10. Gunasekara, A. (2023). AI-Driven Big Data Analytics for Transforming Cybersecurity for Zero-Day Vulnerabilities in E-Commerce Supply Chains. Journal of Advances in Cybersecurity Science, Threat Intelligence, and Countermeasures, 7(12), 17–31.

11. Jiang, L., An, J., Huang, H., Tang, Q., Nie, S., Wu, S., & Zhang, Y. (2024, January 20). BinaryAI: Binary Software Composition Analysis via Intelligent Binary Source Code Matching. arXiv.Org. https://arxiv.org/abs/2401.11161v3.

12. Sane, P. (2020). Is the OWASP Top 10 List Comprehensive Enough for Writing Secure Code? 58–61. Scopus. https://doi.org/10.1145/3437075.3437089.

13. GHSA-8mgj-vmr8-frr6—GitHub Advisory Database. (n.d.). GitHub. Retrieved November 17, 2025, from https://github.com/advisories/GHSA-8mgj-vmr8-frr6.

14. debug-js. (n.d.). (RESOLVED) Version 4.4.2 published to npm is compromised · Issue #1005 · debug-js/debug. GitHub. Retrieved November 17, 2025, from https://github.com/debug-js/debug/issues/1005.

Надійшла 05.11.2025

УДК 004.056:004.4’2
DOI: 10.31673/2409-7292.2025.041213

Мичуда Л.З., Расяк В.І.

ОГЛЯД СУЧАСНИХ ПІДХОДІВ ДО БЕЗПЕЧНОГО КОНФІГУРУВАННЯ DOCKER COMPOSE

У статті проведено аналітичний огляд сучасних підходів до забезпечення безпеки контейнерних конфігурацій, зокрема у контексті використання Docker Compose як інструменту для декларативного опису багатоконтейнерних систем. Визначено ключові проблеми, пов’язані з конфігураційними помилками у файлах docker-compose.yml, які можуть призводити до порушення конфіденційності, цілісності та доступності контейнеризованих сервісів. Проаналізовано можливості та обмеження існуючих інструментів перевірки безпеки: Docker Bench for Security, OpenSCAP, Trivy, Checkov, KICS та інших, з точки зору їхньої ефективності щодо аналізу конфігураційного рівня. Особливу увагу приділено питанням відповідності конфігурацій міжнародним стандартам та рекомендаціям безпеки, зокрема CIS Docker Benchmark і практикам DevSecOps.

У результаті дослідження визначено основні типи ризиків, характерні для середовищ Docker Compose, та запропоновано концептуальну класифікацію таких ризиків за категоріями конфіденційності, цілісності, доступності, ізоляції та відповідності стандартам. Обґрунтовано необхідність створення уніфікованих політик безпечного конфігурування Docker Compose і розроблення автоматизованих інструментів перевірки, інтегрованих у процеси CI/CD. Отримані результати формують теоретичну основу для подальших досліджень у сфері забезпечення безпеки контейнерних конфігурацій та розроблення практичних засобів їх аудиту.

Ключові слова: контейнеризація, Docker Compose, безпека конфігурацій, DevSecOps, Docker Bench for Security, OpenSCAP, контейнерні середовища.

Вступ

У сучасних інформаційних системах контейнеризація є ключовою технологією, що забезпечує ефективність, портативність і масштабованість розгортання програмних застосунків. Серед існуючих інструментів екосистеми контейнеризації платформа Docker посідає провідне місце, а її компонент Docker Compose відіграє важливу роль у спрощенні процесу конфігурації багатоконтейнерних систем. Формат опису у вигляді файлу docker-compose.yml дозволяє декларативно визначати взаємозв’язки між сервісами, параметри мережі, обмеження ресурсів і змінні середовища, що істотно полегшує процес розгортання та супроводу застосунків у середовищах DevOps [10, 17].

Водночас із розширенням масштабів використання контейнеризації спостерігається зростання кількості інцидентів, пов’язаних із конфігураційними помилками. Неправильне визначення привілеїв контейнерів, некоректне налаштування мережевих портів, зберігання секретів у відкритому вигляді чи відсутність обмежень ресурсів можуть призвести до серйозних загроз інформаційній безпеці. Такі недоліки не завжди є результатом навмисних дій розробників, а часто зумовлені відсутністю чітких методичних рекомендацій або автоматизованих засобів перевірки конфігурацій на рівні Docker Compose.

Існуючі рішення для перевірки безпеки контейнерних середовищ, зокрема Docker Bench for Security, OpenSCAP, Trivy чи Checkov, орієнтовані переважно на аналіз образів, операційного середовища або інфраструктурного коду, однак не забезпечують комплексного аудиту саме конфігураційного рівня docker-compose.yml [2-6]. Унаслідок цього залишається нерозв'язаною проблема системної оцінки ризиків, що виникають через помилки у параметрах ізоляції, мережевих взаємодіях чи управлінні доступом між контейнерами.

