УДК 004.056:004.75
doi:10.31799/1684-8853-2026-1-48-60
EDN: AVYJSY

Научные статьи
Articles

# Использование графов для детектирования бесфайловых атак в контейнеризированной инфраструктуре

***Е. О. Здорников***[а], *инженер-программист, orcid.org/0009-0009-0154-5153*
***Е. В. Егоров***[б], *младший научный сотрудник, orcid.org/0009-0006-4321-0069*
***И. Ю. Попов***[а], *канд. техн. наук, доцент, orcid.org/0000-0002-6407-7934, ilyapopov27@gmail.com*
[а]*Университет ИТМО, Кронверкский пр., 49, Санкт-Петербург, 197101, РФ*
[б]*Военно-космическая академия им. А. Ф. Можайского, Ждановская наб., 13, Санкт-Петербург, 197198, РФ*

***Введение:*** *бесфайловые атаки позволяют использовать память для исполнения вредоносного кода без сохранения на диск. Это существенно усложняет их обнаружение и делает традиционные методы защиты неэффективными, особенно в динамических контейнерных средах. Контейнерные системы, например Docker и Kubernetes, по сравнению с виртуальными машинами имеют меньший объем контролируемых данных, что упрощает анализ событий и построение графов активности.* ***Цель:*** *разработать метод обнаружения бесфайловых атак в контейнеризированной инфраструктуре, устойчивый к шуму и эвазивным техникам при неполном мониторинге, обеспечивающий раннюю сигнализацию до достижения критических ресурсов.* ***Результаты:*** *предложен риск-центричный метод, который основан на гетерогенном графе и графе системных вызовов. Выделяются зоны риска вокруг событий, зафиксированных с помощью расширенного фильтра пакетов eBPF, после чего формируется математическая модель зон риска с охраняемым замыканием и вычислением потенциала риска на основе поглощающих случайных блужданий. В модели дополнительно учитываются контексты контейнеров и временные параметры, что дает возможность уменьшить количество ложноположительных срабатываний. Метод позволяет локализовать зоны риска и стабильно работает при потере части событий. Эксперименты проводились на кластере Kubernetes (v1.32) под управлением Ubuntu 24.04 с использованием Tetragon (eBPF) и Falco для контроля качества. Собрано 580 эпизодов поведения контейнеров, включая атакующие и фоновые сценарии, на основе которых формировались гетерогенные графы исполнения и системных вызовов. Предложенный метод превосходит статические правила, n-gram-модели системных вызовов и глобальные графовые методы по метрикам AUROC и AUPRC, демонстрируя повышенную устойчивость к эвазивным техникам.* ***Практическая значимость:*** *разработанный метод подходит для работы с существующими сенсорами и политиками контейнерной безопасности и позволяет администраторам получать интерпретируемые зоны риска и показатели вероятности атаки.* ***Обсуждение:*** *представленная математическая модель и практическая реализация подтверждают применимость риск-центричного анализа для практического обнаружения актуальных бесфайловых угроз в современных контейнерных средах; метод масштабируем, параметризуем и не требует модификации приложений.*

***Ключевые слова*** *— бесфайловые атаки, контейнерная безопасность, eBPF, графотемпоральный анализ, runtime-детекция, графы системных вызовов, Kubernetes.*

**Для цитирования:** Здорников Е. О., Егоров Е. В., Попов И. Ю. Использование графов для детектирования бесфайловых атак в контейнеризированной инфраструктуре. *Информационно-управляющие системы*, 2026, № 1, с. 48–60. doi:10.31799/1684-8853-2026-1-48-60, EDN: AVYJSY

**For citation:** Zdornikov E. O., Egorov E. V., Popov I. Y. Using graphs to detect fileless attacks in containerized infrastructure. *Informatsionno-upravliaiushchie sistemy* [Information and Control Systems], 2026, no. 1, pp. 48–60 (In Russian). doi:10.31799/1684-8853-2026-1-48-60, EDN: AVYJSY

## Введение

Бесфайловые (fileless) атаки — это класс угроз, которые преимущественно используют оперативную память для исполнения вредоносного кода без устойчивых артефактов на диске. Современные информационные системы все чаще сталкиваются с бесфайловыми атаками. Такие атаки, в отличие от традиционных методов, не создают файлы на дисковом пространстве и не оставляют очевидных следов. В этом случае вредоносный код хранится и используется только в оперативной памяти, что затрудняет обнаружение антивирусами и системами обнаружения вторжений (Intrusion Detection System — IDS). Согласно обзорной статье [1], атаки без использования файлов становятся все более распространенными среди нарушителей информационной безопасности (ИБ). Например, исследования Ponemon Institute показывают, что вероятность их успешного проведения примерно в десять раз выше по сравнению с традиционными методами атак [1]. Также около 26 % атак APT-группировок (устойчивые целевые атаки, Advanced Persistent Threat) сопряжены с использованием методов, не связанных с файлами, таких как Living-off-the-Land, а с 2022 г. наблюдается рост бесфайловых атак на 70 % [1]. Также исследования Barr-Smith et al показали, что 80 % антивирусов не смогли обнаружить какие-либо соответствующие бесфайловые атаки, в то время как 20 % могли обнаружить только часть атак [1].

Исследование компании WatchGuard Technologies констатировало рост на 888 % бесфайловых угроз в корпоративных сетях за один год, с 2019-го по 2020-й (https://www.watchguard.com/wgrd-resource-center/security-report-q4-2020). Аналогичный отчет от Aqua Nautilus фиксирует увеличение числа таких угроз на 1400 % за 2023 г., что связано с активным применением подобной техники в облачной инфраструктуре [2]. Эти отчеты показывают, что бесфайловые атаки становятся все более популярным и опасным инструментом у нарушителей ИБ. В научных работах последних лет [1, 3, 4] проанализированы вызовы и различные методы для борьбы с такого рода атаками.

Первые бесфайловые вирусы появились в начале 2000-х годов и сразу стали востребованы среди киберпреступников благодаря своей способности действовать без сохранения на диск, тем самым обходя антивирусы. Такие вредоносные программы использовали уязвимости в браузерах, Microsoft Office или скриптовых движках Windows, чтобы загружаться напрямую в память и выполнять вредоносный код, не оставляя следов на файловой системе. Позже злоумышленники стали активно использовать встроенные системные утилиты Windows — так называемые LOLBins и LOLScripts (например, mshta, wscript, certutil, fodhelper, powershell, rundll32 и др.). Эти программы уже подписаны Microsoft, предустановлены в операционной системе и могут использоваться для загрузки файлов, обхода контроля учетных записей пользователей и запуска вредоносных скриптов в обход антивирусов.

Изначально угрозу бесфайловых вредоносных программ связывали только с Windows-системами, но за последние несколько лет они все чаще стали применяться против Linux-серверов и контейнеризированных сред. В Linux также существует аналог LOLBins, включающий в себя предустановленные исполняемые файлы (awk, tar, find, curl, wget, scp, pkexec, nmap, bash, python, perl, less и др.), которые могут быть использованы для выполнения команд, эскалации привилегий, побега из контейнера и организации обратного подключения (reverse shell). Исследования показывают, что нарушители ИБ применяют такие техники, как внедрение кода через системный вызов ptrace, создание исполняемых сегментов памяти с помощью memfd_create() и запуск вредоносных процессов в пространстве /dev/shm, что делает их невидимыми для традиционных средств защиты [5].

В контейнерных инфраструктурах бесфайловые атаки представляют особую опасность, поскольку вредоносный код может выполняться внутри контейнера и оставаться невидимым для средств мониторинга, развернутых на уровне хоста. Контейнерные системы, например Docker и Kubernetes, характеризуются высоким уровнем изоляции на уровне операционной системы, высокой производительностью за счет отсутствия накладных расходов на эмуляцию оборудования и ограниченным доступом к файловой системе. Несмотря на эти преимущества бесфайловые угрозы остаются актуальным вектором атаки для нарушителя ИБ. При этом меньший объем контролируемых данных в контейнерах по сравнению с виртуальными машинами упрощает анализ событий и построения графов активности, что позволяет точнее выделять зоны риска и снижать уровень ложных срабатываний.

