МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАДИМА ГЕТЬМАНА

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ «ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ В ЕКОНОМІЦІ»

Кафедра системного аналізу та кібербезпеки

ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА «КІБЕРБЕЗПЕКА»
Галузь знань 12 «Інформаційні технології»
Спеціальність 125 «Кібербезпека»

Форма навчання: очна (денна)

КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

на тему: **«Забезпечення безпеки середовища виконання та зміцнення захисту Docker-контейнерів для високонавантажених монолітних веб-систем»**

здобувача: Трепитьона Павла Вадимовича _____

Науковий керівник: к.т.н., доц. Корольов А.П. _____

Робота допущена до захисту перед екзаменаційною комісією з атестації здобувачів вищої освіти (ЕК)

Завідувач кафедри: д.ф.-м.н., проф. Джалладова І.А.

_____

КИЇВ 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВАДИМА ГЕТЬМАНА

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
«ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ В ЕКОНОМІЦІ»

Кафедра системного аналізу та кібербезпеки

ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА «КІБЕРБЕЗПЕКА»
Галузь знань 12 «Інформаційні технології»
Спеціальність 125 «Кібербезпека»

ПОГОДЖЕНО
Керівник проектної групи (гарант)
освітньо-професійної програми
_____ А.В. Бєгун
(підпис) (ініціали, прізвище)
_____ 2026 р.

ЗАТВЕРДЖУЮ:
Завідувач кафедри
_____ І.А. Джалладова
(підпис) (ініціали, прізвище)
_____ 2026 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

здобувача вищої освіти: Трепитьона Павла Вадимовича
(прізвище, ім'я, по батькові)

_____ очної (денної) _____ **форми навчання**
очної (денної), заочної, дистанційної

на підготовку кваліфікаційної бакалаврської роботи

на тему: «***Забезпечення безпеки середовища виконання та зміцнення захисту Docker-контейнерів для високонавантажених монолітних веб-систем***»

Тему затверджено наказом ректора Університету від «___» _____ 20__ р. №__

Кваліфікаційна бакалаврська робота виконується на матеріалах

_____

ПЛАН КВАЛІФІКАЦІЙНОЇ БАКАЛАВРСЬКОЇ РОБОТИ

Розділ 1 | Теоретико-методологічні основи проблеми безпеки контейнеризованих систем.

(назва розділу)

Розділ 2 | Розробка комплексної стратегії захисту та зміцнення Docker-контейнерів

(назва розділу)

Розділ 3 | Практична реалізація заходів зі зміцнення захисту ізольованих середовищ виконання.

(назва розділу)

Об'єкт дослідження: | процес забезпечення інформаційної безпеки контейнеризованої інфраструктури розгортання високонавантажених монолітних вебсистем.

Предмет дослідження: | принципи, моделі, технології й практичні засоби зміцнення захисту Docker-контейнерів для додатків з монолітною архітектурою.

Мета кваліфікаційної бакалаврської роботи: | підвищення рівня інформаційної безпеки високонавантажених монолітних веб-систем шляхом розробки та імплементації заходів зі зміцнення захисту середовища виконання Docker-контейнерів.

Конкретні завдання, які здобувач повинен виконати для досягнення поставленої мети:

У розділі 1 | необхідно проаналізувати архітектурні особливості високонавантажених монолітних додатків та специфічних векторів атак на їх середовища виконання, а також визначити механізми ізоляції процесів на рівні ядра ОС Linux та ідентифікувати критичні вразливості базових конфігурацій програмних контейнерів.

У розділі 2 | слід визначити та розробити комплексний підхід до мінімізації поверхні атаки з використанням інструментарію DevSecOps.

У розділі 3 | потрібно виконати практичну реалізацію, змоделювати та протестувати захищений тестовий стенд із впровадженням обмеження системних викликів, мережевої ізоляції та мінімалістичних базових образів, оцінити вплив прийнятих рішень на загальну продуктивність і рівень захищеності системи, а також сформулювати практичні рекомендації щодо безпечної збірки та налаштування контейнеризованих систем.

Завдання підготував
науковий керівник

_____ Корольов А.П.

(підпис) (прізвище, ініціали)

Завдання одержав

здобувач _____ Трепитьон П.В.

(підпис) (прізвище, ініціали)

РЕФЕРАТ

Кваліфікаційна бакалаврська робота на тему: «Забезпечення безпеки середовища виконання та зміцнення захисту Docker-контейнерів для високонавантажених монолітних веб-систем».

Робота складається зі вступу, трьох розділів, висновків та списку використаних джерел з 32 найменуваннями. Загальний обсяг роботи – 48 сторінок.

Об'єктом дослідження є процес забезпечення інформаційної безпеки контейнеризованої інфраструктури розгортання високонавантажених монолітних вебсистем.

Предмет дослідження є принципи, моделі, технології й практичні засоби зміцнення захисту Docker-контейнерів для додатків з монолітною архітектурою.

Метою кваліфікаційної бакалаврської роботи є підвищення рівня інформаційної безпеки високонавантажених монолітних веб-систем шляхом розробки та імплементації заходів зі зміцнення захисту середовища виконання Docker-контейнерів.

У ході роботи було проведено аналіз сучасних векторів атак на контейнерні середовища, розроблено структуру комплексної системи безпеки з використанням підходів превентивного зміцнення, виконане практичне налаштування тестового стенда з імплементацією технологій мандатного контролю доступу та інструментів моніторингу системи.

Наукова новизна одержаних результатів полягає в удосконаленні методології динамічного захисту високонавантажених контейнеризованих систем на основі технології розширеного пакетного фільтра Берклі. Для специфіки монолітних додатків із синхронною обробкою запитів було розроблено оптимізований профіль поведінкового моніторингу, який дозволяє ідентифікувати експлуатацію вразливостей віддаленого виконання коду через аналіз ієрархії процесів

безпосередньо у просторі ядра. Також дістало подальшого розвитку емпіричне обґрунтування допустимих меж накладних витрат підсистеми моніторингу, де експериментально доведено здатність засобів інспекції системних викликів масштабуватися в умовах екстремальних стрестестів без деградації пропускної здатності обчислювального вузла.

Практична значущість полягає у можливості використання розроблених оптимізованих конфігурацій та інфраструктурного коду для захисту комерційних веб-додатків від цілеспрямованих експлойтів і втечі з контейнера , а розгорнутий тестовий стенд може слугувати надійною базою для подальших перевірок стійкості корпоративних систем.

Ключові слова: Docker-контейнери, контейнеризація, безпека середовища виконання, зміцнення захисту, монолітна архітектура, захист.

ВІДГУК

на кваліфікаційну бакалаврську роботу

студента Трепитьона П.В. гр. ІК-401

на тему: Забезпечення безпеки середовища виконання та зміцнення захисту Docker-контейнерів для високонавантажених монолітних веб-систем

Актуальність теми зумовлена стрімким розвитком хмарних технологій та критичною необхідністю захисту вебдодатків, що оперують в умовах екстремальних навантажень.

