Журавлев Глеб Дмитриевич

Студент,РГУ нефти и газа (НИУ) имени И.М. Губкина

АВТОМАТИЗИРОВАННАЯ ФУНКЦИОНАЛЬНАЯ ВЕРИФИКАЦИЯ РАННЕ-АКТИВИРУЕМЫХ МЕР ЗАЩИТЫ В LINUX-СИСТЕМАХ

Аннотация: Рассматривается подход к автоматизированной функциональной верификации мер защиты, активируемых на ранних этапах загрузки Linux-систем. В отличие от статической проверки наличия пакета или конфигурационного файла, подход проверяет фактическое выполнение меры и формирует статус OK, DEGRADED или FAIL. Демонстрационным профилем служит шифрование блочного устройства LUKS/dm-crypt, проверяемое в initramfs dracut-модулем. Приведено сопоставление с systemd-cryptsetup и IMA/EVM по объекту контроля и артефактам проверки, а также количественная оценка влияния модуля на время загрузки по данным systemd-analyze.

Ключевые слова: функциональная верификация, ранняя загрузка, initramfs, dracut, LUKS, dm-crypt, IMA, EVM, TPM, systemd-cryptsetup.

Abstract: The paper presents an automated functional verification approach for security controls activated during the early Linux boot chain. Unlike static checks of package presence or configuration files, it verifies the actual security effect and classifies the result as OK, DEGRADED or FAIL. A LUKS/dm-crypt block-device encryption profile is verified inside initramfs by a dracut module. The approach is compared with systemd-cryptsetup and IMA/EVM by control object and verification artifact, and its impact on system boot time is quantified from systemd-analyze measurements.

Keywords: functional verification, early boot, initramfs, dracut, LUKS, dm-crypt, IMA, EVM, TPM, systemd-cryptsetup.

Введение

Средства защиты в Linux-системах часто оценивают по формальному признаку: установлен ли пакет, присутствует ли конфигурационный файл, включён ли сервис. Такой подход согласуется с логикой проверки выполнения

требований [2] и с моделью оценки доверия ГОСТ Р ИСО/МЭК 15408 [3], но недостаточен для суждения о практической применимости меры. Механизм может быть установлен и активирован, но работать с ослабленной конфигурацией, не принимать корректные входные данные или не давать требуемый защитный результат [7].

Функциональная верификация понимается здесь как проверка полного цикла применения меры защиты: от обнаружения объекта и анализа конфигурации до положительного и отрицательного сценариев и проверки итогового результата. По существу [15] речь идёт о подтверждении того, что механизм не только присутствует, но и даёт заявленный защитный эффект. Для мер ранней загрузки [10] это особенно важно: их некорректная работа подрывает доверие к последующим этапам цепочки [4], а компрометация этого этапа выделена в MITRE ATT&CK в отдельный класс техник Pre-OS Boot [5].

Обзор существующих исследований

Задача подтверждения того, что мера защиты действительно работает, а не просто установлена, решается в литературе с разных сторон. В [6] систематизированы меры усиления защищённости серверов Linux — минимальные привилегии, усиленная аутентификация, сплошной аудит, — но оценивается конфигурация, а не результат её применения. В [7] система аудита ОС Linux применена при сертификационных испытаниях: штатное журналирование пригодно для автоматического сбора доказательств, однако объектом контроля остаются события сборки, а не защитный эффект. В [12] тот же auditd отслеживает изменения файлов конфигурации — снова контроль состояния, а не функции.

Доверие к ранним стадиям загрузки разобрано в [10]: классифицированы средства доверенной загрузки уровня BIOS, плат расширения и загрузочной записи и показано, что корректная цепочка проверок защищает от буткитов и руткитов, — но объектом контроля выступает неизменность самой цепочки, а не корректность меры, активируемой внутри неё. Формальную сторону раскрывает [11], где построена верифицированная иерархическая модель с

мандатным управлением доступом и контролем целостности; такая модель доказывает свойства проекта, но не отвечает, применена ли мера на конкретном экземпляре.

Шифрование блочных устройств исследовано в [8], где систематизированы свойства схем полнодискового шифрования, и в [9], где разобраны LUKS, dm-crypt и режимы LUKS2 и отмечено, что аутентифицированное шифрование хранимых данных остаётся экспериментальным. Обе работы описывают, каким должно быть корректное шифрование, но не дают процедуры проверки того, что профиль действительно применён. Актуальность подтверждают [13], где незакрытая уязвимость Astra Linux приводит к повышению привилегий, и практика киберполигонов [1], где настройка средств защиты на стенде требует объективного подтверждения.

