизирован s_sorm
        ELSE SormInitialized
    \* удалять или исполнять o_sorm нельзя
    /\  IF o = o_sorm
        \* иначе нарушится SormInitialized
        THEN r \notin {"exec","create","delete"}
        ELSE TRUE
    \* правила для выполнения свойств корректности модели ИПСС
    /\  IF \E obj \in SelectObjects: obj = o
        THEN IF r \in QueriesStateChange
             \* нельзя изменять/удалять чужие ассоциированные объекты
             THEN   /\ o.subj_assoc \subseteq {s.sid}
                    \* s_0 и другие не должны изменить o_sorm
                    /\ o # o_sorm
             ELSE
             IF r \in QueriesAssocChange
             THEN   \* контроль целостности: нельзя изменять ассоц.
                    \* объекты с помощью измененных объектов данных
                    o.state \in {s.sid, s_0.sid} \* # StateChanged
                    \* системные объекты – исключение
             ELSE   TRUE
        ELSE
            \* создание возможно только для личных объектов
            /\ o.subj_assoc \subseteq {s.sid}
            /\ o.state = s.sid
    \* другие дополнительные проверки (правила доступа и т.д.)
```

Верификации описанной спецификации относительно заданных инвариантов и темпоральных свойств позволяет утверждать о доказательстве в автоматическом режиме следующей теоремы в TLA+ нотации, которая подтверждает выполнение требований безопасности для всех возможных состояний и реализаций системы:

```
\* Теорема, учитывающая инварианты и свойства: доказывается при верификации
THEOREM Spec => /\ []TypeInv
                /\ []ConsistencyInv
                /\ []BlockedInv
                /\ []OSKernelExists
                /\ []SormInits
                /\ []Correctness
                /\ []AbsCorrectnessOpp
                /\ OSUsabilityLiveness
                /\ AbsCorrectness
```

При этом инструментальное средство верификации TLC моделирует всевозможные состояния и реализации модели с учетом некоторых ограничений – модельных значений, заданных

для того, чтобы процесс верификации завершился и при этом несущественно ограничивалась сама моделируемая система. Так в TLA+ нотации кроме системного субъекта s_0 и подсистемы управления доступом s_{sorm} могут одновременно существовать еще два субъекта-пользователя и один системный субъект. В состоянии, когда все перечисленные субъекты активны у них может существовать по одному ассоциированному объекту-процессу и дополнительно может быть создано несколько объектов, ассоциированных или неассоциированных с субъектами доступа. Увеличение приведенных модельных значений количества возможных субъектов или объектов не повлияет на моделируемую систему, но значительно увеличит время верификации, количество возможных состояний и реализаций системы.

Верификация разработанной модели проводилась с помощью инструментального средства TLC2 v2.15 на СВТ с Intel Core i5-9400 (3.80 ГГц) и объемом оперативной памяти 16 ГБ в 64-разрядной ОС Linux с ядром v5.4.38. Время, затраченное на верификацию с использованием 6 отдельных потоков, составило 20 часов 25 минут или 200 часов в зависимости от выставленных опций верификации – проверки модели с итеративным углублением, начиная с заданной начальной глубины (опция dfid 6 или 7). Общее количество проанализированных состояний – 124299849 или 671088640 соответственно.

Проведенная верификация позволила:

– выявить и исправить неточности в математической нотации модели ИПСС;

– подтвердить идентичность свойств абсолютной корректности субъектов в прямом и обратном смысле из Определений 4 и 8 для выполнения условий Базовой теоремы ИПСС и Следствия из раздела 2.1;

– подтвердить необходимость использования предложенных в разделах 2.2–2.4 способов и алгоритмов обеспечения безопасности для исключения возможности нарушения действующей в системе политики управления доступом;

– описать конкретные правила доступа, заданные в виде предикатов *SormCheckSubj(o,sc,r)* и *SormCheckPerm(s,id,r)*, которые дополнительно к предложенным алгоритмам защиты позволяют выполнять в системе требуемые свойства корректности субъектов доступа и все условия Следствия из Базовой теоремы ИПСС.

4.2. Анализ результатов применения разработанного средства разграничения доступа в ОС GNU/Linux

4.2.1. Подтверждение возможности встраивания разработанной подсистемы управления доступом в ОС GNU/Linux

Под встраиванием понимается процесс внедрения дополнительных внешних по отношению к системе программных элементов, осуществляемый таким образом, чтобы, с одной стороны, сохранялось функционирование самой системы, а с другой, – расширялись или изменялись ее функциональные возможности [48].

Для механизмов защиты в ОС GNU/Linux существует много различных способов встраивания: в пользовательском режиме с помощью перехвата обращений к стандартной библиотеке (на-

пример, с помощью возможностей LD_PRELOAD); на уровне ядра ОС с помощью модификации соответствующих функций (обработчиков системных вызовов); на уровне ядра ОС с помощью модификации таблицы системных вызовов; на уровне ядра ОС с использованием возможностей Linux (технологии LSM). Рассмотрим подробнее данные подходы, выделим их положительные и отрицательные стороны, а также особенности, которые могут повлиять на применимость в составе подсистемы управления доступом.

Встраивание функций защиты данных на уровне пользовательских приложений не может гарантировать их активизации при осуществлении любого типа доступа (что противоречит требованиям из разделов 2.4 и 3.1). Также при таком встраивании защитные механизмы относительно легко можно отключить, чему достаточно сложно противодействовать со стороны встраиваемого СЗИ НСД. Достоинством данного способа является простота реализации и независимость от используемой системы (с точностью до используемой версии стандартной библиотеки GNU/Linux).

Базовым способом встраивания является изменение некоторых функций ядра Linux, однако он архитектурно зависим вследствие необходимости определения размера заменяемого пролога функций с помощью дизассемблирования [48], а также нарушает целостность компонент системы. Также при модификации функций ядра необходимо обращать особое внимание на корректность встраивания в многопроцессорные системы, в результате изменений не должна нарушаться когерентность [48]. Тем не менее данный способ позволяет осуществлять динамическое встраивание в любые существующие внутренние функции, однако наличие определенных функций может зависеть от используемой версии ядра Linux.

Встраивание на уровне таблицы системных вызовов часто также может зависеть от используемой архитектуры СВТ (см. разд. 3.3), за счет того, что модифицировать необходимо записи в памяти ядра Linux, доступной только на чтение. Однако в данном случае возможно встраивать защитные механизмы СЗИ НСД во все POSIX/SUS-совместимые ОС, а не только в ОС GNU/Linux. Также необходимо учитывать, что, например, в 64-разрядных версиях GNU/Linux в действительности существует две таблицы системных вызовов: sys_call_table и ia32_sys_call_table (для совместимости). В соответствии с этим замену адресов обработчиков системных вызовов необходимо проводить для соответствующих ячеек в обеих таблицах. С помощью данного способа встраивания возможно внедрить механизмы защиты подсистемы управления доступом непосредственно до вызова соответствующих стандартных механизмов ОС GNU/Linux (DAC, Capabilities и так далее, см. разд. 1.1.2).

Возможность использования технологии LSM в ядре Linux зависит от того, был ли включен параметр CONFIG_SECURITY в конфигурации ядра перед сборкой. При отсутствии данного параметра при сборке ядра Linux соответствующие LSM-обработчики не будут добавлены в обработчики системных вызовов и их невозможно будет использовать для встраивания защитных механизмов подсистемы управления доступом. Также при использовании LSM на современных дистрибутивах GNU/Linux возникают следующие сложности в реализации [48], накладывающие ограничение на возможность использования этой технологии во встраиваемой подсистеме управления доступом:

– с версии ядра 2.6.24 указатель на структуру security_ops, используемую для регистрации LSM-обработчиков, не экспортируется как символ ядра Linux;

– с версии 2.6.35 функция register_security(), реализующая регистрацию LSM-обработчиков, не экспортируется ядром ОС;

– с версии ядра 4.1 вместо security_ops используются списки LSM-обработчиков (security_hook_list), также не экспортируемые ядром ОС.

В соответствии с этим в современных ядрах Linux не может быть зарегистрирована внешняя подсистема управления доступом, использующая технологию LSM. Тем не менее для использования существующих LSM-обработчиков (в случае если ядро собрано с CONFIG_SECURITY) достаточно найти в памяти ядра Linux расположение структуры security_ops и функции register_security() (или security_hook_heads для ядра 4.1 и новее) и беспрепятственно использовать их неэкспортируемые в ядре ОС указатели для регистрации собственных функций-обработчиков.

Важным отличием LSM от других способов встраивания является то, что LSM-обработчики вызываются после стандартных механизмов доступа ОС GNU/Linux, а при регистрации этих обработчиков не существует ограничения по используемой аппаратной платформе (не требуется модификации защищенной от записи памяти ядра), сложностей встраивания на многопроцессорных системах или других особенностей – архитектура LSM гарантирует корректность встраивания. Таким образом, использование технологии LSM для встраивания в ядро ОС позволяет внедрять механизмы защиты подсистемы управления доступом кросс-платформенно, однако только в отношении ОС на базе ядра Linux и только в относительно современных версиях ядра. При этом LSM-обработчики обеспечивают достаточное покрытие существующих системных вызовов, необходимых для подсистемы управление доступом (см. раздел 3.3 и [48]).

Кроме особенностей способов встраивания в GNU/Linux для модулей ядра (LKM), в виде которых целесообразней всего реализовывать подсистему управления доступом в ОС (см. раздел 2.4), также существуют некоторые ограничения. Начиная с версии ядра Linux 3.7 существует возможность ограничить использование модулей ядра LKM (опции CONFIG_MODULE_SIG и CONFIG_MODULE_SIG_FORCE в конфигурации ядра). При использовании строгого режима запрещается загрузка всех модулей, у которых нет или некорректна электронная подпись, в разрешительном режиме запрещена загрузка только модулей с некорректной подписью. Эту особенность необходимо учитывать для конкретных дистрибутивов GNU/Linux.

Кроме того, для модулей ядра, используемых в Linux, достаточно остро стоит вопрос лицензирования и статуса исходных кодов. С одной стороны для модулей LKM требуется соответствие свободной лицензии GNU GPL (а исходные коды должны быть открытыми), так как модуль загружается в адресное пространство ядра Linux, которое распространяется по этой лицензии. С другой стороны, LKM в обычном смысле могут не использовать в своей работе исходные коды с лицензией на открытое ПО (по отношению к ядру – GPL v2 и другие, GPL/BSD, MIT/GPL, MPL/GPL). Однако некоторая функциональность ядра Linux может требовать от LKM использования только лицензии GPL.

В данном разделе проанализированы особенности различных способов встраивания подсистемы управления доступом в GNU/Linux, выделены их положительные и отрицательные характеристики. Для внедрения подсистемы управления доступом можно использовать любой из описанных в разделе 2.4 способов динамического встраивания в ядро Linux, однако выбор подходящего способа полностью зависит от целей и задач встраиваемой подсистемы управления доступом – в случае если необходимо обеспечивать совместимость между POSIX/SUS-системами (или не существует требуемого LSM-обработчика) необходимо встраиваться на уровне системных вызовов, для хорошей переносимости между различными дистрибутивами GNU/Linux можно использовать LSM. При этом часть описанных способов встраивания необходимо адаптировать для разных аппаратных платформ (x86, x86_64, arm, sparc, power и так далее), а защитные механизмы в зависимости от способа встраивания могут работать как до стандартных механизмов ОС GNU/Linux, так и после них.

Необходимо отметить, что наиболее простым и безопасным способом встраивания является использование технологии LSM (при всех ее ограничениях по использованию), при применении которой не существует необходимости уделять внимание дополнительным особенностям, не связанным с функциями разграничения доступа напрямую и свойственным для остальных способов встраивания. В случае, если ограничения по использованию LSM не влияют на выполнение требований, предъявляемых к подсистеме управления доступом – необходимо использовать именно этот вариант встраивания.

4.2.2. Подтверждение корректности взаимодействия разработанной подсистемы управления доступом с другими средствами защиты информации

Как было показано в разделах 1.1.2, 1.4 и 2.4 в GNU/Linux встроенные механизмы защиты ОС изначально представляют собой сочетание нескольких формальных моделей безопасности (дискреционная политика управления доступом с помощью атрибутов файловых объектов, ролевая политика управления доступом с помощью групп пользователей и так далее). При этом, как правило, встраивание подсистемы управления доступом не отменяет, но дополняет встроенные средства защиты ОС. Таким образом, для ОС GNU/Linux актуален вопрос корректности и непротиворечивого сочетания или взаимодействия правил всех используемых средств защиты информации от НСД (и реализованных ими формальных моделей безопасности КС).

4.2.2.1. Корректность загрузки и выгрузки функций защиты средств разграничения доступа

При построении подсистемы управления доступом в виде загружаемого модуля ядра Linux дополнительно необходимо особое внимание обратить на безопасность загрузки и выгрузки (в случае удаления) с сохранением работоспособности системы, а также на корректность ее функционирования совместно с другими СЗИ НСД. Рассмотрим показательный пример [22, 36], на основе которого будет продемонстрирована опасность одновременного использования нескольких одинаковых по функциональности модулей, вносящих изменения в некоторые функции ядра Linux – предположим, в таблицу системных вызовов (см. разделы 2.4, 3.3 и 4.2.1).

Допустим, есть 2 загружаемых модуля ядра (*module_A* и *module_B*) и каждый из них изменяет адрес одного и того же системного вызова «open» на адрес своей функции (*open_A* и *open_B* соответственно). Условимся, что первым загружается *module_A* и заменяет адрес в *sys_call_table[__NR_open]* на адрес своей функции *open_A*. Затем загружает *module_B* и заменяет адрес *open_A* адресом своей функции *open_B* (сам *module_B* при этом подозревает, что подменил оригинальный системный вызов).

Теперь если *module_B* выгрузится первым (например, при окончании работы системы, непосредственно перед перезагрузкой или выключением, либо в других обстоятельствах) – в системе в дальнейшем не произойдет ошибок, при выгрузке *module_A* в таблице системных вызовов будет восстановлен оригинальный вызов «open» и работоспособность системы не нарушится.

Однако если же первым выгрузится *module_A*, вначале будет восстановлен оригинальный системный вызов «open», а в случае дальнейшей выгрузки *module_B* системный вызов «open» будет заменен на адрес *open_A* (кода которой уже не существует в памяти ядра Linux). На первый взгляд, для исправления приведенной в примере ситуации можно в каждом модуле (*module_A*, *module_B* и всех других) заменять адреса обработчиков обратно только в случае, если адрес в таблице системных вызовов совпадает с адресом функции самого модуля (*open_A* или *open_B*), но в таком случае при выгрузке в качестве первого модуля *module_A* – адрес оригинального системного вызова «open» не будет возвращен в *sys_call_table*, таким образом данный подход также не исправляет ситуацию.

Кроме того, так как с выгрузкой модуля выгружается вся используемая в нем функциональность (код), также может возникнуть ситуация, когда некоторый поток, осуществляющий системный вызов, перешел в состояние сна, а в этот момент будет инициирована выгрузка подсистемы управления доступом с кодом обработчика этого вызова. В соответствии с этим необходимо корректным образом выполнять загрузку и дальнейшую выгрузку модулей ядра (иногда для этого требуется «невыгрузка» некоторых его функций), в случае если такая возможность вообще предусматривается (конкретный модуль можно удалить из списка используемых модулей ядра Linux, после чего выгрузить его до окончания работы ОС GNU/Linux будет невозможно).

Отметим, что применение способа встраивания подсистемы управления доступом с использованием LSM или предложенного в разделе 3.2 модуля контроля механизма защиты страниц памяти в ядре Linux (в отношении памяти СЗИ НСД и его точек встраивания) исключает приведенную ситуацию. Также заметим, что до недавнего времени в LSM исключалась возможность одновременного использования нескольких модулей ядра для регистрации одинаковых обработчиков – при регистрации LSM-модуля текущие используемые обработчики заменялись новыми. Однако последние изменения в ядре Linux позволяют корректным образом реализовать работу нескольких «конкурентных» модулей безопасности, регистрирующих свои LSM-обработчики (англ. LSM stacking), при этом разработчикам таких модулей более не требуется беспокоиться о каких-либо особенностях встраивания.

4.2.2.2. Корректность сочетания нескольких средств разграничения доступа

Организовав встраивание подсистемы управления доступом в соответствии со способами, алгоритмами и техническими решениями из разделов 2.4 и 3.3 становится возможным реализовать сочетание нескольких используемых в ОС GNU/Linux СЗИ НСД (одного за другим по порядку, зависящему от способа встраивания). Любые встроенные механизмы защиты в ОС GNU/Linux при этом будут отрабатывать непосредственно до или после встраиваемой подсистемы управления доступом.

Для каждого СЗИ НСД порядок вызова его механизмов защиты в общем случае не важен – ведь эти механизмы посредством использования описанных способов и алгоритмов встраивания в любом случае будут гарантированно вызываться (при использовании LSM это вообще обеспечивается архитектурно), в случае если доступ разрешен по правилам средств разграничения доступа, чьи механизмы отрабатывают раньше (в случае же запрета на доступ управление не будет передано, но доступ будет запрещен, а не разрешен). Поэтому в такой последовательности работы механизмов защиты (реализующих формальные модели безопасности КС) доступ в результате будет разрешен только в единственном случае – если он разрешен абсолютно во всех используемых средствах разграничения доступа (и правилах формальных моделей).

Таким образом, используя теории алгоритмов и автоматов систему с внедрением средств разграничения доступа при использовании предложенных способов и алгоритма встраивания (результирующую подсистему управления доступом ОС) можно рассматривать в виде соединения нескольких автоматов [13, 93], где каждое средство защиты представляется автоматом, принимающим на вход запрос о доступе (атрибуты объекта и субъекта, тип доступа) и либо запрещающий, либо разрешающий этот доступ. Соответственно сочетание нескольких СЗИ НСД будет представляться в виде последовательного соединения автоматов.

При этом если рассматривать существующие (встроенные) средства защиты от НСД в Linux как разные автоматы, то они могут влиять на свои состояния и функции переходов (например, права владения объектами и capabilities или права доступа в файловых атрибутах), при этом не представляют собой автоматы с обратной связью или параллельное соединение автоматов (в том числе с общим входом). Поэтому далее все встроенные средства разграничения доступа в ОС будем представлять в виде одного автомата (со сложными функциями переходов), а все другие автоматы – встраиваемые в ОС СЗИ НСД (например, SELinux, AppArmor, Tomoyo, SecretNet, Аккорд-X и другие), которые не могут оказывать влияние на состояния или функции переходов других автоматов.

Соединение двух (или более) автоматов при любом описанном виде встраивания является односторонним блокировочным [5, 93], то есть таким ограничивающим соединением, в котором один из автоматов может перейти в некоторое состояние, при котором все последующие автоматы будут находиться в, так называемом, заблокированном состоянии, их вход будет подавляться и без изменений транслироваться на выход, не влияя при этом на внутренние состояния. Например, при встраивании средства разграничения доступа (A_1) до встроенных механизмов защиты ОС (A_2) в случае поступления на вход A_1 доступа субъекта к объекту, который запрещен в соответствии с правилами A_1, предикат блокировки $\chi_2 = 0$, то есть A_2 будет находиться в забло-

кированном состоянии и вход A_2 будет транслирован без изменений на выход (то есть доступ будет запрещен вне зависимости от правил политики управления доступом A_2). При поступлении на вход A_1 доступа, осуществление которого не противоречит правилам A_1, предикат блокировки $\chi_2 = 1$ и A_2 будет находиться в разблокированном состоянии, то есть доступ будет направлен на вход A_2 (будет проверен на соответствие правилам A_2). Аналогичным образом соединение автоматов будет работать и для случая встраивания средства разграничения доступа после встроенных механизмов защиты ОС (с точностью до перестановки обозначений A_1 и A_2), а также в случае большего количества используемых СЗИ НСД ($n > 2$).

В итоге результаты работы приведенной совокупности средств разграничения доступа вне зависимости от последовательности подключения автоматов A_1 и A_2 будут идентичны: доступ будет разрешен только в случае одновременного соответствия правилам A_1 и A_2 и запрещен – в случае нарушения правил хотя бы одного средства разграничения доступа. Важным условием в таком представлении КС является гарантия активизации A_1 и A_2, которое включает требование по невозможности запуска СВТ без включения подсистемы управления доступом (см. разд. 3.1). Однако посредством использования рассмотренного алгоритма встраивания обеспечивается невлияние любых стандартных и встраиваемых механизмов разграничения доступа друг на друга.

4.2.3. Подтверждение соответствия реализации разработанных алгоритмов обеспечения безопасности управления доступом ОС GNU/Linux предложенной модели безопасности

Используемые в различных средствах разграничения доступа формальные модели безопасности должны формировать подсистему управления доступом, обладающую свойством полноты (с точки зрения полноты механизмов контроля для всех возможных типов доступа субъектов к объектам). Аналогичное требование постулируется, например, в модели полного перекрытия [70], в которой существует требование наличия, по крайней мере, одного средства для обеспечения безопасности на каждом возможном пути воздействия некоторого абстрактного нарушителя на систему.

Разработанная в данной работе подсистема управления доступом в GNU/Linux позволяет реализовать необходимые механизмы для организации защиты информации (объектов ОС) от угроз конфиденциальности и целостности (от угроз доступности только в части удаления объектов). В соответствии с этим, последующие рассуждения будут приведены в контексте противодействия только этим двум видам угроз.

