Рубрика 1. Информатика и информационные процессы

***УДК** 004.056*

Д. Д. Новикова, С. С. Чурсина
ФГАОУ ВО «РГУ нефти и газа (НИУ) имени И.М. Губкина», г. Москва, Российская Федерация

СРАВНИТЕЛЬНЫЙ АНАЛИЗ ЭФФЕКТИВНОСТИ СРЕДСТВ МОНИТОРИНГА БЕЗОПАСНОСТИ В РЕАЛЬНОМ ВРЕМЕНИ НА БАЗЕ ПОДСИСТЕМЫ АУДИТА ЯДРА И eBPF-ТЕХНОЛОГИЙ В ОС АЛЬТ

Аннотация. *В статье представлены результаты сравнительного экспериментального исследования средств мониторинга безопасности в реальном времени в среде отечественной операционной системы Альт Linux. Рассматриваются два принципиально различных подхода к обнаружению угроз: классическая подсистема аудита ядра auditd и современный eBPF-инструмент Tracee. В ходе работы выявлены практические ограничения совместимости популярных eBPF-решений с виртуализированными средами, а также проведена количественная оценка накладных расходов инструментов на производительность системы и их способности обнаруживать типовые сценарии аномальной активности. Экспериментальная среда развёрнута на базе ОС Альт Рабочая Станция 11.1 с применением гипервизора VirtualBox. Материал будет полезен специалистам по защите информации, системным администраторам и разработчикам защищённых конфигураций на базе отечественных дистрибутивов Linux.*

Ключевые слова: *мониторинг безопасности, защита в процессе выполнения, auditd, eBPF, Tracee, ОС Альт Linux, системные вызовы, обнаружение аномалий, производительность ядра, виртуализация.*

Введение

Обеспечение безопасности операционных систем в условиях непрерывного усложнения ландшафта угроз становится одной из ключевых задач современной информационной безопасности. Традиционные средства защиты, ориентированные на предотвращение вторжений на периметре, всё чаще оказываются недостаточными: злоумышленники, получившие доступ к системе, способны действовать незаметно на протяжении длительного времени. В этом контексте особую значимость приобретают инструменты защиты в процессе выполнения – средства мониторинга поведения системы в режиме реального времени, способные обнаруживать аномальную активность непосредственно в момент её возникновения [1].

Актуальность настоящей работы определяется несколькими факторами. Во-первых, отечественные операционные системы, в частности ОС Альт Linux, получают всё более широкое распространение в государственных и корпоративных информационных системах, однако вопросы применимости современных инструментов мониторинга безопасности в реальном времени в данной среде остаются недостаточно изученными. Во-вторых, существующие сравнительные исследования инструментов этого класса, как правило, ориентированы на контейнеризированные и облачные среды, тогда как поведение данных инструментов на традиционных рабочих станциях под управлением отечественных дистрибутивов Linux практически не исследовано [2]. В-третьих, выбор между инструментами с принципиально различными архитектурами – классическим аудитом на уровне ядра и современными eBPF-решениями – требует эмпирического обоснования, поскольку теоретические характеристики инструментов не всегда соответствуют их поведению в конкретной операционной среде.

Новизна работы заключается в том, что впервые выполнено прямое экспериментальное сравнение классического аудита ядра и eBPF-инструмента в среде отечественной ОС Альт Linux с применением виртуализации, а также эмпирически установлены границы совместимости современных eBPF-решений (Falco, Sysdig) с конфигурацией «VirtualBox + ядро 6.12». В отличие от существующих обзоров, фокусирующихся на контейнерных средах, данное исследование рассматривает поведение инструментов мониторинга на традиционной рабочей станции, что приближает результаты к реальным условиям эксплуатации автоматизированных рабочих мест.

Теоретические основы мониторинга безопасности в реальном времени

Защита в процессе выполнения (runtime security) – это процесс защиты приложений, рабочих нагрузок и систем от угроз непосредственно во время их активного выполнения. Принципиальное отличие данного подхода от статических методов безопасности состоит в том, что

он направлен не на предотвращение уязвимостей на этапе разработки или развёртывания, а на обнаружение и нейтрализацию угроз, которые проявляются исключительно в процессе работы системы. Сканирование образов и статический анализ кода не способны выявить атаки, эксплуатирующие легитимные системные механизмы уже после запуска приложения – именно этот пробел закрывает мониторинг безопасности в реальном времени.

Все взаимодействия между пользовательскими приложениями и ресурсами операционной системы осуществляются через системные вызовы – интерфейс, предоставляемый ядром Linux. Поскольку практически любое действие в операционной системе сопровождается обращением к ядру через системный вызов, именно этот уровень является оптимальной точкой наблюдения для инструментов мониторинга безопасности: здесь можно отследить открытие файлов, запуск процессов, сетевые соединения и изменения прав доступа. Современные инструменты используют два основных механизма перехвата: подсистему аудита ядра, встроенную в Linux, и технологию eBPF.

Подсистема аудита (auditd) присутствует в ядре Linux начиная с версии 2.6. Архитектурно она состоит из модуля в пространстве ядра, перехватывающего системные вызовы, и демона в пространстве пользователя, который получает события через сетевой интерфейс ядра и записывает их в журнал [4]. Правила аудита задаются с помощью утилиты auditctl и определяют, какие именно события подлежат регистрации. Важной характеристикой auditd является детерминированность: инструмент регистрирует ровно те события, для которых явно определены правила, что обеспечивает предсказуемость и минимальный уровень ложных срабатываний.

Технология eBPF (Extended Berkeley Packet Filter) встроена в ядро Linux и позволяет безопасно исполнять пользовательские программы непосредственно в ядре без изменения его исходного кода и без загрузки дополнительных модулей [3]. Программа, написанная на подмножестве языка C, компилируется в eBPF-байткод, проходит строгую верификацию и после динамической компиляции исполняется с производительностью, близкой к нативному коду ядра. Ключевым преимуществом eBPF перед традиционными модулями ядра является безопасность: верификатор полностью исключает некорректное поведение программы до её запуска, тогда как ошибочный модуль ядра способен вызвать критический сбой системы [5].

Для присоединения eBPF-программ к точкам перехвата используется механизм kprobes (Kernel Probes), появившийся в ядре версии 2.6.9. При регистрации kprobe сохраняется копия целевой инструкции, а её первый байт заменяется инструкцией программного прерывания. Когда процессор достигает этой точки, управление передаётся обработчику eBPF-программы. Такая схема позволяет динамически устанавливать точки перехвата практически в любом месте кода ядра без его перекомпиляции и перезагрузки системы.