В настоящей работе развивается идея риск-центричной детекции: вместо моделирования всего приложения формируются и классифицируются подграфы, что и определяет цель исследования — разработку метода обнаружения бесфайловых атак в контейнерных средах на основе графовых моделей для снижения числа ложных срабатываний и повышения устойчивости детекции при неполном мониторинге.

Предлагаемое решение основано на риск-центричном анализе системных событий контейнера, в котором данные (системные вызовы, сетевые взаимодействия и различные метаданные контейнера), полученные с помощью eBPF (extended Berkeley Packet Filter), преобразуются в гетерогенный граф. На полученном графе выделяются локальные зоны риска, формируется математическая модель взаимосвязей зон риска с охраняемым замыканием и с использованием поглощающих случайных блужданий, что разрешает вычислять потенциал риска для каждого узла и выделять аномальные активности, указывающие на бесфайловую атаку. Такой подход позволяет учитывать контекст контейнера и временную динамику событий, что уменьшает количество ложноположительных срабатываний.

Входными данными для предложенного метода являются телеметрические события, собираемые eBPF-датчиками, включая системные вызовы процессов, сетевые взаимодействия, обращения к файловым дескрипторам, операции с пространствами имен и изменениями привилегий. Эти данные агрегируются в эпизоды наблюдения за поведением контейнера и служат основой для построения гетерогенного графа исполнения.

Актуальность и научная новизна работы заключаются в интеграции риск-центричного подхода с динамическим контекстом контейнеров и временными параметрами событий, что обеспечивает устойчивость к неполному мониторингу, а также снижает уровень ложноположительных срабатываний.

Предложена модель зон риска с охраняемым замыканием для анализа поведения в контей-

нерных средах на основе графов системных вызовов.

Для достижения поставленной цели в работе решаются следующие задачи.

1. Определение модели угроз, учитывающей специфику бесфайловых атак и контейнерных сред.

2. Построение графового представления поведения процессов с классификацией подграфов, пересекающих «зоны риска»:

2.1 разрешения процессов (Linux process capabilities);

2.2 критические пути ввода-вывода и сетевые сокеты;

2.3 переходы между пространствами имен (namespace);

2.4 выполнение кода в памяти (in-memory execution).

3. Формализация математической модели зон риска и правил их пересечения.

4. Экспериментальная проверка.

Экспериментальная проверка показала, что предложенный метод способен стабильно локализовать зоны риска даже при потере части событий. При тестировании на кластере Kubernetes (v1.32) с использованием Tetragon (eBPF) и Falco метод продемонстрировал преимущество по метрикам AUPRC = 0,87 и AUROC = 0,94 по сравнению с сигнатурными правилами Falco/Tetragon, n-gram-моделями системных вызовов и глобальными графовыми подходами без локализации. Отмечено также более медленное падение качества при эвазивных $k$-модификациях, что указывает на повышенную устойчивость детекции в реальных условиях эксплуатации.

Такой подход снижает поток событий и уровень шума при анализе бесфайловых техник, оставаясь совместимым с eBPF-датчиками и практиками политик контроля во время выполнения (runtime).

## Обзор существующих решений

С развитием угроз развивались и средства защиты. Например, в октябре 2018 г. Azure Security Center от компании Microsoft выпустил первое решение для Windows, а уже в 2020-м представил предварительное решение для обнаружения бесфайловых атак на Linux-системах (https://azure.microsoft.com/en-us/blog/fileless-attack-detection-for-linux-in-preview/). Тем не менее эти подходы имеют существенные недостатки. Стандартные IDS и SIEM (управление событиями и информацией безопасности, Security Information and Event Management) малоэффективны против новых бесфайловых угроз. Антивирусные решения в основном используют статический и сигнатурный анализ и не в состоянии детектировать процессы, которые исполняются только в оперативной памяти [5, 6]. IDS-системы, работающие на основе анализа системных вызовов контейнеров, перегружены и довольно часто могут давать большое количество ложных срабатываний, что снижает их практическую ценность [6]. К современным подходам можно отнести следующие решения.

1. Анализ оперативной памяти (memory forensics) позволяет выявить следы вредоносной активности, но требует сложных дампов и ручной обработки [4].

2. Модели на основе системных вызовов (Bag-of-Syscalls) обеспечивают обнаружение аномалий в реальном времени, не требуя предварительных знаний о поведении контейнера [7], однако чувствительны к шуму и могут выдавать большое количество ложных срабатываний при изменении рабочей нагрузки.

3. Использование eBPF-сенсоров (расширенный фильтр пакетов Беркли, extended Berkeley Packet Filter) и kernel-based мониторинга дает возможность отслеживать вызовы на уровне ядра и выявлять нетипичные взаимодействия процессов [8], но требует повышенных привилегий и может оказывать влияние на производительность при высокой интенсивности событий. Эти вызовы могут обрабатываться как статическими правилами, так и методами машинного обучения и нейросетями [9].

4. Графовые модели поведения — один из наиболее перспективных подходов: строятся ориентированные графы системных вызовов, где выделяются подграфы зон риска (например, обращения к docker.sock, попытки изменения пространств имен или доступ к привилегированным файлам). Эти подграфы анализируются с использованием методов машинного обучения и графовых нейронных сетей (Graph Neural Network — GNN), что позволяет обнаруживать сложные и ранее неизвестные паттерны атак [10–12].

Работы последних двух лет показывают эффективность графов зависимостей/происхождения для анализа внутриконтейнерных атак и побегов:

— метод Container Escape Detection [13] на основе графов зависимостей реконструирует причинно-следственные связи и маркирует «контейнерные» процессы на графе, повышая полноту обнаружения escape-сценариев;

— Phoenix [14] строит граф происхождения данных из событий аудита контейнера, что позволяет выявлять «злонамеренные последовательности» и поддерживает динамическую защиту при наличии уязвимых компонентов;

— CORAL [14] использует логические графы атак для онлайн-оценки рисков и выявления ла-

терального перемещения в контейнерных средах.

Данные работы не лишены недостатков. Container Escape Detection ограничен узким сценарием побега из контейнера и требует полного построения графа зависимостей, что повышает чувствительность к шуму и нагрузку на ресурсы. Phoenix ориентирован на заранее известные последовательности системных вызовов и слабо применим к бесфайловым и in-memory атакам. CORAL зависит от полной картины событий и ресурсов контейнера, что снижает эффективность при неполном мониторинге.

## Графовое представление поведения системы с выделением зон риска

В настоящей работе модель угроз и способ представления поведения системы рассматриваются совместно, поскольку выбранный способ наблюдения и формализации исполнения напрямую определяет класс обнаруживаемых атак и допустимые допущения безопасности. Основной акцент в работе сделан на исследовании бесфайловых угроз в Linux-контейнерах (Docker/CRI-O, Container Runtime Interface/containerd) под управлением Kubernetes и аналогичных оркестраторов. Фокус — рантайм-поведение, извлекаемое из ядра (eBPF), так как традиционные антивирусы/IDS/SIEM слабо наблюдают in-memory активность и перегружаются шумом, особенно внутри namespaces/cgroups контейнеров. Эти ограничения отмечены и в нашем введении. Предполагается:

1. Доверенная база — ядро хоста и eBPF-сенсор (Falco/Tetragon) не скомпрометированы; доступ к данным телеметрии защищен. На практике именно ядровый контроль исполнения (check-on-execution) показал применимость против in-memory кода до перехода в пространство пользователя (userspace), что важно для бесфайловых сценариев.

2. Наблюдаемость — перехват системных вызовов и LSM-событий (модули безопасности Linux, Linux Security Modules) с контейнерным контекстом (cgroup/kubernetes-labels) возможен и экономичен за счет in-kernel фильтрации/действий (Tetragon Selectors, Falco).

3. Граница нашей работы — анализ рантайм-поведения (включая попытки побега/латерали), но мы не решаем уязвимости цепочек поставок (supply chain) и не рассматриваем компрометацию ядра/сенсора.

Атакующий — удаленный либо внутренний пользователь, получивший начальную точку входа в контейнере: удаленное выполнение кода в приложении, скомпрометированные секреты/образы, ошибочные настройки RBAC (Role-Based Access Control)/Pod-Security, уязвимые скрипты. Привилегии — от непривилегированных до полупривилегированных.