Каждая отдельно взятая ОС GNU/Linux уникальна: ядро ОС собрано с уникальными настройками и параметрами конфигурации, в различных версиях ядра Linux может быть разный набор системных вызовов или их обработчиков, в качестве стандартных приложений могут быть установлены различные варианты прикладных программ и так далее. С одной стороны, для таких уникальных дистрибутивов GNU/Linux (или классов ОС) невозможно применять типовые СЗИ НСД. С другой стороны, все дистрибутивы в той или иной степени стандартизованы (на основании стандартов типа POSIX, SUS [91] и LSB [100]) и, предусмотрев, необходимые раз-

личия для более ранних версий дистрибутивов GNU/Linux, можно обеспечить переносимость и полноту защитных механизмов.

Предложенная в разделе 3.3 подсистема управления доступом представляет собой встраиваемое или «навесное» СЗИ НСД, универсальное в части учета различий между версиями ядра Linux 2.6 – 6.∗ (посредством учета описанных в соответствующих подразделах отличий во внедряемых обработчиках для разных версий). Обоснуем полноту используемых в ней защитных механизмов на основе [93].

В большинстве формальных моделей безопасности фиксируется набор объектов и субъектов доступа. Однако в современных КС (ОС) данное ограничение выглядит искусственным и сложнореализуемо – в процессе работы постоянно появляются, как минимум, новые объекты доступа (в том числе временные файлы или объекты в памяти, не имеющие файлового объекта в рамках ФС). Субъекты доступа в целях безопасности ограничивать необходимо, при этом в ходе работы подсистемы управления доступом посредством контроля порождения субъектов в КС (см. разд. 2.2.1 и 3.3) становится невозможным несанкционированное возникновение новых субъектов во время работы системы. Для ограничения множества объектов можно использовать «мягкий» режим работы подсистемы управления доступом, с помощью которого возможно настроить права в соответствии с принципом наименьших привилегий для всех субъектов системы. Недостатком применения данного принципа является необходимость периодической корректировки прав разграничения доступа. Для минимизации вносимых изменений можно адаптировать подсистему управления доступом, то есть сделать ее более интеллектуальной или даже «самообучаемой» (адаптивной), способной фиксировать возникновение новых объектов в ОС (например, загруженных по сети и сохраненных в ФС), пополнять списки объектов доступа [77] и права доступа.

Полнота механизмов защиты в части контроля всех доступных каналов (типов) доступа субъектов к объектам в ОС GNU/Linux напрямую связана с используемыми в подсистеме управления доступом механизмами встраивания (см. разд. 2.4, 3.3 и 4.2.1). Для рассматриваемой в данной работе подсистемы управления доступом полнота механизмов защиты основывается на полноте обработчиков LSM и системных вызовов [48]. Как было выявлено в разделе 3.3 контролировать необходимо любые доступы субъектов к объектам, в том числе *read*, *write*, *execute*, *create/mknode/mkdir*, *unlink/rmdir*, *rename* и *link/symlink*, доступы-посредники *brk/mmap*, *mount/umount*, *pivot_root* и *chroot*, а также доступы, влияющие на изменение дескрипторов безопасности процессов – *setuid* и *fork*. При этом полнота приведенного списка типов доступа основывается на полноте возможных доступом в соответствии со стандартами POSIX, SUS и LSB. Однако для различных версий ядра Linux набор функций-обработчиков, а также используемые параметры могут отличаться, поэтому в подсистеме управления доступом необходимо обеспечить совместимость и однозначность всех реализованных функций, а также унификацию для любого поддерживаемого набора используемых обработчиков. При этом количество доступных для настройки атрибутов доступа в правилах СЗИ НСД непринципиально, так как большинство операций с объектами можно описать с помощью базовых атрибутов *read*, *write*, *execute*.

Также рассмотренная в разделе 3.3 подсистема управления доступом, кроме вопросов конфиденциальности, рассматривает вопросы целостности данных. Вместо совмещения моделей

конфиденциальности с моделями целостности [53, 76] используется дополнительный контроль целостности данных при реализации всех попыток доступа к подконтрольным объектам, то есть целостность объектов контролируется, а внесение в них несанкционированных изменений фиксируется в журнале работы подсистемы управления доступом.

Кроме вопроса полноты механизмов защиты разработанной подсистемы управления доступом, необходимо исследовать ожидаемый эффект от ее применения, особенно в части достижимости и поддержания изолированной программной среды субъектов в ОС [29, 30, 93, 96].

В отличие от СО-модели ИПС (см. разделы 1.3 и 2.1) для подсистемы управления доступом, рассмотренной в разделе 3.3, существует возможность контролируемой загрузки ядра защиты и контроля целостности всех связанных с ним объектов до загрузки системы (СВТ с ОС GNU/Linux). Посредством реализации предложенных в разделах 2.3 и 2.4 способов и алгоритмов обеспечивается невозможность загрузки системы без включения соответствующих механизмов защиты подсистемы разграничения доступа, а сама система становится шире СО-модели. Процесс «ступенчатой» загрузки ОС с внедренной подсистемой управления доступом представляет собой следующие этапы, представленные на Рисунке 4.1, в сравнении с Рисунком 1.3 из раздела 1.3:

1. Начало загрузки системы ($t = 0$).
2. Контроль целостности или обеспечение неизменности компонент системы, ядра защиты и связанных с ним объектов (интервал $[0, t_0]$).
3. Загрузка ядра защиты, начало действия процедур контроля целостности, контроля порождения субъектов и разграничения доступа (начало организации ИПСС – t_0, в данный момент в системе существует всего два реальных субъекта – системный процесс s_0 и ядро защиты s_{sorm} [26]).
4. Дальнейшая активизация компонент системы, порождение субъектов системных процессов (интервал $[t_0, t_3]$).
5. Стационарная фаза функционирования системы (до *login()*), порождение субъектов-пользователей (после *login()*, интервал от t_1 и далее).

При этом достижение ИПСС в системе с внедренной подсистемой управления доступом обеспечивается тем, что:

– ядро защиты гарантированно запускается (порождается), а связанные с ним объекты контролируются на целостность аппаратно до активации (с помощью АМДЗ), либо их неизменность обеспечивается технологически;

– ядро защиты работает с момента ранней загрузки системы и сразу после инициализации реализует функции разграничения доступа и контроля порождения субъектов в соответствии с разделом 2.1, обеспечивая абсолютную корректность новых порождаемых субъектов относительно существующих;

– после активизации ядра защиты в системе существует только два субъекта – само ядро защиты и системный процесс s_0, которые абсолютно корректны относительно друг друга в обратном смысле.

Рисунок 4.1 – Схематичное представление этапов функционирования ОС GNU/Linux с использованием аппаратного контроля целостности (технологической неизменности), контроля порождения субъектов и разграничения доступа средствами «Аккорд-X»

Перед описанным процессом загрузки необходимо выделить множество разрешенных субъектов доступа в системе с использованием «мягкого» режима работы (режима обучения) подсистемы управления доступом. Данный режим работы позволяет настроить «предопределенное выполнение начальной фазы функционирования КС» [13] лишь предполагаемое в формальной СО-модели. В ходе экспериментальных исследований дальнейшей работы подсистемы управления доступом в различных ОС GNU/Linux было показано [96], что в результате использования предложенных в работе способов и алгоритмов защиты в системе невозможно порождение несанкционированных субъектов доступа – в процессе загрузки и дальнейшей работы в ОС могут существовать только разрешенные субъекты, например, представленные в виде графа на Рисунке 4.2.

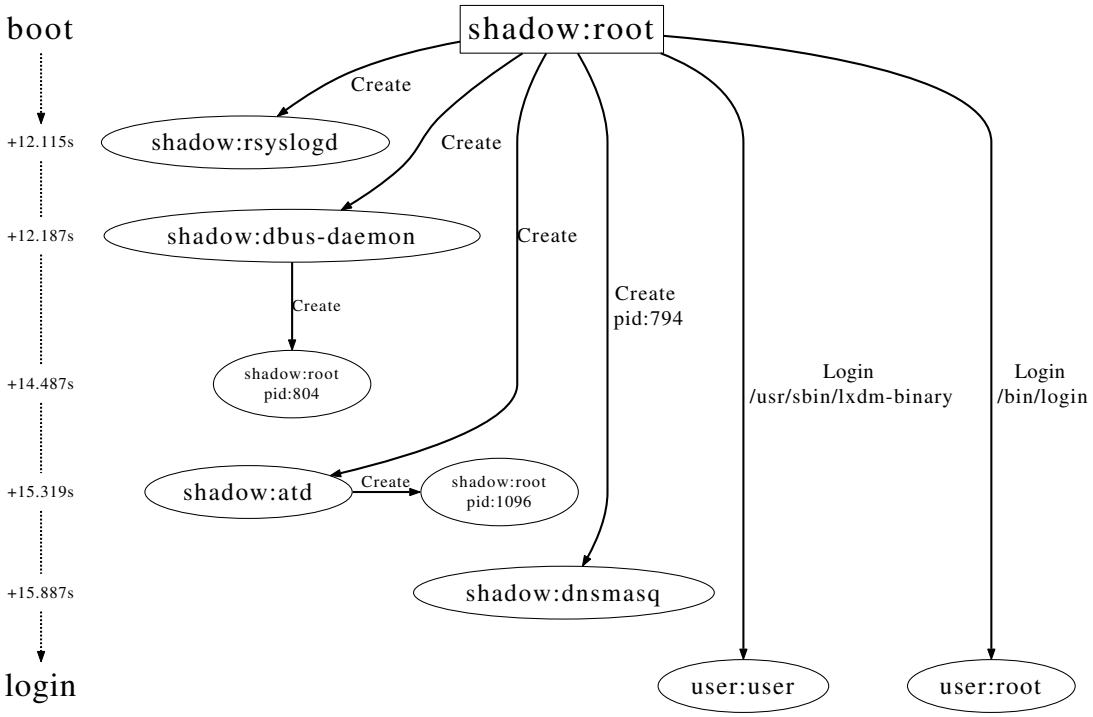

Рисунок 4.2 – Граф контролируемого порождения субъектов доступа в процессе загрузки Linux

Достижение и поддержание изолированной программной среды субъектов были также подтверждены с помощью моделирования системы, в которой реализованы предложенные способы и алгоритмы обеспечения безопасности и модель ИПСС из разделов 2.1–2.4, на языке темпоральной логики действий Лэмпорта с последующей верификацией на соответствие инвариантам безопасности (см. раздел 4.1).

Как было отмечено в разделе 2.1, ассоциированные с ядром защиты объекты (особенно данные о разрешенных субъектах доступа) играют ключевую роль в проектировании ИПСС, так как при их изменении возможно «размыкание» программной среды. В этих условиях возникает вопрос возможности безопасного администрирования подсистемы управления доступом, для решения которого предлагается использовать политику абсолютного разделения административных и пользовательских полномочий [13] (см. также раздел 3.3). Для повышения безопасности системы (в данном случае ОС GNU/Linux) следует искать компромисс между двумя противоположностями – необходимостью обеспечить защиту данных и предоставлением возможности администрирования доверенным пользователям. Однако на основании проведенных исследований полноты и эффективности разработанной подсистемы управления доступом при введении описанных в разделе 2.1 ограничений используемая политика абсолютного разделения административных и пользовательских полномочий позволяет обезопасить процесс администрирования.

Подводя итог, необходимо отметить, что ОС GNU/Linux при соответствующей настройке правил контроля доступа и целостности «Аккорд-X» (см. условия Базовой теоремы ИПСС и ее Следствия, выводы к разделу 2.1, а также предикаты *SormCheckSubj(o,sc,r)* и *SormCheckPerm(s,id,r)* из раздела 4.1) представляет собой ИПСС с обеспечением и контролем загрузки ядра защиты системы и дополнительной реализацией дискреционной и мандатной политик управления доступом, очистки оперативной памяти при ее освобождении или перераспределении [37], очистки остаточной информации при удалении объектов, динамического и статического контроля целостности [25] и других функций защиты. Кроме того в разработанной подсистеме управления доступом учтены: возможность безопасного администрирования прав разграничения доступа и порождения субъектов, возможность настройки прав доступа в соответствии с принципом наименьших привилегий, возможности учета новых объектов доступа, а также возможность обеспечения целостности данных. Таким образом, рассматриваемая ОС с подсистемой управления доступом, с одной стороны, не противоречит формальной модели безопасности ИПСС (исключается возможность возникновения запрещенных информационных потоков) и не конфликтует с существующими СЗИ НСД (функционирует независимо, дополняя их, см. раздел 4.2), с другой стороны, позволяет решить все открытые для формальных моделей вопросы, рассмотренные в разделе 1.3, и при этом обеспечивает защиту как данных (объектов доступа), так и пользовательской среды, в которой непосредственно происходит обработка информации в GNU/Linux.

В ходе экспериментальных исследований была подтверждена достижимость и поддержание ИПСС, а также исследована изолированность пользовательских сред различных субъектов до-

ступа ОС в процессе работы подсистемы управления доступом [29, 30, 96]. На основе журнала событий безопасности подсистемы управления доступом были построены графы санкционированных обращений субъектов к объектам (пример приведен на Рисунке 4.3). В результате было выявлено, что все вершины в пересечении подграфов являются общедоступными объектами, для которых ограничивается доступ и, главное, контролируется целостность. Таким образом, в разработанной подсистеме управления доступом обеспечивается полная изолировать взаимовлияния сессий субъектов доступа через общедоступные объекты, что и является целью создания ИПСС.

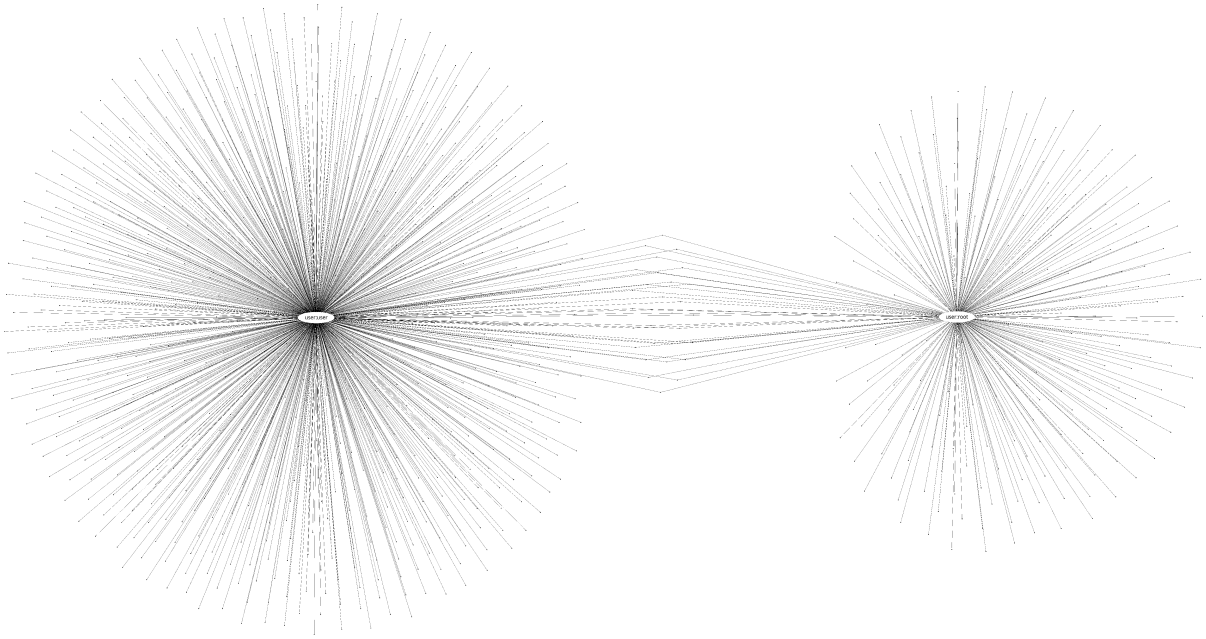

Рисунок 4.3 – Граф доступа нескольких субъектов к объектам в реальной ОС GNU/Linux, построенный на основе журнала событий безопасности «Аккорд-X»

4.2.4. Исследование влияния разработанной подсистемы управления доступом на производительность ОС GNU/Linux

Кроме приведенного в разделе 4.2.3 подтверждения эффективности использования применяемых в подсистеме управления способов и алгоритмов обеспечения безопасности, необходимо провести экспериментальные исследования по оценке возможного отрицательного эффекта. Такая необходимость связана в первую очередь с тем, что подсистема управления доступом может стать помехой для корректной работы ОС, различного прикладного ПО и сервисов, а, в соответствии с этим, требуется подтвердить отсутствие существенного влияния реализованных защитных механизмов на производительность системы [29, 30].

Для исследования оказываемого влияния подсистемой управления доступом на производительность системы проведен ряд микро- и макротестов (стресс- или нагрузочных тестов). Тесты проводились на СВТ с Intel Core i5-3570K (3.40 ГГц) и объемом оперативной памяти 8 ГБ в 32-разрядной ОС Ubuntu 12.04.5 LTS с ядром Linux версии 3.13.0-32-generic в следующих условиях:

– ОС со стандартными настройками сразу после установки (здесь и далее *«os-clean»*);

– ОС с внедренной подсистемой управления доступом «Аккорд-X» (*«os-acx»*);

– аналогично п.2, но с активированным модулем очистки оперативной памяти (*«os-memclean»*);

– аналогично п.2, но с активированным модулем очистки остаточной информации файлов при их удалении (*«os-fileclean»*);

– аналогично п.2, но с активированными модулями очистки оперативной памяти и остаточной информации файлов при их удалении (*«os-memfileclean»*).

Целью проведения микротестов производительности является оценка и сравнение результатов работы низкоуровневых операций ядра Linux, таких как: выполнение системных вызовов и переключение контекста процессов (англ. context switching), доступ к файлам и памяти, а также других. Для выполнения таких микротестов существует специальный инструмент *lmbench*[1], который подтвердил свою эффективность и широко используется для проведения регрессионного тестирования различных изменений ядра Linux [99].

Результаты микротестов производительности приведены в таблице 4.1.

Таблица 4.1 – Результаты микротестов производительности ОС GNU/Linux в различных условиях

Тип теста	**os-clean**	**os-acx**	**os-memclean**	**os-fileclean**	**os-memfileclean**
Базовые операции с процессами, время в мксек.[2]:					
null call	0.05	0.05	0.05	0.05	0.05
null I/O	0.11	0.11	0.11	0.11	0.11
open/read/close	2.62	3.07	3.11	3.11	3.11
fork	127	143	148	151	151
exec	550	633	645	635	645
sh	1295	1515	1523	1526	1526
Переключение контекста процессов, время в мксек.:					
2пр./0Кб	3.07	3.42	3.55	3.50	3.71
2пр./16Кб	4.76[3]	4.62	4.38	4.80	4.59
2пр./64Кб	7.78	7.76[3]	7.81	7.85	7.79
8пр./16Кб	6.79	6.79	6.78	6.78	6.52[3]
8пр./64Кб	11.0	11.0	11.1	10.9	11.2
16пр./16Кб	7.07[3]	5.54	6.90	5.74	6.09
16пр./64Кб	11.4	11.6	11.6	11.7	11.7
Задержки при локальном взаимодействии, время в мксек.:					
pipe	8.246	8.180	8.204	8.182	8.477

[1] В работе использовалась версия lmbench v3.0 – http://www.bitmover.com/lmbench/.

[2] Здесь и далее меньшее значение соответствует лучшему результату.

[3] Возможная экспериментальная ошибка.

Продолжение таблицы 4.1

Тип теста	**os-clean**	**os-acx**	**os-memclean**	**os-fileclean**	**os-memfileclean**
сокеты	6.08[3]	5.28	5.44	6.02[3]	5.56
UDP	12.9[3]	12.2	12.6	12.3	12.3
TCP/IP	14.2	14.3	14.2	14.1	13.8
Задержки в файловой системе и памяти (латентность), время в мксек.:					
создание файла 0Кб	9.5208	11.0	11.0	11.2	11.6
удаление файла 0Кб	6.0851	6.6160	7.0864	9.1659	9.3089
создание файла 10Кб	16.6	18.3	18.3	18.3	18.3
удаление файла 10Кб	7.8884	8.8977	8.9678	11.3	11.6
выделение 100 дескрипторов	0.963	0.969	0.965	0.964	0.964
mmap, munmap	5485	5963.0	5980.0	6117.0	6163.0
Page Fault	0.8069	0.8134	0.8046	0.8044	0.8041

Интерпретируя полученные результаты микротестов, можно сделать следующие выводы:

– множество тестов не показали снижения производительности системы с внедренной подсистемой управления доступом (например, все операции с числами, локальное взаимодействие и память – кэш, основная память, случайный доступ; результаты некоторых тестов в таблице не приводятся), а в ряде случаев накладные расходы были в пределах незначительной погрешности (от 1% и до 5%);

– несколько тестов показали лучшие результаты для системы с внедренной подсистемой управления доступом, что можно списать на экспериментальные ошибки (возможно, связанные с аномалиями и коллизиями кэширования);

– наибольшие накладные расходы при использовании разработанной подсистемы управления доступом были зафиксированы:

– в базовых операциях с процессами (open/read/close, fork, exec и sh), при которых потеря производительности составила в худшем случае от 8 до 16%, что вполне ожидаемо вследствие работы модуля контроля порождения субъектов доступа;

– в переключении контекста процессов – для 2 процессов, 0Кб до 17%, однако в последующих тестах разница во времени постепенно снижается и исчезает (2 процесса, 16Кб – 5%; 2 процесса, 64Кб и последующие – до ~0%), что свидетельствует о фиксированной задерж-

ке вследствие работы внедренных механизмов защиты подсистемы управления доступом без зависимости от прочих входных данных;

– при создании файлов (до 18%) с аналогичным снижением разницы в последующих тестах (до 10%), что также ожидаемо за счет работы механизмов разграничения доступа и идентификации объектов;

– при удалении файлов – 12% и до рекордных 33% при работе подсистемы управления с выключенным и включенным модулем очистки остаточной информации соответственно, что свидетельствует о возможности серьезного падения производительности при очистке областей памяти удаляемых файлов посредством нескольких проходов записи произвольных значений;

– в операциях с памятью – от 8 до 12%, что объясняется работой модуля очистки оперативной памяти.

Необходимо отметить, что полученный в ряде случаев высокий уровень накладных расходов (максимально до 18% при неактивном модуле очистки памяти и до 33% при активном для теста по удалению файлов) не свидетельствует о значительном влиянии внедряемой подсистемы управления доступом на производительность системы. Связано это с тем, что во-первых, даже при добавлении обычных LSM-обработчиков без какой-либо логики и механизмов защиты в ванильное ядро Linux уровень накладных расходов в ряде случаев достигал 7% [99]. Кроме того приведенные микротесты могут не оказывать столь сильного влияния на общую производительность системы вследствие того, что все низкоуровневые операции ядра Linux обычно больше чем в тестах разнесены по времени, а быстродействие системы в результате будет упираться скорее в задержки подсистемы ввода-вывода. В связи с этим необходимо провести макротесты на уровне приложений, которые смогут показать влияние разработанной подсистемы управления доступом на реальную работу системы.

Макротесты проводились в тех же условиях, что и микротесты (см. обозначения *«os-clean»*, *«os-acx»*, *«os-memclean»*, *«os-fileclean»* и *«os-memfileclean»* выше), но в различные этапы работы (состояния) системы: процесс загрузки, стационарная фаза. В первом случае замерялось время работы всех системных сервисов и процессов в ходе загрузки ОС GNU/Linux с помощью инструмента *bootchart* (результаты приведены на Рисунке 4.4).

Необходимо отметить, что процесс загрузки ОС является наиболее критичным периодом работы с точки зрения количества всевозможных системных вызовов, выполняющихся в единицу времени. В то же время выполнение каждого системного вызова контролируется подсистемой управления доступом, что может сказаться на общей производительности системы. Однако, как видно на представленных графиках основные измеряемые показатели в данных макротестах (работа процессора и процессов ОС, подсистемы ввода вывода и самого носителя информации) практически неизменны в различных условиях тестирования: с внедренной подсистемой управления доступом и без нее.

В стационарной фазе работы ОС GNU/Linux запускались стресс-тесты из состава инструмента *phoronix-test-suite*, непосредственно связанные с вводом-выводом и операциями с объектами доступа, оказывающие всестороннее воздействие на тестируемую систему на уровне приложений: компиляция (*build-linux-kernel*) и распаковка ядра (*unpack-linux*), *compilebench* –

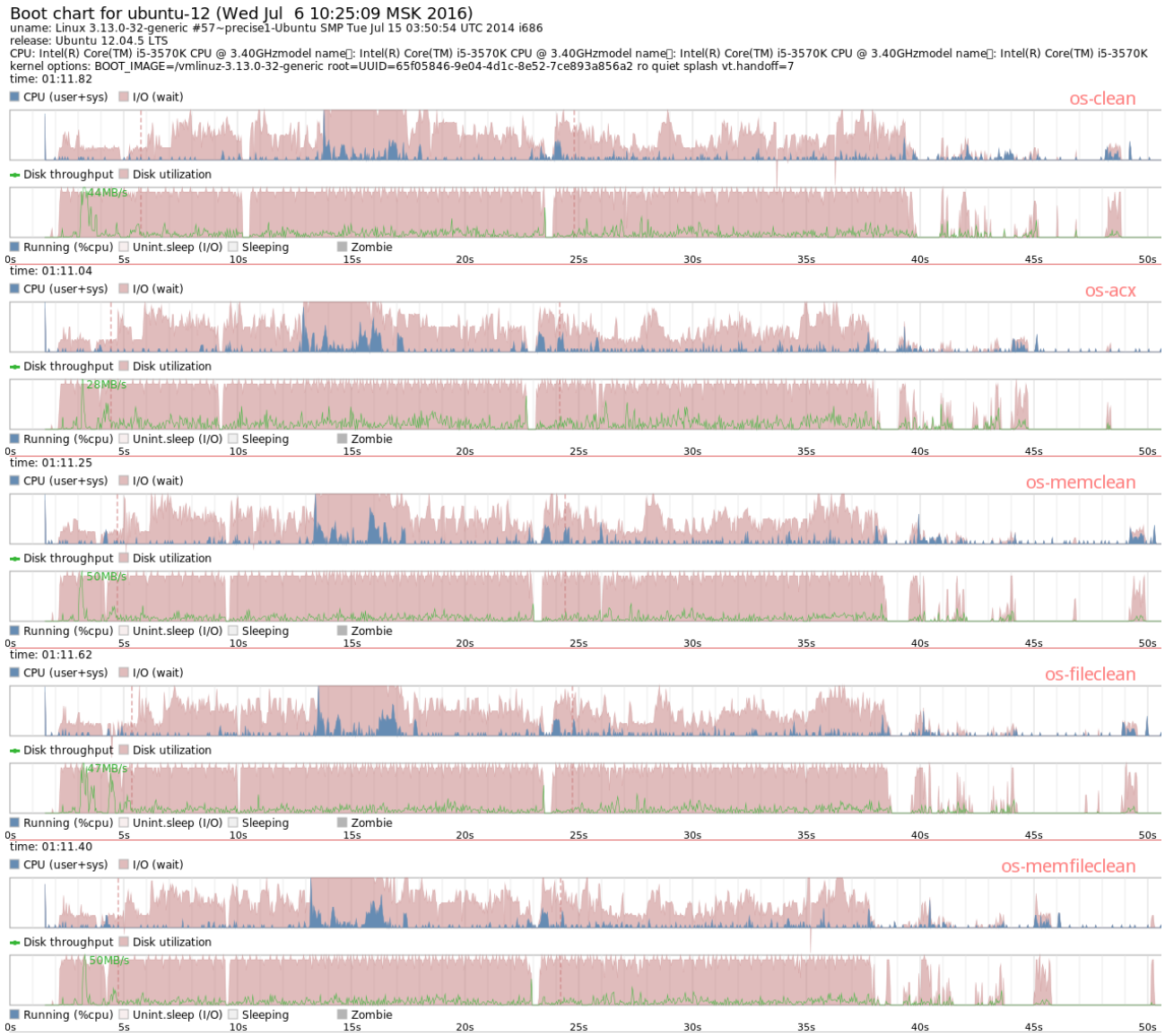

Рисунок 4.4 – Результаты макротестов на этапе загрузки ОС GNU/Linux и работы первоначальных системных сервисов в различных условиях

компиляция других проектов, *compress-gzip* – проведение операции сжатия, *iozone* и *aio-stress* в отношении эффективности различных операций ввода-вывода (результаты приведены на Рисунке 4.5[4]).

Результаты данных макротестов также положительные – существенного влияния внедренных механизмов защиты на производительность системы на уровне приложений не выявлено. Таким образом, можно сделать вывод о том, что при внедрении разработанной автором подсистемы управления доступом в ОС GNU/Linux возникают определенные накладные расходы в микротестах мгновенной производительности, которые в то же время не оказывают значи-

[4]Подробные результаты данных макротестов производительности доступны по следующим ссылкам:
- http://openbenchmarking.org/result/1606307-AR-OSCLEAN8388;
- http://openbenchmarking.org/result/1607012-AR-OSACX963353;
- http://openbenchmarking.org/result/1607028-AR-OSMEMCLEA04;
- http://openbenchmarking.org/result/1607033-AR-OSFILECLE15;
- http://openbenchmarking.org/result/1607051-AR-OSMEMFILE85.

Рисунок 4.5 – Результаты макротестов в стационарной фазе работы ОС GNU/Linux в различных условиях

тельного влияния на общую производительность при проведении макротестов, что говорит о минимальном воздействии предложенных способов и алгоритмов защиты на систему.