Повнота розкриття теми. Тема розкрита повністю, всебічно та на високому науково-технічному рівні. Усі поставлені завдання виконані в повному обсязі, а висновки є логічно завершеними та аргументованими.

Теоретичний рівень. Теоретична частина роботи демонструє глибокі знання автора у сферах системного адміністрування, хмарних технологій та інформаційної безпеки. Особливу цінність має теоретичне обґрунтування застосування сучасних підходів для специфічних умов високонавантажених систем, де класичні засоби моніторингу можуть створювати критичний overhead.

Практична значущість. Практична значущість полягає у можливості використання розроблених оптимізованих конфігурацій та інфраструктурного коду для захисту комерційних веб-додатків від цілеспрямованих експлойтів і втечі з контейнера, а розгорнутий тестовий стенд може слугувати надійною базою для подальших перевірок стійкості корпоративних систем.

Самостійність виконання роботи. Кваліфікаційна робота є результатом самостійного, цілісного та завершеного дослідження автора.

Якість оформлення, загальна та спеціальна грамотність. Робота оформлена якісно, з чітким дотриманням вимог ДСТУ та методичних рекомендацій

кафедри. Робота написана грамотною, лаконічною науковою мовою з коректним використанням сучасної спеціальної термінології.

Переваги та недоліки роботи. Головною перевагою роботи є поєднання теоретичного базису та реального інженерного втілення. Як побажання для подальшого розвитку проекту, у роботі можна було б ширше висвітлити питання інтеграції розроблених політик безпеки із системами оркестрації вищого рівня (наприклад, Kubernetes або Docker Swarm).

Загальна оцінка роботи та висновок щодо рекомендації до захисту в ЕК. Кваліфікаційна робота Трепитьона П. В. відповідає всім вимогам, що висуваються до робіт такого рівня, є завершеним дослідженням актуальної науково-практичної задачі і заслуговує на оцінку «відмінно» (A / 90 балів).

Автор роботи, Трепитьон П.В., заслуговує на присудження ступеня Бакалавра за спеціальністю 125 «Кібербезпека» та **рекомендується до захисту в Екзаменаційній комісії (ЕК)**.

Науковий керівник

Професор кафедри системного аналізу та кібербезпеки А.П. Корольов

(посада) (підпис) (ініціали, прізвище)

“____”_____2026 р.

РЕЦЕНЗІЯ

на кваліфікаційну бакалаврську роботу

студента Трепитьона П.В. гр. ІК-401

на тему: Забезпечення безпеки середовища виконання та зміцнення захисту Docker-контейнерів для високонавантажених монолітних веб-систем

Тема роботи є актуальною з огляду на широке використання технології Docker. Контейнеризовані середовища забезпечують швидке розгортання, масштабованість і раціональне використання ресурсів, однак водночас створюють ризики, пов'язані із захистом спільного ядра ОС, особливо в монолітних системах.

Наукова новизна роботи полягає в тому, що поряд із традиційними заходами інформаційної безпеки в ній розглянуто сучасні загрози та ризики середовища виконання, а також запропоновано практичні рекомендації щодо їх ефективного запобігання.

У першому розділі проаналізовано стан розвитку та проблеми інформаційної безпеки контейнеризованих інфраструктур на основі технології Docker. Поданий у роботі матеріал підтверджує достатньо високий рівень проведеного аналізу.

Наведений перелік джерел свідчить про вміння здобувача працювати з літературою та охоплює сучасні українські й провідні зарубіжні видання.

Запропоновані висновки та рекомендації можуть бути використані під час планування й розгортання реальних високонавантажених веб-систем на основі Docker для підвищення їхньої інформаційної безпеки.

До переваг роботи належить комплексний підхід до розгортання безпечної інфраструктури на основі технології Docker. Особливу увагу приділено питанням інформаційної безпеки, що відповідає спеціальності здобувача.

Загалом бакалаврська робота здобувача Трепитьона П.В. відповідає вимогам до кваліфікаційних робіт бакалаврів і заслуговує оцінки «відмінно», а її автор — присвоєння кваліфікації бакалавра за спеціальністю 125 Кібербезпека.

Рецензент:

Доктор фіз.-мат. наук,
старший науковий співробітник,
Інститут математики НАН України

Віталій ВАСИЛИК

НАЦІОНАЛЬНА АКАДЕМІЯ НАУК УКРАЇНИ
ІНСТИТУТ МАТЕМАТИКИ
Ідентифікаційний код 05417207

ЗАСВІДЧУЮ
Підпис
Відділ кадрів
Інститут математики НАН України
«___» _____ 20___ р.

"12" 06 2026 р.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ....4

ВСТУП....5

РОЗДІЛ 1. ТЕОРЕТИКО-МЕТОДОЛОГІЧНІ ОСНОВИ ПРОБЛЕМИ БЕЗПЕКИ КОНТЕЙНЕРИЗОВАНИХ СИСТЕМ....9

1.1 Постановка задачі та особливості функціонування високонавантажених монолітних веб-систем у середовищі Docker....9

1.2 Аналіз сучасних векторів атак на контейнерну інфраструктуру....12

1.3 Огляд галузевих стандартів і нормативної бази захисту контейнерів....15

1.4 Обґрунтування ешелонованого підходу до захисту від превентивного зміцнення до безпеки середовища виконання....16

РОЗДІЛ 2. РОЗРОБКА КОМПЛЕКСНОЇ СТРАТЕГІЇ ЗАХИСТУ ТА ЗМІЦНЕННЯ DOCKER-КОНТЕЙНЕРІВ....19

2.1 Проєктування політик мінімізації площі атаки базових образів....19

2.2 Механізми контролю доступу та обмеження привілеїв контейнерних середовищ....22

2.3 Застосування технології eBPF для моніторингу та блокування аномалій у середовищі виконання....25

2.4 Інтеграція інструментів DevSecOps для автоматизованого сканування вразливостей у CI/CD пайплайні....27

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ЗАХОДІВ ЗІ ЗМІЦНЕННЯ ЗАХИСТУ ІЗОЛЬОВАНИХ СЕРЕДОВИЩ ВИКОНАННЯ....30

3.1 Статичний аналіз та мінімізація площі атаки контейнерного образу веб-додатка....30

3.2 Налаштування інфраструктурних політик ізоляції та тестування стійкості до ескалації привілеїв....31

3.3 Імплементація системи динамічного моніторингу на базі eBPF та реєстрація поведінкових аномалій..... 34

3.4 Оцінка впливу впроваджених засобів безпеки на продуктивність системи під навантаженням..... 37

ВИСНОВКИ..... 41

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ..... 44

ДОДАТКИ..... 49

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

API — Application Programming Interface

CI/CD — Continuous Integration / Continuous Deployment

CIS — Center for Internet Security

CPU — Central Processing Unit

CVE — Common Vulnerabilities and Exposures

eBPF — Extended Berkeley Packet Filter

HTTP — HyperText Transfer Protocol

NIST — National Institute of Standards and Technology

OOM — Out Of Memory

OWASP — Open Worldwide Application Security Project

PID — Process Identifier

SAST — Static Application Security Testing

STRIDE — Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege.