Важным дополнением к eBPF является механизм CO-RE (Compile Once – Run Everywhere), реализованный через BTF (BPF Type Format). CO-RE позволяет скомпилировать eBPF-программу один раз и запускать её на любом совместимом ядре без повторной компиляции, что существенно упрощает распространение инструментов.

Среди инструментов мониторинга безопасности, использующих eBPF, можно выделить Falco и Tracee. Falco – инструмент обнаружения угроз, разработанный компанией Sysdig и переданный в CNCF (Cloud Native Computing Foundation) [6]. Tracee – инструмент безопасности в процессе выполнения от компании Aqua Security, использующий сигнатурный подход к обнаружению аномалий [7]. Сигнатуры Tracee описывают поведенческие паттерны, характерные для известных техник атак, и организованы в соответствии с базой знаний MITRE ATT&CK.

Рассмотренные инструменты реализуют два принципиально различных подхода к обнаружению аномалий. Первый подход – явный мониторинг на основе правил – представлен auditd. Его эффективность прямо зависит от полноты и корректности настроенных правил. Второй подход – поведенческое обнаружение на основе сигнатур – представлен Tracee. Он не требует ручной настройки правил для каждого типа угроз, однако его действенность определяется актуальностью сигнатурной базы.

Методология эксперимента

Для проведения эксперимента была развёрнута среда на базе гипервизора VirtualBox. Тестовая среда представляла собой виртуальную машину под управлением ОС Альт Рабочая Станция 11.1 со следующими характеристиками: оперативная память – 4096 МБ, количество виртуальных процессоров – 4, объём жёсткого диска – 30 ГБ. Использование виртуальной машины обусловлено тем, что контрольные точки VirtualBox позволяют гарантированно восстанавливать систему в исходное состояние между тестами разных инструментов, обеспечивая воспроизводимость условий эксперимента [10].

На подготовительном этапе на виртуальную машину были установлены инструменты нагрузочного тестирования: stress-ng, fio и sysbench. Для автоматизации сбора метрик производительности и фиксации результатов обнаружения были разработаны два сценария командной оболочки: benchmark.sh обеспечивает последовательный запуск нагрузочных тестов и запись результатов в CSV-файл; attack_scenarios.sh выполняет набор сценариев имитации атак и фиксирует факт срабатывания инструмента мониторинга.

Эксперимент состоял из двух независимых блоков. Блок 1 – оценка производительности: измерение накладных расходов инструмента мониторинга на работу системы в штатном режиме без выполнения атак. Измерения проводились по четырём метрикам: загрузка процессора под нагрузкой, скорость случайной записи на диск (IOPS), скорость случайного чтения с диска (IOPS) и задержка запуска процессов. Блок 2 – оценка точности обнаружения: последовательное выполнение десяти сценариев, имитирующих типовые техники атак, с фиксацией факта и времени срабатывания инструмента.

Порядок выполнения экспериментальных прогонов предусматривал последовательный откат к контрольной точке, установку исследуемого инструмента, запуск сценария сбора метрик в трёх режимах (baseline, auditd, tracee), затем запуск сценария attack_scenarios.sh для соответствующего инструмента. Каждый тест повторялся трижды для минимизации случайных отклонений.

Исследование ограничений совместимости eBPF-стека

Первоначально в качестве исследуемых инструментов защиты в процессе выполнения были выбраны Falco и Sysdig – наиболее широко применяемые решения данного класса с открытым исходным кодом. Однако в ходе развёртывания экспериментальной среды были выявлены ограничения, обусловленные совместным использованием гипервизора VirtualBox и ОС Альт Linux 11.1 с ядром версии 6.12. Данный этап работы является самостоятельным практически значимым результатом: он позволяет установить границы применимости современных eBPF-решений в виртуализированных средах на базе отечественных дистрибутивов Linux.

Falco версий 0.38.2 и 0.39.2 не запустился ни в одном из доступных режимов работы. Режим modern eBPF завершался с ошибкой инициализации: в гостевой ОС под управлением VirtualBox тип eBPF-программ BPF_PROG_TYPE_TRACING, необходимый для работы данного режима, оказался недоступен. Аналогичная проблема зафиксирована в баг-трекере проекта применительно к различным дистрибутивам Linux [8]. Компиляция legacy eBPF-пробы оказалась невозможной из-за отсутствия пакета kernel-devel для ядра 6.12 в репозитории ОС Альт Linux. Sysdig, доступный в репозитории ALT Linux, также несовместим с ядром 6.12: загрузка драйвера завершается ошибкой, сборка модуля через DKMS невозможна [6].

В связи с изложенным итоговый эксперимент представляет собой сравнение двух инструментов с принципиально различной архитектурой: auditd, встроенного в ядро и не требующего внешних зависимостей, и Tracee – eBPF-инструмента, развёртываемого в виде контейнера и использующего механизм CO-RE. Наличие BTF-информации в директории /sys/kernel/btf/vmlinux на тестовой машине было подтверждено, что сделало возможным запуск Tracee в условиях ограниченной поддержки eBPF со стороны гипервизора.

Сценарии имитации атак

Для оценки способности инструментов обнаруживать подозрительную активность был сформирован набор из десяти сценариев, охватывающих основные классы аномального поведения на уровне операционной системы. При формировании набора авторы опирались на

матрицу MITRE ATT&CK, представляющую собой общепризнанную базу знаний о тактиках и техниках злоумышленников [9]. Перечень сценариев представлен в таблице 1.

Таблица 1 – Перечень сценариев имитации атак

№	Идентификатор	Описание
1	write_etc_passwd	Запись в файл /etc/passwd
2	read_etc_shadow	Чтение файла /etc/shadow
3	suid_bit_set	Установка SUID-бита на исполняемый файл
4	write_dev_shm	Создание файла в разделяемой памяти /dev/shm
5	netcat_listen	Открытие сетевого порта через netcat
6	process_spawn	Массовый запуск дочерних процессов
7	chmod_system_file	Изменение прав доступа к системному файлу
8	crontab_modification	Добавление записи в crontab
9	network_download	Загрузка файла из сети через curl
10	read_ssh_key	Попытка чтения приватного SSH-ключа

Каждый сценарий выполнялся в интерактивном режиме: после выполнения команды предусматривалась пауза в 10 секунд для наблюдения за реакцией инструмента, после чего результат фиксировался вручную в файл detection_results.csv.