Однако при работе ряда отдельных приложений ОС GNU/Linux было зафиксировано заметное понижение производительности операции удаления при работе модуля очистки остаточной информации в уничтожаемых файлах (примерно на 30%). В соответствии с этим очистку остаточной информации целесообразно проводить для удаления только защищаемых объектов доступа с целью исключения такой потери производительности в масштабах всей системы.

Также необходимо заметить, что выявленные накладные расходы, как и ожидалось, возникают только для функций ядра, в отношении которых внедряются механизмы защиты (контроль порождения субъектов, разграничение доступа, очистка оперативной памяти и остаточной информации), на прочую функциональность ОС подсистема управления доступом не оказывает влияния даже по результатам микротестов.

4.3. Анализ опыта апробации и внедрения результатов исследования

Разработанная автором подсистема управления доступом предназначена для защиты информации в ОС GNU/Linux и изначально проектировалась для IBM-совместимых СВТ и архитектур x86, x86_64. Однако в основе этого СЗИ НСД используются универсальные способы и алгоритмы защиты, представленные в разделах 2.2–2.4. Выделим отличительные особенности разработанного СЗИ НСД по отношению к аналогичным решениям для защиты других ОС (на примере наиболее распространенных, для ОС семейства Windows [28]), а также исследуем практическую возможность внедрения и использования реализованных в нем способов и алгоритмов защиты на других аппаратных и программных платформах.

4.3.1. Подтверждение возможности использования разработанных алгоритмов обеспечения безопасности управления доступом в операционных системах, отличных от GNU/Linux

В разделе 2.3 вместо общепринятого понятия доверенной загрузки ОС предлагается использовать алгоритм обеспечения доверенной загрузки загрузчика и ОС при пошаговом контроле целостности, при этом, как было отмечено, в отношении ОС GNU/Linux эти два понятия не тождественны. В ОС семейства Windows используется стандартный загрузчик NTLDR (либо более новый Windows Boot Manager), который в отличие от загрузчиков GNU/Linux предусматривает выполнение только предопределенного сценария загрузки системы и не позволяет изменять процесс загрузки в динамике (без нарушения целостности конфигурационных файлов). В связи с этим для стандартных загрузчиков Windows их доверенная загрузка фактически влечет за собой и доверенную загрузку ОС. Однако, для Windows можно использовать нестандартные загрузчики, поставляемые с ОС GNU/Linux. Для загрузчиков ОС GNU/Linux необходимо принимать отдельные меры для обеспечения доверенной загрузки ОС после доверенной загрузки загрузчика (подробнее см. разд. 2.3). Предложенный автором алгоритм обеспечения доверенной загрузки загрузчика и ОС в этом смысле является более общим и имеет широкое применение, вследствие обеспечения доверенной загрузки не только GNU/Linux, но и различных систем, включая POSIX-совместимые (которые используют аналогичные по своим функциям загрузчики – grub legacy, GNU grub, syslinux, zipl и другие), а также вследствие возможности применения на аппаратных платформах, на которых использование аппаратных модулей доверенной загрузки невозможно.

Для различных версий Windows существуют серверная и настольная (англ. desktop) модификации ОС. При этом для настольных ОС Windows характерно существование только одного одновременно работающего пользователя, в том числе и при терминальном (удаленном) доступе. Многопользовательскими являются только серверные модификации ОС Windows. Разные дистрибутивы ОС GNU/Linux при этом также предлагают серверные и настольные модификации ОС, однако их разделение достаточно условно. Любой дистрибутив GNU/Linux представляет собой многопользовательскую систему с возможностью удаленного доступа сразу нескольких пользователей. Поэтому при разработке СЗИ НСД для GNU/Linux в отличие от Windows изна-

чально необходимо предусматривать многопользовательский вариант работы, а также возможность удаленного подключения пользователей. Для разработанной автором подсистемы управления доступом «Аккорд-X» предусмотрена реализация удаленного доступа пользователей, при этом «проброс» аппаратных идентификаторов для идентификации пользователей осуществляется с помощью инкапсуляции в существующие протоколы (например, ssh).

Дополнительно в ОС GNU/Linux, в отличие от Windows, подсистема управления доступом должна учитывать возможность смены учетной записи при продолжении работы пользовательской сессии (операции su и sudo, см. разд. 1.1.1). Поэтому механизмы идентификации и аутентификации, а также контроля порождения субъектов должны действовать не только для стандартной процедуры входа пользователя в ОС (см. раздел 2.1).

В ОС Windows существует достаточно жесткая привязка различных устройств и носителей информации к фиксированным точкам монтирования (например, раздел «C:»). В связи с этим для однозначной идентификации объектов вполне можно использовать абсолютные пути до файлов, если дополнительно осуществлять контроль подключенных носителей информации, что по сути эквивалентно контролю точек монтирования (см. далее), но, как правило, реализуется в АМДЗ. В ОС GNU/Linux для идентификации объектов недостаточно использовать только абсолютные пути, такой способ идентификации необходимо сочетать с контролем точек монтирования. Также в подсистеме управления доступом можно реализовать некоторые другие, более надежные способы идентификации объектов (см. разд. 2.2.2), которые на основании своей универсальности можно применять как в ОС Windows, так и в POSIX-совместимых системах.

Кроме того в ОС Windows отсутствует возможность существования нескольких жестких ссылок на одни и те же данные (что является дополнительной причиной использовать абсолютные пути для идентификации объектов). В ОС GNU/Linux же подсистема управления доступом должна учитывать такую возможность, а для доступа по созданию жестких ссылок необходимо предусмотреть отдельный атрибут доступа (разрешающий или запрещающий создание ссылок).

Также, посредством использования способа контроля порождения субъектов доступа и идентификации субъектов, от имени которых соответствующие процессы ОС GNU/Linux взаимодействуют с объектами системы, из раздела 2.2.1 становится возможных использовать в подсистеме управления доступом дополнительный тип учетных записей для различных системных сервисов. Так, аналогично тому, как ограничиваются в средствах разграничения доступа для Windows возможные действия пользователей, в GNU/Linux можно ограничить действия различных сервисов, в том числе web- или ftp-сервера. Таким образом, даже при некорректной настройке таким сервисам можно будет осуществлять доступ только к строго определенным объектам (например, только в каталог с web-сайтом для web-сервера). В ОС Windows доступ системных процессов обычно не ограничивается напрямую (в том числе вследствие частого существования одного одновременно работающего пользователя), однако необходимость этого присутствует.

В отличие от Windows в GNU/Linux иначе реализована подсистема печати. Модуль контроля печати подсистемы управления доступом для GNU/Linux реализован в виде фильтра CUPS, а не на уровне ядра ОС (как это обычно реализуется в Windows). Однако вследствие кросс-

платформенности сервера печати CUPS предложенные автором программно-технические решения для модуля контроля печати (см. раздел 3.2) могут практически в неизменном виде быть использованы на широком спектре различных систем.

В ОС GNU/Linux и Windows различным образом реализованы стандартные механизмы доступа к объектам. Так, например, в Windows атрибуты доступа для файлов имеют более приоритетное значение, нежели атрибуты для содержащих их каталогов, больший приоритет имеют запретительные правила доступа. Также в ОС Windows существует больше типов доступа к объектам, при этом права доступа можно настроить для любого отдельно взятого пользователя или группы. При этом как и в GNU/Linux в ОС Windows реализуется распределенная схема администрирования, когда пользователи могут изменять и передавать права доступа на объекты, владельцами которых они являются. При реализации защитных механизмов подсистемы управления доступом нужно учитывать подобные особенности.

Еще одним важным отличием является тот факт, что различных версий ОС Windows значительно меньше чем различных версий дистрибутивов GNU/Linux. В связи с этим работоспособность подсистемы управления доступом для Windows достаточно обеспечить для следующих актуальных на сегодняшний день версий Windows: XP, Server 2003, Vista, Server 2008, 7, Server 2008 R2, 8, Server 2012 R2, 8.1, 10 (отдельно для 32-разрядных и 64-разрядных версий). При этом вследствие единообразности и совместимости системного программного и бинарного интерфейсов (API и ABI) зачастую существует возможность использования всего двух сборок подсистемы управления доступом – для 32- и 64-разрядных ОС Windows. Дистрибутивы GNU/Linux в этом плане имеют больше ограничений. Так, например, использование собранных загружаемых модулей возможно только на конкретной версии ядра Linux, а при обновлении ядра в ОС GNU/Linux необходимо соответствующим образом пересобрать и подсистему управления доступом. При использовании какой-либо специализированной сборки ядра GNU/Linux с большой долей вероятности также может потребоваться пересборка всех модулей ядра Linux. Кроме того, существует определенная зависимость бинарных объектных файлов от используемых версий стандартных библиотек. Создать полностью переносимые бинарные файлы (утилит администрирования подсистемы управления доступом) между всеми дистрибутивами GNU/Linux даже с одной архитектурой достаточно проблематично из-за возможного наличия несовместимости в API и/или ABI. Поэтому на основании изложенного выше, а также вследствие возможности произвольным образом изменять ядро Linux возникают сложности в распространении, создании и тестировании подсистемы управления доступом для ОС GNU/Linux.

Таким образом, в разделе выделены отличительные особенности разработанной подсистемы управления доступом для GNU/Linux в сравнении с аналогичными СЗИ НСД для ОС семейства Windows, такие как: гарантированная загрузка не только загрузчика, но и самой ОС; необходимость использования СЗИ НСД для многопользовательской системы с возможностью одновременной работы нескольких субъектов (в том числе и с помощью протоколов удаленного доступа), возможность смены субъекта доступа в процессе работы сессии пользователя, недостаточность идентификации объектов доступа по абсолютному пути, необходимость учета

типа доступа по созданию жестких ссылок, необходимость пересборки СЗИ НСД для различных версий дистрибутивов GNU/Linux и ядра Linux и некоторые другие.

Необходимо, однако, отметить, что несмотря на рассмотренные в разделе отличия ОС GNU/Linux от Windows, касающихся в том числе и разграничения доступа субъектов к объектам (в отношении Unix-подобных систем перечисленные отличия в основном отсутствуют), разработанные в работе способы и алгоритмы защиты из разделов 2.2–2.4, а также требования к подсистеме управления доступом из раздела 3.1 имеют широкое применение и их можно использовать как в отношении ОС Windows, так и для свободно распространяемых и/или коммерческих POSIX/SUS-совместимых ОС или LSB-совместимых дистрибутивов Linux. Разработанная в разделе 2.1 модель безопасности ОС с подсистемой управления доступом применима в отношении POSIX/SUS-совместимых ОС, а также в отношении других ОС и систем. Предложенные автором программно-технические решения по созданию подсистемы управления доступом из разделов 3.2 и 3.3 вследствие своей специфики могут быть применены только в отношении свободно распространяемых и/или коммерческих дистрибутивов GNU/Linux, а также в отношении других POSIX/SUS-совместимых ОС (за исключением использования LSM при встраивании).

4.3.2. Подтверждение возможности использования разработанной подсистемы управления доступом на различных аппаратных платформах

Разработанная в разделе 3.3 подсистема управления доступом представляет собой переносимое решение на различные аппаратные и программные платформы, поддерживаемые ядром GNU/Linux, в том числе x86, x86_64, arm и некоторые другие. При этом подсистема управления доступом является переносимой на уровне исходного кода, однако требует пересборки под конкретную платформу и версию ядра Linux (см. раздел 4.3.1).

Однако широко используемые аппаратные модули доверенной загрузки (например, «Аккорд-АМДЗ» из состава ПАК СЗИ НСД «Аккорд-X») имеют большую зависимость от аппаратной платформы [62] и могут применяться для доверенной загрузки IBM-совместимых СВТ (то есть, в основном, для архитектур x86, x86_64). В соответствии с этим в разделе 2.3 предложено на аппаратных платформах, на которых невозможно использовать стандартные АМДЗ (например, s390x и arm), обеспечивать технологическую невозможность нарушения целостности соответствующих критически важных компонент ОС и подсистемы управления доступом. В результате этого становится возможным использовать полученные результаты исследования на большем количестве аппаратных и программных платформ.

В результате разработанная автором подсистема управления доступом нашла свое применение на мейнфреймах компании IBM, использующих специальные версии GNU/Linux (zLinux) с архитектурой s390x [27, 110]. Рассмотрим отличия и особенности использования разработанной автором подсистемы управления доступом в отношении zLinux.

Термин zLinux (z/Linux) или Linux on System z – это обобщенное название операционных систем GNU/Linux, функционирующих на современной линейке мейнфреймов IBM System z. Платформа System z предполагает использование аппаратной виртуализации (англ. bare-metal virtualization) на базе специального гипервизора (Processor Resource and System Manager, PR/SM),

разделяя при этом «физические» ресурсы мейнфрейма между логическими разделами (англ. Logical Partitions, LPAR) и организуя таким образом первый слой виртуализации.

В рамках LPAR могут быть загружены экземпляры гостевых ОС (виртуальных машин, ВМ), в качестве которых могут выступать z/OS и zLinux. Кроме того, на отдельном LPAR можно организовать второй (а затем третий и так далее), вложенный слой виртуализации с помощью такого средства, как z/VM. С помощью такой «многослойной» виртуализации System z позволяет на одном физическом мейнфрейме запускать одновременно сотни и даже тысячи полностью изолированных друг от друга гостевых ОС, в зависимости от объема доступных ресурсов.

В качестве гостевых ОС на System z могут функционировать [110]:

1. z/OS – 64-битная серверная ОС производства IBM (дальнейшее развитие OS/390).
2. z/VM – 64-битная ОС-гипервизор как N + 1 уровень вложенности (подходит для проектируемых систем и разработки).
3. zLinux – 31-битная (архитектура s390, не используется) или 64-битная ОС Linux (архитектура s390x), которая представлена следующими поддерживаемыми дистрибутивами:
   - RedHat Enterprise Linux Server, SUSE Linux Enterprise Server и Ubuntu Server (официально заявленная поддержка);
   - Debian, Fedora, Slackware, Gentoo и другие (поддерживаются неофициально).

Все гостевые ОС могут оперировать как с реальными, так и с мнимыми (виртуальными) ресурсами, причем суммарно количество виртуальных ресурсов может превосходить тот объем реальных объектов, на которые эти мнимые ресурсы отображаются (включая оперативную память и даже количество процессоров).

Весь архитектурно-зависимый код для поддержки архитектуры s390x внесен в основную ветку ядра Linux, в связи с чем на System z можно с той или иной степенью сложности портировать любой современный дистрибутив GNU/Linux. Рассмотрим возможность применения уже реализованных способов и алгоритмов защиты (см. главы 2 и 3, а также [23–25, 27, 33, 38, 93]), в подсистеме управления доступом для zLinux.

Основное предназначение мейнфрейма – обработка большого количества разнородных операций ввода-вывода. Мейнфрейм может выступать и как средство для проведения сложных вычислений. Однако типовым применением для данной платформы могут быть: работа с большими базами данных, CRM, SAP и какие-то другие серверные решения, требующие в своей работе выполнения большого количества мелких операций. В связи с этим разграничение доступа между пользователями одной отдельно взятой ВМ не является основной требуемой функцией для внедряемой подсистемы управления доступом. В zLinux, скорее всего, либо вообще не существует реальных пользователей, либо их будет очень маленькое количество (2-3, все пользователи – администраторы с различными ролями), так как основная работа ВМ мейнфрейма заключается в работе каких-либо процессов (web-сервер, СУБД и так далее). Однако, с другой стороны, разграничение доступа встраиваемым средством может потребоваться для недопущения повышения привилегий в системе из-за какой-либо уязвимости ОС (ядра ОС, прикладного ПО) – в этом случае повышения привилегий в СЗИ НСД происходить не будет (см. разд. 3.3).

При обработке какой-либо конфиденциальной информации или персональных данных необходимо выполнять ряд нормативных требований РФ (см. разд. 1.2). Чтобы представлять необходимые механизмы защиты информации, которые целесообразно применять по отношению к System z, необходимо учитывать следующие особенности этой платформы:

– процедуру идентификации и аутентификации проходить будут (в основном) только ответственные администраторы с целью первичной настройки, ввода каких-либо сервисов в эксплуатацию, либо для проведения технического обслуживания уже функционирующих сервисов;

– обычных пользователей в ВМ System z фактически нет (при профильном использовании), в связи с чем процедура идентификации и аутентификации не будет интенсивно использоваться при эксплуатации;

– разграничение доступа будет применяться для серверных процессов, запуск которых не инициируется пользователями (администраторами) напрямую;

– System z не предполагает использования каких-либо внешних носителей информации (нет возможности физически подключить какие-либо стандартные носители информации из-за отсутствия соответствующих интерфейсов: в ядре ОС Linux для архитектуры s390x нет USB-стека). В связи с этим требования нормативных документов РФ по очистке (обнулению, обезличиванию) освобождаемых областей памяти на внешних накопителях и по учету внешних носителей в целом являются избыточными;

– System z не предполагает использования печати из ВМ (в настоящее время реальные устройства печати для IBM System z уже просто не выпускаются), однако печать на сетевых принтерах в целом не запрещена;

– на System z на данный момент нет возможности использования интерфейсов PCI, PCI-Express и mini-PCI, кроме как для специальных криптографических, сетевых карт и карт ускорения сжатия данных (при этом большинство АМДЗ используют именно такие интерфейсы).

Таким образом, в части разграничения доступа на zLinux наиболее актуальными механизмами защиты данных можно считать [33, 107]:

1. Идентификацию и аутентификацию субъектов доступа, контроль порождения и однозначную идентификацию субъектов, которые не совершают действий login/logout (демоны, сервисы и другие системные процессы), а также реальных пользователей (как правило, администраторов).

2. Разграничение доступа (в основном для исключения возможности повышения привилегий, а также в отношении контроля доступа субъектов-процессов к объектам доступа), регистрация и учет запуска процессов и попыток НСД.

3. Динамический и статический контроль целостности данных.

4. Очистку оперативной памяти (для высоких классов защищенности).

5. Контроль целостности ОС непосредственно до старта ВМ (то есть обеспечение «доверенной» загрузки ВМ для более высоких классов защищенности).

Реализация пунктов 1–4 представляет собой практически неизмененную реализацию способов и алгоритмов защиты, рассмотренных в главах 2 и 3. При этом в данном случае нет необходимости каким-либо образом изменять или внедрять сторонний код в архитектуру System z,

так как все эти способы и алгоритмы защиты реализуются на уровне ОС GNU/Linux и не затрагивают работу каких-либо средств System z или z/VM. Однако в целях безопасности, возможно, придется отказаться от использования некоторых возможностей System z, таких, как [110]:

– разделяемая память между ВМ (англ. common shared memory);

– разделяемое ядро ОС Linux в памяти (англ. shared kernel in memory) на уровне гипервизора, прозрачно для ВМ;

– возможность подключения дисков к другим ВМ внутри z/VM или LPAR.

При портировании разработанной подсистемы управления доступом с других архитектур (изначально x86, x86_64) возникли следующие особенности:

– невозможность использования архитектурно-зависимого кода (например, код по изменению памяти ядра Linux имеет архитектурную зависимость и должен быть реализован на каждой архитектуре по-разному, см. разд. 3.3, 4.3.1 и пункт далее)[5];

– используется другой формат бинарных данных для всех компонент ОС (если используются особенности бинарного формата, возникнут сложности при портировании СЗИ НСД);

– невозможность использования аппаратных идентификаторов (по интерфейсу USB). Для решения данной задачи проще всего работать с такими идентификаторами локально на клиентском СВТ, инкапсулируя все необходимые данные для идентификации в протоколы удаленного управления (ssh, telnet и другие, см. разд. 4.3.1).

Также в zLinux используется свой системный загрузчик *zipl* (вместо привычных grub, lilo и некоторых других). Его настройка мало чем отличается от настройки grub (см. разд. 2.3), конфигурационный файл /etc/zipl.conf предоставляет возможность задать опции ядра, изменить initrd или ядро Linux, а также сценарий загрузки по умолчанию (этих операций достаточно для корректного встраивания подсистемы управления доступом, см. разд. 3.3):