В отличие от перечисленных работ, данное исследование фокусируется на функциональной верификации меры в момент её активации — в initramfs, до передачи управления пользовательскому пространству — и на формировании воспроизводимого артефакта со статусом OK, DEGRADED или FAIL. Проверяется не наличие пакета и не неизменность загрузочной цепочки, а полный цикл: отклонение неверного ключа, приём корректного, создание mapper-устройства, монтирование ФС и чтение контрольного маркера — с количественной оценкой накладных расходов.

Постановка задачи

Цель работы — разработать и экспериментально проверить ранний модуль, автоматически оценивающий корректность применения меры защиты до перехода к пользовательской работе. Объектом выбран LUKS/dm-crypt [16]: шифрование блочного устройства имеет понятные входные данные, измеримые параметры и проверяемый функциональный результат [9].

Задачи: реализовать проверку в initramfs; подтвердить её выполнение в ранней загрузке; проверить не только наличие LUKS-устройства, но и корректность ключей, создание mapper-устройства, монтирование ФС и чтение маркера; сопоставить подход с systemd-cryptsetup и IMA/EVM [11, 12];

количественно оценить накладные расходы на время загрузки.

Метод работы разработанного модуля

Программа реализована как dracut-модуль 99lukscheck, встраиваемый в initrd по штатному интерфейсу модулей [17]. Основная логика — hook-скрипт lukscheck.sh на стадии pre-mount: она выбрана потому, что проверка выполняется до перехода в пользовательское пространство и до завершения загрузки.

Объект проверки — отдельный LUKS-том на дополнительном диске, не являющемся корневым разделом. Для положительного теста служит ключ test_key.bin, добавленный в слот LUKS; для отрицательного — wrong_key.bin, который не должен открывать контейнер. Внутри тома размещён файл marker.txt с содержимым LUKSCHECK_MARKER_OK.

Алгоритм не ограничивается командой open --test-passphrase: после анализа метаданных выполняются отрицательный и положительный тесты, реальное открытие контейнера, проверка появления mapper-устройства, монтирование ФС и чтение маркера. Проверяется не только ключ, но и фактический результат работы меры.

Таблица 1 – Алгоритм функциональной верификации LUKS-профиля

Этап	**Проверяемое условие**	**Интерпретация**
1	Существует целевое блочное устройство	При отсутствии устройства формируется FAIL
2	Устройство определяется как LUKS	Если объект не является LUKS-контейнером, формируется FAIL
3	Считываются Version, cipher, PBKDF	LUKS2, AES и Argon2 интерпретируются как корректный профиль; отклонения могут дать DEGRADED
4	Неверный ключ отклоняется	Если wrong_key принимается, мера защиты считается отказавшей
5	Корректный ключ принимается	Если test_key не принимается, формируется FAIL
6	Контейнер реально открывается	Проверяется создание /dev/mapper/...
7	ФС монтируется и читается marker.txt	При совпадении маркера подтверждается функциональный результат

8	Формируется итоговый статус	OK, DEGRADED или FAIL записывается в артефакт проверки

Статус OK означает, что контейнер существует, является LUKS, имеет допустимые параметры, отвергает неверный ключ, принимает верный, создаёт mapper-устройство, монтируется и содержит ожидаемый маркер. DEGRADED соответствует случаю, когда мера работает, но имеет ослабляющие признаки: LUKS1, PBKDF2 вместо Argon2 или несовпадение маркера. FAIL означает отказ критического этапа: нет устройства, объект не опознан как LUKS, принят неверный ключ, отвергнут корректный, не создано mapper-устройство или нет доступа к данным.

Теоретическое сопоставление с существующими механизмами

Сопоставление проводится по объекту контроля и носит преимущественно теоретический характер: экспериментально на стенде наблюдались systemd-cryptsetup и IMA/EVM, тогда как по TPM измерена только автоматическая разблокировка контейнера по TPM2 — полная цепочка measured boot с проверкой значений PCR не воспроизводилась. Предлагаемый модуль не дублирует эти механизмы, поскольку они работают в разных слоях доверия. systemd-cryptsetup подключает зашифрованные устройства по crypttab [16], но не классифицирует профиль защиты и не проверяет результат внутри тома. IMA/EVM относится к подсистеме integrity: IMA измеряет и аппрайзит файлы, EVM защищает security-xattrs [11, 12]; на исходной конфигурации стенда TPM отсутствовал, поэтому IMA работала в режиме TPM-bypass.