Результаты тестирования производительности

Эксперимент по оценке производительности проводился поэтапно: сначала на чистой системе без инструментов мониторинга (режим baseline), затем последовательно с запущенным auditd и Tracee. В качестве итоговых значений использовалось среднее арифметическое трёх прогонов.

Первый прогон выполнялся на машине в исходном состоянии. Вывод терминала в процессе сбора метрик представлен на рисунке 1.

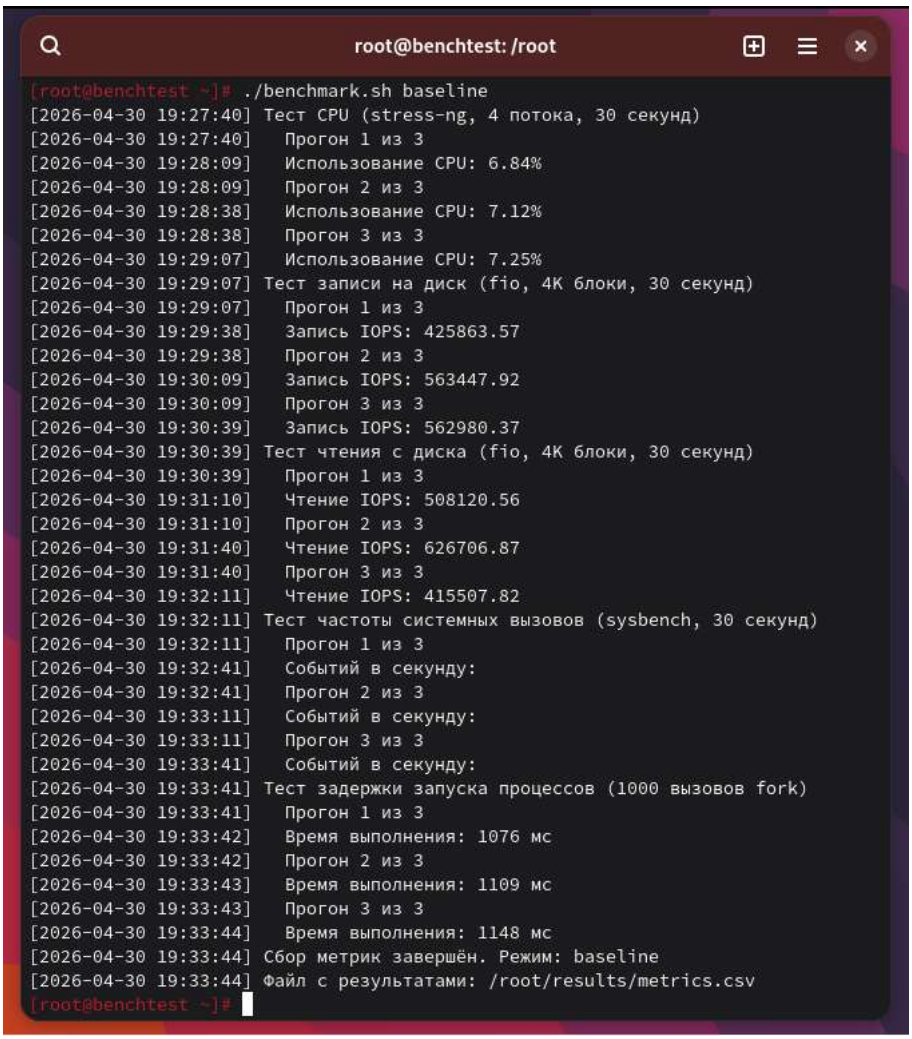

Рис. 1. Вывод терминала при сборе метрик на чистой системе (baseline)

Полученные значения зафиксированы в файле metrics.csv и составили: средняя загрузка процессора – 7,07%, средняя скорость записи на диск – 517 431 IOPS, средняя скорость чтения – 516 778 IOPS, суммарное время запуска 1000 дочерних процессов – 1111 мс. Данные значения служат контрольной точкой для расчёта накладных расходов.

Перед началом второго прогона была выполнена проверка состояния службы auditd; результат подтвердил, что служба активна и функционирует штатно (рисунок 2).

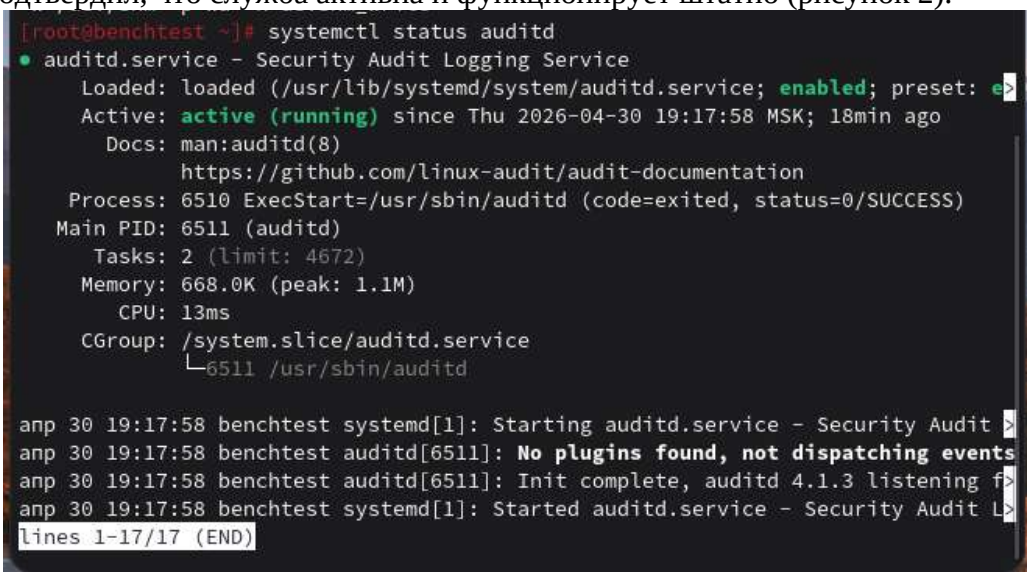

Рис.2. Статус службы auditd (active/running)

Процесс сбора метрик в режиме auditd представлен на рисунке 3.