```
defaultboot!
default=linux+accordx
target=/boot/
linux+accordx!
        image=/boot/vmlinuz-2.6.18-348.el5
        ramdisk=/boot/initrd,0x1800000
        parameters="root=/dev/dasda1 selinux=0"
```

При реализации механизмов защиты 1–4 подсистема управления доступом может соответствовать только низким классам защищенности – оно по своей сути не будет ничем отличаться от стандартных средств защиты ОС GNU/Linux, будет существовать возможность изменить или отключить защитные механизмы, например, на раннем этапе загрузки ВМ. В связи с этим для полноценной защиты zLinux необходимо дополнительно выполнять контроль целостности или неизменности компонент ОС GNU/Linux до старта ВМ (аналогично доверенной загрузке за-

[5] В этой связи для s390x целесообразнее использовать технологию LSM вместо других архитектурно-зависимых способов встраивания подсистемы управления доступом (см. раздел 4.2.1).

грузчика и ОС с использованием АМДЗ). Однако реализовать такой механизм защиты в общем случае непросто из-за следующих причин:

– в System z нельзя использовать стандартные АМДЗ, так как этого не позволяет архитектура, однако код гипервизора загружается из энергонезависимой памяти (доступной только на чтение). В соответствии с этим загрузку гипервизора при загрузке мейнфрейма можно считать «доверенной», так как на сегодняшний день нет возможности вместо гипервизора загружать на мейнфрейме что-либо еще для обеспечения работы множества изолированных ОС типа zLinux;

– в System z какой-либо код на уровне гипервизора можно использовать только с ограничениями (код, который позволил бы контролировать целостность ВМ в LPAR, запускаемых на гипервизоре).

Поэтому для контроля целостности ВМ до старта необходимо применять какой-то иной способ. Например, можно контролировать целостность компонент ОС, запущенной на базе z/VM, из другой ВМ (она будет играть роль «программного» АМДЗ и будет контролировать целостность всех ВМ). В процессе запуска защищаемой ВМ в z/VM можно, используя дополнительный инструментарий z/VM (RACF), перехватить управление и запустить проверку целостности данных на другой ВМ. Если проверка завершится успешно, на z/VM можно продолжить загрузку защищаемой ВМ с дальнейшим запуском ОС и подсистемы управления доступом. В случае возникновения ошибки ВМ должна быть остановлена с созданием соответствующего события в журнале СЗИ НСД. При этом расположить «проверочную» ВМ можно и вне z/VM, она может быть доступна для взаимодействия из z/VM, но недоступна для обычных ВМ. Схематичное представление предлагаемого решения по защите zLinux приведено на Рисунке 4.6.

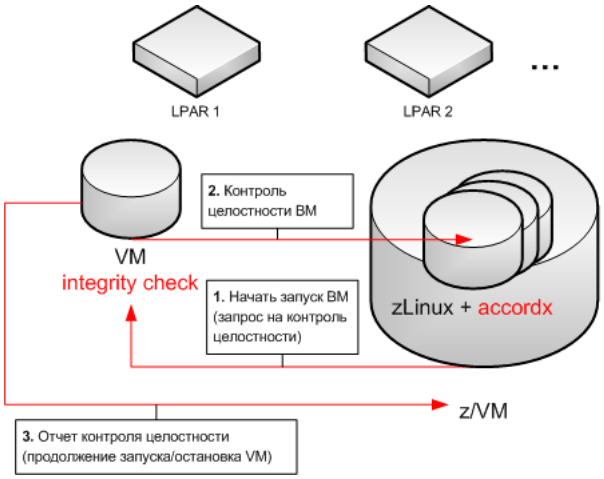

Рисунок 4.6 – Схематичное представление обеспечения целостности компонент виртуальных машин в z/VM на мейнфреймах IBM System z

Такое решение имеет следующие ограничения:

– защищаемые ВМ с zLinux необходимо разворачивать в z/VM (отдельный продукт);

– для организации перехвата процедуры начальной загрузки (IPL) гостевой ОС zLinux необходимо приобретение продукта z/VM RACF;

– ВМ, которая будет играть роль «программного» АМДЗ, должна иметь доступ (на чтение) к дискам тех ВМ, которые требуется защищать, а также быть изолированной от этих ВМ;

– в данном решении осуществляется внедрение в работу z/VM и, возможно, других компонент System z;

– сложность формализации разграничения доступа администраторов z/VM (необходимо ограничивать их полномочия, в том числе с использованием организационных мер).

Существует также другой способ для обеспечения неизменности компонент ОС ВМ и подсистемы управления доступом, при котором нет необходимости внедряться в архитектуру z/VM или гипервизор. Всем ВМ можно выделять некоторые, доступные в режиме только для чтения (англ. read-only, RO) мини-диски для записи на них подсистемы управления доступом, ядра ОС и всех остальных объектов контроля целостности (целостность которых контролировалась с помощью «программного» АМДЗ). Но такой вариант не устраняет сложность разграничения доступа администраторов z/VM, так как режим RO может быть изменен на режим чтения и записи (read-write, RW), а доступ самого гипервизора z/VM всегда RW.

В соответствии с этим для полноценного использования алгоритма обеспечения доверенной загрузки загрузчика и ОС при пошаговом контроле целостности в zLinux необходимо в отношении критически важных компонент ОС и подсистемы управления доступом использовать специальные носители памяти с контролируемым доступом, изначально доступных только для чтение (см. разделы 2.3 и 3.3, а также [43]). При этом использование таких специальных устройств, обеспечивающих технологическую неизменность критически важных компонент системы, нашло свое применение вместе с разработанной автором подсистемой управления доступом не только в отношении архитектуры s390x, но также: в отношении встраиваемых устройств на базе arm (защищенных микрокомпьютеров), а также в средствах организации доверенного сеанса связи с обычной архитектурой x86/x86_64, но для которых невозможно использовать стандартные АМДЗ в силу требуемой мобильности решения.

В диссертации представлена разработанная автором подсистема управления доступом, на основе которой был создан ПАК СЗИ НСД «Аккорд-X», производства компании ОКБ САПР (свидетельство о государственной регистрации № 2015612555 [61]). Комплекс «Аккорд-X» и его подсистема управления доступом обладают сертификатами соответствия требованиям ФСТЭК России № 3079/4447 и удовлетворяют требованиям нормативных документов в области защиты информации РФ для 3/5 класса защищенности СВТ и 2/4 уровня контроля отсутствия недекларированных возможностей соответственно.

Кроме того, результаты проведенного автором исследования и представленная подсистема управления доступом используются ГНИИ ПТЗИ ФСТЭК России в рамках исследований уязвимостей системного ПО для национального банка данных угроз безопасности информации и для верификации функциональных требований к средствам защиты информации от НСД на раннем этапе загрузки ОС.

Также результаты работы применяются в решениях по защите информации, обрабатываемой на мейнфреймах IBM System z в виртуальных машинах zLinux. Для защиты гостевых ОС zLinux в рамках среды виртуализации System z предложенная автором подсистема управления доступом используется для усиления защитных свойств встроенной системы защиты IBM RACF

в части разграничения доступа к конфиденциальной информации серверных систем, функционирующих как на уровне логических разделов, так и на уровне системы вложенной виртуализации IBM z/VM. При этом для аппаратной платформы System z вместо контроля целостности критически важных компонент системы и подсистемы управления доступом обеспечивается технологическая невозможность нарушения их целостности, что позволяет осуществлять доверенную загрузку ОС серверных систем на СВТ с процессорной архитектурой s390x.

Результаты исследования также внедрены в учебный процесс НИЯУ МИФИ по дисциплине «Программно-аппаратные средства защиты информации» (лабораторные работы).

Практическая значимость работы подтверждается положительными итогами использования разработанной подсистемы управления доступом в реальных проектах и совпадением ожидаемых результатов от использования предложенных автором алгоритмов и способов обеспечения безопасности.

4.4. Выводы к главе 4

1. Обоснована корректность взаимодействия и сочетания разработанной подсистемы управления доступом с другими применяемыми в GNU/Linux средствами защиты информации от НСД (и соответствующими формальными моделями безопасности). Показано, что при использовании предложенного алгоритма встраивания сочетание нескольких внедренных в ОС средств разграничения доступа можно представить в виде одностороннего блокировочного соединения автоматов, в результате чего обеспечивается невлияние и отсутствие конфликтов между такими средствами вне зависимости от последовательности их работы.

2. В ходе экспериментальных исследований подтверждены следующие свойства разработанной подсистемы управления доступом: невозможность отключения или несанкционированного изменения правил разграничения доступа, невозможность возникновения несанкционированных субъектов и неучтенных объектов доступа, достижимость и невозможность «размыкания» изолированной программной среды субъектов, полнота реализованных алгоритмов защиты, а также изолированность пользовательских сред и сессий различных субъектов доступа ОС в процессе работы. Таким образом, проведенные исследования подтвердили выполнение ожидаемых свойств ОС со встроенной подсистемой управления доступом, а также соответствие ее реализации предложенной модели безопасности в результате верификации.

3. При проведении сравнительных тестов с различным сочетанием активных функций защиты подсистемы управления доступом были выявлены определенные накладные расходы в микротестах (мгновенная производительность ОС), которые в то же время не оказывают существенного влияния на общую производительность системы при проведении макротестов. Выявленные накладные расходы, в соответствии с ожиданиями, возникают только для функций ядра, в отношении которых внедряются функции защиты подсистемы управления доступом, и не оказывают существенного влияния на реальную производительность системы, что подтверждено результатами проведенных тестов.

4. В результате анализа применимости результатов работы и программно-технических решений на других программных и аппаратных платформах показано, что разработанные алгорит-

мы защиты, а также требования к подсистеме управления доступом имеют широкое применение и их можно использовать как в отношении ОС Windows, так и для POSIX/SUS-совместимых ОС. Разработанная модель безопасности ОС GNU/Linux с подсистемой управления доступом применима в отношении произвольных ОС и других систем. Предложенные автором программно-технические решения по созданию подсистемы управления доступом могут быть применены в отношении различных дистрибутивов GNU/Linux и других POSIX/SUS-совместимых ОС вне зависимости от используемой аппаратной платформы.

ЗАКЛЮЧЕНИЕ

В диссертационной работе сформирована модель изолированной программной среды субъектов, а также необходимые и достаточные условия невозможности нарушения действующей политики управления доступом системы. Для выполнения этих условий на практике разработаны соответствующие алгоритмы обеспечения безопасности управления доступом.

Основные результаты проведенного исследования заключаются в следующем:

1. В результате проведенного анализа существующих средств разграничения доступа для ОС GNU/Linux и реализованных в них способов защиты информации от НСД, результатов известных работ по совершенствованию способов управления доступом в ОС и формальных моделей безопасности выявлены серьезные недостатки, которые могут стать причиной возникновения НСД к данным. Наиболее существенные недостатки заключаются в возможности отключить или исключить активизацию функций защиты информации от НСД на раннем этапе загрузки системы, а также в возможности воздействия на подсистему управления доступом ОС с целью обхода действующих правил политики безопасности.

2. Предложена модель безопасности ОС GNU/Linux с подсистемой управления доступом, обеспечивающая невозможность нарушения заданной политики безопасности в обход ее правил. Модель основана на положениях известной СО-модели изолированной программной среды, и в ней, в отличие от последней, обеспечивается выполнение заданной политики управления доступом в отношении различных пользователей и системных субъектов, задается порядок достижения начального состояния системы с контролем непрерывности работы функций защиты от НСД.

3. Предложен алгоритм обеспечения доверенной загрузки загрузчика и ОС GNU/Linux при пошаговом контроле целостности, устраняющий возможность исключения активизации подсистемы управления доступом до или на раннем этапе загрузки системы на различных аппаратных платформах посредством введения новых процедур по ограничению возможностей загрузчиков, контролю или обеспечению технологической невозможности нарушения целостности критически важных компонент системы. В отличие от аналогов, данный алгоритм применим для широкого спектра СВТ и обеспечивает не только доверенную загрузку загрузчика GNU/Linux, но и дальнейшую доверенную загрузку самой ОС.

4. Предложен алгоритм встраивания функций защиты от НСД, который, в отличие от аналогов, учитывает необходимость обеспечения активизации и целостности подсистемы управления доступом на раннем этапе работы ОС, а также непрерывности дальнейшего функционирования с принудительной блокировкой системы в случае нарушения точек встраивания; применения правил разграничения доступа в отношении всех субъектов, объектов и типов доступа.

5. Комплексное использование предложенных алгоритмов обеспечения безопасности управления доступом позволяет выполнить необходимые и достаточные условия для дальнейшей реализации изолированной программной среды субъектов доступа в соответствии с предложенной моделью безопасности. Кроме того, данные алгоритмы имеют широкое назначение

и могут применяться как в ОС, отличных от Linux, так и по аналогии в системах управления доступом других систем.

6. На базе полученных научных результатов разработана подсистема управления доступом для ОС GNU/Linux и основанный на ней программно-аппаратный комплекс «Аккорд-X», реализующие предложенные способы и алгоритмы защиты и соответствующие сформулированной модели безопасности.

7. Обоснована корректность взаимодействия и сочетания разработанной подсистемы управления доступом с другими применяемыми в GNU/Linux средствами защиты информации от НСД и соответствующими формальными моделями безопасности. Показано, что при использовании предложенного алгоритма встраивания сочетание нескольких внедренных в ОС средств разграничения доступа можно представить в виде одностороннего блокировочного соединения автоматов, в результате чего обеспечивается невлияние и отсутствие конфликтов между такими средствами вне зависимости от последовательности их работы.

8. В ходе экспериментальных исследований подтверждены ожидаемые свойства разработанной подсистемы управления доступом. При проведении сравнительных тестов с различным сочетанием активных функций защиты подсистемы управления доступом были выявлены определенные накладные расходы в микротестах мгновенной производительности ОС, которые в то же время не оказывают существенного влияния на общую производительность системы при проведении макротестов. Таким образом, проведенные исследования подтвердили выполнение ожидаемых свойств ОС со встроенной подсистемой управления доступом, а также соответствие ее реализации предложенной модели безопасности в результате верификации.

В заключении хотелось бы отметить, что в качестве дальнейших перспектив развития темы диссертационной работы целесообразным видится развитие методов адаптивного управления защитой информации от НСД, которые в самом простом виде были реализованы автором в разработанной подсистеме управления доступом в части фиксации возникновения новых или изменения существующих объектов доступа в системе и соответствующего изменения списка прав доступа. Другим возможным направлением развития можно считать совершенствование предложенных или разработку новых алгоритмов обеспечения безопасности управления доступом для различных программных и аппаратных платформ, поддерживаемых GNU/Linux, в том числе: zLinux, arm, sparc, power и других, а также дальнейшее развитие способов технологической гарантии неизменности критически важных компонент ОС и подсистемы управления доступом.

Список сокращений и условных обозначений

GNU	проект по разработке свободного программного обеспечения
LKM	Loadable Kernel Modules
PAM	Pluggable Authentication Modules
АМДЗ	Аппаратный модуль доверенной загрузки
АС	Автоматизированная система
ВМ	Виртуальная машина
ДВС	Доверенная вычислительная среда
ИПС	Изолированная программная среда
ИПСС	Изолированная программная среда субъектов
ИС	Информационная система
ИСПДн	Информационная система персональных данных
КС	Компьютерная система
МРД	Монитор разграничения доступа
НСД	Несанкционированный доступ
ОС	Операционная система
ПАК СЗИ	Программно-аппаратный комплекс средств защиты информации
ПДн	Персональные данные
ПО	Программное обеспечение
РД	Руководящий документ
СВТ	Средство вычислительной техники
СЗИ	Средство защиты информации
СО-модель ИПС	субъектно-ориентированная (субъектно-объектная) модель ИПС
СУБД	Система управления базами данных
ФС	Файловая система
ХРУ	модель матрицы доступов Харрисона-Руззо-Ульмана

Список литературы

1. Алгоритмы: построение и анализ / Т. Кормен [и др.]. — 3-е изд. — М. : Вильямс, 2019. — 1328 с.
2. *Бажитов, И. А.* Обеспечение доверенной среды в ОС Linux с использованием ПАК СЗИ НСД «Аккорд-Х» / И. А. Бажитов // *Комплексная защита информации. Материалы XV Международной конференции 1-4 июня 2010 года, Иркутск (РФ).* — 2010. — С. 32–34.
3. *Бовет, Д.* Ядро Linux / Д. Бовет, М. Чезати. — 3-е изд. — СПб. : БХВ-Петербург, 2007. — 1104 с.
4. ГОСТ Р ИСО/МЭК 15408. Информационная технология. Методы и средства обеспечения безопасности. Критерии оценки безопасности информационных технологий. — М. : Госстандарт России, 2013. — Т. 1–3.
5. *Головинский, И. А.* Блокировки автоматов групп. I. Односторонние блокировки / И. А. Головинский // *Известия РАН. Теория и системы управления*. — 2009. — № 1. — С. 49–73.
6. Гостехкомиссия России. Руководящий документ. Автоматизированные системы. Защита от несанкционированного доступа к информации. Классификация автоматизированных систем и требования по защите информации. — М. : ГТК РФ, 1992. — 39 с.
7. Гостехкомиссия России. Руководящий документ. Концепция защиты средств вычислительной техники и автоматизированных систем от несанкционированного доступа к информации. — М. : ГТК РФ, 1992. — 12 с.
8. Гостехкомиссия России. Руководящий документ. Средства вычислительной техники. Защита от несанкционированного доступа к информации. Показатели защищенности от несанкционированного доступа к информации. — М. : ГТК РФ, 1992. — 24 с.
9. *Грушо, А. А.* Теоретические основы защиты информации / А. А. Грушо, Е. Е. Тимонина. — М. : Издательство Агентства «Яхтсмен», 1996. — 192 с.
10. *Девянин, П. Н.* Анализ безопасности управления доступом и информационными потоками в компьютерных системах / П. Н. Девянин. — М. : Радио и связь, 2006. — 176 с.
11. *Девянин, П. Н.* Базовая ролевая ДП-модель / П. Н. Девянин // *Прикладная дискретная математика*. — 2008. — № 1. — С. 64–70.
12. *Девянин, П. Н.* О разработке моделей безопасности информационных потоков в компьютерных системах с ролевым управлением доступом / П. Н. Девянин // *Материалы III международной научной конференции по проблемам безопасности и противодействия терроризму. МГУ им. М.В. Ломоносова. 25-27 октября 2007 г.* — 2008. — С. 261–265.
13. *Девянин, П. Н.* Модели безопасности компьютерных систем. Управление доступом и информационными потоками. Учебн. пособие для вузов. / П. Н. Девянин. — 2-е изд. — М. : Горячая линия - Телеком, 2015. — 320 с.
14. *Девянин, П. Н.* Применение подтипов и тотальных функций формального метода Event-B для описания и верификации МРОСЛ ДП-модели / П. Н. Девянин, М. А. Леонова // *Программная инженерия*. — 2020. — Т. 11, № 4. — С. 230–241.
15. *Ефанов, Д. В.* Правила изменения домена выполнения процесса в SELinux / Д. В. Ефанов,

П. Г. Рощин // *Спецтехника и связь*. — 2015. — № 3. — С. 35–39.

16. *Ефанов, Д. В.* Практические вопросы разработки политики безопасности SELinux / Д. В. Ефанов, П. Г. Рощин, К. Г. Григорьев // *Спецтехника и связь*. — 2015. — № 4. — С. 23–28.
17. *Зегжда, Д. П.* Принципы и методы создания защищенных систем обработки информации : автореф. дис. д-ра техн. наук : 05.13.19 / Д. П. Зегжда. — СПб. , 2002. — 39 с.
18. *Зегжда, Д. П.* Основы безопасности информационных систем / Д. П. Зегжда, А. М. Ивашко. — М. : Горячая линия - Телеком, 2000. — 452 с.
19. Интеграция мандатного и ролевого управления доступом и мандатного контроля целостности в верифицированной иерархической модели безопасности операционной системы / П. Н. Девянин [и др.] // *Труды Института системного программирования РАН*. — 2020. — Т. 32, № 1. — С. 7–26.
20. *Каннер, А. М.* Идентификация и аутентификация пользователей при работе подсистемы разграничения доступа в ОС Linux / А. М. Каннер // *Безопасность информационных технологий*. — 2012. — № 1кзи. — С. 104–105.
21. *Каннер, А. М.* Контроль печати в ОС Linux / А. М. Каннер // *Безопасность информационных технологий*. — 2012. — № 1кзи. — С. 106–108.
22. *Каннер, А. М.* Особенности перехвата системных вызовов при построении подсистемы разграничения доступа в ОС Linux / А. М. Каннер // *Безопасность информационных технологий*. — 2012. — № 1кзи. — С. 109–111.
23. *Каннер, А. М.* Linux: к вопросу о построении системы защиты на основе абсолютных путей к объектам доступа / А. М. Каннер // *Комплексная защита информации. Материалы XVIII Международной конференции 21-24 мая 2013 года, Брест (РБ)*. — 2013. — № 6. — С. 126–128.
24. *Каннер, А. М.* Linux: о доверенной загрузке загрузчика ОС / А. М. Каннер // *Безопасность информационных технологий*. — 2013. — № 2. — С. 41–46.
25. *Каннер, А. М.* Linux: объекты контроля целостности / А. М. Каннер // *Комплексная защита информации. Материалы XVIII Международной конференции 21-24 мая 2013 года, Брест (РБ)*. — 2013. — № 6. — С. 123–126.
26. *Каннер, А. М.* Linux: о жизненном цикле процессов и разграничении доступа / А. М. Каннер // *Вопросы защиты информации*. — 2014. — № 4. — С. 37–40.
27. *Каннер, А. М.* zLinux: разграничение доступа в мейнфреймах / А. М. Каннер // *Безопасность информационных технологий*. — 2014. — № 4. — С. 27–32.
28. *Каннер, А. М.* Отличительные особенности программно-аппаратных средств защиты информации от несанкционированного доступа для операционных систем GNU/Linux и Windows / А. М. Каннер // *Информация и безопасность*. — 2015. — Т. 18, № 3. — С. 412–415.
29. *Каннер, А. М.* Исследование применимости подсистемы разграничения доступа в операционных системах Linux / А. М. Каннер // *Информация и безопасность*. — 2017. — Т. 20, № 4. — С. 604–609.
30. *Каннер, А. М.* Эффективность внедряемых функций защиты от несанкционированного доступа в операционных системах Linux / А. М. Каннер // *Вопросы защиты информации*. — 2017. — № 2. — С. 3–8.

31. *Каннер, А. М.* Различия подходов к пошаговому контролю целостности: TPM и IMA/EVM или ПАК СЗИ НСД / А. М. Каннер // *Вопросы защиты информации*. — 2018. — № 2. — С. 9–13.
32. *Каннер, А. М.* Требования гарантии выполнения функций защиты встраиваемых средств разграничения доступа к данным в операционных системах / А. М. Каннер // *Вопросы защиты информации*. — 2018. — № 1. — С. 7–12.
33. *Каннер, А. М.* Разграничение доступа в Linux при использовании средства виртуализации kvm / А. М. Каннер // *Вопросы защиты информации*. — 2019. — № 3. — С. 3–7.
34. *Каннер, А. М.* Применение TLA+ нотации для описания модели изолированной программной среды субъектов доступа и ее дальнейшей верификации / А. М. Каннер // *Вопросы защиты информации*. — 2021. — № 3. — С. 8–11.
35. *Каннер, А. М.* Моделирование и верификация подсистемы управления доступом средства защиты информации Аккорд-X / А. М. Каннер, Т. М. Каннер // *Вопросы защиты информации*. — 2020. — № 3. — С. 6–10.
36. *Каннер, А. М.* Особенности доступа к системным функциям ядра ОС GNU/Linux / А. М. Каннер, В. П. Лось // *Вопросы защиты информации*. — 2012. — № 3. — С. 39–44.
37. *Каннер, А. М.* Особенности реализации механизма очистки освобождаемых областей оперативной памяти в GNU/Linux / А. М. Каннер, В. С. Прокопов // *Комплексная защита информации. Материалы XVIII Международной конференции 21-24 мая 2013 года, Брест (РБ)*. — 2013. — № 6. — С. 120–123.
38. *Каннер, А. М.* Управление доступом в ОС GNU/Linux / А. М. Каннер, Л. М. Ухлинов // *Вопросы защиты информации*. — 2012. — № 3. — С. 35–38.
39. *Козачок, А. В.* Спецификация модели управления доступом на языке темпоральной логики действий Лэмпорта / А. В. Козачок // *Труды Института системного программирования РАН*. — 2018. — Т. 30, № 5. — С. 147–162.
40. *Козачок, А. В.* Моделирование и верификация политики безопасности управления доступом для систем электронного документооборота на языке TLA+ / А. В. Козачок // *Методы и технические средства обеспечения безопасности информации*. — 2019. — № 28. — С. 11–13.
41. *Конявский, В. А.* Управление защитой информации на базе СЗИ НСД «Аккорд» / В. А. Конявский. — М. : Радио и связь, 1999. — 325 с.
42. *Конявский, В. А.* Методы и аппаратные средства защиты информационных технологий электронного документооборота : автореф. дис. д-ра техн. наук : 05.13.19 / В. А. Конявский. — М. , 2005. — 47 с.
43. *Конявский, В. А.* Доверенный сеанс связи. Развитие парадигмы доверенных вычислительных систем – на старт, внимание, МАРШ! / В. А. Конявский // *Комплексная защита информации. Материалы XV Международной конференции 1-4 июня 2010 года, Иркутск (РФ)*. — 2010. — С. 166–169.
44. *Лав, Р.* Linux. Системное программирование / Р. Лав. — 2-е изд. — СПб. : Питер, 2014. — 448 с.
45. *Лав, Р.* Ядро Linux. Описание процесса разработки / Р. Лав. — 3-е изд. — М. : Вильямс, 2019.

— 496 с.

46. *Мальцев, А. И.* Алгоритмы и рекурсивные функции / А. И. Мальцев. — М. : Наука, 1986. — 368 с.
47. *Мао, В.* Современная криптография: теория и практика: Пер. с англ. / В. Мао. — М. : Издательский дом «Вильямс», 2005. — 768 с.
48. *Матвейчиков, И. В.* Обзор методов динамического встраивания в ядро операционной системы (на примере Linux) / И. В. Матвейчиков // *Безопасность информационных технологий*. — 2014. — № 4. — С. 75–82.
49. Методический документ ФСТЭК России. Профиль защиты операционных систем типа «А» четвертого класса защиты. ИТ.ОС.А4.ПЗ. — М. : ФСТЭК России, 2017. — 133 с.
50. Моделирование и верификация политик безопасности управления доступом в операционных системах / П. Н. Девянин [и др.]. — М. : Горячая линия - Телеком, 2019. — 212 с.
51. ОС Альт Линукс 8 СП. Руководство администратора ЛКНВ.11100-0190 01 // *Альт Линукс*. — 2020. — 527 с.
52. Программно-аппаратный комплекс средств защиты информации от НСД для ПЭВМ (РС) «Аккорд-АМДЗ». Руководство администратора. 11443195.4012.038 90 // *ОКБ САПР*. — 2019. — 121 с.
53. *Ракицкий, Ю. С.* Модель совмещения двух мандатных политик безопасности / Ю. С. Ракицкий, С. В. Белим // *Безопасность информационных технологий. Материалы XVIII Всероссийской научно-практической конференции «Проблемы информационной безопасности в системе высшей школы»*. — 2011. — № 1. — С. 125–126.
54. *Робачевский, А. М.* Операционная система UNIX / А. М. Робачевский, С. А. Немнюгин, О. Л. Стесик. — СПб. : БХВ-Петербург, 2010. — 656 с.
55. *Родионов, Е. Ю.* Предотвращение несанкционированного запуска ПО: подходы к идентификации приложений / Е. Ю. Родионов, М. В. Сеник // *Безопасность информационных технологий. Материалы XVII Всероссийской научно-практической конференции «Проблемы информационной безопасности в системе высшей школы»*. — 2010. — № 1. — С. 102–105.
56. Российская Федерация. Законы. О лицензировании отдельных видов деятельности, [Федеральный закон от 4 мая 2011 г. № 99-ФЗ]. — М. , 2011. — 26 с.
57. Российская Федерация. Постановления Правительства Российской Федерации. О лицензировании деятельности по разработке и (или) производству средств защиты конфиденциальной информации [№ 171 от 3 марта 2012 г.]. — М. , 2012. — 11 с.
58. Российская Федерация. Приказы. Об утверждении состава и содержания организационных и технических мер по обеспечению безопасности персональных данных при их обработке в информационных системах персональных данных, [Приказ ФСТЭК России № 21: издан ФСТЭК России 18.03.2013]. — М. , 2013. — 20 с.
59. Российская Федерация. Приказы. Об утверждении требований по защите информации, не составляющей государственную тайну, содержащейся в государственных информационных системах, [Приказ ФСТЭК России № 17: издан ФСТЭК России 11.03.2013]. — М. , 2013. — 37 с.

60. Руководящие указания по конструированию прикладного программного обеспечения для операционной системы специального назначения «Astra Linux Special Edition» РУСБ.10015-01 // *НПО РусБИТех*. — 2011. — 63 с.
61. Свидетельство о государственной регистрации программы для ЭВМ. Заявка 2014663519. Российская Федерация. Аккорд-X / Автор Каннер А. М., правообладатель ЗАО «ОКБ САПР». — № 2015612555; заявл. 24.12.2014; опубл. 19.02.2015.
62. *Синякин, С. А.* Особенности совместимости Аккорд-АМДЗ и современных СВТ / С. А. Синякин // *Комплексная защита информации. Материалы XVIII Международной конференции 21-24 мая 2013 года, Брест (РБ)*. — 2013. — № 6. — С. 142–144.
63. Средство защиты информации «Secret Net LSP». Руководство администратора. RU.88338853.501410.017 91 // *Код безопасности*. — 2019. — 130 с.
64. *Стефаров, А. П.* Формирование типовой модели нарушителя правил разграничения доступа в автоматизированных системах / А. П. Стефаров, В. Г. Жуков // *Известия Южного федерального университета. Технические науки*. — 2012. — № 12. — С. 45–54.
65. *Сторожевых, С. Н.* Вопросы обеспечения корректности идентификации пользователя при доступе к ресурсам вычислительной системы. Часть 1. Анализ существующих подходов. Предлагаемое решение / С. Н. Сторожевых, А. Ю. Щеглов // *Вопросы защиты информации*. — 2005. — № 2. — С. 39–43.
66. *Сторожевых, С. Н.* Вопросы обеспечения корректности идентификации пользователя при доступе к ресурсам вычислительной системы. Часть 2. Реализация системы контроля олицетворения / С. Н. Сторожевых, А. Ю. Щеглов // *Вопросы защиты информации*. — 2005. — № 3. — С. 2–5.
67. Теоретические основы компьютерной безопасности: Учебн. пособие для вузов. / П. Н. Девянин [и др.]. — М. : Радио и связь, 2000. — 192 с.
68. *Усов, П. А.* Использование SELinux для защиты информации на уровне приложений / П. А. Усов // *Вопросы защиты информации*. — 2010. — № 3. — С. 50–54.
69. *Ухлинов, Л. М.* Управление безопасностью информации в автоматизированных системах / Л. М. Ухлинов. — М. : МИФИ, 1996. — 112 с.
70. *Хоффман, Л. Дж.* Современные методы защиты информации / Л. Дж. Хоффман. — М. : Советское радио, 1980. — 264 с.
71. *Черемушкин, А. В.* Криптографические протоколы. Основные свойства и уязвимости: Учебн. пособие / А. В. Черемушкин. — М. : Издательский центр «Академия», 2009. — 272 с.
72. *Щеглов, А. Ю.* Защита компьютерной информации от несанкционированного доступа / А. Ю. Щеглов. — СПб. : Наука и техника, 2004. — 384 с.
73. *Щеглов, К. А.* Реализация контроля и разграничения прав доступа по созданию файловых объектов и к системным файловым объектам. [Электронный ресурс] – URL: http://www.itsec.ru/articles2/byauthor/sheglovayu (дата обращения: 25 апреля 2023 г.).
74. *Щеглов, К. А.* Методы идентификации и аутентификации пользователя при доступе к файловым объектам / К. А. Щеглов, А. Ю. Щеглов // *Вестник компьютерных и информационных технологий*. — 2012. — № 10. — С. 47–51.

75. *Щеглов, К. А.* Практическая реализация дискреционного метода контроля доступа к создаваемым файловым объектам / К. А. Щеглов, А. Ю. Щеглов // *Вестник компьютерных и информационных технологий*. — 2013. — № 4. — С. 43–49.
76. *Щеглов, К. А.* Непротиворечивая модель мандатного контроля доступа / К. А. Щеглов, А. Ю. Щеглов // *Известия высших учебных заведений. Приборостроение*. — 2014. — Т. 57, № 4. — С. 12–15.
77. *Щеглов, К. А.* Новый подход к защите данных в информационной системе / К. А. Щеглов, А. Ю. Щеглов // *Известия высших учебных заведений. Приборостроение*. — 2015. — Т. 58, № 3. — С. 157–166.
78. *Щербаков, А. Ю.* Введение в теорию и практику компьютерной безопасности / А. Ю. Щербаков. — М. : Издатель Молгачева С. В., 2001. — 352 с.
79. *Щербаков, А. Ю.* Современная компьютерная безопасность. Теоретические основы. Практические аспекты. Учебное пособие / А. Ю. Щербаков. — М. : Книжный мир, 2009. — 352 с.
80. *Щербаков, А. Ю.* Хрестоматия специалиста по современной информационной безопасности. Том 1. / А. Ю. Щербаков. — Saarbrucken : Palmarium Academic Publishing, 2016. — 272 с.
81. Unix и Linux. Руководство системного администратора / Э. Немет [и др.]. — 5-е изд. — М. : Вильямс, 2020. — 1168 с.
82. *Bell, D. E.* Secure Computer Systems: Unified Exposition and Multics Interpretation / D. E. Bell, L. J. LaPadula. — Bedford, Mass. : MITRE Corp., MTR-2997 Rev. 1, 1976.
83. *Bishop, M.* Computer Security: art and science / M. Bishop. — Addison-Wesley Professional, ISBN 0-201-44099-7, 2002. — 1084 p.
84. Common criteria for information technology security evaluation. — National Security Agency [et al.], 2002.
85. Database Security / S. Castano [et al.]. — Addison Wesley Publishing Company, ACM Press, 1995. — 456 p.
86. *Denning, D.* A Lattice Model of Secure Information Flow / D. Denning // *Communications of the ACM*. — 1976.
87. Filesystem Hierarchy Standard. [Электронный ресурс] – URL: http://www.pathname.com/fhs/ (дата обращения: 25 апреля 2023 г.).
88. *Ford, B.* Multiboot Specification version 0.6.96. [Электронный ресурс] – URL: http://www.gnu.org/software/grub/manual/multiboot/ (дата обращения: 25 апреля 2023 г.).
89. *Frank, J.* Extending the Take-Grant Protection System / J. Frank, M. Bishop // *Department of Computer Science. University of California at Davis*. — 1996. — 14 p.
90. *Harrison, M.* Protection in operating systems / M. Harrison, W. Ruzzo, J. Ullman // *Communications of the ACM*. — 1976. — Vol. 19, no. 8. — Pp. 461–471.
91. IEEE Std 1003.1, 2018 Edition. The Open Group Technical Standard Base Specifications, Issue 7. [Электронный ресурс] – URL: http://pubs.opengroup.org/onlinepubs/9699919799/ (дата обращения: 25 апреля 2023 г.).
92. Intel®64 and IA-32 Architectures Software Developer‘s Manual Volume 3A: System Programming Guide. — 2015.

93. *Kanner, A. M.* Correctness of Data Security Tools for Protection against Unauthorized Access and their Interaction in GNU/Linux / A. M. Kanner // *Global Journal of Pure and Applied Mathematics*. — 2016. — Vol. 12, no. 3. — Pp. 2479–2501.
94. *Kanner, A. M.* Verification of a Model of the Isolated Program Environment of Subjects Using the Lamport's Temporal Logic of Actions / A. M. Kanner, T. M. Kanner // *2020 International Conference on Engineering and Telecommunication (En&T 2020)*. — 2020. — Pp. 1–5.
95. *Kanner, A. M.* Special Features of TLA + Temporal Logic of Actions for Verifying Access Control Policies / A. M. Kanner, T. M. Kanner // *2021 Ural Symposium on Biomedical Engineering, Radioelectronics and Information Technology (USBEREIT 2021)*. — 2021. — Pp. 411–414.
96. *Kanner, A. M.* Verifying Security Properties of the Source Code of Access Control Tools Using Frama-C / A. M. Kanner, T. M. Kanner // *2022 Ural-Siberian Conference on Biomedical Engineering, Radioelectronics and Information Technology (USBEREIT 2022)*. — 2022. — Pp. 255–258.
97. *Lamport, L.* The Temporal Logic of Actions / L. Lamport // *ACM Transactions on Programming Languages and Systems*. — 1994. — Vol. 16, no. 3. — Pp. 872–923.
98. *Lanawehrm, E.* A Security Model for Military Message Systems / E. Lanawehrm, L. Heitmeyer, J. McLean // *ACM Transactions on Computer Systems*. — 1991. — Vol. 9, no. 3. — Pp. 198–222.
99. Linux Security Modules: General Security Support for the Linux Kernel / C. Wright [et al.]. — 2002. — 14 p.
100. Linux Standard Base (LSB). The Linux Foundation. [Электронный ресурс] – URL: http://refspecs.linuxfoundation.org/lsb.shtml (дата обращения: 25 апреля 2023 г.).
101. *Lipton, R.* A Linear Time Algorithm for Deciding Subject Security / R. Lipton, L. Snyder // *Journal of the ACM*. — 1977. — Vol. 24, no. 3. — Pp. 455–464.
102. *McLean, J.* Security Models and Information Flow / J. McLean // *IEEE Press. In Proceeding of the IEEE Symposium on Research in Security and Privacy*. — 1990.
103. *Sandhu, R.* The typed access matrix model / R. Sandhu // *In Proceeding of the IEEE Symposium on Research in Security and Privacy*. — 1992. — Pp. 122–136.
104. *Sandhu, R.* Rationale for the RBAC96 family of access control models / R. Sandhu // *ACM. Proceeding of the* 1*st ACM Workshop on Role-Based Access Control*. — 1997.
105. *Sandhu, R.* Role-Based Access Control, Advanced in Computers / R. Sandhu // *Academic Press*. — 1998. — Vol. 46.
106. *Sandhu, R.* The NIST Model for Role-based Access Control / R. Sandhu, D. F. Ferraiolo, R. Kuhn // *ACM. Proceedings of the* 5*th ACM Workshop on Role-based Access Control*. — 2000. — Pp. 47–63.
107. Security for Linux on System z / L. Parziale [et al.]. — SG24-7728-01 IBM Redbooks, 2013. — 348 p.
108. Specifying and verifying systems with TLA+ / L. Lamport [et al.] // *Proceedings of the 10th workshop on ACM SIGOPS European workshop*. — 2002. — Pp. 45–48.
109. Trusted Computer System Evaluation Criteria. — US Department of Defense, 1985. — DOD 5200.28-STD (supersedes CSC-STD-001-83).
110. The Virtualization Cookbook for IBM z/VM 6.3, RHEL 6.4, and SLES 11 SP3 / L. Parziale [et al.]. — SG24-8147-00 IBM Redbooks, 2014. — 578 p.

Список иллюстративного материала

Список рисунков

Список таблиц

Приложение А

Перечень объектов контроля целостности Linux и правил разграничения доступа к ним для создания изолированной программной среды субъектов

Для обеспечения невозможности отключения функций защиты от НСД необходимо динамически (статически) в процессе работы ОС GNU/Linux или до ее загрузки осуществлять контроль целостности (или обеспечивать неизменность) компонент, перечисленных в Таблице А.1, а также осуществлять разграничение доступа к объектам в соответствии с данными из Таблицы А.2 с помощью любой подходящей политики управления доступом.

Таблица А.1 – Объекты контроля целостности GNU/Linux для создания ИПСС

Объект контроля целостности	**Примечание**
1. Главная загрузочная запись (Master Boot Record, MBR)	Контролировать целостность MBR необходимо для носителя информации, на который устанавливается защищаемая ОС, если в эту область записывается загрузчик (обычно по умолчанию во всех дистрибутивах GNU/Linux)
2. Загрузочный сектор раздела (Partition Boot Record, PBR)	Контролировать целостность PBR необходимо в случае, если в него записывается часть загрузчика ОС GNU/Linux при установке (вместо MBR)
3. Сектора 1-63 относительно начала загрузочного раздела	Контролировать целостность данных секторов необходимо в случае, если часть загрузчика записывается в эту область (например, загрузчик grub записывает в указанные сектора свои компоненты – stage 1.5)
4. Компоненты загрузчика ОС (в зависимости от используемого загрузчика)	Для загрузчика grub это файл /boot/grub/stage2 или /boot/grub/grub и некоторые другие объекты
5. Файлы конфигурации загрузчика ОС (в зависимости от используемого загрузчика)	Для загрузчика grub это /boot/grub/grub.conf или ссылка /boot/grub/menu.lst
6. Ядро Linux	Файлы /boot/vmlinuz-*.*.*-xxx (в различных дистрибутивах GNU/Linux наименование может отличаться)
7. Образ начальной загрузки ОС GNU/Linux (initramfs или initrd)	/boot/initrd-*.*.*-xxx.img (в различных дистрибутивах GNU/Linux наименование может отличаться). При использовании разработанной в данной работе подсистемы управления доступом «Аккорд-X» контроль целостности initrd также обеспечивает неизменность и активизацию ядра защиты СЗИ НСД.

Продолжение таблицы А.1

Объект контроля целостности	**Примечание**
8. Файлы конфигурации и критичные данные ОС	В зависимости от используемого дистрибутива GNU/Linux и набора используемого ПО необходимо контролировать целостность следующих компонент: `/bin/**`[1]; `/boot/**` (дополнительно к объектам выше); `/etc/**`; `/lib/**`[2]; `/lib64/**`[3]; `/sbin/**`; `/usr/bin/**`; `/usr/lib/**`[4]; `/usr/lib64/**`[3]; `/usr/libexec/**`; `/usr/sbin/**`; `/usr/$MACHTYPE/**`[5]; `/usr/local/bin/**`; `/usr/local/lib/**`; `/usr/local/lib64/**`[3]; `/usr/local/sbin/**`. Целостность некоторых описанных компонент необходимо проверять до загрузки ОС (опционально), либо посредством возможности контроля их целостности средствами подсистемы управления доступом (статический или динамический контроль целостности). В число приведенных компонент также включены объекты-источники порождения субъектов-пользователей ОС и связанные с ними объекты, а именно: `/bin/login`, `/bin/su`, `/usr/bin/sudo`, `/usr/sbin/sshd`, `/etc/pam.d/**`, `/lib/security/**` или `/lib64/security/**`, `/etc/passwd`, `/etc/shadow` и некоторые другие, а также бинарные файлы объектов-источников порождения системных сервисов.

[1] Обозначения здесь и далее: «**» – все вложенные файлы и каталоги, «*» – любые символы в имени

[2] `/lib32/**` для 64-разрядных ОС GNU/Linux с поддержкой multilib

[3] для 64-разрядных ОС GNU/Linux

[4] `/usr/lib32/**` для 64-разрядных ОС GNU/Linux с поддержкой multilib

[5] `/usr/i386-linux-gnu/**`, `/usr/x86_64-linux-gnu/**` для 32-/64-разрядных ОС GNU/Linux или аналогичные в зависимости от machine type

Продолжение таблицы А.1

Объект контроля целостности	**Примечание**
9. Настройки используемой подсистемы управления доступом (на примере «Аккорд-X»)	Файлы конфигурации `/etc/accordx/**`, `/usr/lib`[6]`/cups/filter/accord.cnf`. Утилиты администрирования, драйверы и прочие компоненты «Аккорд-X»: `/bin/acx-*`; `/lib`[6]`/acx-core.ko`; `/lib`[6]`/modules/$(uname-r)`[7]`/kernel/drivers/pci/accord-le.ko`; `/lib`[6]`/modules/$(uname-r)`[7]`/kernel/drivers/pci/tmdevice.ko`; `/lib`[6]`/modules/$(uname-r)`[7]`/kernel/drivers/usb/serial/shipka.ko`; `/lib`[6]`/modules/$(uname-r)`[7]`/kernel/drivers/usb/serial/shipka_kc2.ko`; `/lib`[6]`/modules/$(uname-r)`[7]`/kernel/drivers/usb/serial/shipka_kc3.ko`; `/lib`[6]`/modules/$(uname-r)`[7]`/kernel/drivers/usb/serial/tmusb_drv.ko`; `/lib`[6]`/security/pam_acx*`; `/usr/bin/acx-admin*`; `/usr/bin/acx-config`; `/usr/bin/acx-remote*`; `/usr/lib`[6]`/libacx-*`; `/usr/lib`[6]`/libosci*`; `/usr/lib`[6]`/libtmid*`; `/usr/lib`[6]`/tmid-*`; `/usr/lib`[6]`/cups/filter/pstops`.

Таблица А.2 – Пример списка прав непривилегированного пользователя GNU/Linux в запретительной политике управления доступом для создания ИПСС (при использовании подсистемы управления доступом «Аккорд-X»)

Объект контроля доступа	**Права доступа**
1. `/**`	RXCNLMnG[8]
2. `/bin/acx-*`	–

[6] `lib64` для 64-разрядных ОС GNU/Linux

[7] строка с версией ядра Linux

[8] Обозначения атрибутов доступа: **R/W/X** – чтение, запись, исполнение; **C/D/N** – создание, удаление, переименование; **L** – создание ссылки; **M/E/n** – создание, удаление, переименование каталога; **G** – переход в каталог.

Продолжение таблицы A.2

Объект контроля доступа	**Права доступа**
3. /bin/acx-integrity-controller	RX
4. /bin/acx-integrity-controller-db	RX
5. /boot/**	–
6. /dev/null	RW
7. /etc/	RCNL
8. /etc/mtab*	RWCD
9. /etc/accordx/**	–
10. /etc/accordx/db.json	R
11. /home/$USER[9]/**	RWXCDNLMEnG
12. /lib[10]/acx-core.ko	–
13. /lib[10]/modules/$(uname-r)[11]/kernel/drivers/usb/serial/shipka*	–
14. /lib[10]/modules/$(uname-r)[11] /kernel/drivers/usb/serial/tmusb_drv.ko	–
15. /lib[10]/security/pam_acx*	R
16. /run/**	RWXCDNLMEnG
17. /tmp/**	RWXCDNLMEnG
18. /usr/bin/acx-admin*	–
19. /usr/bin/acx-config*	–
20. /usr/bin/acx-admin	RX
21. /usr/bin/acx-admin-log	RX
22. /usr/lib[10]/cups/filter/accord*	R
23. /usr/lib[10]/cups/filter/accord.users/**	RG
24. /usr/lib[10]/cups/filter/pstops	RX
25. /usr/lib[10]/libacx-core*	–
26. /usr/lib[10]/libacx-db*	R
27. /usr/lib[10]/libacx-log*	R
28. /usr/lib[10]/libacx-print*	R
29. /usr/lib[10]/libosci*	R
30. /usr/lib[10]/libtmid*	R
31. /usr/lib[10]/tmid*	R
32. /var/log/accordx/**	RG

[9] имя пользователя
[10] lib64 для 64-разрядных ОС GNU/Linux
[11] строка с версией ядра Linux

Приложение Б

Листинги разработанной подсистемы управления доступом к данным