Постановка проблеми

Розповсюдження контейнерних технологій змінило підходи до розробки, розгортання та супроводу програмних систем. Контейнеризація з використанням Docker забезпечує стандартизоване середовище виконання, спрощує масштабування та зменшує залежність від інфраструктури. Водночас помилки конфігурації можуть становити критичну загрозу для безпеки застосунків та даних.

У контейнеризованих середовищах безпека залежить не лише від коду та бібліотек, а й від коректності параметрів у конфігураційних файлах. Формат docker-compose.yml, що описує багатоконтейнерні архітектури, містить критично важливі налаштування, мережеві політики, томи, привілеї контейнерів, змінні середовища та інші атрибути. Недотримання принципів мінімізації прав, неправильне керування ресурсами або секретами може призвести до ескалації привілеїв, компрометації даних або відмови в обслуговуванні.

Існуючі інструменти перевірки безпеки контейнерів, такі як Docker Bench for Security, OpenSCAP чи Trivy, орієнтовані здебільшого на аналіз хостового середовища, образів контейнерів або операційної системи. Конфігураційний рівень Docker Compose, що визначає взаємодію компонентів системи, залишається поза їхнім основним фокусом. Це призводить до того, що багато потенційних загроз, пов'язаних із помилками у docker-compose.yml, не виявляються автоматизованими засобами аудиту.

Відсутність уніфікованих правил чи стандартів для безпечного конфігурування Docker Compose ускладнює інтеграцію перевірок безпеки в DevSecOps та CI/CD-процеси. Розробники і адміністратори змушені покладатися на фрагментарні практики й власний досвід, що призводить до неоднорідності політик безпеки у різних проєктах.

Науковою проблемою є відсутність цілісного підходу до оцінювання безпеки конфігурацій Docker Compose, який враховував би специфіку ризиків конфігураційного рівня та дозволяв інтегрувати перевірку у процеси безперервної розробки.

Метою дослідження є аналіз сучасних методів і засобів безпеки контейнерних конфігурацій, визначення їхніх обмежень щодо Docker Compose та формування класифікації ризиків, характерних для цього рівня конфігурування. Досягнення мети передбачає систематизацію теоретичних положень, виявлення прогалин у практичних рішеннях і обґрунтування напрямів розвитку автоматизованого аудиту конфігурацій контейнерних середовищ.

Огляд сучасних підходів до безпеки контейнерних конфігурацій

Питання безпеки контейнеризації упродовж останніх років має важливе значення серед досліджень у сфері інформаційних технологій. Поступове перенесення корпоративних систем у хмарні середовища, використання мікросервісної архітектури та автоматизація розгортання через CI/CD-конвеєри призвели до необхідності перегляду традиційних підходів до захисту інфраструктури. На відміну від монолітних застосунків, у контейнеризованих системах безпека значною мірою визначається не лише якістю програмного коду, а й правильністю конфігурацій, які керують ізоляцією процесів, управлінням доступом і мережею.

Перші системні рекомендації щодо безпеки Docker були сформульовані центром інтернет-безпеки (Center for Internet Security, CIS) у вигляді CIS Docker Benchmark [1]. Документ описує перелік перевірок, спрямованих на зміцнення захисту Docker Daemon, контейнерів, образів і середовища хоста. У ньому визначено базові принципи безпечного розгортання контейнерів, зокрема мінімізацію привілеїв, захист сокетів Docker, використання

© Мичуда Л.З., Расяк В.І. (2025) Огляд сучасних підходів до безпечного конфігурування Docker Compose. Сучасний захист інформації, 4(64), 132–140.
https://doi.org/10.31673/2409-7292.2025.041213

перевірених образів і контроль доступу до API. Проте в цьому стандарті практично не розглядаються аспекти, пов'язані з декларативним описом багатосервісних архітектур, що формуються у файлах docker-compose.yml. Тобто CIS Benchmark залишається ефективним для низькорівневого аудиту середовища виконання, але не охоплює конфігураційного рівня, на якому розробники визначають взаємодію між контейнерами.

Одним із найпоширеніших практичних інструментів реалізації CIS-рекомендацій є Docker Bench for Security, сценарій автоматизованого аудиту, що виконує перевірки відповідності налаштувань Docker установленим контрольним точкам [2]. Його перевагою є доступність і простота інтеграції у процес розгортання контейнерів, однак функціональність цього засобу обмежується аналізом хостової системи та параметрів запуску контейнерів. Docker Bench не має механізму для читання та інтерпретації структури файлу Docker Compose, унаслідок чого не може оцінити коректність міжсервісних зв'язків або мережевих правил, що визначаються на рівні YAML-конфігурацій.

Іншим напрямом розвитку є стандарти на основі SCAP (Secure Content Automation Protocol), які реалізуються у фреймворку OpenSCAP [3]. Цей підхід орієнтований на перевірку відповідності системи вимогам безпеки через формалізовані профілі, політики та чек-листи. OpenSCAP має модулі для аудиту контейнерів і хостових операційних систем, а також інтегрується з корпоративними рішеннями керування безпекою. Проте аналогічно до CIS Benchmark, OpenSCAP не містить уніфікованого профілю, який би регламентував безпечне конфігурування Docker Compose, що обмежує можливість автоматизованої оцінки на цьому рівні.