TPM-based measured boot выступает дополняющим, а не конкурирующим слоем: он обеспечивает неизменность загрузочной цепочки и противодействует техникам T1542.001 и T1542.003 [5], тогда как разработанный модуль проверяет корректность применения меры на уровне блочного устройства уже после её инициализации. Различие сохраняется и при доступном TPM: успешное открытие контейнера по TPM2 подтверждает ожидаемое состояние платформы, но ничего не говорит о криптографическом профиле контейнера и о фактическом доступе к данным. Тем самым модуль дополняет известные

методы верификации конфигураций [11], смещая проверку от статического анализа настроек к измерению защитного эффекта [6].

Практическая часть

Эксперимент выполнялся на ALT Workstation в виртуальной машине VirtualBox. Выбор данного отечественного дистрибутива обусловлен его активным внедрением и изучением механизмов обеспечения безопасности в рамках политики импортозамещения [18]. Изолированный виртуальный стенд соответствует практике учебно-исследовательских полигонов [1]. В отличие от ранней версии с файловым образом и post-boot проверкой, текущая использует отдельный блочный диск /dev/sdb с LUKS-контейнером. Модуль встроен в initrd, его выполнение подтверждалось сообщениями journalctl и dmesg.

В ходе отладки выявилась показательная ситуация: корректный ключ принимался и mapper-устройство создавалось, но ФС не монтировалась из-за отсутствия подготовленной файловой системы внутри контейнера. Это подтвердило, что модуль фиксирует и неполную подготовку тома, а не только ошибки ключей.

На рисунках 1–5 показаны анализ параметров LUKS, отрицательная и положительная проверки ключей с монтированием тома, фиксация результата в системном журнале и формирование статуса DEGRADED.

