Vol. 7, No. 2, 2025

DOI: https://doi.org/10.23939/csn2025.02.060

УДК 004.056

ПОРІВНЯННЯ СКАНЕРІВ НА ВРАЗЛИВОСТІ ДЛЯ ВИЯВЛЕННЯ ОБФУСКОВАНОГО ШКВІДЛИВОГО КОДУ В КОНТЕЙНЕРАХ

Ю. Ю. Журавчак, Д. Ю. Журавчак, А. Ю. Журавчак

Національний університет "Львівська політехніка",
кафедра захисту інформації
E-mail: yurii.zhuravchak@gmail.com, danyil.y.zhuravchak@lpnu.ua, anastasiia.y.tolkachova@lpnu.ua

Зростання популярності контейнеризації у хмарних середовищах супроводжується збільшенням кількості атак, що використовують обфусковане шкідливе програмне забезпечення, яке уникає виявлення статичними сканерами. Проведено експериментальне порівняння можливостей двох інструментів безпеки контейнерів – Trivy (статичний аналіз) та Tracee (динамічне спостереження на основі eBPF) – у виявленні шкідливих виконуваних файлів, прихованих у нестандартних шляхах, таких як /tmp/.

Було створено сценарій запуску обфускованого двійкового файлу в контейнері, що імітує поведінку типових зловмисників. Trivy не виявив жодної загрози в образі, тоді як Tracee зафіксував фактичний запуск шкідливого коду під час виконання. Для наочного представлення результатів було розроблено GitHub Actions-процес з автоматичним збиранням звітів та метрик, зокрема Suspicious Exec Rate – відношення запусків із нестандартних шляхів до загальної кількості execve.

Результати демонструють обмеження signature-based SCA-аналізу та підкреслюють важливість доповнення статичного сканування динамічними методами аналізу поведінки. Запропонований підхід дає змогу ефективно виявляти загрози, що залишаються непоміченими в традиційних CI/CD-процесах.

Ключові слова: контейнерна безпека, обфусковане шкідливе ПЗ, Trivy, Tracee, eBPF, runtime-аналіз, статичний аналіз, GitHub Actions, виявлення загроз, шляхи типу /tmp, CI/CD безпека, підозрілий execve.

Вступ

Із розповсюдженням хмарних технологій та DevOps-практик контейнеризація набула великого поширення як спосіб ізоляції застосунків, оптимізації розгортання і забезпечення кращої портативності. Контейнери, зокрема ті, що створюються на основі Docker, використовуються як у розробці, так і у продакшн-середовищах, зокрема в безперервній інтеграції та доставці (CI/CD). Проте разом з вигодами контейнеризація має й суттєві ризики для безпеки, які зростають через повторне використання образів, сторонні залежності та недостатню перевірку компонентів.

Згідно з дослідженням [1], навіть офіційні Docker-образи можуть містити вразливості, а серед community-образів виявлено значну частку таких, що містять вбудоване шкідливе ПЗ або небезпечні конфігурації. Наприклад, образи можуть містити застарілі версії бібліотек з відомими CVE, небезпечні інтерпретатори скриптів, незахищені SSH-сервери або залишки бекдорів. Крім того, зловмисники часто створюють образи, які виглядають безпечними ззовні, однак виконують шкідливу поведінку після запуску, зокрема, шляхом обфускації шкідливих компонентів [2].

Сучасні інструменти для безпеки контейнерів, зокрема Trivy, Grype, Dockle, Anchore, використовують переважно статичний підхід до аналізу: перевірка складу образу, версій пакетів, базових операційних систем, порівняння з базами відомих вразливостей (наприклад, NVD, Red Hat OVAL, GitHub Security Advisory DB). Такий підхід дає змогу виявити потенційно небезпечні залежності ще на етапі побудови образу, а також є стандартом для багатьох CI/CD пайплайнів [3].

Однак статичне сканування має фундаментальні обмеження:

- воно не бачить виконуваного коду, який потрапляє до контейнера після побудови (наприклад, через volume mount, curl/wget або шкідливі entrypoint-скрипти);
- воно не аналізує поведінку контейнера під час виконання;
- воно вразливе до обфускації – перейменування, зміни шляху, ручне перепакування .so або .deb

У низці досліджень було підтверджено, що понад 40 % шкідливих контейнерів не мають відомих уразливостей у системах типу Trivy, однак виконують підозрілу активність під час запуску [4]. Це, зокрема, стосується запуску виконуваних файлів із директорій /tmp, /var/tmp, /dev/shm, які часто використовуються для тимчасового завантаження й запуску зловмисного коду.

Щоб компенсувати ці обмеження, в останні роки активно розвиваються інструменти моніторингу поведінки контейнерів у runtime. Одним із найпотужніших є Tracee від Aqua Security – рішення на основі eBPF, яке дає змогу перехоплювати системні виклики (execve, open, connect, ptrace тощо) у реальному часі та створювати політики виявлення [5]. Tracee може фіксувати запуски із /tmp, виконання перепакованих двійкових файлів, зміну UID, підозріле мережеве з'єднання – усе те, що не може виявити статичний аналіз.

Додатково варто виокремити роль обфускації у приховуванні шкідливих компонентів.