Паралельно з розвитком стандартів виникла низка інструментів статичного аналізу конфігурацій інфраструктурного коду (Infrastructure-as-Code, IaC), які частково охоплюють і контейнерні сценарії. Найвідомішим серед них є Trivy [5], багатофункціональний сканер вразливостей, що поєднує аналіз образів, конфігурацій IaC та виявлення секретів. Завдяки модулю trivy config інструмент здатний обробляти YAML-файли, включно з обмеженою підтримкою Docker Compose. Однак якість результатів значною мірою залежить від набору політик, наявних у базі правил, що не завжди враховують специфіку багатоконтейнерних конфігурацій.

Схожий підхід реалізовано у системі Checkov [6], яка застосовується для перевірки Terraform, CloudFormation, Kubernetes-маніфестів та інших декларативних описів. Вона підтримує написання власних правил перевірки, однак орієнтована переважно на хмарну інфраструктуру. Аналіз Docker Compose у Checkov здійснюється частково й не охоплює повної семантики міжсервісних зв'язків, що знижує точність оцінювання ризиків конфігурації.

Більш вузькоспеціалізованим інструментом є KICS (Keeping Infrastructure as Code Secure), розроблений як універсальний механізм статичного аналізу IaC [7]. Його перевагою є наявність вбудованих запитів для Docker Compose, які дозволяють виявляти окремі типові помилки, пов'язані з мережевими налаштуваннями, портами, обмеженнями ресурсів і політиками доступу. Водночас навіть у цьому випадку набір перевірок не є вичерпним і не охоплює специфічних аспектів безпеки, пов'язаних із міжконтейнерними залежностями чи керуванням секретами.

Інший клас інструментів становлять лінтери контейнерних образів, серед яких Dockle [8]. Цей засіб перевіряє дотримання найкращих практик під час створення Dockerfile і побудови образів, зокрема відсутність команд, що запускаються під користувачем root, використання легковагових базових образів або очищення кешу після встановлення пакетів. Dockle підвищує якість образів, проте не аналізує рівень Compose-конфігурацій.

Зіставлення згаданих підходів дає змогу констатувати, що більшість сучасних інструментів орієнтовані на перевірку або контейнерного середовища виконання, або декларацій інфраструктури у хмарних провайдерах. Конфігураційний рівень Docker Compose, який описує топологію сервісів та їхню взаємодію, залишається недостатньо формалізованим

© Мичуда Л.З., Расяк В.І. (2025) Огляд сучасних підходів до безпечного конфігурування Docker Compose. Сучасний захист інформації, 4(64), 132–140.
https://doi.org/10.31673/2409-7292.2025.041213

[9]. Це створює прогалину між низькорівневими перевірками безпеки та реальними практиками DevSecOps-команд, що використовують Compose як основу для тестового та виробничого розгортання застосунків.

Типові ризики та вразливості у конфігураціях Docker Compose

Безпечність контейнерних середовищ безпосередньо залежить від коректності конфігураційних параметрів, що визначають поведінку контейнерів, рівень їхніх привілеїв та взаємодію між сервісами. Формат docker-compose.yml, який широко використовується для оркестрації багатосервісних застосунків, має декларативний характер, але при цьому залишається вразливим до помилок людини. Розробники нерідко приділяють недостатню увагу деталям безпеки, концентруючись передусім на працездатності системи. У результаті навіть незначні недоліки в конфігурації можуть стати причиною критичних інцидентів безпеки.

Одним із найпоширеніших джерел ризику є неправильне керування секретами. Використання змінних середовища для зберігання паролів, токенів доступу або ключів API у відкритому вигляді створює передумови для компрометації конфіденційних даних. У середовищах розробки такі дані нерідко потрапляють до систем контролю версій або файлів журналів, що робить їх доступними для сторонніх осіб. Рекомендованим підходом є застосування механізмів Docker Secrets або зовнішніх сервісів управління секретами, проте навіть серед досвідчених DevOps-команд цей принцип не завжди дотримується [12].

Інша категорія проблем пов’язана з надмірними привілеями контейнерів [13]. Визначення параметра privileged: true або додавання системних можливостей через директиву cap_add надає контейнеру розширені права доступу до ядра операційної системи. Такі конфігурації суперечать принципу мінімізації привілеїв і підвищують ризик ескалації прав у разі експлуатації вразливостей у застосунках. Згідно з рекомендаціями CIS Docker Benchmark, контейнери мають запускатися з обмеженими правами, без доступу до хостових пристроїв, і використовувати лише ті можливості ядра, які є критично необхідними.