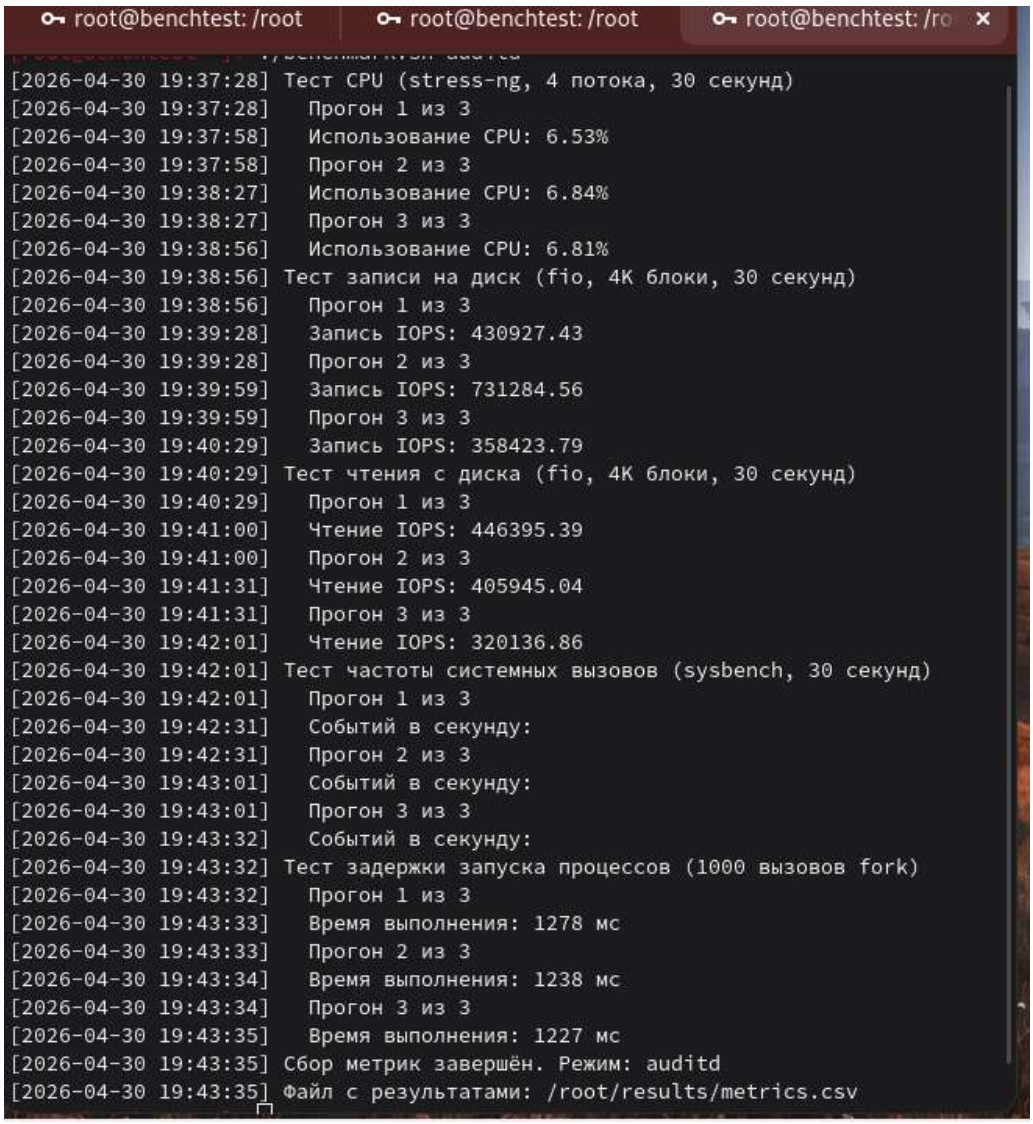

Рис. 3. Вывод терминала при сборе метрик в режиме auditd

Средние значения по трём прогонам для auditd составили: загрузка процессора – 6,73%, запись на диск – 506 878 IOPS, чтение с диска – 390 826 IOPS, задержка запуска процессов – 1248 мс. Прослеживается умеренный рост задержки при запуске процессов относительно baseline при практически неизменной загрузке процессора.

Tracee развёртывался как контейнер, факт его работы подтверждался командой docker ps (рисунок 4).

Рис. 4. Список запущенных контейнеров: Tracee активен

Для наблюдения за потоком событий в реальном времени использовалась команда docker logs -f tracee. Пример вывода с зафиксированными событиями представлен на рисунке 5.

Рис. 5. Вывод событий Tracee в режиме реального времени

Процесс сбора метрик в режиме Tracee представлен на рисунке 6.