```

#if LINUX_VERSION_CODE >= KERNEL_VERSION(2, 5, 0)
#if defined(CONFIG_X86)

#ifdef CONFIG_X86_32
#define START_MEM   0xc0000000 //32bit kernel space
#define END_MEM     0xd0000000
typedef unsigned int address;
#else // CONFIG_X86_32
#define START_MEM 0xffffffff80000000 //64bit kernel space
#define END_MEM 0xffffffffffffffff
typedef unsigned long long int address;
#endif // CONFIG_X86_64

// address typedef from above for 32/64 compat
address *sys_call_table;
// function for searching syscall table
address **find_sys_call_table_ptr(void) {
  address **sctable;
  address i = START_MEM;
  while ( i < END_MEM ) { //bruteforce through address space
    sctable = (address **)i;
    if ( sctable[__NR_close] == (address *) sys_close)
      return &sctable[0];
    i += sizeof(void *);
  }
  return NULL;
}

#endif // CONFIG_X86
#else // LINUX_VERSION_CODE < KERNEL_VERSION(2, 5, 0)
void *sys_call_table[];
#endif
```

Листинг Б.1 – Функция поиска адреса таблицы системных вызовов для 32- и 64-разрядных ОС GNU/Linux

```
static int set_memory_protection(void *addr, unsigned long prot_flags) {
  struct page *pg;
  pgprot_t prot;
```

```
    int res;

    pg = virt_to_page((unsigned long)addr);
    prot.pgprot = prot_flags;
    res = change_page_attr(pg, 1, prot);
    global_flush_tlb();
    return res;
}
```

Листинг Б.2 – Функция установки доступа к страницам памяти

```
#define GPF_DISABLE write_cr0(read_cr0() & (~ 0x10000))
#define GPF_ENABLE write_cr0(read_cr0() | 0x10000)

static asmlinkage int (* orig_mkdir)(struct inode *dir, struct dentry *
 dentry, int mode);

static asmlinkage int hook_mkdir (struct inode *dir, struct dentry *
 dentry, int mode) {
  printk (KERN_ERR "mkdir hijacked!\n");
  return orig_mkdir(dir, dentry, mode);
}

static int patch_sys_call_table(void) {
  #if defined(CONFIG_X86)
  #if LINUX_VERSION_CODE >= KERNEL_VERSION(2, 5, 0)
  GPF_DISABLE;
  #endif

  orig_mkdir = (void *) sys_call_table[__NR_mkdir];
  sys_call_table[__NR_mkdir] = (address) ((void *) hook_mkdir);

  #if LINUX_VERSION_CODE >= KERNEL_VERSION(2, 5, 0)
  GPF_ENABLE;
  #endif
  #endif

  return 0;
}

static int revert_sys_call_table(void) {
  #if defined(CONFIG_X86)
  #if LINUX_VERSION_CODE >= KERNEL_VERSION(2, 5, 0)
  GPF_DISABLE;
```

```
32    #endif
33
34    sys_call_table[__NR_mkdir] = (address) ((void *) orig_mkdir);
35
36    #if LINUX_VERSION_CODE >= KERNEL_VERSION(2, 5, 0)
37    GPF_ENABLE;
38    #endif
39    #endif
40
41    return 0;
42  }
```

Листинг Б.3 – Функции замены обработчиков и возврата изменений оригинального состояния таблицы системных вызовов

```
1   static int inode_mkdir(struct inode *dir, struct dentry *dentry, int mode
     ) {
2     printk (KERN_ERR "mkdir hijacked!\n");
3     return 0;
4   }
5
6   static struct security operations hook_security_ops = {
7     .inode_mkdir = inode_mkdir,
8   };
9
10  int hook_register() {
11    int res = register_security(&hook_security_ops);
12    if (res) {
13      printk(KERN_ERR "error: failed to register lsm: %d\n", res);
14      return res;
15    }
16    return 0;
17  }
18
19  void hook_unregister() {
20    int res = unregister_security(&hook_security_ops);
21    if (res)
22      printk(KERN_ERR "error: failed to unregister lsm: %d\n", res);
23  }
```

Листинг Б.4 – Функции регистрации LSM-обработчика Linux для системного вызова *mkdir()*

Приложение В

Листинги спецификации для системы ИПСС на языке темпоральной логики действий

Полный текст описания системы ИПСС на языке TLA+ приведен в Листингах В.1–В.6.

Конфигурация системы ИПСС в TLA+ нотации имеет вид, представленный в Листинге В.1

```
SPECIFICATION Spec
INVARIANTS    TypeInv
              ConsistencyInv
              BlockedInv
              OSKernelExists
              SormInits
              Correctness
              AbsCorrectnessOpp
PROPERTIES    AbsCorrectness
              OSUsabilityLiveness
```

Листинг В.1 – Конфигурация системы ИПСС

Типы сущностей системы представлены в Листинге В.2. Некоторые значения являются модельными и введены для сокращения времени верификации системы.