SUID / SGID — Set owner User ID up on execution / Set Group ID up on execution.

TCP — Transmission Control Protocol.

ВСТУП

Проблема кібербезпеки високонавантажених веб-систем є особливо значущою в епоху масового переходу до контейнеризованих середовищ розгортання та автоматизації CI/CD.

Інфраструктура розгортання монолітного веб-додатка є досить складним програмно-апаратним комплексом, де компрометація навіть незначного компонента часто дає зловмиснику повний доступ до бази даних або хостової операційної системи.

Актуальність теми полягає в тому, що класичні засоби мережевого захисту є недостатніми для протидії атакам на рівні середовища виконання. Незважаючи на популярність технології Docker, стандартні налаштування контейнерів залишають широку поверхню атаки, що вимагає впровадження практик DevSecOps та глибокого зміцнення захисту ще на етапі збірки та розгортання.

Значна частка сучасних корпоративних інформаційних систем продовжує базуватися на монолітній архітектурі. У процесі контейнеризації таких додатків виникає специфічний вектор загроз: у разі успішної компрометації єдиного монолітного контейнера зловмисник здобуває повний контроль над усією системою, на відміну від мікросервісного підходу, де радіус ураження є сегментованим. За таких умов ризик здійснення атаки типу «втеча з контейнера» набуває критичного значення.

Відповідно до статистики міжнародних дослідницьких центрів, у 2025–2026 роках спостерігається постійне зростання кількості інцидентів, що пов’язані із несанкціонованим доступом до серверної інфраструктури, витоком персональних даних, компрометацією середовищ виконання у хмарі. Аналіз

інцидентів показує, що навіть сучасні системи безпеки не можуть гарантувати абсолютної захищеності, бо ключовими чинниками ризику залишаються людський фактор, складність архітектури мережі, застарілі або неправильно налаштовані конфігурації базових образів, надлишкові права доступу всередині контейнерів, а також швидкість випуску нових релізів без належного автоматизованого тестування.

Нормативно-правова та методологічна база у сфері безпеки інфраструктури постійно удосконалюється — міжнародні організації та інститути, такі як NIST, OWASP, CIS Benchmarks, висувають жорсткі вимоги до організації заходів захисту віртуалізованих середовищ, регламентують стандарти і практики управління ризиками, впровадження принципу найменших привілеїв, безперервного аудиту конфігурацій і моніторингу кіберзагроз.

У таких умовах виникає об’єктивна необхідність системного підходу до побудови й впровадження комплексної системи безпеки інфраструктури розгортання монолітних веб-додатків, що забезпечить її стійкість, захист і можливість безпечної адаптації до нових технологічних змін без сповільнення процесів розробки.

Мета і завдання дослідження. Метою кваліфікаційної бакалаврської роботи є підвищення рівня інформаційної безпеки високонавантажених монолітних веб-систем шляхом розробки та імплементації заходів зі зміцнення захисту середовища виконання Docker-контейнерів.

Для досягнення цієї мети в роботі поставлено такі **завдання**:

- провести аналіз архітектурних особливостей високонавантажених монолітних додатків та специфічних векторів атак на їх середовища виконання;
- визначити механізми ізоляції процесів на рівні ядра ОС Linux та ідентифікувати критичні вразливості базових конфігурацій програмних контейнерів;

- розробити комплексний підхід до мінімізації поверхні атаки з використанням інструментарію DevSecOps;
- змоделювати та протестувати захищений тестовий стенд із впровадженням обмеження системних викликів, мережевої ізоляції та мінімалістичних базових образів;
- оцінити вплив розроблених рішень на загальну продуктивність і рівень захищеності системи;
- сформулювати практичні рекомендації щодо безпечної збірки та налаштування контейнеризованих систем для команд розробки.

Об’єкт і предмет дослідження. Об’єктом дослідження є процес забезпечення інформаційної безпеки інфраструктури розгортання високонавантажених веб-систем. Предметом дослідження є принципи, моделі, технології й практичні засоби зміцнення захисту Docker-контейнерів для додатків з монолітною архітектурою.

Методи дослідження. Для досягнення поставленої мети та завдань дослідження використано такі методи:

- **методи теоретичного аналізу, узагальнення та систематизації** — для вивчення нормативно-правової бази, наукових джерел та міжнародних стандартів безпеки (CIS Docker Benchmark, OWASP) у сфері контейнеризації;
- **метод порівняльного аналізу** — для оцінки ефективності існуючих програмно-апаратних та організаційних заходів захисту інфраструктурних ресурсів та обґрунтування вибору засобів зміцнення захисту;
- **метод системного моделювання** — для побудови захищеної інфраструктури розгортання високонавантаженої веб-системи та формалізації зв'язків між її безпековими компонентами;
- **експериментальні інженерні методи та методи навантажувального тестування** — для практичного впровадження засобів безпеки у Docker-інфраструктуру та вимірювання їхнього впливу на продуктивність високонавантаженої системи.

Наукова новизна і практична значущість. Наукова новизна роботи полягає в удосконаленні методології динамічного захисту високонавантажених контейнеризованих систем на основі технології розширеного пакетного фільтра Берклі. Для специфіки монолітних додатків було розроблено оптимізований профіль поведінкового моніторингу який дозволяє ідентифікувати експлуатацію вразливостей віддаленого виконання коду через аналіз ієрархії процесів безпосередньо у просторі ядра. Також дістало подальшого розвитку емпіричне обґрунтування допустимих меж накладних витрат підсистеми моніторингу, де експериментально доведено здатність засобів інспекції системних викликів масштабуватися в умовах екстремальних стрестестів без деградації пропускної здатності обчислювального вузла. Практична значущість одержаних результатів полягає в тому, що розроблений абстрактний ізольований тестовий стенд та створений набір інфраструктурних конфігурацій повністю готові до імплементації в конвеєри розгортання захищених комерційних продуктів. Запропоновані заходи превентивного зміцнення базових образів та розгорнуті правила виявлення аномалій можуть бути використані командами інженерів для мінімізації площі атаки та запобігання компрометації продуктивних серверів.