Иллюстрации хода эксперимента

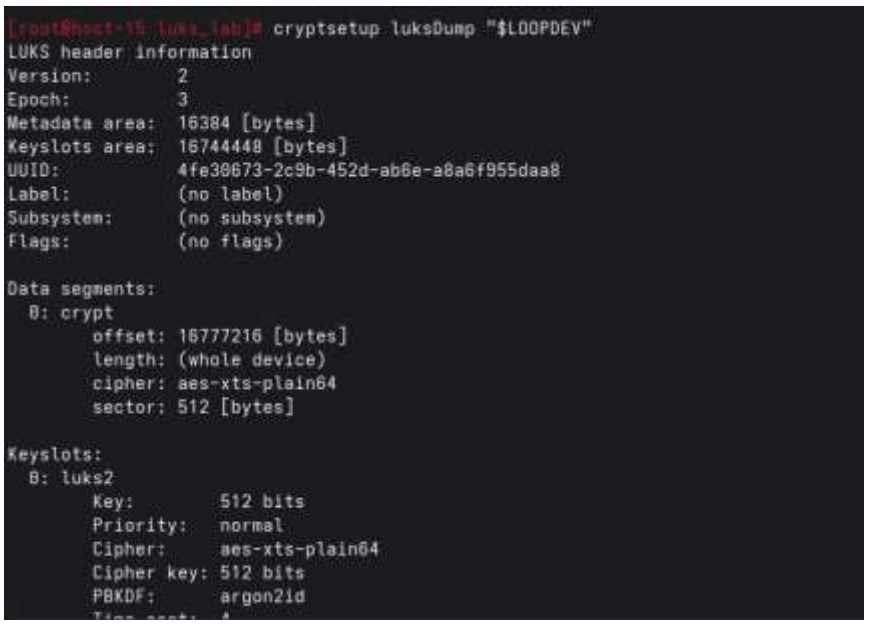

Рисунок 1 – Анализ метаданных LUKS: версия, шифр и PBKDF-профиль

Рисунок 2 – Отрицательная проверка: некорректный ключ отвергается

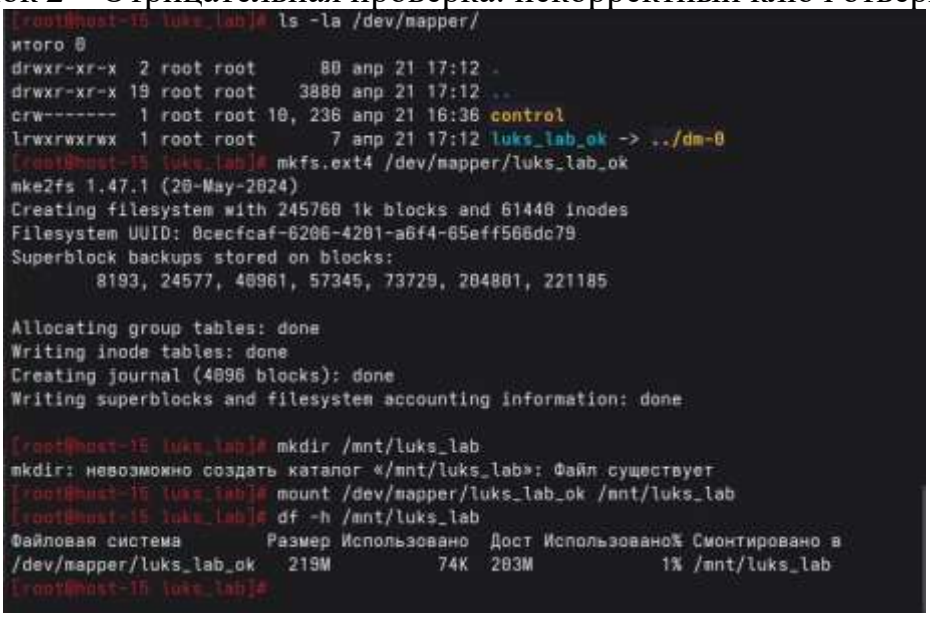

Рисунок 3 – Открытие mapper-устройства, создание ext4 и монтирование тома

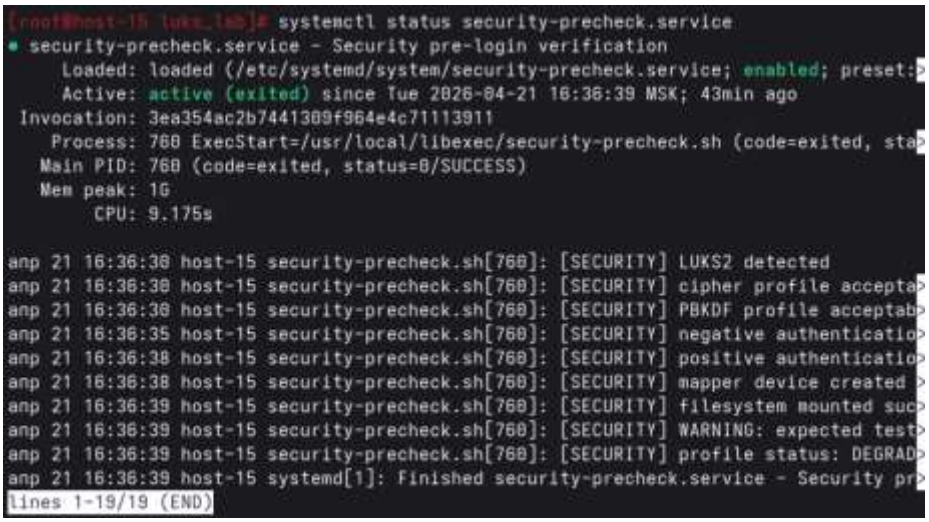

Рисунок 4 – Пример журнала работы контрольного сервиса с классификацией результата