Особливо небезпечними є помилки, пов’язані з неправильним мапуванням портів і мережею [14]. Відкриття контейнерів на адресу 0.0.0.0 або прив’язка зовнішніх портів без потреби роблять сервіси доступними поза межами внутрішньої мережі. Це підвищує ризик несанкціонованого доступу або сканування сервісів із зовні. Багато випадків компрометації контейнерних середовищ пов’язані саме з помилками у налаштуванні мережевої ізоляції або відсутністю обмежень для міжсервісної взаємодії.

Нерідко розробники нехтують обмеженнями на використання ресурсів [15]. Відсутність параметрів deploy.resources.limits у Docker Compose-файлі призводить до того, що контейнер може споживати необмежену кількість оперативної пам’яті або процесорного часу, що у виробничих умовах може викликати відмову в обслуговуванні (DoS) інших сервісів. Принципи керування ресурсами є обов’язковими складовими безпечної конфігурації, однак автоматизовані засоби перевірки не завжди контролюють ці аспекти, що ще раз підкреслює потребу у формалізованих політиках для Compose.

Суттєвим ризиком є також використання нестабільних тегів образів [16], зокрема позначення latest. Оскільки такий тег автоматично оновлюється під час побудови контейнера, розробник не має гарантій щодо відтворюваності середовища. Унаслідок цього можуть виникати розбіжності між тестовими та виробничими середовищами, що створює умови для непередбачуваних збоїв або втрати сумісності. Крім того, оновлення базового образу без контролю версії може призвести до інтеграції вразливостей, яких не було у попередніх релізах.

Не менш критичною є проблема небезпечного монтування томів [17]. Використання директиви volumes із вказанням шляху до системних каталогів хоста без обмеження прав доступу відкриває контейнеру можливість модифікувати або зчитувати файли операційної системи. Особливо ризикованим є сценарій, коли контейнер має доступ до сокета Docker (/var/run/docker.sock), що фактично надає йому адміністративний контроль над іншими

© Мичуда Л.З., Расяк В.І. (2025) Огляд сучасних підходів до безпечного конфігурування Docker Compose. Сучасний захист інформації, 4(64), 132–140.
https://doi.org/10.31673/2409-7292.2025.041213

контейнерами й самим хостом. Окрему увагу слід приділити неправильному керуванню користувачами [18]. Багато контейнерів запускаються з обліковим записом root за замовчуванням, що суперечить базовим вимогам безпеки. Docker надає можливість визначати користувача за допомогою директиви user, проте цей механізм часто ігнорується або використовується некоректно. У поєднанні з відсутністю мережевої ізоляції такі конфігурації можуть дозволити зловмиснику отримати контроль над усією системою.

У сукупності перелічені фактори свідчать про системний характер проблеми: безпечне конфігурування Docker Compose не може ґрунтуватися виключно на індивідуальних практиках розробників. Необхідним є створення формалізованих підходів до оцінювання конфігураційних ризиків, що враховують специфіку міжсервісних взаємодій, керування привілеями, ресурсами й секретами. Саме конфігураційний рівень є тією точкою, де перетинаються аспекти безпеки програмного забезпечення, інфраструктури та процесів DevOps, і саме його недооцінка найчастіше стає причиною вразливостей у контейнерних середовищах.

Концептуальна класифікація ризиків конфігураційного рівня

Забезпечення безпеки контейнерних систем потребує систематичного підходу до аналізу ризиків, що виникають на рівні конфігурацій. На відміну від уразливостей програмного коду, конфігураційні ризики формуються внаслідок порушення принципів побудови безпечної інфраструктури, помилок під час визначення параметрів взаємодії між контейнерами та відсутності стандартизованих політик. Для їх ефективного контролю необхідна концептуальна модель класифікації, яка дозволяє системно описати природу, джерела та наслідки потенційних загроз у середовищі Docker Compose.

Розроблення такої класифікації ґрунтується на принципах тріади CIA (Confidentiality, Integrity, Availability), яка традиційно використовується у сфері інформаційної безпеки. Проте для контейнеризованих середовищ доцільно розширити її додатковими вимірами, що враховують специфіку контейнерної ізоляції та відповідності стандартам. У результаті пропонується п’ятикомпонентна модель класифікації ризиків конфігураційного рівня: конфіденційність, цілісність, доступність, ізоляція та відповідність (табл. 1).