Целями нарушителя могут быть:

1) загрузка и исполнение бесфайловой вредоносной нагрузки;

2) разведка, закрепление, извлечение токенов и внедрение в долгоживущие процессы;

3) попытки выбраться из контейнера (namespace-переходы, mount/pivot_root при наличии CAP_SYS_ADMIN), злоупотребление сокетами рантайма (/var/run/docker.sock, CRI/containerd), доступ к kubelet-артефактам.

После определения модели угроз для бесфайловых атак в контейнерной среде на следующем этапе проводится построение формального графового представления активности системы с помощью графов исполнения, основанных на анализе потока событий ядра, с одновременным выделением зон риска.

1. Поток событий → граф исполнения.

Собирается поток событий ядра (syscalls, LSM-хуки) и преобразуется в графы.

Системные вызовы (syscalls) — интерфейс обращений пользовательских процессов к ядру операционной системы (open, execve, setns, bpf); по ним фиксируются ключевые действия процессов.

LSM-хуки — точки перехвата в ядре, позволяющие контролировать/логировать действия (доступ к файлам, сетям и пр.).

В работе [12] рассматриваются различные графовые модели для анализа вредоносного поведения в системе. Для нашей задачи выявления узких зон риска бесфайловых атак наиболее подходят модели гетерогенного графа (Heterogeneous Graph — HG) и графа системных вызовов (System Call Graph — SCG). Эти модели эффективно отражают ключевые аспекты поведения системы в контейнере: HG моделирует разнообразные объекты и их взаимодействия (процессы, файлы, сокеты, namespaces и т. д.), а SCG фокусируется на последовательности системных вызовов и связях между ними.

Другие рассмотренные модели оказались менее применимыми, так как не учитывают специфику бесфайловых атак или недостаточно полно отражают структуру взаимодействий внутри контейнера.

На основе HG и SCG строим граф исполнения событий, что позволяет выделить узкие зоны риска, характерные для бесфайловых угроз. Такой метод обеспечивает требуемый уровень детализации и полноту анализа, необходимые для эффективного выявления потенциальных атак.

В гетерогенном графе:

— узлы (точки): процессы, файлы, сокеты, namespaces, capability-состояния;

— ребра (линии): взаимодействия между узлами (например, процесс открыл файл, присоединился к namespace);

— метки на ребрах: тип действия, аргументы вызова, время события.

В графе системных вызовов:

— узлы (точки): системные вызовы и процессы;

— ребра (линии): последовательность и связи вызовов (какой syscall вызвал следующий, какие процессы взаимодействуют через syscalls);

— метки на ребрах: тип системного вызова, параметры, временные метки.

Таким образом, формируется графовое представление активности контейнера, в котором можно анализировать взаимодействия и выявлять потенциальные зоны риска. В этом графе сразу выделяем границы — опасные ресурсы и состояния, вроде docker.sock или host namespace.

2. Якоря риска (risk anchors).

Это простые логические события в графе, которые напрямую связаны с известными методами бесфайловых атак. На основе анализа литературы для более четкого представления зон риска в бесфайловых атаках на Linux-контейнеры выделены основные способы, которые чаще всего используют нарушители ИБ.

2.1 Использование временных файловых систем /dev/shm и /run/shm для размещения и исполнения кода в памяти без записи на диск.

2.2 Применение системного вызова memfd_create() для создания анонимных в памяти файлов и исполнения вредоносных бинарников.

2.3 Злоупотребление механизмом LD_PRELOAD для внедрения библиотек и перехвата вызовов функций процесса.

2.4 Использование ptrace для отладки и инъекции кода в другие процессы, обхода защитных механизмов [5].

2.5 Активное применение предустановленных в Linux бинарников из набора GTFObins (например, awk, tar, find, curl, bash) для выполнения команд, установки обратных подключений, эскалации привилегий и побега из контейнеров [15].

2.6 Эксплуатация доступа к сокетам контейнерного рантайма, таким как /var/run/docker.sock, и неправильных namespace-настроек для побега и латерального перемещения [13, 16].

2.7 Извлечение и использование секретов (токенов, ключей) для закрепления и дальнейшего распространения атаки [6].

2.8 Выделение аномальных последовательностей системных вызовов и LSM-событий для раннего обнаружения бесфайловых техник [7, 11].

3. Графотемпоральная модель зоны риска.

Шаги определения зон риска показаны на рис. 1, *а–в*.

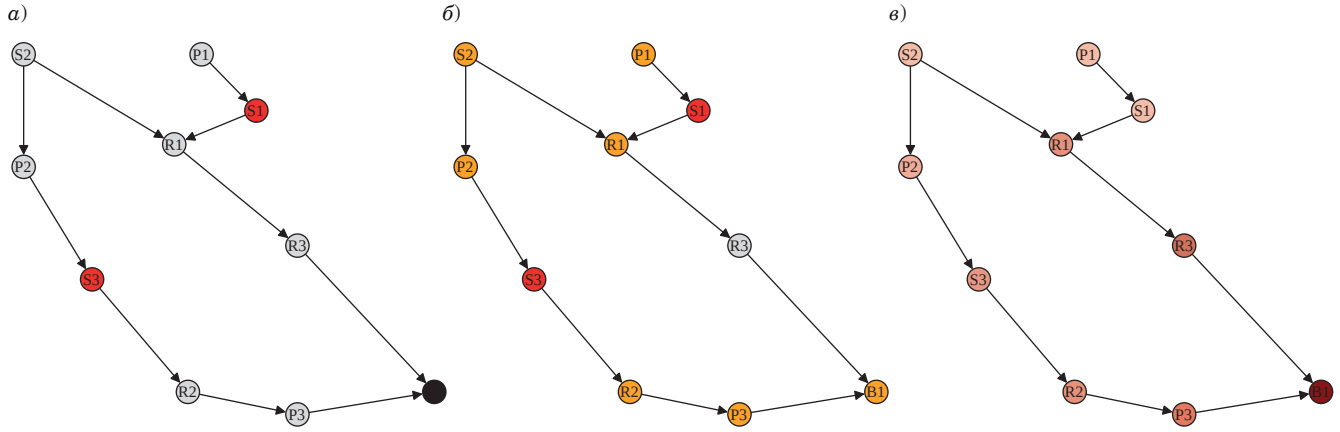
Серые узлы — обычные элементы графа (процессы, ресурсы, syscalls), которые пока считаются безопасными.

Красные узлы — якоря риска, найденные по логическим правилам над событиями ядра. В примере это два системных вызова S1 и S3, которые указывают на возможные признаки бесфайловой атаки (например, memfd_create и fexecve).

Черный узел — граница риска, т. е. точка, за которой находится критичный объект. В нашем случае это может быть сокет Docker (/var/run/docker.sock) или host namespace.

На этом этапе есть понимание только того, где начинается зона опасности (якоря) и где граница, до которой нельзя позволить достигнуть.

Оранжевые узлы — новые вершины, добавленные в процессе охраняемого замыкания (guarded closure). Узлы добавляются, если они находятся в «смысловой» близости к якорям: тот же контейнер, связанные ресурсы или syscalls,




■ ***Рис. 1.*** Графотемпоральная модель зоны риска: *а* — исходный граф с якорями; *б* — фронт замыкания; *в* — тепловая карта риска

■ ***Fig. 1.*** Graph-temporal risk zone model: *а* — initial graph with anchors; *б* — closure front; *в* — risk heat map

ведущие к границе. Полученный подграф образует структурный контур зоны риска, т. е. минимальную окрестность, достаточную для представления всех потенциальных причинно-временных цепочек атаки, которые в дальнейшем могут привести к достижению опасных границ.

Узлы на рис. 1, *в* окрашены по интенсивности красного цвета в зависимости от значения потенциала риска $u(v)$. Чем краснее, тем выше вероятность, что этот узел «дойдет» по допустимым ребрам до границы (в примере — B1). Чем светлее, тем ниже риск — узел либо далеко, либо связан через безопасные пути. Потенциал вычислен через поглощающие случайные блуждания (absorbing random walk): поглощающие состояния — это границы риска, а веса ребер зависят от их «опасности» (syscalls вроде setns, mmap, bpf дают высокий вес). Далее переходим от «геометрии» зоны (замыкание) к картине распределения риска. Это позволяет отсечь шум и выделить только те узлы, которые реально могут привести к атаке. При выборе порога θ получаем финальный контур зоны риска.