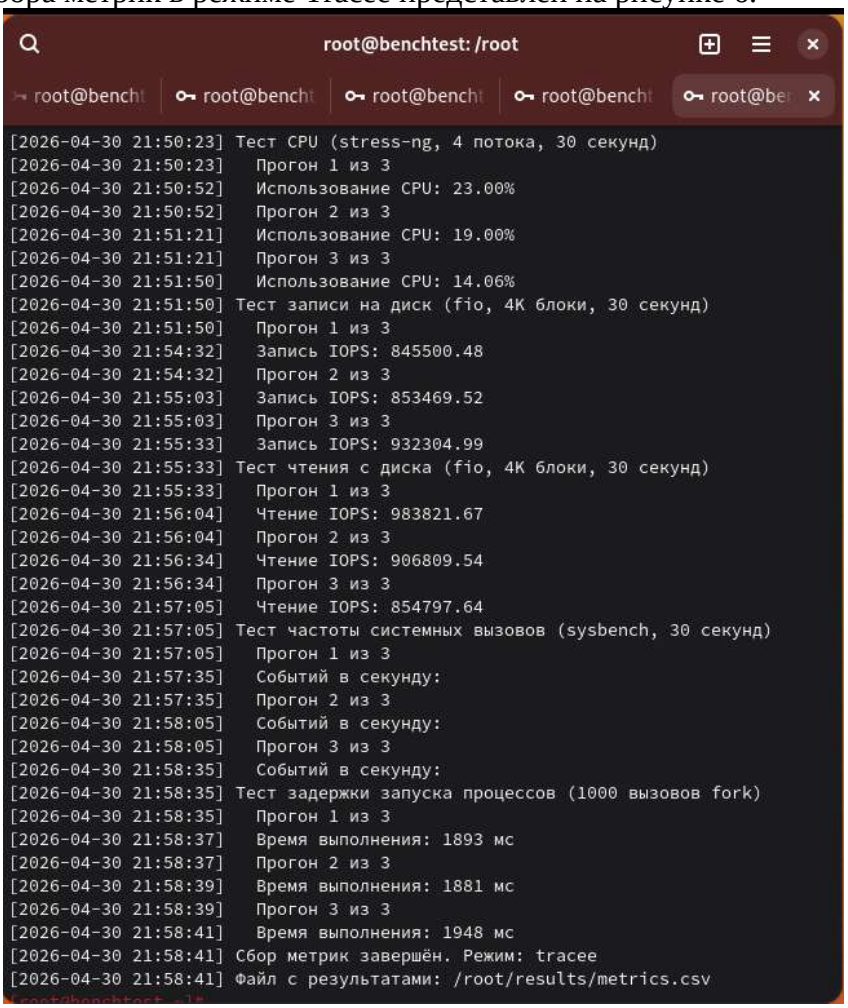

Рис. 6. Вывод терминала при сборе метрик в режиме Tracee

Средние значения для Tracee составили: загрузка процессора – 18,69%, запись на диск – 877 092 IOPS, чтение с диска – 915 143 IOPS, задержка запуска процессов – 1907 мс. Значение загрузки процессора существенно превышает как baseline, так и показатели auditd, что объясняется контейнерной архитектурой Tracee: инструмент обрабатывает поток системных вызовов в пространстве пользователя параллельно с основными рабочими процессами.

Сводное сравнение всех трёх режимов по ключевым метрикам представлено на рисунке 7. Накладные расходы каждого инструмента в процентном выражении относительно baseline – на рисунке 8.