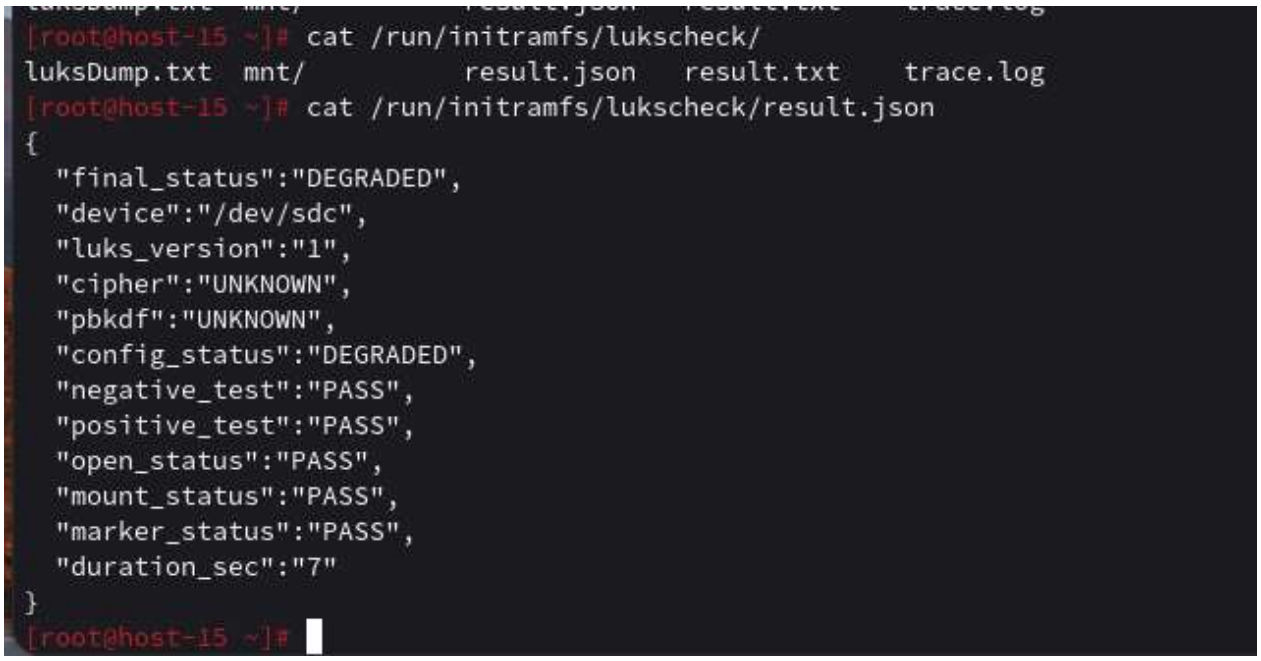

Рисунок 5 – Формирование статуса DEGRADED для ослабленного профиля LUKS1 (содержимое result.json)

На первом этапе выполнялись серии загрузок baseline, systemd_cryptsetup, ima_evm, lukscheck_early, lukscheck_full и ima_evm_lukscheck; данные собирались скриптом collect_boot_metrics. Эта серия подтверждала сам факт срабатывания модуля и фиксировала формируемые статусы. Время загрузки измерялось отдельной серией с сохранением вывода systemd-analyze time и blame (таблица 4).

В таблицах LC — сокращение от lukscheck. По прогонам с сохранённым result.json оценивалось время самой проверки: оно относится к работе скрипта в раннем контуре, а не ко всей загрузке.

Таблица 2 – Время выполнения проверки lukscheck по валидным прогонам

Режим / сценарий	**Статус**	**Прогонов**	**Время, с (ср.; мин.; макс.)**	**Устр.**
LC early / degraded	OK	3	17; 14; 21	/dev/sdc
LC early / fail	FAIL	4	9; 7; 12	/dev/sdb
LC early / ok	OK	3	12; 11; 13	/dev/sdb
LC early / degraded	DEGRADED	2	7; 7; 7	/dev/sdc

Анализ результатов

Во-первых, модуль действительно выполнялся в initramfs и проводил содержательную проверку: в прогонах lukscheck_early со статусом OK отрицательный и положительный тесты, открытие контейнера, монтирование и проверка маркера завершались успешно. Среднее время проверки по валидным OK-прогонам — около 12 секунд.

Во-вторых, воспроизведён отказной сценарий: в серии fail модуль формировал FAIL после того, как корректный ключ не принимался. Среднее время — около 9 секунд, при этом этапы open, mount и marker не запускались, что соответствует логике раннего прекращения проверки.

В-третьих, DEGRADED воспроизведён на реально ослабленном профиле: на /dev/sdc подготовлен LUKS1-контейнер, и модуль по метаданным сформировал DEGRADED при luks_version = 1 и config_status = DEGRADED (рисунок 5). В более ранней серии с тем же названием такого результата не было — профиль /dev/sdc тогда не содержал распознаваемых признаков ослабления, и прогоны дали OK. Это подтверждает, что классификация опирается на измеряемые признаки, а не на имя сценария.

В-четвёртых, IMA/EVM проверены как отдельный integrity-слой: фиксировались ima_policy=appraise_tcb, ima_appraise=fix, evm=fix и audit-события appraisal. На исходной конфигурации TPM отсутствовал, поэтому IMA работала в режиме TPM-bypass; после пересборки стенда с виртуальным TPM 2.0 дополнительно измерен режим автооткрытия по TPM2. Это не противоречит цели работы: IMA/EVM и TPM оценивают доверие и целостность, тогда как lukscheck проверяет функциональный результат конкретной меры шифрования.