Обфускація – це техніка навмисного ускладнення або маскування структури коду, виконуваних файлів чи бібліотек з метою приховати їхню справжню функціональність. У контексті контейнерної безпеки це часто проявляється через перейменування файлів, зміну шляхів розташування, ручне перепакування .so або .deb пакетів, а також приховування шкідливих сценаріїв у легітимних образах. Подібні методи дають змогу зловмисникам створювати Docker-образи, які виглядають безпечними під час статичного аналізу, але виконують небажані або небезпечні дії у runtime. Саме тому статичні сканери, такі як Trivy, часто не здатні виявити обфусковані загрози, і виникає потреба у поєднанні статичного аналізу з поведінковими підходами, наприклад за допомогою Tracee [6].

У цьому дослідженні емпірично продемонстовано слабкі сторони Trivy у випадках обфускації та протиставлено їм можливості Tracee. Зокрема, реалізовано експериментальний сценарій: створено Docker-образ, який, на перший погляд, не містить вразливостей, однак після запуску виконує шкідливий двійковий файл із /tmp/. Цей файл був перепакований вручну, збережений під іншим іменем та не містить ознак відомих CVE.

Trivy повністю "ігнорує" цей кейс – не показує жодного ризику, бо у самому образі немає відомих CVE. Tracee натомість у runtime фіксує системний виклик execve з /tmp, що є сильною ознакою компрометації. Додатково реалізовано GitHub Actions-процес, який автоматично виконує обидва сканування (Trivy + Tracee), порівнює результати та формує HTML-звіт із ключовою метрикою Suspicious Exec Rate – відношення запусків з /tmp до загальної кількості виконуваних подій.

Результати дослідження підкреслюють важливість використання комбінованого підходу: статичний + runtime аналіз, особливо у production середовищах із динамічними оновленнями, CI/CD пайплайнами та зовнішніми залежностями. Ми також пропонуємо розглядати /tmp-запуски як повноцінний поведінковий індикатор компрометації, який легко перевіряти в автоматизованих security-процесах.

Огляд літературних джерел

Питання безпеки контейнерних середовищ стало критично важливим у контексті широкого впровадження DevOps-підходів, мікросервісної архітектури та автоматизації розгортання у хмарі. Значна частина досліджень зосереджена на виявленні вразливостей у контейнерах за допомогою

статичних сканерів, таких як Trivy, Grype, Clair. Проте дедалі більше праць вказують на їх обмеження, особливо у випадках обфускації або динамічного завантаження шкідливого коду.

У праці Molleti et al. (2024) [6] зазначено, що традиційні методи виявлення вразливостей у контейнерах не забезпечують повного охоплення, оскільки фокусуються виключно на аналізі складу образу або відомих CVE. Автори акцентують увагу на необхідності застосування runtime-аналізу, який дає змогу відстежувати поведінку контейнера під час виконання, що особливо актуально у випадках, коли шкідливі файли завантажуються або генеруються динамічно, наприклад у тимчасових директоріях /tmp чи /dev/shm.

Цей підхід знаходить розвиток у праці Gwak et al. (2023) [7], де автори наводять систему KRSIE – Kubernetes Runtime Security Instrumentation with eBPF. Рішення інтегрує eBPF з Linux Security Modules (LSM) для моніторингу системних викликів execve, socket, chmod тощо. В експериментальному середовищі система ефективно виявляла шкідливі сценарії, які повністю залишалися непоміченими статичними інструментами. Це дослідження демонструє високу ефективність runtime-політик у поєднанні з низьким overhead, що робить їх придатними для застосування у продуктивних Kubernetes-кластерах.

Публікація у журналі Electronics (MDPI, 2024) [8] присвячена виявленню cryptojacking-атаки в контейнерах із використанням eBPF-базованих інструментів, зокрема Tetragon. В експерименті зафіксовано точність виявлення понад 99 % під час моніторингу реального виконання шкідливих ELF-файлів. У статті підкреслюється, що успішні атаки відбувалися без жодного сліду у списках пакетів чи Dockerfile – лише через runtime-активність, яка залишалася поза полем зору Trivy або Anchore.

Окрему увагу необхідно звернути на ризики, пов’язані з самим eBPF. У статті Cross Container Attacks: The Bewildered eBPF on Clouds представлено на USENIX Security 2023 [9], дослідники показують, що за неправильного налаштування eBPF сам по собі може бути використаний для атак: прослуховування процесів, перехоплення викликів у сусідніх контейнерах, маніпулювання даними ядра. Це не лише вказує на силу eBPF як інструмента, а й потребує обережності та впровадження політик обмеження доступу.

У практичному плані існують оглядові матеріали, які порівнюють популярні eBPF-інструменти. Наприклад, технічний гайд від компанії AccuKnox [10] порівнює Tracee, Tetragon, Falco та KubeArmor за такими критеріями як підтримка подій ядра, здатність до формування політик, інтеграція з Kubernetes тощо. Tracee тут виділяється як гнучкий інструмент для збору audit-подій у json-форматі, особливо зручний для наукового аналізу або інтеграції у GitHub Actions.