Данный способ отличается от стандартных методов, использующих полный граф или сигнатурные правила. В научной литературе встречаются подходы на основе графов зависимостей (dependency graphs), происхождения данных (provenance graphs) [17] и GNN-моделей [18, 19]. Однако подход, в котором зона риска сначала формируется с помощью охраняемого темпорального замыкания (guarded closure), а затем ее граница уточняется с использованием поглощающего случайного блуждания с ограничениями по меткам (label-constrained absorbing random walk), в рассмотренной нами литературе не выявлен. Метод обладает устойчивостью к шуму, так как замыкание гарантирует полноту включения релевантных узлов, а расчет потенциала отсекает малозначимые ветви.

## Математическая модель зоны риска

*Поток событий*. Пусть $E_{evt} = \{e_i\}_{i=1}^{N}$ — упорядоченный по времени поток событий. Каждое ядровое событие задаем кортежем

$$e_i = (p_i, s_i, a_i, r_i, c_i, t_i),\ i = 1, \ldots, N, \qquad (1)$$

где $p_i$ — процесс (PID, имя, контейнерный PID); $s_i \in S$ — системный вызов или LSM-событие (например, execve, memfd_create, setns); $a_i \in A$ — релевантные аргументы вызова (дескриптор, флаги, маска capabilities); $r_i \in R$ — ресурс (файл, сокет, объект namespace и т. п.); $c_i \in C$ — контейнерный контекст (cgroup, pod, image, набор событий); $t_i \in R^+$ — время события.

*Граф исполнения*. Из потока событий строим темпоральный атрибутированный ориентированный мультиграф

$$G = (V, E, \lambda, \tau), \qquad (2)$$

где $V$ — множество вершин: процессы, ресурсы, пространства имен, состояния привилегий; $E \subseteq V \times V$ — мультимножество ориентированных ребер (причинно-временные связи между узлами); $\lambda$: $E \to \Lambda$ — метка ребра (тип вызова, объем байтов, код возврата, изменения capabilities); $\tau$: $E \to R_{>0}$ — функция временных меток (время события).

*Границы*. Критические вершины (границы риска) задаем множеством

$$B \subseteq V, \qquad (3)$$

это вершины, достижение которых нарушителем считается критическим (например, /var/run/docker.sock, containerd.sock, host-namespace, состояния с CAP_SYS_ADMIN).

*Якоря риска* — событие $e \in E_{evt}$, удовлетворяющее булевой формуле $\Phi(e)$ над предикатами. В предыдущем разделе были перечислены основные способы атаки, отражающие ключевые приемы бесфайловых атак на Linux-контейнеры. Для формализации математической модели эти способы объединены в шесть формальных предикатов, каждый из которых может включать несколько исходных способов атаки, предложенных в табл. 1.

В случае появления новых способов бесфайловых атак модернизация формальных предикатов в математической модели не представляет сложности, так как любой новый способ возможно отнести к одному из уже определенных предикатов, что обеспечивает гибкость модели и упрощает ее расширение для поддержки новых угроз.

Множество якорей

$$A = \{e \in E_{evt} | \Phi(e) = 1\}. \qquad (4)$$

*Охраняемое темпоральное замыкание*. Чтобы из множества «точек-якорей» получить структурный контур зоны риска, мы распространяем влияние по графу исполнения, ограничивая допустимые переходы как по смыслу, так и по времени. Процедура состоит из трех логически связанных шагов.

1. *Охрана ребер*. Зададим предикат

$$\Gamma: E \to \{0, 1\}, \qquad (5)$$

который разрешает распространяться только по допустимым ребрам (например, в том же контейнере; или явные переходы границы с фик-

■ ***Таблица 1.*** Способы бесфайловых атак и формальные предикаты
■ ***Table 1.*** Fileless attack techniques and formal predicates

| Исходный способ атаки | Формальный предикат |
|---|---|
| Использование временных файловых систем для размещения и исполнения кода в памяти без записи на диск | $P_{mem}$ |
| Применение системного вызова memfd_create() для создания анонимных файлов и исполнения исполняемых файлов | $P_{mem}$ |
| Злоупотребление механизмом LD_PRELOAD для внедрения библиотек и перехвата вызовов функций процесса | $P_{mem}$ |
| Использование ptrace для отладки и инъекции кода в другие процессы, обхода защитных механизмов | $P_{inj}$ |
| Активное применение LOLBins (GTFObins) — выполнение команд, подключений, эскалации и побега из контейнера | $P_{sock}$ |
| Эксплуатация доступа к сокетам контейнерного рантайма и namespace-настроек для латерального перемещения | $P_{sock}, P_{ns}$ |
| Извлечение и использование секретов Linux capabilities для закрепления и дальнейшего распространения атаки | $P_{cap}$ |
| Аномальные последовательности системных вызовов и LSM-событий | $P_{bpf}$ |

сацией capabilities). На основе предиката $\Gamma$ для произвольного множества узлов $S \subseteq V$ вводится оператор охраняемого темпорального замыкания.

2. *Оператор замыкания*. Для множества узлов $S \subseteq V$ определим

$$C^{k,\Delta}_{\Gamma,L}(S) = S \bigcup \{v \in V \mid \exists u \in S : u \overset{<k,\Delta}{\underset{\Gamma,L}{\Rightarrow}} v\}. \qquad (6)$$

Здесь $k \in N$ — радиус по ребрам (максимальное число переходов в путях), чем больше $k$, тем выше полнота, но возврат затраты и шум; $\Delta \in R$ — максимально допустимое «временное растяжение» пути (секунды — та же единица, что у $t_i$).

3. *Скелет зоны риска* — наименьшая фиксточка (минимальное неподвижное множество) этого оператора, порожденная узлами якорей. Оператор $C_r^{k,\Delta}$ монотонен, а значит, существует минимальное неподвижное множество

$$Z^* = \mu SC_r^{k,\Delta}(S \bigcup nodes(A)), \qquad (7)$$

где $nodes(A) = \{src(e), dst(e) | e \in A\}$; $A \subset E_{evt}$ — якоря-события. После выделения якорей риска расширяем зону вокруг якорей так далеко, как это необходимо и достаточно, чтобы покрыть все потенциальные причинно-временные цепочки атаки в пределах $k$ переходов и окна $\Delta$, но не дальше.

Приведем ключевые свойства оператора: (i) $C$ монотонен $\Rightarrow$ $\mu$-точка существует; (ii) оператор вычисляется итеративно локально вокруг $A$, и время оценивается как $O(k|E_{\text{лок}}|)$, где $E_{\text{лок}} \to \subseteq E$ — подмножество ребер, достижимых из $nodes(A)$ при $\Gamma(e) = 1$ и в пределах ограничений $k$, $\Delta$; (iii) параметры $k$, $\Delta$ управляют «шириной» зоны и бюджетом телеметрии.

*Поле риска на подграфе (выделение финальной зоны).* Скелет $Z^*$ может содержать «хвосты», которые структурно близки, но практически малоопасны. Поэтому вводим полевую оценку риска на индуцированном подграфе. Отделяем исходный мультиграф и индуцированный:

$$H = G[Z^*] = (V_H, E_H, \lambda|E_H, \tau|E_H), \qquad (8)$$

где $V_H = Z^*$ и $E_H = \{e \in E | src(e), dst(e) \in V_H\}$.

Далее формируем поглощающие случайные блуждания с ограничениями по меткам:

— веса ребер $w$: $E_H \to R_{>0}$: опасные метки (например, setns, mmap, bpf) получают больший вес;

— поглощение на границах: узлы $B \cap V_H$ делаем поглощающими (если блуждание дошло до границы, оно «исчезает» там);

— ограничение допустимых цепочек: фиксируем язык $L$, это регулярные паттерны TTP (Tactics, Techniques and Procedures) в терминологии MITRE ATT & CK. Запрещаем все переходы, которые не соответствуют $L$. Технически это можно реализовать как декартово произведение графа $H$ с автоматом $A_L$.