Таблица 3 – Покрытие проверок безопасности разными механизмами

Критерий	**systemd-cryptsetup**	**IMA/EVM**	**lukscheck**
Ранняя стадия загрузки	Да	Да/частично	Да, pre-mount hook
Проверка наличия LUKS	Косвенно	Нет	Да

Проверка LUKS2/PBKDF/cipher	Нет	Нет	Да
Отрицательный тест ключа	Нет	Нет	Да
Положительный тест ключа	Частично, через unlock	Нет	Да
Создание mapper	Да	Нет	Да
Монтирование и marker.txt	Нет	Нет	Да
Контроль целостности файлов	Нет	Да	Нет
Формальный статус OK/DEGRADED/FAIL	Нет	Нет в данной модели	Да

Количественная оценка влияния на время загрузки

Выполнена повторная серия загрузок с сохранением вывода systemd-analyze time и blame в четырёх режимах: эталонная загрузка без модуля (baseline), загрузка с модулем на корректном LUKS2-профиле (OK), на намеренно ослабленном LUKS1-профиле (DEGRADED) и с автоматическим открытием вспомогательного контейнера по TPM2 (tpm_autounlock). В каждом режиме — по три загрузки; в таблице 4 приведены значения в формате «среднее / медиана».

Таблица 4 – Влияние модуля верификации на время загрузки системы

Режим загрузки	**kernel, с**	**initrd, с**	**userspace, с**	**total, с**
Baseline (без модуля)	1,01 / 0,77	10,51 / 8,88	59,12 / 69,71	70,65 / 78,98
lukscheck_early, OK (LUKS2)	0,72 / 0,70	20,65 / 20,41	90,48 / 103,73	111,85 / 124,82
lukscheck_early, DEGRADED (LUKS1)	0,83 / 0,79	14,99 / 15,01	74,98 / 58,43	90,80 / 74,39
Автооткрытие по TPM2 (tpm_autounlock)	1,85 / 1,50	7,92 / 6,46	99,98 / 116,96	109,75 / 126,31

Показатели userspace и total обладают высокой дисперсией: коэффициент вариации по ним 32–41 %. Основной источник разброса — сервис plymouth-quit-wait.service, длительность которого менялась от 21,5 до 93,3 с независимо от наличия модуля. Поэтому суммарное время загрузки не

может служить мерой накладных расходов, и далее анализируются стадии kernel и initrd: их коэффициент вариации при активном модуле не превышает 13 %, а для ключевой стадии initrd — 3 %.

Показательна стадия initrd, на которой и работает модуль. По медиане её длительность составила 8,88 с в эталонном режиме, 15,01 с в сценарии DEGRADED и 20,41 с в сценарии OK; прирост равен 6,13 с и 11,53 с (по среднему — 4,47 с и 10,14 с). Различие объясняется объёмом криптографической работы: LUKS1 с PBKDF2 проверяется заметно быстрее, чем LUKS2 с Argon2, где основное время занимает вычисление ключа памяти-затратной функцией. Стоимость проверки зависит не от факта её выполнения, а от стойкости проверяемого профиля.

Прямая оценка стоимости получена из systemd-analyze blame по юниту dracut-pre-mount.service, который выполняет hook-скрипты стадии pre-mount и присутствует в журнале только при встроенном модуле. Его длительность — 9,01 с (8,56; 9,33) в сценарии DEGRADED и 15,18 с (14,99; 15,35) в сценарии OK при коэффициенте вариации менее 5 %. Это согласуется с приростом initrd и подтверждает, что накладные расходы локализованы в проверочном скрипте. Время инициализации ядра во всех режимах с модулем не превышало 0,95 с и практически не отличалось от эталонного (медиана 0,77 с).

Режим tpm_autounlock сопоставим с остальными строками лишь качественно. Он измерялся на пересобранной конфигурации виртуальной машины с виртуальным TPM 2.0, где изменился и эмулируемый чипсет; кроме того, вспомогательный контейнер открывался по crypttab уже в пользовательском пространстве, поэтому расшифрование отражается не в стадии initrd, а в юните systemd-cryptsetup@tpmtest.service длительностью 3,07 с (2,60; 3,79). Именно поэтому initrd в этом режиме короче эталонной: модуль верификации в неё не встраивался.

Ограничения эксперимента

Количественная часть опирается на выборку из трёх загрузок на режим. Этого достаточно, чтобы оценить порядок накладных расходов и устойчиво измерить стадии kernel и initrd, но недостаточно для проверки статистических гипотез о суммарном времени. Замеры выполнены в виртуальной машине, где дисперсия существенно выше, чем на физическом оборудовании: значительная часть разброса вносится сервисами графической подсистемы и ожидания сети. Для строгого сравнения нужна серия не менее чем из десяти загрузок на режим с отключением plymouth-quit-wait.service и NetworkManager-wait-online.service и с использованием медианы.