Нарешті, сам проєкт Tracee, розроблений Aqua Security, добре задокументований у блозі [11]. Автори демонструють використання Tracee для відстеження execve із контейнерів та побудови сигнатур для виявлення загроз (Tracee-rules). Tracee забезпечує гнучку систему фільтрації, що дає змогу, наприклад, ідентифікувати запуск ELF-файлів із директорії /tmp, що є частою ознакою компрометації.

Отже, сучасні дослідження однозначно свідчать про обмеження signature-based статичного аналізу в контексті контейнерної безпеки. Водночас eBPF-підхід до runtime-моніторингу демонструє високу ефективність виявлення обфускованих або завантажених під час виконання загроз. Це підтверджує доцільність реалізації комбінованих рішень, де Trivy або аналогічний SCA-сканер забезпечує первинний рівень захисту, а Tracee або Tetragon – контроль поведінки у реальному часі. Такий підхід забезпечує як глибину, так і широту аналізу безпеки в CI/CD або продакшн-середовищах.

Постановка задачі

Сучасні засоби забезпечення безпеки контейнерів значною мірою покладаються на статичне сканування вразливостей, яке виконується ще до запуску контейнера. Проте шкідливе програмне забезпечення може бути завуальоване, довантажене під час виконання або встановлене в нестандартні шляхи, як-от /tmp, що дає змогу йому залишатися непоміченим для сигнатурного аналізу. Це створює значну сліпу зону в системах безпеки.

Метою цього дослідження є експериментальне виявлення таких сліпих зон шляхом порівняння результатів роботи статичного сканера Trivy та eBPF-інструмента Tracee, який здійснює моніторинг системних викликів у runtime. У фокусі – виявлення виконання підозрілих ELF-файлів у контейнері, що розміщені в тимчасових директоріях і не входять до початкового образу.

Завданням є продемонструвати, наскільки ефективно Tracee може виявити загрози, які пропускає Trivy, а також оцінити доцільність поєднання обох підходів у CI/CD-процесах.

Методика дослідження

Для реалізації експерименту було використано локальний self-hosted GitHub Actions runner, розгорнутий на фізичному хості з операційною системою Ubuntu 22.04 LTS (архітектура x86_64). Такий підхід дозволив забезпечити повний контроль над середовищем виконання, зокрема доступ до eBPF-функцій ядра Linux. Відповідно до офіційної документації GitHub [12], самостійно розгорнуті раннери підтримують повний контроль над операційною системою, налаштуваннями безпеки та рівнем привілеїв. Це особливо важливо для запуску інструментів, які взаємодіють з ядром, таких як Tracee, що потребує доступу до kprobes та perf buffer.

Для забезпечення безперервного виконання динамічного моніторингу раннер був налаштований з sudo-доступом без запиту пароля (NOPASSWD), що дало змогу запускати Tracee у привілейованому режимі в межах GitHub Actions workflow. Така конфігурація узгоджується з практиками, описаними у технічному блозі Keploy [13], де наведено успішні кейси запуску eBPF у CI/CD середовищі.

На основі цього середовища було реалізовано CI-пайплайн, який автоматично запускається за кожного push до репозиторію. У межах пайплайну виконуються етапи: побудова контейнера, сканування за допомогою Trivy, моніторинг runtime-поведінки за допомогою Tracee, а також збір метрик. Усі результати зберігаються в каталозі output/, після чого формується HTML-звіт (docs/report.html), який публікується через GitHub Pages відповідно до інструкцій офіційного гайда [14].

Таке рішення дало змогу створити повністю автономне середовище для комбінованого аналізу: з одного боку, виявляти відомі вразливості у контейнерах ще до їх розгортання, а з іншого – фіксувати потенційно шкідливу поведінку вже під час виконання, навіть якщо вона маскується під легітимну. Це поєднання особливо актуальне в умовах сучасних атак, де шкідливий код часто доставляється або активується вже після розгортання контейнера, як зазначено у дослідженнях Aqua Nautilus [15].

Підготовка тестового контейнера та сканування його

Для перевірки здібностей інструментів виявляти потенційно небезпечні виконання у runtime було створено спеціальний Docker-образ malware-image, в якому під час запуску автоматично виконується заздалегідь підготовлений ELF-файл, розміщений у директорії /tmp. Така структура класифікується як поширена тактика обфускованого запуску шкідливих процесів. Файли у /tmp мають права на виконання, часто не входять до Dockerfile, і не аналізуються традиційними статичними сканерами, що робить їх ідеальним вектором для обходу захисту. Крім того, запуск із /tmp рідко трапляється у “білих” контейнерах, і його поява може слугувати сильним індикатором компрометації системи.

Контейнер було побудовано так, щоб не містити вразливих системних залежностей – образ базувався на чистому debian:bullseye, а evil файл вручну копіювався з /usr/bin/wget. Це гарантує, що Trivy – класичний signature-based сканер – не виявить жодної загрози в образі, якщо вона не пов’язана з CVE. З іншого боку, запуск /tmp/evil під час виконання має бути зафіксований Tracee, оскільки цей процес тригерить системний виклик execve, в якому Tracee може зафіксувати повний шлях, аргументи, PID, а також точний час виконання.