В результате, решая стандартную задачу поглощения, получаем вектор вероятностей

$$\overline{\mathbf{u}} : V^H \to 0,1, \overline{\mathbf{u}}(v) = \Pr\{\text{дойти из } v \text{ до } B \bigcap V_H\}. \qquad (9)$$

Это и есть «потенциал риска»: чем ближе узел к опасным границам по допустимым ребрам, тем больше $\bar{\mathbf{u}}(v)$.

Граничные условия для потенциала риска:

1) множество поглощающих вершин: $B \cap V_H$ — узлы, достижение которых критично (границы риска). Для них фиксируем $\bar{\mathbf{u}}(v) = 1$;

2) множество безопасных опорных вершин (специальные вершины в графе исполнения, которые принимаются как «абсолютно безопасные» начальные точки для расчета потенциала риска): $S_{safe} \subseteq V_H$ — вершины, гарантированно не вовлеченные в атаку (например, долгоживущие системные процессы, инфраструктурные демоны). Для них фиксируем $\bar{\mathbf{u}}(v) = 0$;

3) для остальных вершин $\mathbf{u}$ определяется как решение задачи гармоничности

$$\bar{\mathbf{u}}(v) = \frac{\sum_{(v,x)\in E_H} w(v, x) * \bar{\mathbf{u}}(x)}{\sum_{(v,x)\in E_H} w(v, x)},$$
$$v \in V_H (B \cup S_{safe}). \qquad (10)$$

Финальная зона риска — уровневое множество потенциала

$$Z_\theta = \{v \in V^H \mid \bar{\mathbf{u}}(v) \geq \theta\}, \qquad (11)$$

где $\theta \in (0, 1)$ — заданный порог (калибруется по ROC/PR на валидации).

Эквивалентно можно решить дискретную задачу Дирихле для взвешенного лапласиана на $H$: задать $\mathbf{u} = 1$ на $B$, $\mathbf{u} = 0$ на безопасных «сидерах» (например, долгоживущие системные процессы) и получить гармоническое поле $\mathbf{u}$; затем порогом $\theta$ выделить $Z_\theta$.

Сигнал тревоги подается после вычисления $H$, $u$, $Z_\theta$, если выполнены следующие условия:

1) существуют $a \in A_v$ и $b \in B \cap V_H$ такие, что внутри $H$

$$a \overset{\Gamma,\ L}{\underset{<k,\Delta}{\Rightarrow\Rightarrow}} b; \qquad (12)$$

2) $\max_{v \in A_v} u(v \geq \theta)$.

## Испытательный стенд

Практическая часть работы проводилась на операционной системе Ubuntu (24.04.3 LTS), с развернутым Kubernetes (v1.32, containerd v2.0) кластером в конфигурации один мастер и два воркер-сервера. В качестве сетевого плагина использован Cilium. Все LSM-события и системные вызовы собраны с помощью Tetragon, который в свою очередь использует eBPF и развернут в виде DaemonSet (privileged, hostPID, hostNetwork). В Tetragon заданы минимальные правила, которые не блокируют, а только логируют указанные вызовы с аргументами и контекстом. Для контроля качества параллельно запускали Falco с включенными правилами “fileless via memfd_create” и “open docker.sock”. Falco похож на Tetragon, но регистрирует только системные вызовы (syscalls). В качестве целевого образа контейнера был взят nginx:1.25-alpine. Этот образ широко используется в промышленной эксплуатации и содержит типичный набор утилит BusyBox. Построение графа и зон риска реализовано на Python 3.11.

Паттерны S1–S6 представляют собой заведомо подозрительное поведение и являются якорями, B1–B2 — доброкачественные эпизоды (команды администрирования, статика NGINX, сетевой шум без доступа к границам). В табл. 2 сведены все сценарии генерации телеметрии.

Для каждого сценария проведено по 60 повторов с разными задержками, вставками безвредных вызовов и перестановками (эвазивность) и по 100 повторов фоновой активности. Итого 580 эпизодов, из них 240 с достижением границы риска и 340 без достижения границ. Для каждого повтора S-сценария мы случайно выбирали:

1) задержки между вызовами из диапазона 0,1–2,0 с;

2) вставки до $m = 5$ нейтральных системных вызовов из словаря {stat, read, getpid и др.};

3) перестановки соседних блоков длиной 1–3 вызова. Параметр сложности эвазии обозначаем $k$ (число модификаций).

Эпизод получает метку 1, если в полном графе событий эпизода действительно зафиксиро-

■ ***Таблица 2.*** Сценарии генерации телеметрии
■ ***Table 2.*** Telemetry generation scenarios

| ID | TTP-паттерн | Правила генерации (упрощенно) |
|---|---|---|
| S1 | memfd → exec | memfd_create → fexecve в ≤5s; ресурс memfd://… |
| S2 | X∧W | mmap/mprotect с prot = X |
| S3 | Ptrace | Одиночный ptrace_attach к целевому pid в том же pod |
| S4 | /dev/shm | write в /dev/shm/foo → execve / tmp/x (обычный true) |
| S5 | setns | setns PID/MOUNT → mount read-only |
| S6 | docker.sock | open /var/run/docker.sock (без сокетного обмена) |
| B1 | Benign web | open/read/write статики nginx |
| B2 | Benign batch | fork/exec воркера, сетевые подключения к базе данных |

вано достижение границы риска: открыт docker.sock/containerd.sock, выполнен setns в host-namespace, получена повышающая capability и другие подобные ситуации. Во всех прочих случаях метка 0. Разметка производится автоматически по событиям ядра и проверяется вручную на случай неоднозначностей. Разбиение на тренировочную и валидационную выборку случайное.

Построение графа и зоны риска состоит из четырех этапов.

1. Формирование графа исполнения. Для каждого эпизода из сырых eBPF-событий сформировали гетерогенный граф исполнения. Узлы представляли процессы, файлы/сокеты, пространства имен и состояния привилегий; ребра кодировали факты взаимодействия. Параллельно строился граф системных вызовов, фиксирующий причинные последовательности по PID/TID (Process/Thread ID) и времени. Такой двойной взгляд (HG+SCG) сохраняет и структуру ресурсов, и темпоральную причинность.

2. Локализация зоны вокруг якорей. Из множества якорей индуцировали локальную зону риска, расширяя граф от якорей на ограниченную глубину и в заданном временном окне. Фильтрация по предикату релевантности исключала нерелевантные переходы (другие контейнеры, несвязанную активность). Результат — индуцированный подграф $H$, существенно меньший по размеру, но сохранивший достижимые пути к границам риска.

3. Ранжирование по потенциалу риска. На подграфе $H$ вычислялся потенциал риска: границы риска (например, docker/containerd-сокеты, host-namespace, получение критичных capabilities) рассматривались как поглощающие вершины, ребра взвешивались по типу действия и контексту. Полученное поле значений интерпретировалось как степень достижимости границ из соответствующих вершин. Этот расчет выполнялся пакетно для каждого эпизода и не изменял исходную телеметрию.

4. Сводка в скор и оценка метрик. Для эпизода агрегировали потенциал в один числовой скор (максимум или высокий перцентиль по вершинам подграфа). Набор пар (score, label) по тестовой выборке использовали для построения PR/ROC-кривых и вычисления AUPRC/AUROC (площадь под PR-кривой / площадь под ROC-кривой), а также для оценки задержки детекции и устойчивости к эвазивности и потере событий.