```
---------------------------- MODULE types ----------------------------
EXTENDS Integers, FiniteSets
----------------------------------------------------------------------

\* Множества идентификаторов
SubjectIDs  == 0..4
ObjectIDs   == 0..12

\* Модельные значения состояний объектов
\* 0 — объект создан или изменялся субъектом с sid = 0 (системой)
\* 1 — объект создан или изменялся субъектом с sid = 1
\* ...
\* 4 — объект изменялся другим субъектом (целостность нарушена)
ObjectStates == 0..Cardinality(SubjectIDs)
StateChanged == Cardinality(ObjectStates) - 1

\* Типы сущностей системы
SubjTypes == {"users","system","sorm"}
ObjTypes == {"func","data","na"}

\* Типы запросов к системе
QueryTypes == {"change_blocked", "initial", \* только в начальном состоянии
```

```
                "create_process", "delete_process",
                "create_user", "create_shadow", "delete_subject",
                "exec",
                "read", "write", "create", "delete"}

QueriesSystem == {"change_blocked", "initial"}
QueriesStateChange == {"write", "create", "delete",
                       "create_process", "delete_process",
                       "delete_subject"}
QueriesAssocChange == {"create_user", "create_shadow",
                       "exec", "read"}


-----------------------------------------------------------------------------


\* Субъекты доступа
Subjects == [sid: SubjectIDs,
             type: SubjTypes,
             \* заблокирован, не зарегистрирован
             is_blocked: BOOLEAN]

\* Объекты доступа
Objects  == [oid: ObjectIDs,
             type: ObjTypes,
             \* субъект, с которым ассоциирован
             subj_assoc: SUBSET SubjectIDs,
             \* o_t — состояние объекта
             state: ObjectStates]

\* Запросы к системе
Queries  == [subj: Subjects,
             proc: Objects,
             dent: Objects \cup Subjects,
             type: QueryTypes]

\* Сущности системы: субъекты и объекты
Entities == Subjects \cup Objects
```

Листинг B.2 – Модуль types.tla с описанием модельных значений и сущностей системы

Начальное состояние системы и переменные модели, изменяемые при переходах системы из состояния в состояние, описаны в Листинге B.3.

```
------------------------------ MODULE init ------------------------------
EXTENDS TLC, Integers, types
-------------------------------------------------------------------------
```

```
4
5  VARIABLES S_active, O_func, O_data, O_na, S, Q
6
7  \* Переменные модели:
8  \*
9  \* S — множество субъектов модели ИПСС
10 \*
11 \* V = S_active x O_func x O_data x O_na
12 \*   — множество всевозможных состояний модели, где:
13 \* S_active — множество активных субъектов системы
14 \* O_func   — множество функционально ассоциированных с субъектами объектов
15 \* O_data   — множество ассоциированных с субъектами объектов—данных
16 \* O_na     — множество неассоциированных с субъектами объектов
17 \*
18 \* Q = S x O x E
19 \*   — множество всевозможных запросов к системе
20 vars == <<S_active, O_func, O_data, O_na, S, Q>>
21
22 \* Модельные значение
23
24 \* Зарегистрированные субъекты системы
25 \* s_0 — первоначальный системный субъект (ядро ОС)
26 s_0 ==      [sid        |-> 0,
27              type       |-> "system",
28              is_blocked |-> FALSE]
29
30 \* s_sorm — подсистема управления доступом
31 s_sorm ==   [sid        |-> 1,
32              type       |-> "sorm",
33              is_blocked |-> FALSE]
34
35 \* пользователь 1
36 s_2 ==      [sid        |-> 2,
37              type       |-> "users",
38              is_blocked |-> FALSE]
39
40 \* пользователь 2
41 s_3 ==      [sid        |-> 3,
42              type       |-> "users",
43              is_blocked |-> FALSE]
44
45 \* системный субъект
46 s_4==       [sid        |-> 4,
```

```
                type        |-> "system",
                is_blocked  |-> FALSE]

\* Начальные объекты системы
\* o_0 — процесс системного субъекта s_0
o_0 ==          [oid        |-> 0,
                 type       |-> "func",
                 subj_assoc |-> {0},
                 state      |-> 0]

\* o_sorm — ассоциированный объект-данные s_sorm модели ИПСС
o_sorm ==       [oid        |-> 1,
                 type       |-> "na",
                 subj_assoc |-> {},
                 state      |-> 1]

\* o_2 — объект-данные s_0 (корневой каталог)
o_2 ==          [oid        |-> 2,
                 type       |-> "na",
                 subj_assoc |-> {},
                 state      |-> 0]

\* инициализирующий запрос
q_0 ==          [subj        |-> s_0,
                 proc        |-> o_0,
                 dent        |-> o_0,
                 type        |-> "initial"]
```

Листинг B.3 – Модуль init.tla с описанием начального состояния системы и переменных

Спецификация системы ИПСС, всевозможные операции системы, а также инварианты и темпоральные свойства описаны в Листинге B.4.

```
---------------------------- MODULE ipes ----------------------------
EXTENDS init, types, select, sorm
---------------------------------------------------------------------

\* CreateProcessD
\* Создание функционально ассоциированного объекта
CreateProcess(s,o,o_c) ==
        /\ O_func' = O_func \cup {o_c}
        /\ Q' = Append(Q, [subj |-> s, proc |-> o,
                           dent |-> o_c,
                           type |-> "create_process"])
        /\ UNCHANGED <<S_active, O_data, O_na, S>>
```

```
13
14 CreateProcessD ==
15         \* Активизирующее воздействие субъекта и его процесса
16         \E s \in S_active:
17         \E o \in SelectSubjProc(s):
18         \E st \in ObjectStates:
19
20         \E x \in ObjectIDs:
21
22             \* обозначим новый объект
23             LET o_c == [oid |-> x,
24                         type |-> "func",
25                         subj_assoc |-> {s.sid},
26                         state |-> st] IN
27
28             /\  IF o.state # s.sid
29                 THEN st = StateChanged
30                 \* форк наследует состояние
31                 ELSE st = o.state
32
33             \* Выбор свободного идентификатора
34             /\ \A obj \in SelectObjects: obj.oid # o_c.oid
35
36             \* Правила доступа s_sorm
37             /\ SormCheckPerm(s,o_c,"create_process")
38
39             \* Постусловия
40             /\ CreateProcess(s,o,o_c)
41
42 \* DeleteProcessD
43 \* Уничтожение функционально ассоциированного объекта
44 DeleteProcess(s,o) ==
45         /\ O_func' = O_func \ {o}
46         \* оставим техническую возможность удаления чужих процессов
47         \* (будет запрещено свойством Correctness)
48         /\  \/  /\ o.state # s.sid
49                 /\ Q' = Append(Q, [subj |-> s, proc |-> o,
50                                    \* удалили чужой объект
51                                    dent |-> [ o EXCEPT![ "state"]=
52                                                 StateChanged],
53                                    type |-> "delete_process"])
54             \/  /\ o.state = s.sid
55                 /\ Q' = Append(Q, [subj |-> s, proc |-> o,
```

```
                                    \* удалили свой объект
                                    dent |-> o,
                                    type |-> "delete_process"])
        /\ UNCHANGED <<S_active, O_data, O_na, S>>

DeleteProcessD ==
        \E s \in S_active:
        \E o \in SelectProc:

            \* Нельзя удалять последний процесс -> DeleteSubjectD
            /\ Cardinality(SelectSubjProc(s)) > 1

            \* Правила доступа s_sorm
            /\ SormCheckPerm(s,o,"delete_process")

            \* Постусловия
            /\ DeleteProcess(s,o)

\* CreateUserD
\* Порождения субъекта-пользователя из объекта
CreateUser(s,o,s_u) ==
        /\ S_active' = S_active \cup {s_u}
        /\ O_func' = (O_func \ {o}) \cup
                                {[oid |-> o.oid,
                                  type |-> o.type,
                                  subj_assoc |-> {s_u.sid},
                                  state |-> s_u.sid]}
        /\ Q' = Append(Q, [subj |-> s,
                           proc |-> (CHOOSE f \in O_func': f.oid = o.oid),
                           dent |-> s_u,
                           type |-> "create_user"])
        /\ UNCHANGED <<O_data, O_na, S>>

CreateUserD ==
        \E s \in S_active:
        \E o \in SelectSubjProc(s):
        \* оставим техническую возможность мультисессий и каскадных
        \* сессий (будет запрещено s_sorm)
        \E s_u \in S:

            \* Новый субъект типа "users"
            /\ s_u.type = "users"

```

```
99              \* Правила порождения s_sorm
100             /\ SormCheckSubj(o, s_u, "create_user")
101             /\ SormCheckPerm(s_u, o, "create_user")
102
103             \* Нельзя порождать из последнего процесса
104             /\ Cardinality(SelectSubjProc(s)) > 1
105
106             \* Постусловия
107             /\ CreateUser(s,o,s_u)
108
109 \* CreateShadowD
110 \* Порождения системного субъекта из объекта
111 CreateShadow(s,o,s_w) ==
112         /\ S_active' = S_active \cup {s_w}
113         /\ O_func' = (O_func \ {o}) \cup
114                                 {[oid |-> o.oid,
115                                   type |-> o.type,
116                                   subj_assoc |-> {s_w.sid},
117                                   state |-> s_w.sid]}
118         /\ Q' = Append(Q, [subj |-> s,
119                           proc |-> (CHOOSE f \in O_func': f.oid = o.oid),
120                           dent |-> s_w,
121                           type |-> "create_shadow"])
122         /\ UNCHANGED <<O_data, O_na, S>>
123
124 CreateShadowD ==
125         \E s \in S_active:
126         \E o \in SelectSubjProc(s):
127         \* оставим техническую возможность мультисессий и каскадных
128         \* сессий (будет запрещено s_sorm)
129         \E s_w \in S:
130
131             \* Новый субъект типа "system" или "sorm"
132             /\ s_w.type \in {"system", "sorm"}
133
134             \* Правила порождения s_sorm
135             /\ SormCheckSubj(o, s_w, "create_shadow")
136             /\ SormCheckPerm(s_w, o, "create_shadow")
137
138             \* Нельзя порождать из последнего процесса
139             /\ Cardinality(SelectSubjProc(s)) > 1
140
141             \* Постусловия
```

```
            /\ CreateShadow(s,o,s_w)

\* DeleteSubjectD
\* Уничтожение всех функционально ассоциированных объектов
DeleteSubject(s,o) ==
        \* Для всех процессов выполняется DeleteProcess()
        /\ \A proc \in SelectSubjProc(s):
            /\ O_func' = O_func \ {proc}
            \* Объекты-данные перестают быть ассоциированными
        /\  \/  /\ Cardinality(SelectSubjData(s)) > 0
                /\ \A d \in SelectSubjData(s):
                        \* Неассоциированный объект должен перейти в O_na
                    /\  \/  /\ Cardinality(d.subj_assoc) = 1
                            /\ O_data' = O_data \ {d}
                            /\ O_na' = O_na \cup
                                {[oid |-> d.oid,
                                  type |-> "na",
                                  subj_assoc |-> {},
                                  state |-> d.state]}
                        \* Из ассоциированных объектов исключается s.sid
                        \/  /\ Cardinality(d.subj_assoc) > 1
                            /\ O_data' = (O_data \ {d}) \cup
                                {[ d EXCEPT!["subj_assoc"]=
                                    (d.subj_assoc \ {s.sid})]}
                            /\ O_na' = O_na
            \/  /\ Cardinality(SelectSubjData(s)) = 0
                /\ O_data' = O_data
                /\ O_na' = O_na
        /\ S_active' = S_active \ {s}
        /\ Q' = Append(Q, [subj |-> s, proc |-> o,
                           dent |-> s,
                           type |-> "delete_subject"])
        /\ UNCHANGED <<S>>

DeleteSubjectD ==
        \E s \in S_active:
        \E o \in SelectSubjProc(s):
        \E s_d \in S_active:

            \* Правила удаления s_sorm
            /\ SormCheckSubj(o, s_d, "delete_subject")
            /\ SormCheckPerm(s_d, o, "delete_subject")

```

```
185             \* Постусловия
186             /\ DeleteSubject(s_d,o)
187
188 \* ExecD
189 \* Загрузка бинарного образа для выполнения
190 Exec(s,o,o_e) ==
191         \* Бинарный файл загружается в процесс
192         /\ IF o_e.state # s_0.sid
193            THEN O_func' = (O_func \ {o}) \cup
194                     {[ o EXCEPT![ "state"]=
195                         o_e.state]}
196            ELSE O_func' = O_func
197             \* Объект перестает быть ассоциированным и переходит в O_na
198         /\  \/  /\ Cardinality(o_e.subj_assoc) = 1
199                 /\ O_data' = O_data \ {o_e}
200                 /\ O_na' = O_na \cup
201                     {[oid |-> o_e.oid,
202                       type |-> "na",
203                       subj_assoc |-> {},
204                       state |-> o_e.state]}
205             \* Из ассоциированного объекта исключается s.sid
206             \/  /\ Cardinality(o_e.subj_assoc) > 1
207                 /\ O_data' = (O_data \ {o_e}) \cup
208                     {[ o_e EXCEPT![ "subj_assoc"]=
209                         (o_e.subj_assoc \ {s.sid})]}
210                 /\ O_na' = O_na
211         /\ Q' = Append(Q, [subj |-> s,
212                            proc |-> (CHOOSE f \in O_func': f.oid = o.oid),
213                            dent |-> (CHOOSE d \in O_data' \cup O_na':
214                                              d.oid = o_e.oid),
215                            type |-> "exec"])
216         /\ UNCHANGED <<S_active, S>>
217
218 ExecD ==
219         \E s \in S_active:
220         \E o \in SelectSubjProc(s):
221         \* Объект уже прочитан
222         \E o_e \in SelectSubjData(s):
223
224             \* Правила доступа s_sorm
225             /\ SormCheckPerm(s,o_e,"exec")
226
227             \* Постусловия
```

```
            /\ Exec(s,o,o_e)

\* ReadD
\* Реализация информационного потока на чтение
Read(s,o,o_r) ==
        \* Процесс читает данные и изменяет свое состояние
        /\ IF o_r.state # s_0.sid
           THEN O_func' = (O_func \ {o}) \cup
                    {[ o EXCEPT![\"state\"]=
                        o_r.state]}
           ELSE O_func' = O_func
        \* Объект с данными становится ассоциированным
        /\ O_data' = (O_data \ {o_r}) \cup
            {[oid |-> o_r.oid,
              type |-> \"data\",
              subj_assoc |-> (o_r.subj_assoc \cup {s.sid}),
              state |-> o_r.state]}
        /\ O_na' = O_na \ {o_r}
        /\ Q' = Append(Q, [subj |-> s,
                           proc |-> (CHOOSE f \in O_func': f.oid = o.oid),
                          dent |-> (CHOOSE d \in O_data': d.oid = o_r.oid),
                           type |-> \"read\"])
        /\ UNCHANGED <<S_active, S>>

ReadD ==
        \E s \in S_active:
        \E o \in SelectSubjProc(s):
        \E o_r \in SelectObjects \ SelectProc:

            \* Правила доступа s_sorm
            /\ SormCheckPerm(s,o_r,\"read\")

            \* Постусловия
            /\ Read(s,o,o_r)

\* WriteD
\* Реализация информационного потока на запись
Write(s,o,o_w) ==
        \* изменяем состояние в зависимости от того, чей объект
        /\  \/  /\ o_w.state # s.sid
                /\  \/  /\ o_w \in O_na
                        /\ O_na' = (O_na \ {o_w}) \cup
                            \* состояние изменено
```

```
                                {[ o_w EXCEPT![\"state\"]= StateChanged]}
                            /\ O_data' = O_data
                        \/  /\ o_w \in O_data
                            /\ O_data' = (O_data \ {o_w}) \cup
                                \* состояние изменено
                                {[ o_w EXCEPT![\"state\"]= StateChanged]}
                            /\ O_na' = O_na
                \* состояние задано субъектом s
                \/  /\ o_w.state = s.sid
                    /\ O_data' = O_data
                    /\ O_na' = O_na
        /\ Q' = Append(Q, [subj |-> s, proc |-> o,
                           dent |-> (CHOOSE d \in O_data' \cup O_na':
                                              d.oid = o_w.oid),
                           type |-> \"write\"])
        /\ UNCHANGED <<S_active, O_func, S>>

WriteD ==
        \E s \in S_active:
        \E o \in SelectSubjProc(s):
        \E o_w \in SelectObjects \ SelectProc:

            \* Правила доступа s_sorm
            /\ SormCheckPerm(s,o_w,\"write\")

            \* Постусловия
            /\ Write(s,o,o_w)

\* CreateD
\* Реализация информационного потока на создание объекта
Create(s,o,o_c) ==
        /\ O_na' = O_na \cup {o_c}
        /\ Q' = Append(Q, [subj |-> s, proc |-> o,
                           dent |-> o_c,
                           type |-> \"create\"])
        /\ UNCHANGED <<S_active, O_func, O_data, S>>

CreateD ==
        \E s \in S_active:
        \E o \in SelectSubjProc(s):

        \E x \in ObjectIDs:

```

```
314             \* обозначим новый объект
315             LET o_c == [oid |-> x,
316                         type |-> "na",
317                         subj_assoc |-> {},
318                         \* начальное состояние задано s
319                         state |-> s.sid] IN
320
321             \* Выбор свободного идентификатора
322             /\ \A obj \in SelectObjects: obj.oid # o_c.oid
323
324             \* Правила доступа s_sorm
325             /\ SormCheckPerm(s,o_c,"create")
326
327             \* Постусловия
328             /\ Create(s,o,o_c)
329
330 \* DeleteD
331 \* Реализация информационного потока на удаление объекта
332 Delete(s,o,o_d) ==
333         /\ O_na' = O_na \ {o_d}
334         /\ O_data' = O_data \ {o_d}
335         \* изменяем состояние в зависимости от того, чей объект
336         /\  \/  /\ o_d.state # s.sid
337                 /\ Q' = Append(Q, [subj |-> s, proc |-> o,
338                                    \* удалили чужой объект
339                                    dent |-> [ o_d EXCEPT!["state"]=
340                                                 StateChanged],
341                                    type |-> "delete"])
342             \/  /\ o_d.state = s.sid
343                 /\ Q' = Append(Q, [subj |-> s, proc |-> o,
344                                    \* удалили свой объект
345                                    dent |-> o_d,
346                                    type |-> "delete"])
347         /\ UNCHANGED <<S_active, O_func, S>>
348
349 DeleteD ==
350         \E s \in S_active:
351         \E o \in SelectSubjProc(s):
352         \E o_d \in SelectObjects \ SelectProc:
353
354             \* Правила доступа s_sorm
355             /\ SormCheckPerm(s,o_d,"delete")
356
```