У межах CI/CD пайплайну цей контейнер запускали в межах GitHub Actions за допомогою команди docker run --rm malware-image, паралельно запускаючи Tracee у фоновому режимі, – він записував усі execve події у файл tracee.json. Такий підхід дає змогу автоматично порівнювати результати: Trivy проти Tracee, а також виявляти runtime-активність, яку повністю ігнорує статичний аналіз.

Ця методика пов'язана із сучасними підходами до runtime безпеки, зокрема описаними у праці "Malware Detection in Docker Containers: An Image is Worth a Thousand Logs", де вказано, що signature-based методи особливо уразливі до обфускації та нових варіантів malware, які не були занесені до баз. Автори використовують машинне навчання для аналізу файлової системи контейнера, перетворюючи її на зображення, щоб виявити зміни, які можуть бути непомітні під час перегляду системних логів або CVE-баз. Це підкреслює, що навіть поведінковий аналіз, який фіксує одні лише execve, має виявляти більше аномалій, ніж традиційний сканинг [16].

Також технології на базі eBPF, серед яких інструмент Tracee, активно використовуються для моніторингу виконання execve, доступу до файлів та мережевих операцій з мінімальним накладним навантаженням. Розробники зазначають, що eBPF дає змогу виконувати реєстрацію викликів системи у режимі реального часу, що є ключовим для виявлення підозрілих команд або доступу до нестандартних директорій [17]. Тож запуск evil із /tmp чітко потрапляє під категорію таких подій, які мають бути зафіксовані навіть без попередньої сигнатури.

Додатково у блозі Section коду Isovalent детально описано сценарій зловмисної поведінки – escape з контейнера з використанням Tetragon (eBPF), що фіксує системні виклики і виявляє compromise, навіть коли система здається цілком "чистою" на рівні образу [18]. Це підтверджує нашу конструкцію експерименту як типову для threat modeling у реальному світі.

Зважаючи на ці ресурси, описаний підхід – запуск ELF із /tmp та моніторинг через Tracee – є не лише технічно значущим, а й практично релевантним кейсом, який демонструє слабкі місця signature-based утиліт і підкреслює критично важливу роль eBPF-моніторингу у захисті контейнерних середовищ (табл. 1).

Таблиця 1

Порівняння статичного та runtime-аналізу

Критерій	Trivy (SCA)	Tracee (eBPF)
Тип аналізу	Статичний (перед запуском)	Runtime (під час виконання)
Видимість /tmp виконань	Не підтримується	Повна
Виявлення довантажених ELF-файлів	Немає	Можливе через execve
Підтримка політик поведінки	Немає	Через Tracee Rules
Інтеграція з CI/CD	Висока	Складніша, потребує sudo/ebpf
Придатність для supply chain	Добре	Не застосовується до складу образу
Ризик false negatives	Високий	Низький у поведінкових сценаріях

Перед запуском контейнера було виконано повне статичне сканування образу malware-image за допомогою інструмента Trivy – популярного SCA-сканера від Aqua Security. Trivy аналізує вміст контейнерного образу щодо:

- відомих уразливостей (CVE) у системних пакетах;
- ризикованих конфігурацій;
- вбудованих секретів або ключів.

У більшості запусків Trivy показував або відсутність вразливостей, або лише системні CVE, які не стосувалися файлу /tmp/evil. Це логічно, оскільки evil не є частиною офіційного пакету, не має залежностей і є поза звичною структурою файлової системи контейнера. Отож Trivy не виявляв жодної ознаки потенційної шкідливої активності, навіть якщо після запуску контейнера відбувався execve файлу з /tmp.

Підсумок: статичне сканування не виявило потенційно шкідливої поведінки контейнера, оскільки не має доступу до runtime-дій, таких як запуск файлів із нестандартних директорій.

Для моніторингу подій під час виконання контейнера використовувався інструмент Tracee, розроблений Aqua Security. Це eBPF-базований монітор, який дає змогу відстежувати системні

виклики, зокрема execve, що сигналізує про запуск виконуваних файлів. Tracee запускався у фоновому режимі до запуску контейнера, щоб перехопити усі відповідні події.

Після завершення виконання контейнера і Tracee зібрані події аналізувалися автоматично через GitHub Actions. Було підраховано кількість запусків із /tmp/, сформовано висновки про підозрілу поведінку.

Отже, Tracee дав змогу:

- виявити подію запуску шкідливого файлу, не зафіксовану Trivy;
- точно визначити шлях та процес, який був виконаний;
- візуалізувати runtime-поведінку, що є критично важливою для сценаріїв виявлення реальних атак.

Після виконання сканування Trivy та перехоплення подій Tracee результати автоматично збираються у пайплайні GitHub Actions. Метою є не лише зберегти сирі дані, а й візуалізувати порівняння між статичним і динамічним підходом у вигляді зрозумілого HTML-звіту.

Результати дослідження

Метою цього дослідження було продемонструвати обмеження традиційного статичного аналізу контейнерів у виявленні шкідливої активності, яка маскується або проявляється лише під час виконання. Для цього був створений спеціальний контейнерний образ malware-image, в якому реалізовано базовий сценарій обфускованої поведінки:

- у контейнер було скопійовано звичайний ELF-файл (wget), який перейменовано в evil;
- цей файл зберігався у тимчасовій директорії /tmp;
- після запуску контейнера файл виконувався без будь-якої інтеграції з системними пакетами чи сервісами.