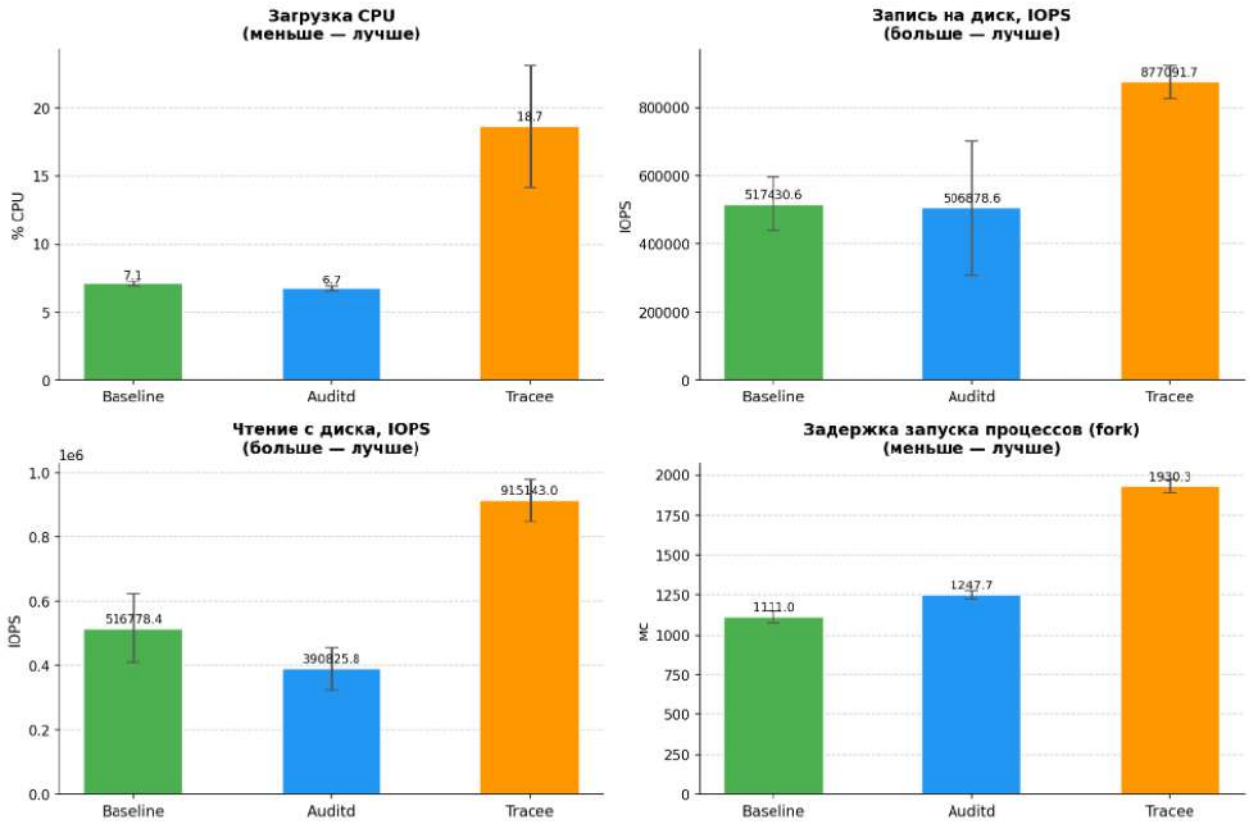

Рисунок 7 – Сравнение производительности: Baseline, Auditd, Tracee

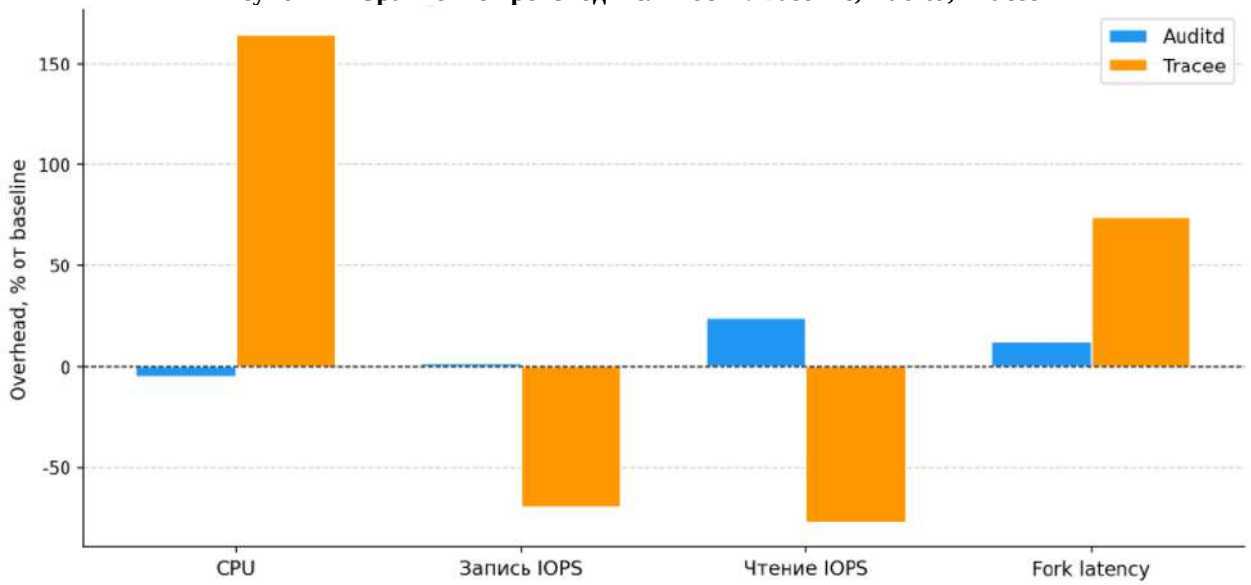

Рис. 8. Накладные расходы Auditd и Tracee относительно baseline

Анализ полученных данных позволяет сделать следующие выводы. Auditd практически не оказывает измеримого влияния на загрузку процессора: значение 6,73% незначительно отличается от базового показателя 7,07%, что укладывается в пределы статистического разброса. Tracee, напротив, демонстрирует загрузку процессора на уровне 18,69% – увеличение на 164% относительно baseline, что обусловлено дополнительными накладными расходами на межпроцессное взаимодействие.

Задержка запуска процессов наглядно демонстрирует различие в архитектурах. Baseline: 1111 мс на 1000 процессов. Auditd: 1248 мс (+12,3%). Tracee: 1907 мс (+71,6%). Рост задержки при Tracee объясняется тем, что каждый запуск процесса порождает событие, которое должно быть обработано eBPF-программой и передано в пользовательское пространство контейнера.

Показатели IOPS демонстрируют значительный разброс между прогонами, что является характерной особенностью виртуализированной среды: планировщик ввода-вывода хостовой системы вносит существенную нестабильность в результаты дисковых тестов. По этой причине метрики IOPS в рамках данного эксперимента следует интерпретировать с осторожностью.

Результаты тестирования обнаружения аномалий

В ходе тестирования для auditd наблюдение производилось за журналом /var/log/audit/audit.log посредством утилиты ausearch. Результаты представлены на рисунке 9.

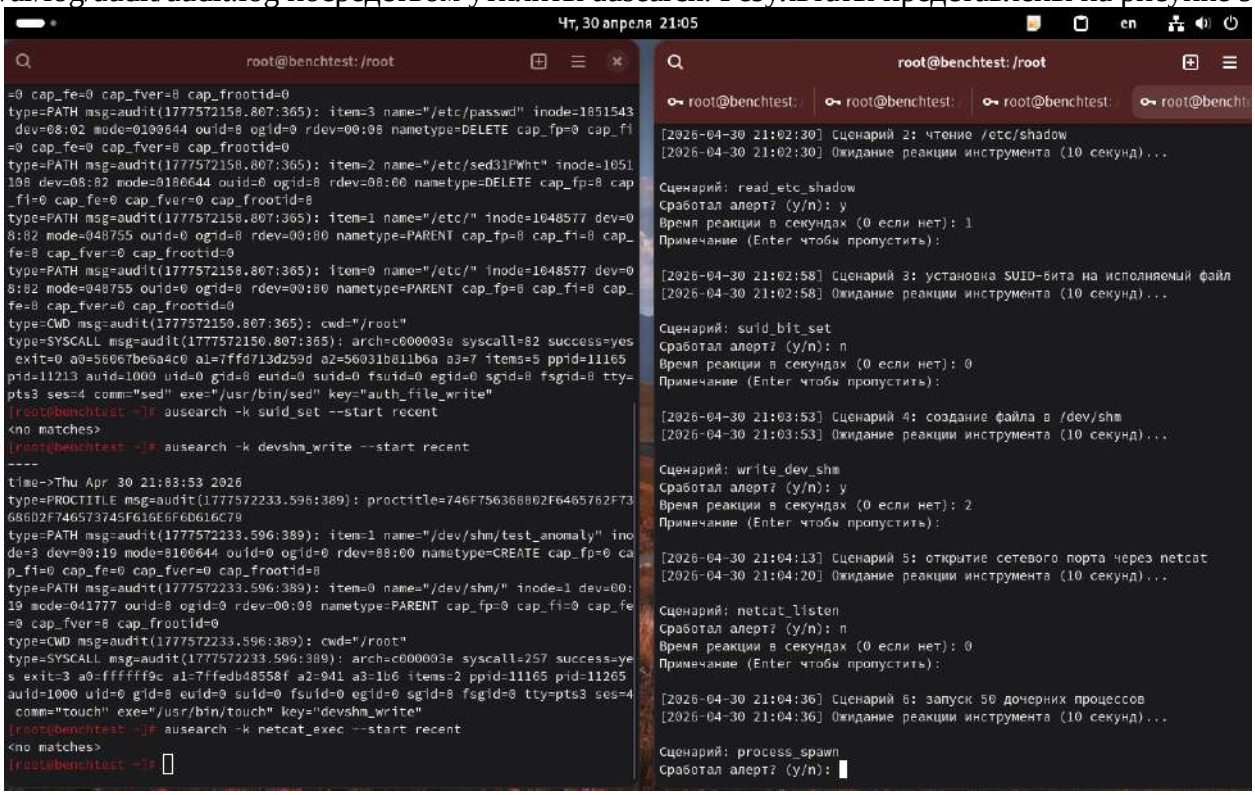

Рис. 9. Срабатывания auditd при моделировании атак

Auditd зафиксировал четыре из десяти сценариев: запись в /etc/passwd, чтение /etc/shadow, создание файла в /dev/shm и модификацию crontab. Время реакции во всех случаях составило не более двух секунд. Шесть сценариев не были обнаружены: правило для установки SUID-бита сформулировано некорректно из-за особенностей интерпретации числовых значений утилитой auditctl; сценарии с сетевым взаимодействием не сопроводились срабатыванием из-за отсутствия соответствующих правил; сценарий с чтением SSH-ключа не выполнился, поскольку файл /root/.ssh/id_rsa отсутствовал на тестовой машине.

Для Tracee наблюдение осуществлялось за выводом контейнера командой docker logs -f tracee. Результаты представлены на рисунке 10.