Предложенный в статье метод рассматривался в сравнении с тремя базовыми методами:

1) статическими правилами (Falco/Tetragon): сигнатуры/правила преобразовывались в оценки через нормированную «важность» срабатывающих правил (максимум по эпизоду);

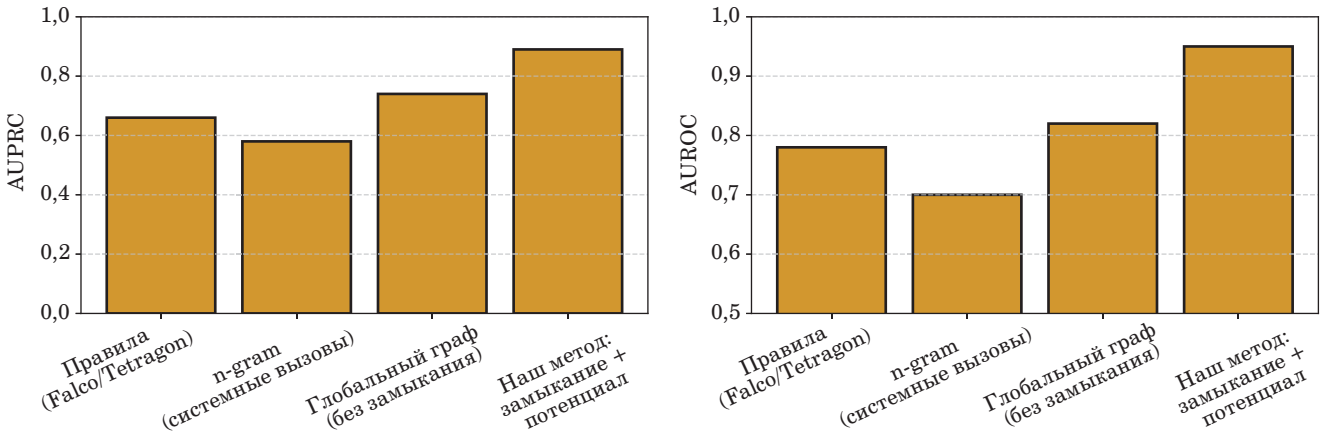
2) n-gram (syscalls): оценка «аномальности» последовательности системных вызовов. Рассматривали последовательность вызовов как язык и оценивали, насколько вероятен текст. Оценка — это мера маловероятности эпизода с последующей нормализацией [0, 1];

3) Global HG: тот же потенциал $u$, но на полном графе $G$ без локализации замыканием.

## Результаты

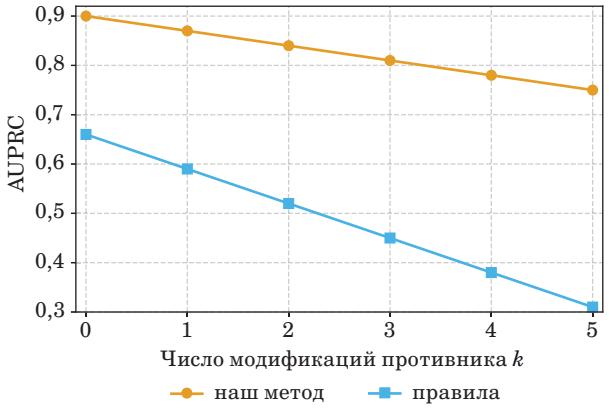
На тестовой выборке предложенный риск-центричный метод показал AUPRC = 0,87 и AUROC = 0,94, что превосходит все базовые подходы (рис. 2).

Атакующие могут маскироваться и выполнять легитимные вызовы между якорями, менять порядок нейтральных действий и добавлять задержки. Эти действия моделируются как




■ ***Рис. 2.*** Сравнение метрик AUPRC и AUROC по методам

■ ***Fig. 2.*** Comparison of metrics AUPRC and AUROC by methods




■ ***Рис. 3.*** Сравнение робастности к эвазивности
■ ***Fig. 3.*** Comparison of robustness against evasion

$k$-модификаций в эпизоде. На рис. 3 показано снижение качества детекции при усложнении (маскировке) действий противника.

При росте числа модификаций противника $k$ качество нашего метода падает значительно медленнее: AUPRC снижается с 0,90 до 0,75 (≈ 17 %), тогда как для правил — с 0,66 до 0,31 (≈ 53 %). Это подтверждает робастность локализованного анализа «зоны риска» к эвазивным изменениям и сохраняет практическую точность даже при $k = 5$. На практике это означает более стабильные оповещения без постоянной перенастройки правил при попытках обхода детектирования.

DaemonSet Tetragon на рабочем узле потреблял 1,5 % CPU (медиана; p95 — 3,5 %), память 180 мегабайт при средней интенсивности около 1200 событий в секунду. Построение локального подграфа $H$ из сырых eBPF-событий занимало в среднем 12 мс на эпизод. Суммарная задержка формирования скора эпизода (парсинг $\rightarrow H \rightarrow u \rightarrow$ агрегация) составляла 36 мс.

## Обсуждение

Исследование [20] по n-gram/STIDE/BoSC для Docker/LXC фиксирует высокие показатели точности (до 97 %), однако эти подходы чувствительны к эвазии через вставки нейтральных вызовов и не моделируют достижимость именно контейнерных границ риска. Наши эксперименты прямо оценивают падение качества при эвазии, что делает сравнение на этой оси прозрачным. В работе [11] рассматривается онлайн-детектор атак в контейнерах на частотах syscalls с самосупервизией. Этот подход показывает уменьшение ложных тревог на 33–93 % по сравнению с наборами классических ML-схем при сопоставимой полноте. В отличие от данных частотных/оконных признаков, наш метод строит гетерогенный HG/SCG-граф из eBPF-телеметрии и агрегирует «достижимость» реально опасных границ, что повышает переносимость между сервисами при сохранении интерпретации причинных путей.

Сравнивая предложенный риск-центричный метод с известными подходами к обнаружению бесфайловых атак в контейнерах, можно выделить следующие особенности.

1. Статические и сигнатурные методы (IDS, SIEM, антивирусы) малоэффективны против бесфайловых угроз и дают большое количество ложных срабатываний при динамической нагрузке, в то время как предложенный метод анализирует локальные подграфы зон риска и учитывает контекст контейнера, что повышает точность детекции.

2. Анализ оперативной памяти (memory forensics) требует сложных дампов и ручной обработки, тогда как предложенный метод работает онлайн на основе потоковых eBPF-событий и не блокирует контейнеры.

3. Модели на основе системных вызовов (Bag-of-Syscalls) чувствительны к шуму и изменениям нагрузки; риск-центричный подход снижает влияние шума за счет выделения локальных зон риска и учета временных параметров событий.

4. eBPF-сенсоры и kernel-based-мониторинг обеспечивают сбор событий ядра, но могут повышать нагрузку при высокой интенсивности. Предложенный в статье метод также использует eBPF, но за счет локализации зон риска и ограниченного построения графа сохраняет производительность и масштабируемость.

5. Графовые модели поведения (ориентированные графы системных вызовов, GNN, логические графы атак) являются наиболее близкими к предложенному подходу. В отличие от Container Escape Detection, Phoenix и CORAL [14] разработанный риск-центричный метод обнаружения бесфайловых угроз в контейнерных средах:

— не требует полного построения графа всех событий, снижая чувствительность к шуму и нагрузку на ресурсы;

— ориентирован на детекцию бесфайловых (in-memory) атак, а не только на заранее известные последовательности или escape-сценарии;

— учитывает динамический контекст контейнера и временные зависимости, что повышает устойчивость при неполном мониторинге.

Таким образом, предложенный метод сочетает преимущества графового подхода и динамического анализа, обеспечивая интерпретируемость, устойчивость к шуму и масштабируемость, что делает его более применимым в современных контейнерных инфраструктурах по сравнению с существующими решениями.

## Заключение

В работе предложен риск-центричный метод обнаружения бесфайловых угроз в контейнерных средах, который смещает фокус от анализа полного графа происхождения событий к выделению локальных подграфов «зон риска», индуцируемых eBPF-телеметрией и заданными якорями. В основе метода лежит графотемпоральная модель исполнения, где события ядра и системные вызовы представляются в виде гетерогенного графа. Для локализации потенциальных областей атаки введены формальные предикаты охраны ребер, ограничивающие возможные траектории распространения, и оператор замыкания, формирующий скелет зоны риска вокруг якорей.

Для оценки подготовлено 580 эпизодов поведения контейнера nginx:1.25-alpine: 240 с достижением границы риска (переходы к docker.sock/containerd.sock, host-namespace, повышающие capabilities и др.) и 340 без достижения; для каждого сценария S1–S6 выполнено по 60 повторов с эвазивными модификациями плюс 100 повторов фоновой активности.

По сравнению с подходами, основанными на сигнатурных правилах Falco/Tetragon, n-gram по системным вызовам и глобальном графе без локализации, предлагаемый метод показывает преимущество по AUPRC = 0,87 и AUROC = 0,94, а также более медленное падение качества при эвазивных $k$-модификациях.