Более ранняя серия решала задачу качественной проверки срабатывания модуля и не предназначалась для измерения времени; статус DEGRADED в ней также не был зафиксирован, поскольку профиль /dev/sdc тогда не содержал признаков ослабления. Оба пробела закрыты отдельными сериями (таблица 2, рисунок 5 и таблица 4).

Режим автооткрытия по TPM2 измерялся на пересобранной конфигурации: для появления в гостевой ОС устройств /dev/tpm0 и /dev/tpmrm0 потребовалось включить виртуальный TPM 2.0, что повлекло смену эмулируемого чипсета, — поэтому на длительность стадий влияет не только наличие TPM. Совмещённый режим, в котором TPM-открытие корневого контейнера сочеталось бы с верификацией в initramfs, не исследовался.

Заключение

В работе представлен подход к автоматизированной функциональной верификации ранне-активируемых мер защиты. На примере LUKS/dm-crypt показано, что проверка выполнима не после входа пользователя, а в раннем контуре initramfs. Модуль lukscheck проверяет наличие устройства, тип LUKS, криптографический профиль, отрицательный и положительный сценарии ключа, создание mapper-устройства, монтирование ФС и чтение маркера. Экспериментально подтверждено формирование всех трёх классов

статуса, включая DEGRADED на ослабленном LUKS1-профиле, и измерена цена проверки: по медиане она добавляет к стадии initrd от 6,1 до 11,5 с в зависимости от профиля, не изменяя времени инициализации ядра.

Разработанный подход занимает отдельную нишу: он не заменяет штатное подключение LUKS, integrity-контроль файлов или аппаратное измерение загрузочной цепочки, но закрывает задачу проверки фактического результата применения конкретной меры защиты. Именно это отличает функциональную верификацию от проверки наличия механизма или контроля неизменности файлов. Практическое применение модуля предполагает его сопровождение в рамках процессов разработки безопасного программного обеспечения [14].

Список литературы

1. Греков, В. С. Стратегии создания модели сети предприятия в рамках киберполигона для эффективной подготовки кадров в области кибербезопасности / В. С. Греков, А. Г. Уймин // Актуальные проблемы комплексной безопасности критически важных объектов топливно-энергетического комплекса : Тезисы докладов 78-й Международной молодежной научной конференции, Москва, 22–26 апреля 2024 года. – Москва : Российский государственный университет нефти и газа (национальный исследовательский университет) им. И.М. Губкина, 2025. – С. 17-18. – EDN RVOSAN. – Текст : непосредственный.
2. Об утверждении Требований о защите информации, не составляющей государственную тайну, содержащейся в государственных информационных системах : Приказ ФСТЭК России от 11.02.2013 № 17 (ред. от 28.08.2024) [Электронный ресурс]. – URL: https://www.consultant.ru/document/cons_doc_LAW_147084/ (дата обращения: 15.08.2026). – Текст : электронный.
3. ГОСТ Р ИСО/МЭК 15408-1-2015. Информационная технология. Методы и средства обеспечения безопасности. Критерии оценки безопасности информационных технологий. Часть 1. Введение и общая модель. – Москва : Стандартинформ, 2016. – URL: http://docs.cntd.ru/document/1200112238 (дата обращения: 20.05.2024). – Текст : электронный.
4. NIST SP 800-193: Platform Firmware Resiliency Guidelines [Электронный ресурс] / National Institute of Standards and Technology. – Gaithersburg, MD : NIST, 2018. – URL: https://doi.org/10.6028/NIST.SP.800-193 (дата обращения: 20.05.2026). – Текст : электронный.