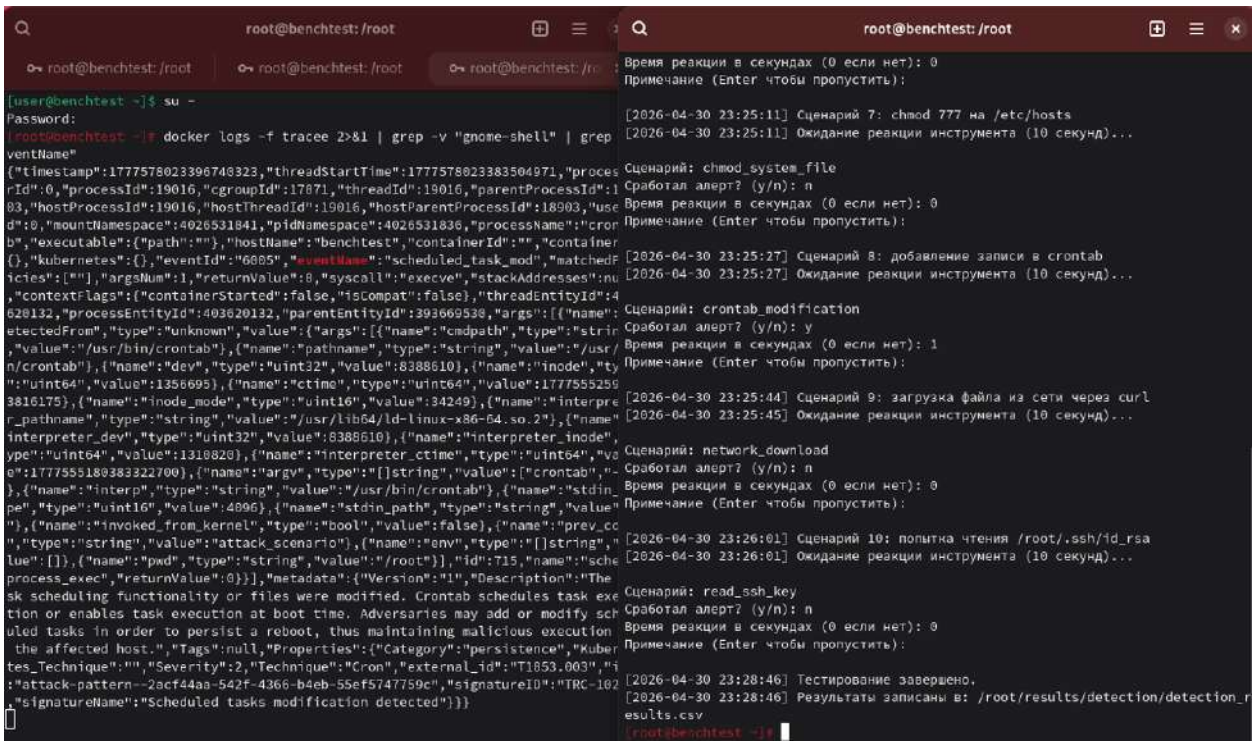

Рис. 10. Срабатывания Tracee при моделировании атак

Tracee обнаружил один из десяти сценариев – модификацию crontab. Инструмент идентифицировал событие как scheduled_task_mod с привязкой к технике MITRE ATT&CK T1053.003 и зафиксировал его в течение одной секунды. Низкий показатель объясняется ориентацией инструмента на контейнеризированные рабочие нагрузки: библиотека сигнатур оптимизирована под угрозы, актуальные для контейнерных сред [7]. Модификация системных файлов непосредственно на хосте без контейнерного контекста не попадает в приоритетные сценарии обнаружения по умолчанию.

Сводная матрица обнаружения по всем сценариям и обоим инструментам представлена на рисунке 11.

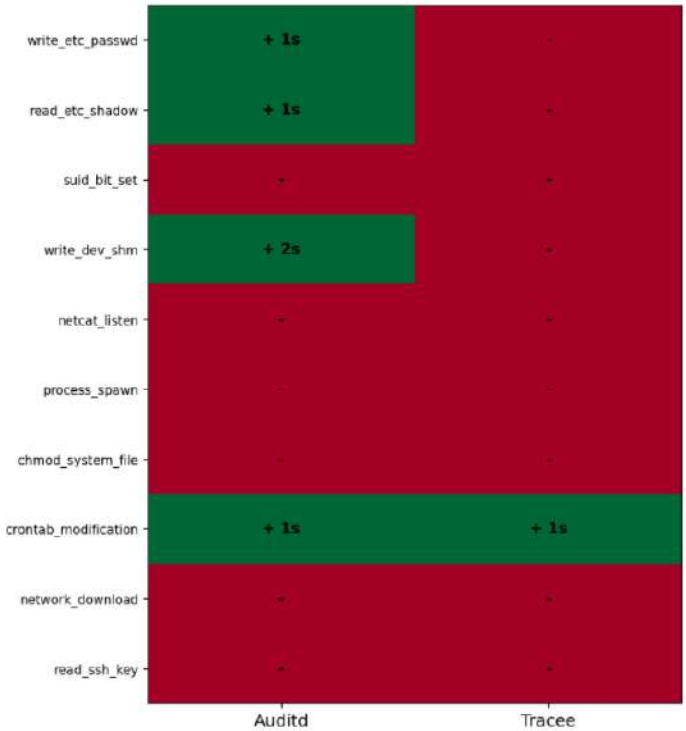

Рис. 11. Матрица обнаружения атак: Auditd и Tracee

Обобщённые результаты сравнительного анализа характеристик инструментов сведены в таблицу 2.

Таблица 2 – Сравнительный анализ характеристик инструментов

Характеристика	Auditd	Tracee
Архитектура перехвата	Подсистема аудита ядра	eBPF (kprobes + tracepoints)
Способ развёртывания	Системный пакет (в составе ОС)	Контейнер
Настройка обнаружения	Ручная (правила auditctl)	Автоматическая (сигнатуры)
Загрузка процессора (отн. baseline)	–4,8% (в пределах погрешности)	+164,4%
Задержка процессов (отн. baseline)	+12,3%	+71,6%
Обнаружено сценариев (из 10)	4	1
Зависимость от гипервизора	Отсутствует	Частичная (требуется BTF)

Выводы

Проведённое экспериментальное исследование позволило сформулировать ряд обобщающих выводов.

Во-первых, установлено, что современные eBPF-инструменты (Falco, Sysdig) несовместимы с рядом виртуализированных конфигураций на базе отечественных дистрибутивов Linux. Проблема обусловлена ограниченной поддержкой типов eBPF-программ со стороны гипервизора VirtualBox в сочетании с отсутствием необходимых заголовочных файлов ядра новых версий в репозиториях ОС Альт. Данный результат имеет самостоятельную практическую значимость для проектирования защищённых сред.