Практическая ценность — в интерпретируемости (контур зоны риска, $u(v)$) и снижении ложных срабатываний, что важно для раннего реагирования.

Научная новизна работы заключается в применении риск-центричного подхода к анализу поведения контейнеров с учетом временных зависимостей событий, что позволяет локализовать зоны риска и повышает устойчивость обнаружения бесфайловых атак.

Таким образом, предложенная строгая математическая модель и практическая реализация на реальном кластере Kubernetes формируют интерпретируемую и устойчивую основу в системы ИБ, подтверждая применимость риск-центричного метода для обнаружения актуальных бесфайловых угроз в условиях современных контейнерных инфраструктур.

Дальнейшие исследования включают открытые бенчмарки и трассы промышленной эксплуатации с более широким покрытием TTP; адаптивную калибровку порога $\theta$ и весов ребер под стоимость ошибок (cost-sensitive ROC/PR); комбинирование локальной зоны риска с легкими графовыми нейросетевыми приор-оценками на этапе ранжирования.

## Литература


1. **Liu S., Peng G., Zeng H., Fu J.** A survey on the evolution of fileless attacks and detection techniques. *Computers & Security*, 2024, vol. 137, Art. 103653. doi:10.1016/j.cose.2023.103653/issn0167-4048
2. **Doherty A., Zigdon Y., Ron A.** *Aqua Nautilus Research Finds 1,400% Surge in Memory-Based Attacks as Hackers Evade Traditional Cloud Security Defenses. Aqua Security Research Report*. 2023. https://www.aquasec.com/news/aqua-nautilus-research-finds-1400-surge-in-memory-based-attacks/ (дата обращения: 24.07.2025).
3. **Sudhakar, Kumar S.** An emerging threat Fileless malware: A survey and research challenges. *Cybersecurity*, 2020, vol. 3, Art. 1. doi:10.1186/s42400-019-0043-x
4. **Kara I.** Fileless malware threats: Recent advances, analysis approach through memory forensics and research challenges. *Expert Systems with Applications*, 2023, vol. 214, Art. 119133. doi:10.1016/j.eswa.2022.119133
5. **Wu M.-H., Hsu F.-H., Huang J.-H., Wang K., Hwang Y.-L., Wang H.-J., Chen J.-X., Hsiao T.-C., Yang H.-T.** Enhancing linux system security: A kernel-based approach to fileless malware detection and mitigation. *Electronics*, 2024, vol. 13, Art. 3569. doi:10.3390/electronics13173569
6. **Levy Rocha Savio, Lopes de Mendonca Fabio Lucio, Staciarini Puttini Ricardo, Rabelo Nunes Rafael, Amvame Nze Georges Daniel Less**. DCIDS — Distributed Container IDS. *MDPI AG*, 2023, vol. 13, iss. 16, Art. 9301. doi:10.3390/app13169301
7. **Amr S. Abed, T. Charles Clancy, David S. Levy.** Applying bag of system calls for anomalous behavior detection of applications in Linux containers. *IEEE Globecom Workshops (GC Wkshps)*, 2015, doi:10.1109/GLOCOMW.2015.7414047
8. **Ковалев М. Г.** Трассировка сетевых пакетов в ядре Linux с использованием eBPF. *Труды Института системного программирования РАН*, 2020, т. 32, вып. 3, с. 71–77. doi:10.15514/ISPRAS-2020-32(3)-6
9. **Котенко И. В., Мельник М. В.** Обнаружение аномалий в контейнерных системах: применение частотного анализа и гибридной нейронной сети. *Программные продукты и системы*, 2025, т. 38, № 3, с. 426–437. doi:10.15827/0236-235X.151.426-437
10. **Pope J., Liang J., Kumar V., Raimondo F., Sun X., McConville R., Pasquier T., Piechocki R., Oikonomou G., Luo B., Howarth D., Mavromatis I., Sanchez Mompo A., Carnelli P., Spyridopoulos T., Khan A.** Resource-interaction graph: Efficient graph representation for anomaly detection. *arXiv preprint*, 2022. doi:10.48550/arXiv.2212.08525

11. **Tunde-Onadele O., Lin Y., Gu X., He J., Latapie H.** Self-supervised machine learning framework for online container security attack detection. *ACM Transactions on Autonomous and Adaptive Systems*, 2024, vol. 19, iss. 3. doi:10.1145/3665795
12. **Shokouhinejad H., Razavi-Far R., Mohammadian H., Rabbani M., Ansong S., Higgins G., Ghorbani A. A.** Recent advances in malware detection: Graph learning and explainability. *arXiv preprint IEEE*, 2025. arXiv:2502.10556. doi:10.48550/arXiv.2502.10556
13. **Chen K., Zhao Y., Guo J., Gu Z., Han L., Tang K. A**. Container escape detection method based on a dependency graph. *Electronics*, 2024, vol. 13, no. 23, Art. 4773. doi:10.3390/electronics13234773
14. **Tayouri D., Sgan Cohen O., Maimon I., Mimran D., Elovici Y., Shabtai A.** CORAL: Container Online Risk Assessment with Logical attack graphs. *Computers & Security*, 2025, vol. 150, iss. C, Art. 104296. doi:10.1016/j.cose.2024.104296
15. **Boros T., Cotaie A., Stan A., Vikramjeet K., Malik V., Davidson J.** Machine learning and feature engineering for detecting living off the land attacks. *Proceedings of the 7th International Conference on Internet of Things, Big Data and Security (IoTBDS 2022)*, 2022, pp. 133–140. doi:10.5220/0011004500003194
16. **Kermabon-Bobinnec H., Jarraya Y., Wang L., Majumdar S., Pourzandi M.** Phoenix: Surviving unpatched vulnerabilities via accurate and efficient filtering of syscall sequences. *Network and Distributed System Security Symposium (NDSS) 2024*, San Diego, USA, February 26–March 1, 2024. doi:10.14722/ndss.2024.24582
17. **Milajerdi S. M., Gjomemo R., Eshete B., Sekar R., Venkatakrishnan V. N.** HOLMES: Real-time APT detection through correlation of suspicious information flows. *IEEE Symposium on Security and Privacy*, San Francisco, CA, USA, 2019, pp. 1137–1152. doi:10.1109/SP.2019.00026
18. **Ziqi L., Chaochao C., Xinxing Y., Jun Z., Xiaolong L., Le S.** Heterogeneous graph neural networks for malicious account detection. *arXiv preprint*, 2020. arXiv:2002.12307. doi:10.48550/arXiv.2002.12307
19. **Haoyu Wu, Dongyu L., Luxin Z., Xiaoshan Z., Jiajun Z., Shanqing Yu, Qi X.** A graph neural network framework for dynamic malware detection using api calls and lightweight containers. *2025 5th International Conference on Intelligent Communications and Computing (ICICC)*, Nanjing, China, 2025, 15–17 August. doi:10.1109/ICICC66840.2025.11199642
20. **Castanhel G. R., Heinrich T., Ceschin F., Maziero C.** Taking a peek: An evaluation of anomaly detection using system calls for containers. *Conference: 26th IEEE Symposium on Computers and Communications (ISCC)*, Athens, Greece, 2021, 5–8 September, IEEE, 2021. doi:10.1109/ISCC53001.2021.9631251


UDC 004.056:004.75
doi:10.31799/1684-8853-2026-1-48-60
EDN: AVYJSY

**Using graphs to detect fileless attacks in containerized infrastructure**

E. O. Zdornikov[a], Programmer Engineer, orcid.org/0009-0009-0154-5153
E. V. Egorov[b], Junior Researcher, orcid.org/0009-0006-4321-0069
I. Y. Popov[a], PhD, Tech., Associate Professor, orcid.org/0000-0002-6407-7934, ilyapopov27@gmail.com
[a]ITMO University, 49, Kronverksky Pr., 197101, Saint-Petersburg, Russian Federation
[b]A. F. Mozhaiskii Military Space Academy, 13, Zhdanovskaia Emb., 197198, Saint-Petersburg, Russian Federation