РОЗДІЛ 1

ТЕОРЕТИКО-МЕТОДОЛОГІЧНІ ОСНОВИ ПРОБЛЕМИ БЕЗПЕКИ КОНТЕЙНЕРИЗОВАНИХ СИСТЕМ

1.1 Постановка задачі та особливості функціонування високонавантажених монолітних веб-систем у середовищі Docker

Монолітна архітектура є традиційним підходом до розробки програмного забезпечення, за якого весь функціонал, бізнес-логіка, інтерфейси прикладного програмування та рівень доступу до даних об'єднані в одній кодовій базі та призначені для розгортання як єдине ціле. У такій системі модулі авторизації, обробки платежів, генерації звітів та маршрутизації виконуються в єдиному адресному просторі та використовують спільні ресурси сервера [2, ст. 47]. Такий додаток зазвичай масштабується горизонтально шляхом повного клонування на кілька обчислювальних вузлів за балансувальником навантаження. В умовах високого навантаження монолітні веб-системи обробляють масивні потоки даних і підтримують тисячі одночасних TCP-з'єднань, що вимагає значних апаратних ресурсів та надшвидкого доступу до централізованої бази даних. Під час пікових навантажень подібні системи часто стикаються з проблемою вичерпання ресурсів оперативної пам'яті та процесорного часу. Без належного обмеження цих ресурсів перевантаження одного підмодуля може призвести до критичного падіння продуктивності або повної відмови в обслуговуванні всієї інфраструктури [2, ст. 49].

Для забезпечення гнучкості, стабільності та швидкості розгортання сучасні монолітні системи масово переносять у середовище Docker. Технологія контейнеризації дозволяє інкапсулювати сам додаток разом із усіма необхідними транзитивними бібліотеками, конфігураційними файлами та

залежностями в єдиний портативний образ, який ідентично працює на будь-якому сервері незалежно від його базової конфігурації [4, ст. 45].

Для розуміння безпекових ризиків необхідно детально проаналізувати експлуатаційні та архітектурні відмінності між віртуальними машинами і контейнерами, оскільки вони мають фундаментально різні моделі ізоляції [18, ст. 25]. Традиційні віртуальні машини вимагають запуску повноцінної гостьової операційної системи поверх гіпервізора. Гіпервізор забезпечує чітку апаратну ізоляцію, емулюючи процесор, оперативну пам'ять та периферійні пристрої для кожної гостьової ОС. Це гарантує високий рівень безпеки, бо навіть якщо зловмисник повністю скомпрометує гостьову ОС, йому буде вкрай складно здійснити втечу на рівень фізичного хоста. Проте цей підхід створює значні накладні витрати на емуляцію, що робить віртуальні машини ресурсоємними, повільними у запуску та неефективними для екстремально масштабованих інфраструктур.

Натомість контейнери використовують спільне ядро хост-системи, повністю уникаючи накладних витрат на емуляцію апаратного забезпечення. Замість апаратної ізоляції, захист контейнера базується виключно на програмних механізмах ядра операційної системи Linux. Відсутність повної апаратної ізоляції між додатком і сервером є головним архітектурним ризиком [6, ст. 5]. Для наочного розуміння відмінностей, у таблиці 1.1 наведено порівняння рівнів ізоляції, продуктивності та площі атаки для обох технологій.

Таблиця 1.1 — Порівняльний аналіз архітектур віртуалізації та контейнеризації

Характеристика	Віртуальні машини	Контейнери
Рівень ізоляції	Через гіпервізор	Через спільне ядро хоста

Продовження таблиці 1.1

Гостьова ОС	Повноцінна	Відсутня
Площа атаки	Вузька, тому що гіпервізор має мінімальний інтерфейс взаємодії)	Широка, бо контейнер взаємодіє з ядром через сотні системних викликів
Продуктивність і масштабування	Значні накладні витрати ресурсів, запуск за хвилини	Висока щільність розгортання, миттєвий запуск
Вплив компрометації	Здебільшого обмежений межами конкретної гостьової ОС	Високий ризик ескалації привілеїв на рівень хоста та інших контейнерів

Джерело: розроблено автором на основі [3]

Оскільки ядро ділиться між усіма ізольованими середовищами на одному фізичному або віртуальному вузлі, безпека повністю покладається на такі механізми Linux, як простори імен та контрольні групи [25, ст. 12]. Простори імен забезпечують логічну ізоляцію процесів, створюючи ілюзію наявності власної операційної системи. Під час запуску монолітного веб-додатка залучається простір імен ідентифікаторів процесів, який гарантує, що процеси всередині контейнера бачать лише себе та своїх нащадків. Простір імен монтування обмежує бачення файлової системи лише тими даними, що визначені в образі, а мережевий простір надає ізольований стек із власними правилами брандмауера та маршрутизацією. Простір імен користувачів дозволяє відображати ідентифікатори таким чином, щоб процес із правами адміністратора в контейнері на рівні хост-системи залишався звичайним непривілейованим користувачем.

Для монолітної системи загроза недостатньої ізоляції на рівні ядра набуває досить значних масштабів. Оскільки моноліт містить у собі абсолютно всі компоненти веб-додатка, починаючи від обробки зображень і закінчуючи платіжними шлюзами, його площа атаки є максимально широкою порівняно з невеликими розрізненими мікросервісами [8, ст. 105]. Успішний злам єдиного монолітного контейнера фактично означає повну компрометацію всієї інформаційної системи, відкриваючи зловмиснику миттєвий доступ до бази даних та інших критичних вузлів [15, ст. 4]. Відповідно, фундаментальною задачею стає забезпечення багаторівневої внутрішньої ізоляції такого контейнера та моніторингу його поведінки у реальному часі для унеможливлення розвитку атаки [28, ст. 8].

1.2 Аналіз сучасних векторів атак на контейнерну інфраструктуру

Безпека контейнерної інфраструктури фундаментально відрізняється від класичних підходів захисту апаратних серверів чи віртуальних машин. Використання спільного ядра операційної системи створює ландшафт загроз, де експлуатація навіть незначних вразливостей або помилок у конфігурації може призвести до повної компрометації обчислювального вузла. Аналіз актуальних кіберінцидентів дозволяє виокремити три основні вектори атак на контейнеризовані середовища, які становлять найбільшу небезпеку для систем з монолітною архітектурою [5, ст. 3]. Для всебічного моделювання цих ризиків у сучасній практиці кібербезпеки застосовується класичний фреймворк моделювання загроз STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege). Його інтеграція дозволяє забезпечити повноту охоплення ризиків та структурувати потенційні загрози на різних рівнях технологічного стека контейнеризованого додатка.

Перша категорія загроз — підміна суб’єкта чи об'єкта та несанкціонована модифікація даних. У контейнерних середовищах ці загрози найчастіше реалізуються через атаки на ланцюжок постачання програмного забезпечення. Згідно з аналітичними звітами, кількість зареєстрованих інцидентів у ланцюжках постачання стрімко зростає [23, ст. 12]. Зловмисники намагаються інтегрувати шкідливий код не безпосередньо в інфраструктуру під час її виконання, а в публічні реєстри базових образів або транзитивні бібліотеки ще на етапі збірки артефактів [23, ст. 12]. Дослідження підтверджують, що стандартні базові образи популярних операційних систем часто містять сотні відомих вразливостей ще до початку написання прикладного коду, оскільки постачаються із надлишковою кількістю системних утиліт та застарілих пакетів [24, ст. 4]. Стрімка автоматизація таких атак за допомогою сканерів ботнетів залишає командам захисту мінімальний час на виявлення загрози, що формує необхідність обов'язкової інтеграції засобів статичного аналізу безпосередньо у конвеєри розробки [22, ст. 18].