Во-вторых, сравнительное тестирование auditd и Tracee показало, что классическая подсистема аудита ядра обеспечивает минимальное влияние на производительность системы (прирост задержки процессов около 12%) и высокую предсказуемость обнаружения при условии корректной настройки правил. Современный eBPF-инструмент Tracee демонстрирует значительные накладные расходы (прирост загрузки процессора на 164%) и ограниченную применимость вне контейнеризированных сред.

В-третьих, ни один из инструментов не является универсальным решением, одинаково пригодным для всех сценариев применения. Выбор между auditd и eBPF-инструментами должен определяться условиями эксплуатации: для статических рабочих станций с чётко определённой моделью угроз предпочтителен auditd; для динамических контейнеризированных сред с широким спектром угроз целесообразно применять eBPF-инструменты при условии обеспечения совместимости программно-аппаратного стека.

Таким образом, защита рабочих станций на базе отечественных операционных систем требует комбинирования классических механизмов аудита ядра с дополнительными средствами контроля целостности, а внедрение eBPF-инструментов должно предваряться оценкой совместимости конкретных версий гипервизора, ядра и прикладного инструмента.

Библиографический список

1. Шаньгин В. Ф. Информационная безопасность компьютерных систем и сетей: учебное пособие. М.: ИД «ФОРУМ»: ИНФРА-М, 2019. 416 с.
2. Таненбаум А. С., Бос Х. Современные операционные системы. 4-е изд. СПб.: Питер, 2019. 1120 с.
3. What is eBPF? // eBPF.io. URL: https://ebpf.io/what-is-ebpf/ (дата обращения: 11.05.2025).
4. Linux Audit Documentation // The Linux Kernel Archives. URL: https://docs.kernel.org/audit/ (дата обращения: 11.05.2025).
5. Райс Л. Изучаем eBPF: программирование ядра Linux для наблюдаемости и безопасности. М.: ДМК Пресс, 2023. 350 с.

6. What is Falco? Open source runtime threat detection // Sysdig. URL: https://www.sysdig.com/learn-cloud-native/what-is-falco (дата обращения: 11.05.2025).
7. The Story of Tracee: The Path to Runtime Security Tool // Aqua Security. URL: https://www.aquasec.com/blog/open-source-container-runtime-security/ (дата обращения: 11.05.2025).
8. [Modern eBPF] libpman: tracing program type is not supported // GitHub – falcosecurity/falco, issue #2792. URL: https://github.com/falcosecurity/falco/issues/2792 (дата обращения: 11.05.2025).
9. MITRE ATT&CK Framework // MITRE Corporation. URL: https://attack.mitre.org/ (дата обращения: 11.05.2025).
10. Уймин А. Г. Сетевое и системное администрирование. Демонстрационный экзамен КОД 1.1: учебно-методическое пособие. СПб.: Лань, 2020. 480 с.

References

1. Shangin V. F. Information Security of Computer Systems and Networks: A Training Manual. M.: ID «FORUM»: INFRA-M, 2019. 416 p. (In Russ).
2. Tanenbaum A. S., Bos H. Modern Operating Systems. 4th ed. SPb.: Piter, 2019. 1120 p. (In Russ).
3. What is eBPF? Available from: https://ebpf.io/what-is-ebpf/ (accessed: 11.05.2025).
4. Linux Audit Documentation. Available from: https://docs.kernel.org/audit/ (accessed: 11.05.2025).
5. Rice L. Learning eBPF: Programming the Linux Kernel for Enhanced Observability and Security. M.: DMK Press, 2023. 350 p. (In Russ).
6. What is Falco? Open-source runtime threat detection. Available from: https://www.sysdig.com/learn-cloud-native/what-is-falco (accessed: 11.05.2025).
7. The Story of Tracee: The Path to Runtime Security Tool. Available from: https://www.aquasec.com/blog/open-source-container-runtime-security/ (accessed: 11.05.2025).
8. [Modern eBPF] libpman: tracing program type is not supported. Available from: https://github.com/falcosecurity/falco/issues/2792 (accessed: 11.05.2025).
9. MITRE ATT&CK Framework. Available from: https://attack.mitre.org/ (accessed: 11.05.2025).
10. Uimin A. G. Network and System Administration. Demonstration Exam COD 1.1: A Training Manual. SPb.: Lan, 2020. 480 p. (In Russ).

Информация об авторах

Чурсина Серафима Сергеевна – студент кафедры «Математических методов обеспечения безопасности систем», ФГАОУ ВО «РГУ нефти и газа (НИУ) имени И.М. Губкина», г. Москва, e-mail: chursinass@mail.ru.

Новикова Дарья Дмитриевна – студент кафедры «Математических методов обеспечения безопасности систем», ФГАОУ ВО «РГУ нефти и газа (НИУ) имени И.М. Губкина», г. Москва, e-mail: novikova.d@mail.ru

COMPARATIVE ANALYSIS OF REAL-TIME SECURITY MONITORING TOOLS BASED ON KERNEL AUDIT SUBSYSTEM AND eBPF TECHNOLOGIES IN ALT LINUX

D. D. Novikova[1], S. S. Chursina[1]

[1]National University of Oil and Gas «Gubkin University», Moscow, Russian Federation

Abstract. *This paper presents the results of a comparative experimental study of real-time security monitoring tools in the Alt Linux operating system environment. Two fundamentally different approaches to threat detection are examined: the classical kernel audit subsystem (auditd) and a modern eBPF-based tool (Tracee). The study identifies practical compatibility limitations of popular eBPF solutions with virtualized environments and provides a quantitative assessment of the performance overhead and anomaly detection capabilities of the tested tools. The experimental environment is deployed on Alt Linux Workstation 11.1 using the VirtualBox hypervisor. The findings contribute to the informed selection of security monitoring tools for domestic Linux distributions.*

Keywords: *security monitoring, runtime protection, auditd, eBPF, Tracee, Alt Linux, system calls, anomaly detection, kernel performance, virtualization.*

Information about the authors

Chursina Serafima Sergeevna – student of the Department of Mathematical Methods of System Safety Assurance, Gubkin Russian State University of Oil and Gas (National University), Moscow, e-mail: chursinass@mail.ru.

Novikova Daria Dmitrievna – student of the Department of Mathematical Methods of System Safety Assurance, Gubkin Russian State University of Oil and Gas (National University), Moscow, e-mail: novikova.d@mail.ru