**Introduction:** Fileless attacks exploit memory to execute malicious code without storing artifacts on disk. This significantly complicates their detection and makes traditional protection methods ineffective, especially in dynamic containerized environments where processes frequently start and terminate automatically, generating additional noise. Container systems such as Docker and Kubernetes have a smaller volume of monitored data compared to virtual machines, which simplifies event analysis and activity graphing. **Purpose:** To develop a method for detecting fileless attacks in containerized infrastructures that remains robust to noise and evasive techniques under incomplete monitoring and provides early alerting before critical resources are reached. **Results:** We propose a risk-oriented method based on a heterogeneous graph and a system call graph. Risk zones are identified around events logged using eBPF, after which a mathematical model of risk zones with secure closure and risk potential calculation based on random walk absorption is formed. The model additionally takes into account the container context and timing parameters, which reduces the number of false positives. The method enables the localization of risk zones and operates reliably even with the loss of some events. Experiments have been conducted on a Kubernetes cluster (v1.32) running Ubuntu 24.04 using Tetragon (eBPF) and Falco for quality control. We have collected 580 container behavior episodes, including attack and background scenarios, and have then used them to generate heterogeneous execution and system call graphs. The proposed method outperforms static rules, n-gram system call models, and global graph methods in terms of AUROC and AUPRC metrics, demonstrating increased resilience to invasive techniques. **Practical relevance:** The developed method is compatible with existing sensors and container security policies. It provides system administrators with interpretable risk zones and quantitative indicators of attack probability, while also supporting integration into existing incident response systems. **Discussion:** The presented mathematical model and practical implementation confirm the applicability of risk-centric analysis for detecting contemporary fileless threats in modern containerized environments; the approach is scalable, parameterizable, and does not require application modification, which makes it suitable for deployment in industrial clusters.

**Keywords** — fileless attacks, container security, eBPF, graph-temporal analysis, runtime detection, system call graphs, Kubernetes.

**For citation:** Zdornikov E. O., Egorov E. V., Popov I. Y. Using graphs to detect fileless attacks in containerized infrastructure. *Informatsionno-upravliaiushchie sistemy* [Information and Control Systems], 2026, no. 1, pp. 48–60 (In Russian). doi:10.31799/1684-8853-2026-1-48-60, EDN: AVYJSY

**References**

1. Liu S., Peng G., Zeng H., Fu J. A survey on the evolution of fileless attacks and detection techniques. *Computers & Security*, 2024, vol. 137, Art. 103653. doi:10.1016/j.cose.2023.103653/issn0167-4048
2. Doherty A., Zigdon Y., Ron A. *Aqua Nautilus Research Finds 1,400% Surge in Memory-Based Attacks as Hackers Evade Traditional Cloud Security Defenses. Aqua Security Research Report*. 2023. https://www.aquasec.com/news/aqua-nautilus-research-finds-1400-surge-in-memory-based-attacks/ (дата обращения: 24.07.2025).
3. Sudhakar, Kumar S. An emerging threat Fileless malware: A survey and research challenges. *Cybersecurity*, 2020, vol. 3, Art. 1. doi:10.1186/s42400-019-0043-x
4. Kara I. Fileless malware threats: Recent advances, analysis approach through memory forensics and research challenges. *Expert Systems with Applications*, 2023, vol. 214, Art. 119133. doi:10.1016/j.eswa.2022.119133
5. Wu M.-H., Hsu F.-H., Huang J.-H., Wang K., Hwang Y.-L., Wang H.-J., Chen J.-X., Hsiao T.-C., Yang H.-T. Enhancing linux system security: A kernel-based approach to fileless malware detection and mitigation. *Electronics*, 2024, vol. 13, Art. 3569. doi:10.3390/electronics13173569
6. Levy Rocha Savio, Lopes de Mendonca Fabio Lucio, Staciarini Puttini Ricardo, Rabelo Nunes Rafael, Amvame Nze Georges Daniel Less. DCIDS — Distributed Container IDS. *MDPI AG*, 2023, vol. 13, iss. 16, Art. 9301. doi:10.3390/app13169301
7. Amr S. Abed, T. Charles Clancy, David S. Levy. Applying bag of system calls for anomalous behavior detection of applications in Linux containers. *IEEE Globecom Workshops (GC Wkshps)*, 2015, doi:10.1109/GLOCOMW.2015.7414047
8. Kovalev M. G. Network packet tracing in the Linux kernel using eBPF. *Proceedings of the Institute for System Programming of the Russian Academy of Sciences*, 2020, vol. 32, iss. 3, pp. 71–77 (In Russian). doi:10.15514/ISPRAS-2020-32(3)-6
9. Kotenko I. V., Melnik M. V. Anomaly detection in container systems: application of frequency analysis and hybrid neural network. *Software & Systems*, 2025, vol. 38, no. 3, pp. 426–437 (In Russian). doi:10.15827/0236-235X.151.426-437
10. Pope J., Liang J., Kumar V., Raimondo F., Sun X., McConville R., Pasquier T., Piechocki R., Oikonomou G., Luo B., Howarth D., Mavromatis I., Sanchez Mompo A., Carnelli P., Spyridopoulos T., Khan A. Resource-interaction graph: Efficient graph representation for anomaly detection. *arXiv preprint*, 2022. doi:10.48550/arXiv.2212.08525
11. Tunde-Onadele O., Lin Y., Gu X., He J., Latapie H. Self-supervised machine learning framework for online container security attack detection. *ACM Transactions on Autonomous and Adaptive Systems*, 2024, vol. 19, iss. 3. doi:10.1145/3665795
12. Shokouhinejad H., Razavi-Far R., Mohammadian H., Rabbani M., Ansong S., Higgins G., Ghorbani A. A. Recent advances in malware detection: Graph learning and explainability. *arXiv preprint IEEE*, 2025. arXiv:2502.10556. doi:10.48550/arXiv.2502.10556
13. Chen K., Zhao Y., Guo J., Gu Z., Han L., Tang K. A. Container escape detection method based on a dependency graph. *Electronics*, 2024, vol. 13, no. 23, Art. 4773. doi:10.3390/electronics13234773
14. Tayouri D., Sgan Cohen O., Maimon I., Mimran D., Elovici Y., Shabtai A. CORAL: Container Online Risk Assessment with Logical attack graphs. *Computers & Security*, 2025, vol. 150, iss. C, Art. 104296. doi:10.1016/j.cose.2024.104296
15. Boros T., Cotaie A., Stan A., Vikramjeet K., Malik V., Davidson J. Machine learning and feature engineering for detecting living off the land attacks. *Proceedings of the 7th International Conference on Internet of Things, Big Data and Security (IoTBDS 2022)*, 2022, pp. 133–140. doi:10.5220/0011004500003194
16. Kermabon-Bobinnec H., Jarraya Y., Wang L., Majumdar S., Pourzandi M. Phoenix: Surviving unpatched vulnerabilities via accurate and efficient filtering of syscall sequences. *Network and Distributed System Security Symposium (NDSS) 2024*, San Diego, 2024. doi:10.14722/ndss.2024.24582
17. Milajerdi S. M., Gjomemo R., Eshete B., Sekar R., Venkatakrishnan V. N. HOLMES: Real-time APT detection through correlation of suspicious information flows. *IEEE Symposium on Security and Privacy*, San Francisco, CA, USA, 2019, pp. 1137–1152. doi:10.1109/SP.2019.00026
18. Ziqi L., Chaochao C., Xinxing Y., Jun Z., Xiaolong L., Le S. Heterogeneous graph neural networks for malicious account detection. *arXiv preprint*, 2020. arXiv:2002.12307. doi:10.48550/arXiv.2002.12307
19. Haoyu Wu, Dongyu L., Luxin Z., Xiaoshan Z., Jiajun Z., Shanqing Yu, Qi X. A graph neural network framework for dynamic malware detection using api calls and lightweight containers. *2025 5th International Conference on Intelligent Communications and Computing (ICICC)*, Nanjing, China, 2025. doi:10.1109/ICICC66840.2025.11199642
20. Castanhel G. R., Heinrich T., Ceschin F., Maziero C. Taking a peek: An evaluation of anomaly detection using system calls for containers. *Conference: 26th IEEE Symposium on Computers and Communications (ISCC)*, Athens, Greece, 2021, IEEE, 2021. doi:10.1109/ISCC53001.2021.9631251