```
357             \* Постусловия
358             /\ Delete(s,o,o_d)
359
360 \* SormBlockSubjectD
361 \* Изменение блокировки субъекта (разрешенный / неразрешенный)
362 SormBlockSubject(s,o,s_b) ==
363         /\ S' = (S \ {s_b}) \cup
364             {[ s_b EXCEPT![“is_blocked”]= (\neg s_b.is_blocked) ]}
365         /\ Q' = Append(Q, [subj |-> s, proc |-> o,
366                           dent |-> (CHOOSE su \in S': su.sid = s_b.sid),
367                           type |-> “change_blocked”])
368         /\ UNCHANGED <<S_active, O_func, O_data, O_na>>
369
370 SormBlockSubjectD ==
371         \E s \in S_active:
372         \E o \in SelectSubjProc(s):
373         \* для уменьшения возможных состояний блокировать можно
374         \* только неактивного пользователя
375         \E s_b \in S \ S_active:
376
377             \* Правила s_sorm
378             /\ SormCheckSubj(o, s_b, “change_blocked”)
379
380             \* Постусловия
381             /\ SormBlockSubject(s,o,s_b)
382
383 ----------------------------------------------------------------------------
384
385 \* Type Invariant
386 \* Инвариант типов
387 TypeInv ==  /\ S_active \subseteq Subjects
388             /\ O_func \subseteq Objects
389             /\ O_data \subseteq Objects
390             /\ O_na \subseteq Objects
391             /\ S \subseteq Subjects
392             /\ SelectPrevQuery(Q) \in Queries
393
394 \* Consistency Invariant
395 \* Инвариант консистентности множеств сущностей системы
396 ConsistencyInv ==
397     /\ \A proc \in O_func:
398         \* объект м.б. функционально ассоциирован только с одним субъектом
399         /\ Cardinality(proc.subj_assoc) = 1
```

```
400         /\ \E subj \in S_active: proc.subj_assoc = {subj.sid}
401         \* объект не может состоять в другом множестве
402         /\ proc \notin O_data \cup O_na
403         \* state должен соответствовать субъекту
404         /\ proc.state \in proc.subj_assoc
405     /\ \A data \in O_data:
406         \* объект м.б. ассоциирован как данные с несколькими субъектами
407         /\ Cardinality(data.subj_assoc) >= 1
408         /\ \E subj \in S_active: subj.sid \in data.subj_assoc
409         /\ data \notin O_func \cup O_na
410     /\ \A obj \in O_na:
411         \* объект м.б. не ассоциирован ни с одним субъектом
412         /\ Cardinality(obj.subj_assoc) = 0
413         /\ obj \notin O_func \cup O_data
414     \* уникальность идентификаторов субъектов и объектов
415     /\ \A sid \in SubjectIDs:
416         /\ Cardinality({s \in S: s.sid = sid}) <= 1
417     /\ \A oid \in ObjectIDs:
418         /\ Cardinality({o \in SelectObjects: o.oid = oid}) <= 1
419
420 \* Blocked Invariant
421 \* Неразрешенные субъекты не могут быть активными и
422 \* иметь ассоциированные объекты
423 BlockedInv ==
424     \A s \in S:
425         /\  \/  /\ s.is_blocked = TRUE
426                 /\ s \notin S_active
427                 /\ ~\E o \in SelectObjects: s.sid \in o.subj_assoc
428             \/ s.is_blocked = FALSE
429
430 \* OSKernelExists
431 \* В любой момент времени существует s_0
432 OSKernelExists ==
433     /\ s_0 \in S_active
434     /\ s_0.is_blocked = FALSE
435
436 \* SormInits
437 \* В начальный момент времени инициализирован s_sorm
438 \* либо функционирует только s_0
439 SormInits ==
440     /\  \/  /\ s_sorm \in S_active
441             /\ s_sorm.is_blocked = FALSE
442         \/  /\ s_sorm \notin S_active
```

```
443             /\ S_active = {s_0}
444
445 \* Correctness
446 \* Свойство корректности субъектов при переходе системы
447 Correctness ==
448 LET ent == CHOOSE e \in Entities: e = SelectPrevQueryDent(Q)
449     subj == CHOOSE s \in Subjects: s = SelectPrevQuerySubj(Q)
450     proc == CHOOSE p \in Objects: p = SelectPrevQueryProc(Q)
451     r == CHOOSE q \in QueryTypes: q = SelectPrevQueryType(Q) IN
452     \* Нельзя изменять состояние ассоциированных объектов другого субъекта
453     IF  r \in QueriesStateChange
454     THEN
455         \* "write", "delete", ...
456         IF ent \in Objects /\ ent.type \in {"func", "data"}
457         THEN ent.subj_assoc \subseteq {subj.sid}
458         ELSE IF ent \in Subjects \* delete_subject
459              THEN proc.subj_assoc \subseteq {ent.sid}
460              ELSE TRUE
461     ELSE IF (r \in QueriesAssocChange
462             /\ ent \in Subjects) \* create_user, create_shadow
463     THEN    proc.state \in {ent.sid, s_0.sid}
464     ELSE
465         TRUE
466
467 \* Вспомогательные предикаты для свойств корректности
468 EntityStateChanged(subj, proc, ent) ==
469     IF ent \in Objects
470     \* в прошлом был "write", "delete", ... чужих объектов
471     THEN ent.state \notin {subj.sid, s_0.sid} \* = StateChanged
472     ELSE IF ent \in Subjects
473     \* в прошлом был "create_process", ... чужих объектов
474     THEN proc.state = StateChanged
475     ELSE FALSE
476
477 EntityStateChanging(ent) ==
478     \* "create_user", "create_shadow", "exec", "read"
479     /\ SelectPrevQueryType(Q) \in QueriesAssocChange
480
481 \* AbsCorrectnessOpp
482 \* Свойство абсолютной корректности субъектов в обратном смысле
483 AbsCorrectnessOpp ==
484     LET ent == CHOOSE e \in Entities: e = SelectPrevQueryDent(Q)
485         subj == CHOOSE s \in Subjects: s = SelectPrevQuerySubj(Q)
```

```
486         proc == CHOOSE p \in Objects: p = SelectPrevQueryProc(Q) IN
487     \* Ассоциированным объектом не может стать измененный ранее объект
488     IF  EntityStateChanging(ent)
489     THEN
490         IF      EntityStateChanged(subj, proc, ent)
491                 \* была нарушена целостность
492         THEN    FALSE
493         ELSE    TRUE
494     ELSE TRUE
495
496 \* AbsCorrectness
497 \* Свойство абсолютной корректности субъектов
498 AbsCorrectness ==
499     LET ent == CHOOSE e \in Entities: e = SelectPrevQueryDent(Q)
500         subj == CHOOSE s \in Subjects: s = SelectPrevQuerySubj(Q)
501         proc == CHOOSE p \in Objects: p = SelectPrevQueryProc(Q) IN
502     \* если объект измененен
503     (EntityStateChanged(subj, proc, ent))
504     \* объект в будущем не станет ассоциированным с другим субъектом
505         ~>  <>[] (IF    SelectPrevQueryDent(Q) = ent
506                   THEN  ~EntityStateChanging(ent)
507                   ELSE  TRUE)
508
509 \* OSUsabilityLiveness
510 \* Свойство возможности использования ОС
511 OSUsabilityLiveness ==
512     \* хотя бы в одном состоянии есть субъекты кроме
513     \* s_0 и s_sorm: пользователь или системный субъект
514     <> (Cardinality(S_active) > 2)
515
516 ----------------------------------------------------------------------
517
518 \* Init
519 \* Инициализация модели
520 Init == /\ S_active = {s_0}
521         \* изначально существует только s_0 и его процесс o_0
522         /\ O_func = {o_0}
523         /\ O_data = {}
524         /\ O_na = {o_sorm, o_2}
525         \* остальные субъекты еще не активизированы
526         /\ S = {s_0, s_sorm, s_2, s_3, s_4}
527         /\ Q = <<q_0>>
528
```

```
529 \* Next
530 \* Возможные дальнейшие действия модели (запросы к системе)
531 Next ==
532         \* Запросы к системе
533         \/ CreateProcessD
534         \/ DeleteProcessD
535         \/ CreateUserD
536         \/ CreateShadowD
537         \/ DeleteSubjectD
538         \/ ExecD
539         \/ ReadD
540         \/ WriteD
541         \/ CreateD
542         \/ DeleteD
543         \* Административные действия
544         \/ SormBlockSubjectD
545
546 \* TemporalAssumption
547 \* Темпоральное liveness-свойство
548 TemporalAssumption ==
549     \* если действие доступно - нужно его выполнить
550     \* (иначе система останется в состоянии stuttering)
551     /\ WF_vars(Next)
552
553 \* Spec
554 \* Спецификация модели
555 Spec == Init /\ [][Next]_vars /\ TemporalAssumption
556
557 \* Invariants and Temporal Properties
558 \* Теорема, учитывающая инварианты и свойства: доказывается при верификации
559 THEOREM Spec => /\ []TypeInv
560                 /\ []ConsistencyInv
561                 /\ []BlockedInv
562                 /\ []OSKernelExists
563                 /\ []SormInits
564                 /\ []Correctness
565                 /\ []AbsCorrectnessOpp
566                 /\ OSUsabilityLiveness
567                 /\ AbsCorrectness
```

Листинг В.4 – Модуль ipes.tla с описанием спецификации модели: операций системы и свойств безопасности

Правила подсистемы управления доступом, необходимые для выполнения инвариантов и темпоральных свойств TLA+ приведены в Листинге B.5.

```
---------------------------------- MODULE sorm ----------------------------------
EXTENDS types, init, select
---------------------------------------------------------------------------------

\* Предикаты проверки подсистемы управления доступом

SormInitialized ==
    /\ s_sorm \in S_active
    \* прочитан ассоциированный объект o_sorm (правила доступа)
    /\ \E o \in SelectSubjData(s_sorm):
        /\ o.oid = 1
        /\ s_sorm.sid \in o.subj_assoc

SormCheckSubj(o, sc, r) ==
    /\  IF  \neg SormInitialized
        THEN \* активизируется s_sorm
             /\ sc.type = "sorm"
        ELSE \* запрос возможен только при активизированном s_sorm
             /\ SormInitialized
    \* delete_subject
    /\  IF  r \in QueriesStateChange
        THEN \* s_0, s_sorm не могут уничтожиться после активизации
             /\ sc \notin {s_0, s_sorm}
    \* create_user, create_shadow
        ELSE IF r \in QueriesAssocChange
        THEN \* субъект не должен быть заблокирован
             /\ sc.is_blocked = FALSE
    \* change_blocked
        ELSE IF r \in QueriesSystem
        THEN \* Административные действия выполняет только s_sorm
             /\ o.subj_assoc = {s_sorm.sid}
             \* Блокировать s_0 или s_sorm нельзя
             /\ sc \notin {s_0, s_sorm}
        ELSE TRUE
    \* дополнительные проверки (идентификация/аутентификация и т.д.)

SormCheckPerm(s,o,r) ==
    \* запрос разрешен s_0 и s_sorm
    /\  IF s.sid \in {s_0.sid, s_sorm.sid}
        THEN TRUE
        \* либо должен быть активизирован s_sorm
```

```
        ELSE SormInitialized
    \* удалять или исполнять o_sorm нельзя
    /\  IF o = o_sorm
        \* иначе нарушится SormInitialized
        THEN r \notin {"exec","create","delete"}
        ELSE TRUE
    \* правила для выполнения свойств корректности модели ИПСС
    /\  IF \E obj \in SelectObjects: obj = o
        THEN IF r \in QueriesStateChange
             \* изменение может совершать ассоц. субъект, либо
             \* объект должен быть в O_na (нельзя изменять/удалять
             \* чужие ассоциированные объекты)
             THEN   /\ o.subj_assoc \subseteq {s.sid}
                    \* s_0 и другие не должны изменить o_sorm
                    /\ o # o_sorm
             ELSE
             IF r \in QueriesAssocChange
             THEN   \* контроль целостности: нельзя изменять ассоц.
                    \* объекты с помощью измененных объектов данных;
                    \* субъект может породиться сам (без
                    \* каскадных сессий) или от имени s_0
                    o.state \in {s.sid, s_0.sid} \* # StateChanged
                    \* системные объекты — исключение
             ELSE   TRUE
        ELSE
            \* создание возможно только для личных объектов
            /\ o.subj_assoc \subseteq {s.sid}
            /\ o.state = s.sid
    \* другие дополнительные проверки (правила доступа и т.д.)
    \* при нарушении — запрос запрещен
```

Листинг В.5 – Модуль sorm.tla с описанием правил подсистемы управления доступом

Вспомогательные функции, применяемые в TLA+ нотации приведены в Листинге В.6.

```
---------------------------- MODULE select ----------------------------
EXTENDS init, Sequences
-----------------------------------------------------------------------

\* множество O_t модели ИПСС
SelectObjects == { o \in O_func \cup O_data \cup O_na: TRUE }

\* функционально ассоциированные объекты: процессы
SelectProc == { o \in O_func: TRUE }

```

```
\* функционально ассоциированные объекты субъекта
SelectSubjProc(s) == { o \in O_func: s.sid \in o.subj_assoc }

\* ассоциированные объекты–данные субъекта
SelectSubjData(s) == { o \in O_data: s.sid \in o.subj_assoc }

\* выбор последнего совершенного запроса и его параметров
SelectPrevQuery(Sq) == Sq[Len(Sq)]
SelectPrevQuerySubj(Sq) == SelectPrevQuery(Sq).subj
SelectPrevQueryProc(Sq) == SelectPrevQuery(Sq).proc
SelectPrevQueryDent(Sq) == SelectPrevQuery(Sq).dent
SelectPrevQueryType(Sq) == SelectPrevQuery(Sq).type

\* выбор множества запросов определенного типа
SelectQueries(Sq, L, Types) ==
    {
        q \in Queries:
        \E idx \in 1..L:
            /\ q = Q[idx]
            /\ q.type \in Types
    }
```

Листинг В.6 – Модуль select.tla с описанием вспомогательных функций

Приложение Г

Документы, подтверждающие внедрение результатов диссертации

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2015612555

Аккорд-Х

Правообладатель(ли): ***Закрытое акционерное общество "Особое Конструкторское Бюро Систем Автоматизированного Проектирования" (RU)***

Автор(ы): ***Каннер Андрей Михайлович (RU), не указаны***

Заявка № **2014663519**

Дата поступления **24 декабря 2014 г.**

Дата государственной регистрации в Реестре программ для ЭВМ ***19 февраля 2015 г.***

Руководитель Федеральной службы по интеллектуальной собственности

Г.П. Ивлиев

Экз. № 1

УТВЕРЖДАЮ
Начальник ФАУ «ГНИИИ ПТЗИ ФСТЭК России»
д-р техн. наук, ст. науч. сотр.

А.В. Анищенко

«11» февраля 2022 г.

Акт

о внедрении результатов диссертационной работы Каннера Андрея Михайловича на тему «Модель и алгоритмы обеспечения безопасности управления доступом в операционных системах семейства Linux» в Федеральном автономном учреждении «Государственный научно-исследовательский институт проблем технической защиты информации Федеральной службы по техническому и экспортному контролю»

Комиссия в составе кандидата технических наук, доцента Соловьева С.В., кандидата технических наук Мамуты В.В. и старшего научного сотрудника Тарелкина М.А. составила настоящий акт о том, что результаты диссертационной работы Каннера А. М., связанные с защитой информации от несанкционированного доступа в операционных системах семейства Linux, а именно:

– алгоритм обеспечения доверенной загрузки загрузчика и операционной системы при пошаговом контроле целостности;

– алгоритм встраивания функций защиты от несанкционированного доступа на раннем этапе загрузки операционной системы;

– требования для гарантии выполнения системой защиты информации от несанкционированного доступа целевых функций в операционной системе Linux,

используются в ФАУ «ГНИИИ ПТЗИ ФСТЭК России» при исследованиях уязвимостей системного программного обеспечения семейства Linux, связанных с процессами загрузки операционной среды и при верификации функциональных требований, предъявляемых к защитным механизмам на раннем этапе загрузки операционной системы.

Применение указанных алгоритмов и требований позволило проводить исследования уязвимостей свободного программного обеспечения на примере операционной системы Linux в интересах ведения национального банка данных угроз безопасности информации в ФАУ «ГНИИИ ПТЗИ ФСТЭК России».

Председатель комиссии
начальник управления
кандидат техн. наук, доцент С.В. Соловьев

Члены комиссии:
начальник отдела
кандидат техн. наук В.В. Мамута

старший научный сотрудник М.А. Тарелкин

ОКБ САПР ОСОБОЕ КОНСТРУКТОРСКОЕ БЮРО
СИСТЕМ АВТОМАТИЗИРОВАННОГО ПРОЕКТИРОВАНИЯ

ЗАО «ОКБ САПР» 115114, г. Москва, 2-й Кожевнический переулок, 12 Тел. +7 (495) 994-72-62
ИНН 7725739137, КПП 772501001 okbsapr@okbsapr.ru, www.okbsapr.ru

«УТВЕРЖДАЮ»
Генеральный директор
ЗАО «ОКБ САПР»,
к.т.н.
_____ И.Г. Назаров
«__» _____ 2022 г.

АКТ
о практическом применении (внедрении) результатов диссертационного исследования Каннера А.М. на тему «Модель и алгоритмы обеспечения безопасности управления доступом в операционных системах семейства Linux»

Настоящим комиссия в составе заместителя директора Счастного Д.Ю., начальника отдела программирования средств защиты информации (СЗИ) Мозолиной Н.В., начальника отдела инновационных разработок Батракова А.Ю. удостоверяет, что следующие результаты проведенного Каннером А.М. диссертационного исследования на тему «Модель и алгоритмы обеспечения безопасности управления доступом в операционных системах семейства Linux»:

– модель безопасности ОС со средством разграничения доступа, обеспечивающая выполнение заданной политики безопасности;
– алгоритм обеспечения доверенной загрузки загрузчика и ОС при пошаговом контроле целостности;
– алгоритм встраивания функций защиты от несанкционированного доступа на раннем этапе загрузки ОС;
– подсистема управления доступом для ОС Linux

обладают актуальностью, представляют практический интерес, были изучены и использованы компанией «ОКБ САПР» при разработке ряда программно-аппаратных комплексов СЗИ от несанкционированного доступа в ОС Linux, а именно: «Аккорд-X», «Аккорд-X K» и «Аккорд-XL».

Проведенное диссертантом исследование послужило как теоретическим фундаментом, так и практическим руководством для создания перечисленных перспективных средств защиты информации, которые были внедрены в ряд государственных информационных систем, в том числе и в подразделения Банка России.

Результаты исследования Каннера А.М. позволили решить ряд актуальных проблем систем разграничения доступа к информации в свободном программном обеспечении. В связи с этим, по мнению комиссии, выполненное исследование отвечает требованиям научности, а созданные на его основе новые программные и программно-аппаратные решения позволили в значительной степени повысить общий уровень защищенности информационных систем заказчиков компании.

Председатель комиссии:

Зам. генерального директора _____ Д.Ю. Счастный

Члены комиссии:

Начальник отдела программирования СЗИ _____ Н.В. Мозолина

Начальник отдела инновационных разработок _____ А.Ю. Батраков

«УТВЕРЖДАЮ»
И.о. проректора
НИЯУ МИФИ
_____ Е.Б. Весна
«__» _____ 2022 г.

А К Т

**о внедрении результатов диссертационной работы Каннера А.М.
на тему «Модель и алгоритмы обеспечения безопасности управления доступом в операционных системах семейства Linux»
в курс «Программно-аппаратные средства защиты информации»
кафедры «Криптология и кибербезопасность» (№ 42) НИЯУ МИФИ**

Комиссия в составе: зам. заведующего кафедрой № 42 к.т.н. Когоса К.Г., д.т.н., проф. Иваненко В.Г., д.т.н., доцента Запечникова С.В. составила настоящий акт о том, что результаты диссертационной работы Каннера А.М., связанные с защитой информации от несанкционированного доступа (НСД) в операционных системах (ОС) семейства Linux, а именно:

– модель безопасности ОС со средством разграничения доступа, обеспечивающая выполнение заданной политики безопасности;

– алгоритм обеспечения доверенной загрузки загрузчика и ОС при пошаговом контроле целостности, необходимый для достижения начального состояния системы в рамках разработанной модели безопасности;

– алгоритм встраивания функций защиты от НСД на раннем этапе загрузки ОС;

– подсистема управления доступом для ОС Linux и основанный на ней программно-аппаратный комплекс «Аккорд-X»

используются при проведении лабораторных занятий по курсу «Программно-аппаратные средства защиты информации» на кафедре «Криптология и кибербезопасность» Института интеллектуальных кибернетических систем НИЯУ МИФИ.

Применение приведенных выше результатов диссертационной работы позволяет слушателям данного учебного курса знакомиться с практическим использованием алгоритмов обеспечения активизации и непрерывности функционирования подсистемы управления доступом операционных систем семейства Linux, а также реализующим их программно-аппаратным комплексом «Аккорд-X».

к.т.н. Когос К.Г.

д.т.н. Иваненко В.Г.

д.т.н. Запечников С.В.