Таблиця 1

Концептуальна класифікація ризиків конфігураційного рівня Docker Compose

Категорія ризику	**Типові приклади**	**Потенційні наслідки**	**Рекомендовані заходи**
Конфіденційність	Зберігання паролів у environment-змінних; передавання секретів без TLS	Витік облікових даних; компрометація сервісів	Використання Docker Secrets, шифрування з’єднань, централізоване керування секретами
Цілісність	Використання тегу latest; спільні томи з правами запису	Модифікація даних, втручання у виконання застосунків	Фіксування версій образів, read-only томи, обмеження прав контейнерів
Доступність	Відсутність resource limits; некоректна політика restart	Відмова в обслуговуванні (DoS); зупинка критичних сервісів	Встановлення ресурсних лімітів; тестування стійкості конфігурацій
Ізоляція	Використання privileged: true; спільні мережеві простори; доступ до /var/run/docker.sock	Ескалація привілеїв; компрометація хоста	Запуск контейнерів без привілеїв; окремі мережі; rootless-контейнери
Відповідність	Відсутність перевірок відповідності CIS або NIST SP 800-190	Порушення стандартів безпеки; невідповідність аудиту	Автоматизований аудит конфігурацій; інтеграція з DevSecOps pipeline

© Мичуда Л.З., Расяк В.І. (2025) Огляд сучасних підходів до безпечного конфігурування Docker Compose. Сучасний захист інформації, 4(64), 132–140.
https://doi.org/10.31673/2409-7292.2025.041213

До категорії ризиків конфіденційності належать порушення, що призводять до розкриття або витоку чутливих даних. Найтиповішими прикладами є зберігання паролів і токенів доступу у відкритому вигляді в environment-змінних, надання контейнерам доступу до зовнішніх систем без шифрування або передавання секретів через мережеві з'єднання без TLS. Такі помилки часто трапляються у невеликих розробницьких командах, де відсутня централізована система керування секретами.

Ризики цілісності охоплюють помилки, які можуть призвести до несанкціонованої модифікації даних або порушення узгодженості середовища. До них належить використання тегу latest замість фіксованої версії образу, спільний доступ до томів із правами запису, а також надмірні привілеї контейнерів, які дозволяють змінювати системні ресурси хоста. Такі вразливості створюють можливість втручання у функціонування інших компонентів системи або навіть у підлеглі рівні операційної системи.

Категорія ризиків доступності пов'язана з помилками у налаштуванні ресурсних лімітів і політик перезапуску контейнерів. Відсутність директив deploy.resources або неправильне визначення політики restart може спричинити перевитрату ресурсів, зупинку критичних сервісів або ефект каскадного відмовлення у межах багатоконтейнерної архітектури. Порушення доступності часто мають непрямі наслідки, оскільки впливають не лише на окремий контейнер, а й на залежні від нього сервіси.

Особливу увагу слід приділити ризикам ізоляції, які є фундаментальними для контейнерної безпеки. Їх причиною стає неправильне розмежування привілеїв, спільне використання мережевих просторів або надання контейнерам доступу до системного сокета Docker. Такі помилки фактично порушують межі віртуалізації, перетворюючи контейнер на повноцінну точку атаки на хостову систему. Сучасні дослідження демонструють, що понад 20% інцидентів у середовищах Docker спричинені саме проблемами ізоляції на рівні конфігурацій.

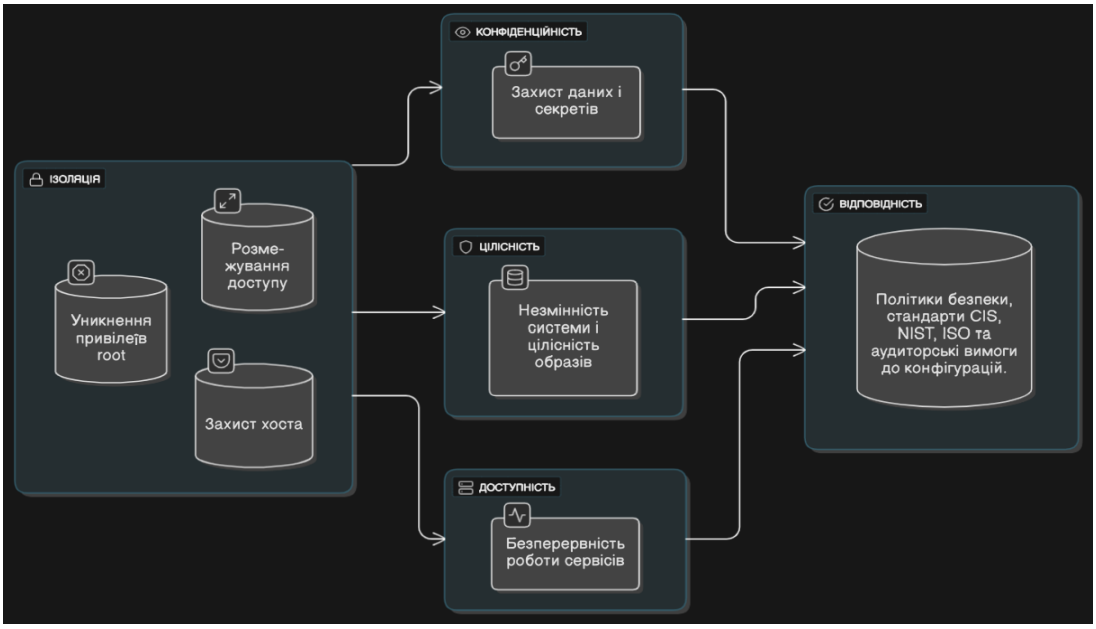

Рис. 1. Узагальнена модель взаємозв'язку категорій ризиків конфігураційного рівня Docker Compose