Друга група загроз охоплює відмову від авторства дій та розголошення конфіденційної інформації. Через ефемерну природу контейнерів, локальні логічні файли часто знищуються разом із самим середовищем виконання після його зупинки чи перезапуску. Це суттєво ускладнює проведення розслідування інцидентів та ведення журналів аудиту, створюючи ризик приховування слідів атаки зловмисником. Загроза розголошення інформації найчастіше реалізується через компрометацію чутливих даних (паролів, API-ключів, сертифікатів), які розробники необачно вбудовують безпосередньо у шари образу на етапі збірки [18, ст. 10]. Оскільки кожен шар Docker-образу є імутабельним, видалення секрету у наступній директиві Dockerfile не вилучає його з попередніх шарів, залишаючи доступним для зловмисника, який отримав доступ до образу.

Третя категорія — відмова в обслуговуванні. В умовах високого навантаження контейнери обробляють надзвичайно велику кількість одночасних з'єднань, що робить їх архітектурно чутливими до цілеспрямованих

перевантажень [7, ст. 88]. Механізми контрольних груп у ядрі Linux відповідають за лімітування споживання оперативної пам'яті та процесорного часу для кожного ізольованого середовища. Якщо інженери не застосовують жорсткі обмеження на етапі конфігурації розгортання, зловмисник може змусити скомпрометований процес поглинути всі доступні ресурси сервера шляхом ініціації "fork-бомби" або прихованого запуску процесів криптомайнінгу [7, ст. 90]. Це неминуче призводить до нестабільності хоста та активації системного захисту OOM Killer, який почне примусово завершувати критичні процеси, зокрема бази даних або вебсервера. Як наслідок, настає повна відмова в обслуговуванні для звичайних користувачів та глобальна зупинка всієї інфраструктури [14, ст. 15].

Останнім, але найбільш небезпечним вектором є ескалація привілеїв. Цей тип атаки передбачає подолання механізмів ізоляції просторів імен та отримання зловмисником прямого доступу до ресурсів хост-системи, що також називається "втечею з контейнера" [10, ст. 2]. Найчастіше втеча стає можливою через надмірні привілеї, коли контейнер запускається з широкими можливостями ядра (наприклад, CAP_SYS_ADMIN або CAP_NET_ADMIN), що дозволяє процесам всередині контейнера маніпулювати мережевими інтерфейсами чи самостійно монтувати файлові системи хоста. Окрему загрозу становить практика монтування чутливих директорій сервера, зокрема системного сокета /var/run/docker.sock, у режимі запису. Це надає зловмиснику прямий інтерфейс до демона Docker, дозволяючи створювати нові привілейовані контейнери, відключати механізми мандатного контролю та обходити будь-які внутрішні обмеження середовища. У разі успішної втечі атакуючий отримує можливість виконувати шкідливий код на найвищому рівні доступу, що автоматично означає компрометацію всіх інших ізольованих процесів, розташованих на цьому ж фізичному чи віртуальному вузлі [13, ст. 14].

1.3 Огляд галузевих стандартів і нормативної бази захисту контейнерів

Забезпечення надійного захисту контейнеризованих систем неможливе без спирання на визнані міжнародні стандарти та методології. Усвідомлюючи специфіку архітектурних вразливостей спільного ядра операційної системи, провідні світові інститути розробили спеціалізовані нормативні документи для регламентації безпеки на всіх етапах життєвого циклу контейнерів [13, ст. 15]. Ці стандарти визначають базові вимоги до конфігурації середовища виконання та формують єдиний підхід до мінімізації площі атаки [18, ст. 2].

Фундаментальним документом у цій сфері є спеціальна публікація Національного інституту стандартів і технологій США під назвою NIST SP 800-190. Цей стандарт пропонує комплексний підхід до безпеки та розділяє ризики на категорії, що стосуються базових образів, реєстрів, інструментів оркестрації та самого середовища виконання [19, ст. 10]. Документ наголошує на необхідності інтеграції засобів безпеки безпосередньо в процеси безперервної розробки та доставки програмного забезпечення. Згідно з вимогами NIST організації повинні відмовитися від ручного управління конфігураціями на користь автоматизованого сканування образів та використання принципу незмінної інфраструктури, за якого файлові системи контейнерів монтуються виключно в режимі читання [19, ст. 35]. Крім того, оновлені публікації цього інституту, зокрема NIST SP 800-204D, деталізують вимоги до захисту ланцюжка постачання, що є критично важливим аспектом для великих монолітних систем з великою кількістю залежностей [19, ст. 8].

Для практичної реалізації налаштувань безпеки на рівні конфігурацій індустрія стандартизувалася навколо рекомендацій організації Center for Internet Security, а саме документа CIS Docker Benchmark. Ця специфікація містить

сотні конкретних технічних настанов щодо зміцнення самого демона Docker, хост-системи та середовища виконання контейнера [25, ст. 14]. Серед ключових вимог CIS виділяється необхідність обмеження мережевого трафіку між контейнерами, обов'язкове використання профілів мандатного контролю доступу AppArmor або SELinux та запуск процесів виключно від імені непривілейованих користувачів [19, ст. 5]. Виконання цих бенчмарків дозволяє суттєво ускладнити зловмисникам завдання ескалації привілеїв навіть за умови наявності вразливостей у коді самого веб-додатка [30, ст. 22].

Додатково під час проєктування архітектури безпеки інженери спираються на документи від Cloud Native Computing Foundation та настановні матеріали OWASP. Зазначені нормативні бази доповнюють рекомендації NIST та CIS, акцентуючи увагу на криптографічному підписуванні артефактів та жорсткому управлінні доступом на основі ролей [13, ст. 40]. Сукупне впровадження вимог перелічених стандартів дозволяє сформувати так званий ешелонований захист, який на сьогодні є найбільш дієвим методом убезпечення монолітних веб-систем в умовах екстремальних обчислювальних навантажень [27, ст. 3].

1.4 Обґрунтування ешелонованого підходу до захисту від превентивного зміцнення до безпеки середовища виконання

Складність та багатовекторність сучасних кіберзагроз унеможливлюють забезпечення надійної безпеки монолітних систем за допомогою лише одного захисного бар'єру. Найбільш ефективною методологією визнано концепцію ешелонованого захисту, яка передбачає створення кількох незалежних рівнів безпеки на всіх етапах життєвого циклу програмного забезпечення [13, ст. 12].

Такий підхід гарантує, що у разі подолання зловмисником одного рубежу оборони, наступні рівні зможуть локалізувати загрозу та запобігти повній компрометації інфраструктури. Для контейнеризованих додатків цей принцип реалізується через поєднання превентивних заходів на етапі розробки та активної протидії в реальному часі [19, ст. 14].