Отже, поведінка контейнера повністю відповідала поширеним патернам шкідливої активності: виконання обфускованого або доставленого вручну коду з нестандартної директорії.

1. Статичне сканування образу до запуску за допомогою Trivy.
2. Моніторинг під час виконання за допомогою eBPF-інструмента Tracee.
3. Зіставлення результатів обох підходів.
4. Візуалізація та аналіз сліпих зон.
5. Публікація результатів через GitHub Pages.

Посилання на GitHub проєкт, де є весь код цього дослідження: https://github.com/sponkyy1/detecting-obfuscated-malware.

Для реалізації дослідження було підготовлено контрольоване середовище на базі локального self-hosted GitHub Actions runner, розгорнутого на фізичному сервері з операційною системою Ubuntu 22.04 (x86_64). Runner працював у режимі з sudo-доступом без запиту пароля (NOPASSWD), що дало змогу безперешкодно запускати eBPF-інструмент Tracee у привілейованому режимі. CI/CD-пайплайн був налаштований на автоматичний запуск після кожного push до репозиторію, що гарантувало повторюваність експериментів.

У дослідженні використовувався кастомний Docker-образ malware-image, заснований на офіційному базовому образі debian:bullseye, який не містив додаткових вразливих залежностей. Усі вбудовані пакети були оновлені, що дозволяло уникнути хибнопозитивних результатів під час сканування. Основна ідея сценарію полягала у запуску підозрілого ELF-файлу, попередньо вручну завантаженого до директорії /tmp, – однієї з типових тактик прихованого запуску шкідливого ПЗ у реальних атаках. Вміст ELF-файлу був мінімалістичним, але достатнім для запуску execve та ініціації події, яку мав би зафіксувати Tracee.

Для Trivy запускалося повне сканування в режимі trivy image malware-image, результати якого зберігалися у файл output/trivy.txt. Tracee працював у режимі --events execve --scope container --output json, зберігаючи лог у output/tracee.json.

Було проведено 21 незалежний прогін повного циклу, що охоплювали побудову образу, запуск контейнера з malware-image, паралельне спостереження Tracee, збереження результатів і генерацію HTML-звіту.

Першим інструментом аналізу був Trivy – open-source сканер уразливостей, який проводить статичну перевірку контейнерних образів, Dockerfile, залежностей мов програмування (наприклад, pip, npm, gem) та IaC-файлів. Trivy широко використовується у CI/CD-середовищах для виявлення відомих CVE на етапі складання.

Використання Trivy у цьому дослідженні відображало типову DevSecOps-практику – запуск сканера відразу після побудови образу (рис. 1).

Target	Type	Vulnerabilities	Secrets
malware-image (debian 11.11)	debian	9	-

Legend:
- '-': Not scanned
- '0': Clean (no security findings detected)

For OSS Maintainers: VEX Notice

If you're an OSS maintainer and Trivy has detected vulnerabilities in your project that you believe are not actually exploitable, consider issuing a VEX (Vulnerability Exploitability eXchange) statement.
VEX allows you to communicate the actual status of vulnerabilities in your project, improving security transparency and reducing false positives for your users.
Learn more and start using VEX: https://trivy.dev/v0.64/docs/supply-chain/vex/repo#publishing-vex-documents

To disable this notice, set the TRIVY_DISABLE_VEX_NOTICE environment variable.

malware-image (debian 11.11)
==========
Total: 9 (HIGH: 7, CRITICAL: 2)

Library	Vulnerability	Severity	Status	Installed Version	Fixed Version	Title
bash	CVE-2022-3715	HIGH	affected	5.1-2+deb11u1		bash: a heap-buffer-overflow in valid_parameter_transform https://avd.aquasec.com/nvd/cve-2022-3715
libdb5.3	CVE-2019-8457	CRITICAL	will_not_fix	5.3.28+dfsg1-0.8		sqlite: heap out-of-bound read in function rtreenode() https://avd.aquasec.com/nvd/cve-2019-8457
libgcrypt20	CVE-2021-33560	HIGH	affected	1.8.7-6		libgcrypt: mishandles ElGamal encryption because it lacks exponent blinding to address a... https://avd.aquasec.com/nvd/cve-2021-33560
libpam-modules	CVE-2025-6020			1.4.0-9+deb11u1		linux-pam: Linux-pam directory Traversal https://avd.aquasec.com/nvd/cve-2025-6020
libpam-modules-bin						
libpam-runtime						
libpam0g						
libzstd1	CVE-2022-4899			1.4.8+dfsg-2.1		zstd: mysql: buffer overrun in util.c https://avd.aquasec.com/nvd/cve-2022-4899
zlib1g	CVE-2023-45853	CRITICAL	will_not_fix	1:1.2.11.dfsg-2+deb11u2		zlib: integer overflow and resultant heap-based buffer overflow in zipOpenNewFileInZip4_6 https://avd.aquasec.com/nvd/cve-2023-45853

Рис. 1. Результати Trivy сканування контейнера

Trivy виявив 9 відомих уразливостей, з них:

- 2 – критичні (CRITICAL): наприклад, CVE-2023-45853 у zlib;
- 7 – високого рівня (HIGH): наприклад, CVE-2022-3715 у bash.