Остання категорія – ризики відповідності (compliance risks), яка охоплює відхилення від установлених стандартів безпеки, корпоративних політик чи галузевих вимог. Відсутність механізмів аудиту docker-compose.yml у контексті стандартів CIS або NIST SP 800-190 унеможливлює автоматизоване підтвердження відповідності системи вимогам безпеки. Для

© Мичуда Л.З., Расяк В.І. (2025) Огляд сучасних підходів до безпечного конфігурування Docker Compose. Сучасний захист інформації, 4(64), 132–140.
https://doi.org/10.31673/2409-7292.2025.041213

великих організацій це створює додаткові труднощі при сертифікації або внутрішніх перевірках.

Запропонована класифікація дозволяє розглядати конфігураційні ризики Docker Compose у системному контексті, що поєднує технічні, організаційні та процесні аспекти. Вона також слугує теоретичною основою для розроблення скорингових моделей ризику, де кожна категорія може бути кількісно оцінена за рівнем критичності, ймовірністю виникнення та потенційним впливом. Такий підхід дає змогу інтегрувати аналіз безпеки конфігурацій у безперервні процеси DevSecOps, формуючи комплексну систему управління конфігураційними ризиками.

На рисунку 1 подано модель, яка відображає п’ять взаємопов’язаних доменів безпеки, де ізоляція є базовою умовою для забезпечення конфіденційності, цілісності та доступності, а відповідність є горизонтальним виміром, що інтегрує всі попередні категорії у нормативно-керований контекст DevSecOps [10-18].

Напрями вдосконалення підходів до перевірки безпеки Docker Compose

Розвиток практик DevSecOps визначає потребу у переході від реактивного до проактивного контролю безпеки конфігурацій контейнеризованих середовищ. У контексті Docker Compose це означає не лише виявлення помилок після розгортання, а й інтеграцію механізмів перевірки на етапі розроблення, тестування та CI/CD-процесів.

Існуючі інструменти, такі як Docker Bench for Security, OpenSCAP чи Trivy, орієнтовані переважно на аудит рівня Docker Engine або образів контейнерів, що залишає поза увагою специфічні аспекти конфігураційних файлів docker-compose.yml. Основним напрямом вдосконалення має стати розроблення єдиної системи політик безпеки для Docker Compose, яка б поєднувала машинно-читані правила перевірки з гнучкими механізмами оновлення у відповідності до змін міжнародних стандартів.

Такі політики доцільно формувати у форматах YAML або JSON, із можливістю їхнього автоматичного оновлення через централізовані репозиторії, аналогічно до підходів, реалізованих у Open Policy Agent (OPA) [15]. Це дозволить забезпечити незалежність правил від конкретної реалізації інструменту перевірки, а також сприятиме уніфікації аудиту між різними етапами життєвого циклу контейнерного застосунку.

Другим ключовим напрямом розвитку є інтеграція перевірок конфігурацій у CI/CD-пайплайн, що забезпечить постійний моніторинг безпеки конфігураційних файлів під час кожного оновлення репозиторію [16]. Такий підхід відповідає принципу Shift Left Security, коли безпека вбудовується у найраніші етапи розроблення. Автоматизовані сканери можуть бути налаштовані для запуску під час коміту, пул-реквесту або збірки Docker Compose, формуючи звіт про ризики у зручному форматі (JSON, HTML, SARIF).

Крім технічної інтеграції, важливо забезпечити адаптивність системи перевірки до різних контекстів розгортання: від локальних середовищ розробки до хмарних оркестраторів (Docker Swarm, Kubernetes). Для цього необхідна підтримка динамічних політик, які враховують контекст використання, наприклад, допустимі порти, ресурси або мережеві обмеження в залежності від середовища.

Третім напрямом є створення скорингової моделі оцінки ризику конфігурацій, яка б на основі категорій, описаних у попередньому розділі, автоматично присвоювала оцінку рівня критичності. Така модель може враховувати як вагові коефіцієнти ризику, так і історичні дані про виявлені порушення в конкретному проєкті. Це дозволяє формувати зведений індекс безпеки конфігурації (Configuration Security Index), який може використовуватися для моніторингу динаміки покращень безпеки у часі.

Окрему роль у підвищенні ефективності перевірки відіграє застосування методів машинного навчання для класифікації ризиків і виявлення аномалій у конфігураціях. Використання моделей, навчених на відкритих репозиторіях Docker Compose, дозволить прогнозувати потенційно небезпечні шаблони, навіть якщо вони не відповідають явно

© Мичуда Л.З., Расяк В.І. (2025) Огляд сучасних підходів до безпечного конфігурування Docker Compose. Сучасний захист інформації, 4(64), 132–140.
https://doi.org/10.31673/2409-7292.2025.041213

визначеним політикам. Зрештою, стратегічним напрямом удосконалення є інституціоналізація перевірки безпеки Docker Compose, тобто включення таких інструментів до корпоративних DevSecOps-стандартів, а також розроблення рекомендацій на рівні міжнародних організацій (CNCF, OWASP, ISO). Це забезпечить єдність вимог, спрощення аудиту та сумісність результатів перевірки між різними інструментами.