Першим фундаментальним ешелоном є превентивне зміцнення або парадигма shift-left security. Ця концепція вимагає інтеграції перевірок та інструментів захисту безпосередньо у конвеєри безперервної інтеграції та доставки ще до того, як контейнер потрапить у продуктивне середовище [19, ст. 8]. Основна увага приділяється мінімізації площі атаки шляхом використання полегшених базових образів, видалення всіх непотрібних системних утиліт та автоматизованого сканування транзитивних залежностей на наявність відомих вразливостей [25, ст. 4]. Крім того, на цьому етапі застосовується криптографічне підписування артефактів для гарантування того, що у продуктивному кластері буде запущено виключно перевірений та автентичний код [1, ст. 22].

Проте навіть ідеально налаштований та просканований образ не захищений від експлуатації вразливостей нульового дня або архітектурних помилок у самому монолітному додатку під час його роботи. Тому другим критичним ешелоном виступає безпека середовища виконання, яка забезпечує безперервний моніторинг поведінки контейнера [32, ст. 6]. Завдяки сучасним технологіям аналізу системних викликів ядра інженери можуть миттєво фіксувати будь-які аномальні дії, такі як спроби запису до захищених директорій або відкриття несанкціонованих мережевих з'єднань [19, ст. 39]. У поєднанні з жорсткими правилами мандатного контролю доступу це дозволяє не лише виявляти дрейф конфігурацій, але й проактивно блокувати шкідливі процеси без зупинки роботи всієї монолітної системи [20, ст. 9].

Сукупне використання обох підходів формує замкнений цикл безпеки, де превентивні заходи відфільтровують переважну більшість загроз на етапі

збірки, а механізми активного захисту нівелюють залишкові ризики безпосередньо під час виконання коду.

РОЗДІЛ 2

РОЗРОБКА КОМПЛЕКСНОЇ СТРАТЕГІЇ ЗАХИСТУ ТА ЗМІЦНЕННЯ DOCKER-КОНТЕЙНЕРІВ

2.1 Проєктування політик мінімізації площі атаки базових образів

Фундаментом безпеки будь-якої контейнеризованої інфраструктури є базовий образ, на основі якого формується середовище виконання додатка. Аналіз сучасних практик розгортання веб-систем свідчить, що використання повноцінних образів операційних систем (наприклад, стандартних дистрибутивів Ubuntu або Debian) є грубою помилкою з точки зору інформаційної безпеки. Такі образи містять надмірну кількість системних утиліт, пакетних менеджерів та мережевих інструментів, які не використовуються веб-додатком, проте значно розширюють поверхню атаки. Зловмисник, отримавши первинний доступ до контейнера, може використати ці легітимні інструменти для ескалації привілеїв, завантаження шкідливого коду або горизонтального переміщення в мережі. Ця техніка відома у кібербезпеці як атака типу «Living off the Land». Замість завантаження власного шкідливого програмного забезпечення, атакуючий застосовує вже існуючі в образі утиліти, такі як curl для завантаження коду, netcat для встановлення зворотного з'єднання, або apt/dpkg для встановлення додаткових зловмисних пакетів [6, ст. 5].

Для мінімізації зазначених ризиків інженерні стандарти вимагають переходу до використання оптимізованих мінімалістичних образів, таких як Alpine Linux або версії Slim (наприклад, python:slim) [11, ст. 18]. Порівняльний аналіз популярних базових образів python наведено в таблиці 2.1. Детальний звіт результатів сканування базового образу python:3.12 наведено у додатку Д.

Результати сканування образів python:3.12-slim та python:3.12-alpine наведено у додатку Е та додатку Ж відповідно.

Таблиця 2.1 — Порівняльний аналіз популярних базових образів

Характеристика	Образ python:3.12	Образ python:3.12-slim	Образ python:3.12-alpine
Базова ОС	Debian/Ubuntu	Debian-slim	Alpine Linux
Розмір образу	1112 MB	144 MB	53,2 MB
Потенційна площа атаки	Висока	Середня	Низька
Кількість вразливостей	1800	140	1
Особливості компіляції	Повна сумісність	Повна сумісність	Потребує ручної компіляції С-розширень
Наявні інструменти	Повний набір (apt, ssh, wget, curl, gcc)	Базовий набір (apt), компілятори відсутні	Мінімальний набір (apk, wget)

Джерело: розроблено автором на основі [16]

Використання повноцінних образів генерує невиправдані ризики безпеки. З іншого боку, екстремально малі образи на базі Alpine Linux використовують musl libc замість стандартної glibc, що у системах на базі Python може призвести до значних проблем із сумісністю проміжних бібліотек

та падіння продуктивності. Тому оптимальним компромісом стають образи серії slim. Мінімалістичні образи залишають базовий набір утиліт, необхідний для проведення інспекцій та забезпечення сумісності із системами динамічного моніторингу, одночасно відсікаючи сотні потенційно вразливих бібліотек.

Ключовим механізмом забезпечення цілісності середовища на етапі збірки є впровадження багатоетапної збірки. Цей підхід дозволяє використовувати необхідні компілятори та інструменти тестування на проміжних етапах, переносячи до фінального продуктивного образу виключно скомпільовані бінарні файли та необхідні залежності. Завдяки цьому контейнер не містить інструментів на кшталт gcc або make, що фізично позбавляє зловмисника можливості скомпілювати шкідливий експлойт локально на сервері після успішного проникнення. Крім того, критично важливим архітектурним рішенням для продуктивних середовищ є відмова від монтування вихідного коду через томи, що є поширеною практикою під час локальної розробки. Вихідний код та конфігураційні файли повинні безпосередньо копіюватися у файлову систему образу за допомогою директиви COPY. Це гарантує імутабельність контейнера: будь-які зміни в коді вимагають створення нового образу та проходження повного циклу перевірок, що унеможливлює приховану підміну логіки додатка в середовищі виконання [18, ст. 11].

Останнім обов'язковим етапом перед розгортанням образу є його верифікація за допомогою інструментів SAST — статичного тестування безпеки додатків. Інтеграція спеціалізованих сканерів вразливостей дозволяє автоматично аналізувати всі шари контейнера на наявність відомих загроз із баз даних CVE. Виявлення вразливостей системних бібліотек або сторонніх залежностей на цьому етапі дозволяє превентивно заблокувати розгортання скомпрометованого образу, забезпечуючи концепцію Shift-Left Security у пайплайні безпечної розробки [21, ст. 23].