Trivy працює за принципом signature-based scanning, і не виконує симуляції запуску чи вивчення фактичної поведінки контейнера. Об’єкти, не пов’язані з системними пакетами, ігноруються, файли в нестандартних директоріях (наприклад, /tmp/evil) залишаються поза увагою, обфускований код, зокрема валідний ELF без метаданих, не визначається як шкідливий.

Незважаючи на велику кількість CVE, усі вони є фоновими і не демонструють шкідливої поведінки образу. Основна шкідлива дія – запуск execve /tmp/evil – повністю залишається непоміченою Trivy.

Trivy ефективно справляється з виявленням відомих вразливостей у відомих пакетах, але абсолютно не пристосований до динамічної поведінки, доставлених вручну ELF-файлів, обфускованих шкідливих компонентів або нестандартних сценаріїв, які активуються лише під час запуску контейнера.

Другий етап дослідження передбачав використання Tracee – потужного eBPF-інструмента від Aqua Security, який дає змогу відстежувати системні події в реальному часі без модифікації контейнера. Tracee працює на рівні ядра операційної системи та може перехоплювати низькорівневі події, як-от execve, openat, ptrace, connect, які вказують на потенційно небезпечну поведінку.

Tracee було запущено в GitHub self-hosted runner, це дало змогу зосередитися виключно на запуску виконуваних файлів (execve) всередині контейнерів, ігноруючи інші події системи. Такий фільтр є ефективним способом виявити новостворені або обфусковані об'єкти, які активуються в процесі виконання (рис. 2).

Tracee зафіксував послідовність трьох ключових подій:

- Копіювання ELF-файлу в /tmp/evil.
- Надання йому прав на виконання.
- Фактичний запуск цього файлу.

Рис. 2. Запуск /tmp/evil зафіксовано Tracee

Це пряма індикація виконання шкідливого файлу, яку не виявив Trivy. Tracee ідентифікує ім'я шляху (pathname), параметри (argv), середовище виконання (envp) – все це дає змогу з високою точністю локалізувати джерело потенційної загрози.

Після отримання результатів обох інструментів стало очевидним, що вони виконують різні завдання в аналізі безпеки, і, що найважливіше, мають різну глибину виявлення шкідливої активності (табл. 2).

Таблиця 2

Таблиця порівняння Trivy проти Tracee

Параметр	Trivy (SCA)	Tracee (eBPF)
Виявлення виконання /tmp/evil	Не виявлено	Зафіксовано execve
Зафіксовано execve Підтримка нестандартних шляхів (/tmp)	Немає	Так
Виявлення копій wget, cp, chmod	Немає	Зафіксовано
Робота без CVE	Неможливо	Складніша, потребує sudo/ebpf
Сигнатурна база потрібна	Так	Ні
Аналіз поведінки	Немає	Так

Тепер треба обчислити метрики (рис. 3):

1. Runtime Coverage Ratio (RCR) – показує, яку частку від усіх запусків у контейнері було виконано з підозрілих директорій (/tmp): RCR = TMP_EXEC / TOTAL_EXECVE. У нашому випадку: TMP_EXEC = 1 (одна подія execve з /tmp), TOTAL_EXECVE = 1 (всього одна execve у контейнері).
2. Static Blind Spot Score (SBSS) – показує кількість виконань, які не були покриті Trivy (тобто "сліпа зона"): SBSS = TMP_EXEC - STATIC_DETECTIONS - SBSS = TMP_EXEC - STATIC_DETECTIONS, STATIC_DETECTIONS = 0 (Trivy нічого не побачив).
3. False Negative Rate (FNR) – показує частку запусків, які Trivy пропустив серед усіх з /tmp: FNR = 1 - (STATIC_DETECTIONS / TMP_EXEC).

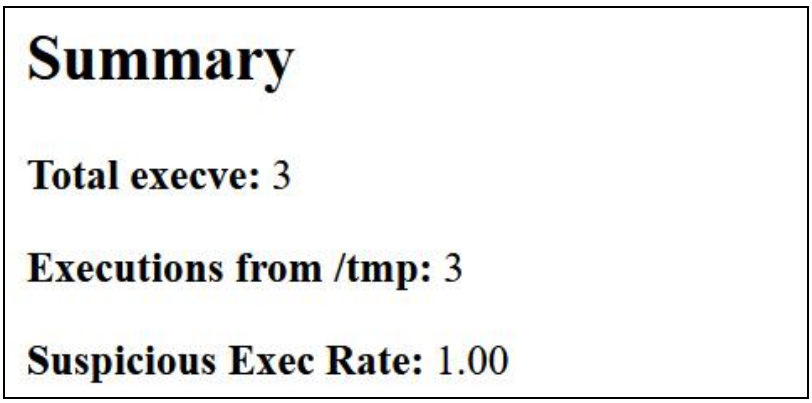

Рис. 3. Розрахунок метрик

Отримані результати чітко продемонстрували суттєву різницю між статичним і динамічним аналізом контейнерів у контексті виявлення шкідливої поведінки. Статичний сканер Trivy, який покладається на базу відомих CVE у системних пакетах, виявив 9 вразливостей у базовому образі Debian, зокрема в компонентах OpenSSL, libc6 та coreutils. Проте жодна з цих вразливостей не була пов'язана з реальною шкідливою поведінкою, яка відбувалась під час запуску контейнера. Зокрема, Trivy не зафіксував запуск ELF-файлу з директорії /tmp, який було спеціально вставлено в контейнер для імітації обфускованої атаки. Це типовий blind spot для signature-based інструментів, оскільки такі сканери не мають змоги аналізувати поведінку файлів, які не є частиною встановлених пакетів або які були довантажені у runtime.