Таким чином, сучасний підхід до безпеки конфігурацій Docker Compose має ґрунтуватися на поєднанні формалізованих політик, автоматизації перевірок та інтеграції з CI/CD-процесами. Це не лише знижує ймовірність людських помилок, а й забезпечує відтворюваність результатів аудиту, підвищуючи загальний рівень захищеності контейнерних середовищ.

На рисунку 2 подана модель, яка відображає етапи формування політик, інтеграції перевірок у CI/CD pipeline, виконання автоматизованого сканування, оцінки ризиків та формування зворотного зв'язку для вдосконалення безпеки контейнерних конфігурацій.

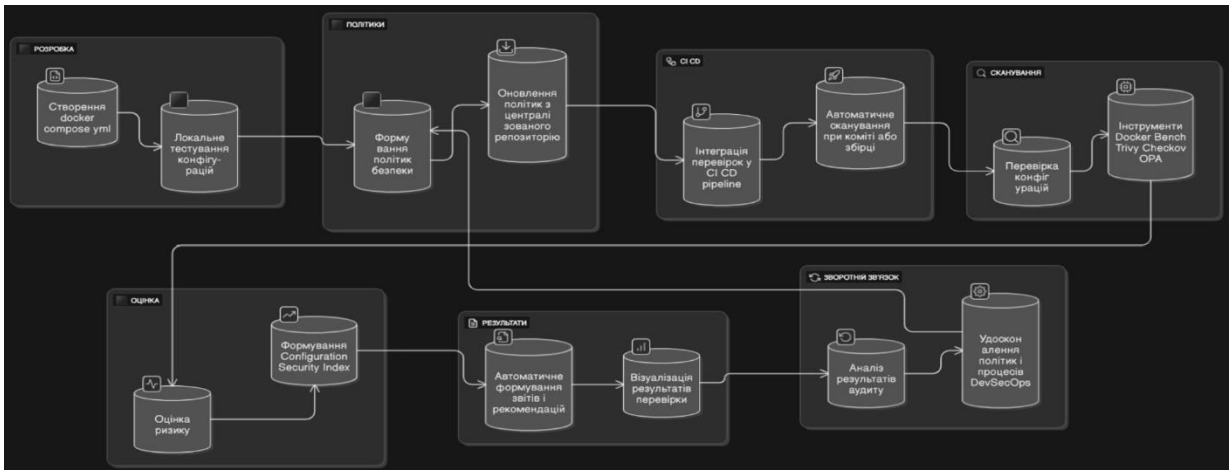

Рис. 2. Модель інтеграції перевірки безпеки Docker Compose у DevSecOps-процес

Висновки

У ході дослідження проведено комплексний аналіз сучасних підходів до забезпечення безпеки конфігурацій Docker Compose, а також виявлено основні ризики та вразливості, що виникають на конфігураційному рівні контейнеризованих середовищ. Встановлено, що більшість наявних інструментів перевірки безпеки орієнтовані на низькорівневі аспекти контейнерної інфраструктури (Docker Engine, образи контейнерів) і не забезпечують достатньої підтримки аналізу декларативних конфігураційних файлів docker-compose.yml.

На основі проведеного аналізу запропоновано концептуальну класифікацію ризиків конфігураційного рівня, що охоплює п'ять взаємопов'язаних категорій: конфіденційність, цілісність, доступність, ізоляцію та відповідність. Така класифікація дозволяє системно описати природу загроз, визначити їхні джерела та вплив на безпеку контейнерних систем, а також сформувати основу для побудови скорингових моделей ризику. Розроблена модель інтеграції перевірки безпеки Docker Compose у процеси DevSecOps передбачає використання формалізованих політик безпеки, автоматизованих інструментів аудиту та зворотного зв'язку для постійного вдосконалення політик і процесів. Інтеграція таких механізмів у CI/CD-пайплайн забезпечує безперервний контроль безпеки конфігурацій, знижує ризик людських помилок і підвищує рівень прозорості та відповідності стандартам CIS, NIST і ISO. Узагальнені результати дослідження формують наукове підґрунтя для подальшої розробки автоматизованої системи перевірки безпеки конфігурацій Docker Compose, що може бути реалізована у вигляді незалежного інструменту або як модуль для інтеграції у DevSecOps-процеси. Подальші дослідження можуть бути спрямовані на побудову скорингових моделей

© Мичуда Л.З., Расяк В.І. (2025) Огляд сучасних підходів до безпечного конфігурування Docker Compose. Сучасний захист інформації, 4(64), 132–140.
https://doi.org/10.31673/2409-7292.2025.041213

ризику, розширення бази політик безпеки та застосування машинного навчання для виявлення аномалій у конфігураціях.