5. Pre-OS Boot: Technique T1542 [Электронный ресурс] // MITRE ATT&CK : [официальный сайт]. – URL: https://attack.mitre.org/techniques/T1542/ (дата обращения: 20.05.2026). – Текст : электронный.
6. Романов, М. А. Методы повышения безопасности при администрировании серверов Linux / М. А. Романов, А. О. Бадалин, Д. В. Рыбалка, В. В. Борисов, Д. Д. Новикова // Программные системы и вычислительные методы. – 2025. – № 4. – DOI 10.7256/2454-0714.2025.4.77322. – EDN UHRMPL. – Текст : электронный.
7. Самаров, В. В. Использование системы аудита операционных систем семейства Linux при проведении сертификационных испытаний программных изделий / В. В. Самаров // Надежность и качество сложных систем. – 2021. – № 1. – С. 144–150. – URL: https://cyberleninka.ru/article/n/ispolzovanie-sistemy-audita-operatsionnyh-sistem-semeystva-linux-pri-provedenii-sertifikatsionnyh-ispytaniy-programmnyh-izdeliy (дата обращения: 15.08.2026). – Текст : электронный.
8. Алексеев, Е. К. Защищённое хранение данных и полнодисковое шифрование / Е. К. Алексеев, Л. Р. Ахметзянова, А. А. Бабуева, С. В. Смышляев // Прикладная дискретная математика. – 2020. – № 49. – URL: https://cyberleninka.ru/article/n/zaschischyonnoe-hranenie-dannyh-i-polnodiskovoe-shifrovanie (дата обращения: 15.08.2026). – Текст : электронный.
9. Васильева, И. Н. Криптографическая защита цифровых носителей информации / И. Н. Васильева // Технико-технологические проблемы сервиса. – 2019. – URL: https://cyberleninka.ru/article/n/kriptograficheskaya-zaschita-tsifrovyh-nositeley-informatsii (дата обращения: 15.08.2026). – Текст : электронный.
10. Авезова, Я. Э. Вопросы обеспечения доверенной загрузки в физических и виртуальных средах / Я. Э. Авезова, А. А. Фадин // Вопросы кибербезопасности. – 2016. – URL: https://cyberleninka.ru/article/n/voprosy-

obespecheniya-doverennoy-zagruzki-v-fizicheskih-i-virtualnyh-sredah (дата обращения: 15.08.2026). – Текст : электронный.

11. Девянин, П. Н. Интеграция мандатного и ролевого управления доступом и мандатного контроля целостности в верифицированной иерархической модели безопасности операционной системы / П. Н. Девянин, В. В. Кулямин, А. К. Петренко, А. В. Хорошилов, И. В. Щепетков // Труды Института системного программирования РАН. – 2020. – Т. 32, № 1. – С. 7–26. – DOI 10.15514/ISPRAS-2020-32(1)-1. – Текст : электронный.
12. Цянь, Л. В. Функционал auditd для отслеживания изменений файлов системной конфигурации / Л. В. Цянь // Вестник науки. – 2024. – URL: https://cyberleninka.ru/article/n/funktsional-auditd-dlya-otslezhivaniya-izmeneniy-faylov-sistemnoy-konfiguratsii (дата обращения: 15.08.2026). – Текст : электронный.
13. Степанова, Е. А. Эксплуатация уязвимости операционной системы Astra Linux / Е. А. Степанова, Д. А. Елизаров, Е. Е. Мурзаева // Прикаспийский журнал: управление и высокие технологии. – 2025. – URL: https://cyberleninka.ru/article/n/ekspluatatsiya-uyazvimosti-operatsionnoy-sistemy-astra-linux (дата обращения: 15.08.2026). – Текст : электронный.
14. ГОСТ Р 56939-2024. Защита информации. Разработка безопасного программного обеспечения. Общие требования. – Москва : Российский институт стандартизации, 2024. – URL: https://docs.cntd.ru/document/1310017763 (дата обращения: 15.08.2026). – Текст : электронный.
15. RFC 4949. Internet Security Glossary, Version 2 [Электронный ресурс] / IETF. – 2007. – URL: https://datatracker.ietf.org/doc/html/rfc4949 (дата обращения: 20.05.2026). – Текст : электронный.
16. cryptsetup(8) : manage plain dm-crypt and LUKS encrypted volumes [Электронный ресурс] // Linux manual pages : [официальный сайт]. – URL:

https://man7.org/linux/man-pages/man8/cryptsetup.8.html (дата обращения: 20.05.2024). – Текст : электронный.

17. dracut.modules(7) : dracut module interface [Электронный ресурс] // Linux manual pages : [официальный сайт]. – URL: https://man7.org/linux/man-pages/man7/dracut.modules.7.html (дата обращения: 20.05.2026). – Текст : электронный.
18. Ерёмина, Е. А. Изоляция процессов в операционной системе Альт с использованием механизмов cgroups v2 / Е. А. Ерёмина, А. Н. Простова // Профессионалитет. — 2025. — Т. 3, № 8. — С. 6–17. — URL: https://www.elibrary.ru/item.asp?id=91902025 (дата обращения: 30.08.2026). Текст : электронный.