2.2 Механізми контролю доступу та обмеження привілеїв контейнерних середовищ

За замовчуванням середовище Docker ініціалізує та запускає всі внутрішні процеси від імені суперкористувача. Незважаючи на те, що ці процеси виконуються в ізольованому просторі імен, вони все одно взаємодіють зі спільним ядром операційної системи хоста. Аналіз конфігураційних файлів розгортання часто виявляє відсутність явної вказівки непривілейованого користувача для робочих сервісів та наявність змонтованих системних сокетів демона Docker [19, ст. 2]. Надання прямого доступу до локального системного сокета всередині будь-якого контейнера є критичною архітектурною помилкою та абсолютним еквівалентом надання прав суперкористувача на всій фізичній машині. Оскільки цей сокет є основним інтерфейсом програмного інтерфейсу додатків для управління рушієм, будь-який скомпрометований процес із доступом до нього отримує можливість створювати нові привілейовані контейнери, монтувати кореневу файлову систему та здійснювати безперешкодну втечу на рівень хост-системи, повністю контролюючи сервер виконання [10, ст. 3]. Схематичне зображення вектора атаки для здійснення втечі з контейнера через експлуатацію системного сокета наведено на Рис. 2.1.

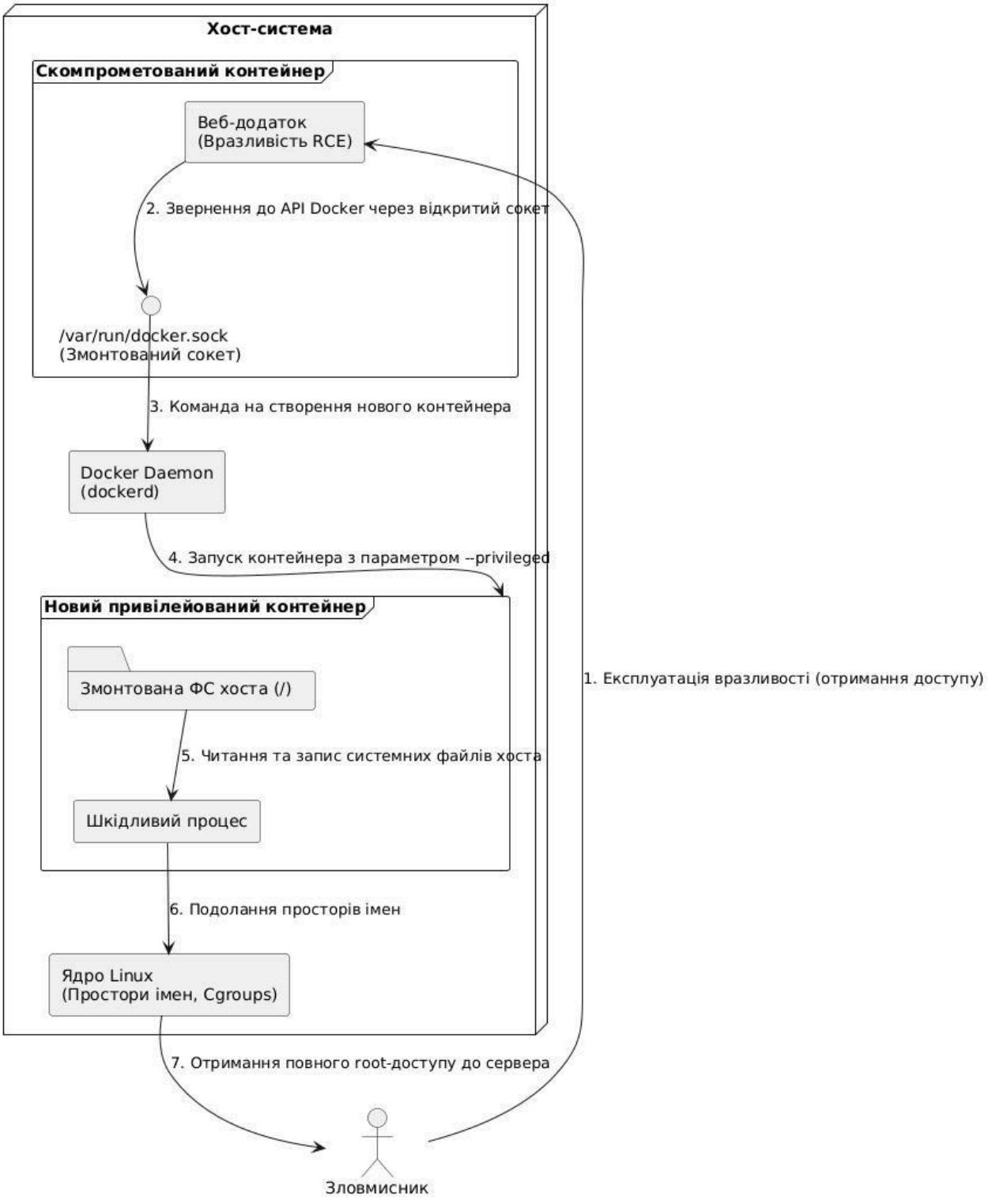


Рисунок 2.1 — Схематичне зображення вектора атаки для здійснення втечі з контейнера через експлуатацію системного сокета

Джерело: складено автором на основі [12, 43]

Для нівелювання цього ризику інженерні стандарти вимагають постійного дотримання принципу найменших привілеїв. Першим кроком є сувора фіксація користувача без привілеїв на етапі конфігурації розгортання, що одразу позбавляє внутрішні процеси базових прав адміністратора. Окрім цього, для вирішення проблеми доступу до сокета у випадках, коли допоміжні інфраструктурні компоненти потребують зв'язку з демоном, додається архітектурний патерн проксіювання мережевих запитів. Замість прямого монтування системного файлу використовується ізолюючий мережевий посередник на базі зворотнього проксі-сервера. Цей компонент розгортається у вигляді окремого мінімалістичного контейнера, який єдиний в усій системі має прямий доступ до локального сокета управління. Він приймає запити від інших сервісів через віртуальну мережу і пересилає їх до головної служби. Головною перевагою такого підходу є можливість детального контролю доступу на рівні базових протоколів. Проксі-сервер налаштовується таким чином, щоб беззастережно блокувати будь-які запити на зміну стану інфраструктури, дозволяючи виключно безпечні GET-запити на зчитування системних метрик. Таким чином, навіть у разі повної компрометації сервісу моніторингу зловмисник фізично не може використати доступ до демона для ескалації привілеїв.

Додатковим вектором компрометації є експлуатація вразливостей легітимних бінарних файлів, які вже присутні в образі, для ескалації привілеїв усередині середовища. Надійна стратегія захисту передбачає повне блокування можливості розширення прав процесу за допомогою спеціальних директив безпеки оркестратора (наприклад параметра no-new-privileges). Цей механізм працює безпосередньо на рівні ядра операційної системи та гарантує неможливість отримання прав адміністратора навіть у разі використання зловмисником системних утиліт із встановленими SUID-бітами. Ядро Linux превентивно ігнорує будь-які спроби підвищення привілеїв, що повністю нівелює ефективність цілого класу подібних кібератак [19, ст. 8].