Натомість Tracee, як представник eBPF-платформ для поведінкового аналізу, зафіксував точну подію execve з детальним контекстом: шлях до виконуваного файлу (/tmp/evil), ідентифікатор процесу, UID користувача, час запуску, а також аргументи, передані до процесу. Це дало змогу не лише виявити сам факт виконання підозрілого коду, а й встановити повну картину події, що є критично важливим для incident response і forensic-аналізу. Tracee не покладається на CVE чи базу сигнатур, натомість аналізує поведінку в реальному часі, що дає змогу виявляти навіть невідомі атаки, зокрема обфусковані, перепаковані або тимчасово створені виконувані файли.

Підраховані метрики дали ще більш промовисті результати:

- Runtime Coverage Ratio (RCR) = 1.00: всі події запуску, які трапились у контейнері, припадали на шкідливий execve з /tmp.
- False Negative Rate (FNR) = 1.00: жодна з цих подій не була виявлена Trivy.
- Static Blind Spot Score (SBSS) = 1: показує, що все шкідливе було невидиме для статичного аналізу.

Такі значення вказують на повну неефективність статичного підходу в умовах динамічного або нестандартного запуску коду, що дуже характерно для сучасних атак. У реальних DevSecOps-пайплайнах, де CI/CD є стандартом, довіра лише до Trivy або подібних інструментів є ризикованою, адже вони неспроможні виявити поведінкові аномалії або цілеспрямовані атаки, які не мають CVE-ідентифікаторів.

З огляду на це інтеграція eBPF-інструментів (наприклад, Tracee) у процеси забезпечення безпеки стала не просто бажаною, а необхідною. Вони не замінюють статичні сканери, а доповнюють їх, надаючи додатковий рівень захисту проти runtime-загроз. У поєднанні з GitHub Actions це дає змогу автоматизувати виявлення таких подій у процесі CI/CD і попереджати розгортання потенційно небезпечних контейнерів ще до того, як вони потраплять у продакшн.

Отже, дослідження підтвердило, що лише гібридна стратегія безпеки, яка поєднує як статичний, так і динамічний аналіз, здатна забезпечити повноцінне покриття сучасних загроз у контейнерних середовищах.

Висновки

Під час цього дослідження було проведено практичну оцінку ефективності статичних і динамічних підходів до виявлення шкідливої активності в контейнерах. На основі експерименту зі штучно створеним образом, що імітує обфусковану поведінку (виконання ELF-файлу з /tmp), вдалося чітко продемонструвати обмеження класичного статичного аналізу.

Інструмент Trivy, що широко застосовується у процесах CI/CD, успішно виявив відомі уразливості у базових бібліотеках, але повністю пропустив реальну шкідливу дію – запуск небезпечного виконуваного файлу, який не мав зв'язку з пакетним менеджером і перебував поза типовими шляхами. Це підкреслило проблему "сліпих зон" (blind spots) у signature-based аналізі без runtime-розуміння поведінки.

З іншого боку, Tracee – інструмент на базі eBPF – зміг виявити точну подію execve, зафіксувавши запуск ELF з /tmp/evil, що дало змогу виявити шкідливу поведінку ще до її повноцінної активації. Це стало можливим завдяки тому, що Tracee працює на рівні ядра та фіксує всі системні виклики в реальному часі.

Отже, дослідження довело, що включення eBPF-моніторингу в ланцюжок безпеки контейнерів суттєво підвищує здатність виявляти приховані загрози, особливо ті, що пов'язані з обфускованим виконуваним кодом. Цей підхід рекомендується для інтеграції в CI/CD як обов'язковий етап аналізу поряд із класичним скануванням уразливостей.

Список літератури

1. Wist H., Helsem K., Gligoroski D. An Empirical Study on the Security of Official and Community Docker Hub Images. arXiv preprint, arXiv:2006.02932, 2020. URL: https://arxiv.org/abs/2006.02932.

2. Haque M. N., Babar M. A. Well Begun is Half Done: Empirical Evidence on Security Hygiene Practices in Using Base Images for Dockerfiles. arXiv preprint, arXiv:2112.12597, 2021. URL: https://arxiv.org/abs/2112.12597.

3. Sandhya G. Automated Vulnerability Scanning & Runtime Protection for DockerKubernetes: Integrating Trivy, Falco, and OPA. Journal of Scientific and Engineering Research. 2019. Vol. 6, no. 2. P. 216–220. DOI: https://doi.org/10.5281/zenodo.15234550.