Перелік посилань

1. Center for Internet Security. CIS Docker Benchmark v1.6.0 [Електронний ресурс] // CIS. 2023. Режим доступу: https://www.cisecurity.org/benchmark/docker.

2. Docker. Docker Bench for Security [Електронний ресурс] // GitHub Repository. 2024. Режим доступу: https://github.com/docker/docker-bench-security.

3. Aqua Security. Trivy: A Simple and Comprehensive Vulnerability Scanner for Containers and IaC [Електронний ресурс] // Aqua Security. 2024. Режим доступу: https://aquasec.com/tools/trivy.

4. Bridgecrew. Checkov: Infrastructure as Code static analysis tool [Електронний ресурс] // Bridgecrew. 2024. Режим доступу: https://www.checkov.io/.

5. Checkmarx. KICS – Keeping Infrastructure as Code Secure [Електронний ресурс] // Checkmarx Docs. 2024. Режим доступу: https://docs.kics.io/latest/queries/dockercompose-queries.

6. GoodwithTech. Dockle – Container Image Linter for Security [Електронний ресурс] // GitHub Repository. 2023. Режим доступу: https://github.com/goodwithtech/dockle.

7. OWASP Foundation. Container Security Verification Standard (CSVS), Draft v0.9 [Електронний ресурс] // OWASP. 2023. Режим доступу: https://owasp.org/.

8. Docker. Managing Secrets in Docker Compose [Електронний ресурс] // Official Documentation. 2024. Режим доступу: https://docs.docker.com/compose/use-secrets/.

9. OWASP Foundation. Docker Security Cheat Sheet [Електронний ресурс] // OWASP Cheat Sheet Series. 2023. Режим доступу: https://cheatsheetseries.owasp.org/cheatsheets/Docker_Security_Cheat_Sheet.html.

10. Docker Documentation. Networking in Compose [Електронний ресурс] // Docker Docs. 2024. Режим доступу: https://docs.docker.com/compose/networking/.

11. Docker Documentation. Compose Deploy Resources [Електронний ресурс] // Docker Docs. 2024. Режим доступу: https://docs.docker.com/compose/compose-file/deploy/#resources.

12. Snyk Ltd. Container Security: Volume Mount Risks [Електронний ресурс] // Snyk Learn. 2023. Режим доступу: https://snyk.io/learn/docker-security/.

13. Red Hat. Running Containers as Non-root [Електронний ресурс] // Red Hat Container Security. 2022. Режим доступу: https://www.redhat.com/en/topics/containers.

14. National Institute of Standards and Technology (NIST). Application Container Security Guide (SP 800-190) [Електронний ресурс] // NIST Publications. 2017. Режим доступу: https: // nvlpubs. nist. gov / nistpubs / SpecialPublications/NIST.SP.800-190.pdf.

15. Open Policy Agent. Rego Policy Language Documentation [Електронний ресурс] // OPA Docs. 2024. Режим доступу: https://www.openpolicyagent.org/docs/latest/.

16. GitLab Documentation. Shift Left Security in CI/CD Pipelines [Електронний ресурс] // GitLab Docs. 2023. Режим доступу: https://docs.gitlab.com/ee/topics/security/shift_left_security.html.

17. Docker Documentation. Compose File Reference (v3.9) [Електронний ресурс] // Docker Docs. 2024. Режим доступу: https://docs.docker.com/compose/compose-file/.

18. Google Cloud Security. Machine Learning for Misconfiguration Detection [Електронний ресурс] // Google Cloud Security Blog. 2023. Режим доступу: https://cloud.google.com/security.

Надійшла 10.11.2025

УДК 004.056.5:004.7
DOI: 10.31673/2409-7292.2025.041215

Прокопович-Ткаченко Д.І., Зубченко Н.С., Черкаський О.В., Черкаський Д.О., Тихоненко І.М.

МЕТОДИ ВИЯВЛЕННЯ ШКІДЛИВИХ ВЛАСТИВОСТЕЙ АЛЬТЕРНАТИВНИХ ПРОШИВОК МАРШРУТИЗАТОРІВ У КОНТЕКСТІ СУЧАСНИХ МЕРЕЖЕВИХ ЗАГРОЗ

У статті досліджується використання альтернативних та модифікованих прошивок маршрутизаторів як одного з ключових чинників формування ботнет-мереж і реалізації розподілених DDoS-атак. Показано, що підроблені або змінені прошивки можуть містити приховані бінарні модулі, алгоритми псевдовипадкової генерації доменів, схеми відкладеної активації та елементи архітектури block-safe, що забезпечують стійкі та малопомітні канали C2-взаємодії. Проведений аналіз трафіку, порівняння контрольних сум прошивок та часткова зворотна інженерія виявили типові ознаки компрометації: періодичні cron-виклики, звернення до нестандартних доменних зон, використання нетипових TCP/UDP-портів, а також активацію шкідливого функціоналу у відповідь