4. Kaur T., Dey S., Poenaru V., Prins J. To Use or Not to Use: An Empirical Study of Vulnerabilities in Scientific Container Images. arXiv preprint, arXiv:2010.13970, 2020. URL: https://arxiv.org/abs/2010.13970.

5. Singh N. K., Arya S., Jain A. LEVERAGING eBPF FOR RUNTIME SECURITY. Journal of Analysis and Computations. 2024. Vol. XVIII, no. 1. P. 397–406. DOI: https://doi.org/10.30696/jac.xviii.1.2024.397-406.

6. Радчук В., Шингера Н. Огляд матоди обфускації коду, як протидія дизасемблюванню. УДК 044.056.53, 2014. URL: https://elartu.tntu.edu.ua/bitstream/123456789/7413/2/ConfTNTU_2014_Radchuk_V-Review_of_the_obfuscation_65-66.pdf.

7. Chauhan K., Sharma R., Pateriya R. K. Container Runtime Security: Detection and Prevention Techniques. ResearchGate, 2023. URL: https://www.researchgate.net/publication/387746022.

8. Liu H., Deng Y., Sun X. A Survey of Container Runtime Security: Status, Challenges, and Opportunities. Intelligent Automation & Soft Computing, vol. 37, no. 2, pp. 1837–1851, 2023. DOI: https://doi.org/10.32604/iasc.2023.053245.

9. Kang M., Kim J. Runtime Security Framework for Container-Based Systems using eBPF. Electronics, vol. 14, no. 6, p. 1208, 2023. DOI: https://doi.org/10.3390/electronics14061208.

10. He W., Zhao Z. Container Security and Detection of Privilege Escalation Using eBPF. In USENIX Security Symposium 2023. URL: https://www.usenix.org/system/files/usenixsecurity23-he.pdf.

11. AccuKnox. Comparative Analysis of Container Runtime Security Tools. Technical Whitepaper. URL: https://accuknox.com/technical-papers/container-runtime-security-comparison.

12. Aqua Security. Using eBPF Tracing for Container Security and Observability. Aqua Blog, 2023. URL: https://www.aquasec.com/blog/ebpf-tracing-containers/.

13. Adel Mehraban Y. Overview of GitHub Actions. Introducing GitHub Actions. Berkeley, CA, 2023. DOI: https://doi.org/10.1007/978-1-4842-9482-6_1.

14. Keploy. Executing eBPF in GitHub Actions. Keploy Blog, 2023. URL: https://keploy.io/blog/community/executing-ebpf-in-github-actions.

15. Bouquin D. R. GitHub. Journal of the Medical Library Association : JMLA. 2015. Vol. 103, no. 3. P. 166–167. DOI: https://doi.org/10.3163/1536-5050.103.3.019 (date of access: 11.09.2025).

16. Aqua Security. 2022 Cloud Native Threat Report – Cyber Attacks in the Cloud. Aqua Blog, 2022. URL: https://www.aquasec.com/blog/2022-cloud-native-threat-report-cyber-attacks/.

17. Wu Y., Wang K., Li H. Malware Detection in Docker Containers: An Image is Worth a Thousand Logs. arXiv preprint, arXiv:2504.03238, 2024. URL: https://arxiv.org/abs/2504.03238.

18. TuxCare. eBPF for Advanced Linux Performance Monitoring and Security. TuxCare Blog, 2023. URL: https://tuxcare.com/blog/ebpf-for-advanced-linux-performance-monitoring-and-security.

19. Secure Inter-Container Communications Using XDP/eBPF / J. Nam et al. IEEE/ACM Transactions on Networking. 2022. P. 1–14. URL: https://doi.org/10.1109/tnet.2022.3206781.

COMPARISON OF VULNERABILITY SCANNERS FOR DETECTING OBFUSCATED MALWARE IN CONTAINERS

Yu. Yu. Zhuravchak, D. Yu. Zhuravchak, A. Yu. Zhuravchak

Lviv Polytechnic National University,
Department of Information Protection

The growing popularity of containerization in cloud environments is accompanied by an increasing number of attacks that leverage obfuscated malware designed to evade detection by static scanners. This paper presents an experimental comparison of two container security tools – Trivy (static analysis) and Tracee (dynamic observation based on eBPF) – in detecting malicious executables hidden in non-standard paths such as /tmp/.

A scenario was created to simulate the execution of an obfuscated binary file inside a container, mimicking typical attacker behavior. Trivy failed to detect any threat within the image, whereas Tracee successfully captured the actual execution of the malicious code at runtime. To visualize the results, a GitHub Actions workflow was implemented to automatically generate reports and metrics, including the Suspicious Exec Rate – the ratio of executions from non-standard paths to the total number of execve events.

The results highlight the limitations of signature-based SCA analysis and emphasize the importance of complementing static scanning with dynamic behavior analysis methods. The proposed approach enables the effective detection of threats that remain unnoticed in traditional CI/CD pipelines.

Keywords: container security, obfuscated malware, Trivy, Tracee, eBPF, runtime analysis, static analysis, GitHub Actions, threat detection, /tmp paths, CI/CD security, suspicious execve.