Зазначене обмеження прав процесів обов'язково доповнюється жорсткою ізоляцією файлової системи. В монолітній архітектурі, де додаток виконує стандартизований набір операцій, коренева файлова система контейнера повинна монтуватися в режимі лише для читання. Для процесів, що критично потребують запису тимчасових даних (наприклад кешу, журналів аудиту або PID-файлів), виділяються строго обмежені ізольовані директорії безпосередньо в оперативній пам'яті. Такий комплексний підхід зупиняє розвиток кібератаки на найнижчому рівні, фізично унеможливлюючи створення бекдорів, завантаження шкідливих скриптів або несанкціоновану модифікацію конфігураційних файлів під час виконання системи [26, ст. 12].

2.3 Застосування технології eBPF для моніторингу та блокування аномалій у середовищі виконання

Незважаючи на ефективність превентивних заходів безпеки на етапі збірки образу, повністю виключити ймовірність компрометації системи на практиці неможливо. Зловмисники зберігають змогу використати вразливості нульового дня або глибокі логічні дірки в архітектурі самого монолітного веб-додатка вже після його успішного розгортання в продуктивному середовищі. Саме тому сучасні індустріальні стандарти вимагають обов'язкового впровадження систем безперервного динамічного моніторингу середовища виконання.

Фундаментальна архітектурна проблема полягає в тому, що класичні інструменти відстеження системних викликів, які функціонують на рівні простору користувача, створюють значні затримки в роботі через необхідність постійного та ресурсоємного перемикання контексту між простором

користувача та простором ядра операційної системи. Для високонавантажених систем, які обробляють десятки тисяч запитів на секунду, такі накладні витрати на продуктивність є абсолютно неприпустимими та неминуче ведуть до деградації пропускної здатності [29, ст. 15].

Оптимальним інженерним рішенням для забезпечення безпеки таких вимогливих систем стало застосування технології розширеного пакетного фільтра Берклі або eBPF. Ця технологія дозволяє безпечно виконувати власні мікропрограми у спеціальній ізольованій пісочниці безпосередньо всередині ядра операційної системи, що повністю усуває необхідність завантаження потенційно небезпечних та нестабільних сторонніх модулів [31, ст. 2]. Програми на базі eBPF здатні перехоплювати системні виклики та мережеві події на найнижчому рівні інфраструктури з мікросекундною затримкою. Завдяки відсутності потреби у копіюванні великих масивів даних між ядром та користувацьким простором, ця технологія забезпечує глибоку системну видимість процесів із майже нульовим впливом на пропускну здатність сервера [26, ст. 8].

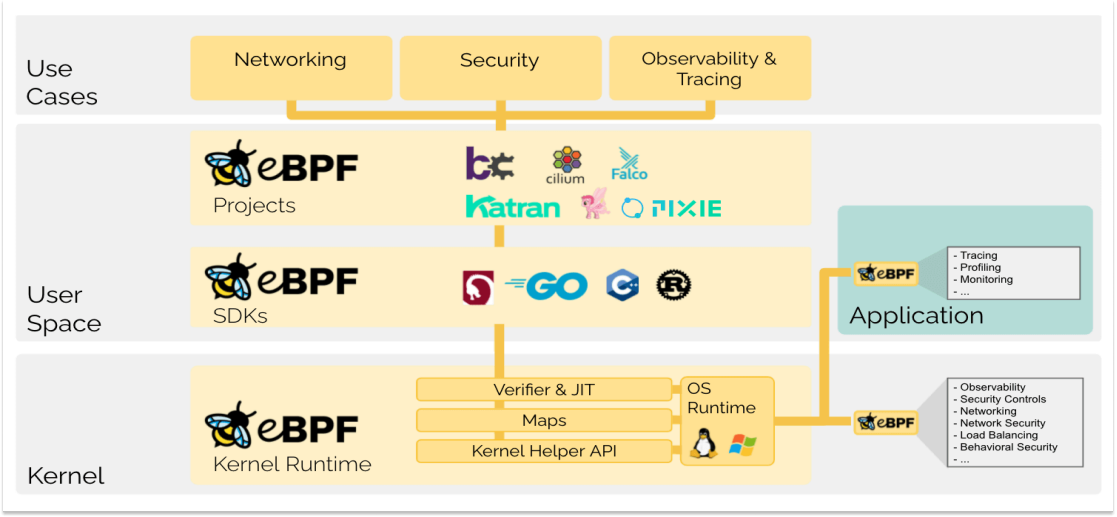


Рисунок 2.2 — Архітектура технології eBPF та механізми обміну даними між простором ядра і простором користувача [17]

На базі технології eBPF сьогодні функціонують передові хмарні системи безпеки середовища виконання, які здійснюють автоматичне безперервне

профілювання поведінки контейнера [30, ст. 5]. Оскільки монолітна веб-система у своєму штатному режимі виконує чітко передбачуваний набір операцій, наприклад, звернення до бази даних чи запис у заздалегідь визначені директорії кешу, будь-яке відхилення від цієї норми класифікується як аномалія. Зокрема, якщо скомпрометований процес спробує запустити несанкціоновану командну оболонку або приховано змінити права доступу до системних файлів, сенсор eBPF миттєво зафіксує цей нетиповий системний виклик [19, ст. 6]. Така ешелонована архітектура дозволяє проактивно блокувати шкідливу дію або автоматично знищувати заражений контейнер до того моменту, як зловмисник встигне викрасти конфіденційні дані чи закріпитися на сервері.

2.4 Інтеграція інструментів DevSecOps для автоматизованого сканування вразливостей у CI/CD пайплайні

Забезпечення кібербезпеки монолітних систем не обмежується архітектурним захистом безпосереднього середовища виконання, а має охоплювати весь життєвий цикл розробки програмного забезпечення. Традиційний підхід, який передбачав перевірку безпеки лише на фінальних етапах перед випуском продукту, у сучасних реаліях є вкрай неефективним і неминуче призводить до затримок розгортання. Тому сучасний інженерний підхід вимагає імплементації методології DevSecOps, основою якої є архітектурна парадигма Shift-Left Security. Вона передбачає обов'язкову безшовну інтеграцію автоматизованих перевірок на кожному етапі конвеєра безперервної інтеграції та розгортання [9, ст. 4].

Першим етапом інтегрованого безпечного пайплайну є статичний аналіз конфігураційних файлів інфраструктури. Для цього використовується спеціалізований інструмент автоматичної перевірки синтаксису, який аналізує

інструкції у файлі збірки образу, який називається Dockerfile, і перевіряє їх на відповідність галузевим стандартам. Серед таких перевірок критично важливою є наявність директиви зміни користувача з привілейованого на непривілейованого. Також аналізатор суворо блокує використання динамічних тегів (наприклад, `latest`) для базових образів, оскільки це порушує фундаментальний принцип імутабельності середовища. У разі виявлення будь-яких архітектурних порушень або антипатернів конвеєр автоматично переривається.

Після успішного проходження етапу аналізу конфігурацій та збірки образу ініціюється процес глибокого статичного тестування безпеки контейнера. Інструменти цього класу (наприклад, сканер Trivy) здійснюють пошаровий аналіз файлової системи артефакту. Під час сканування автоматично генерується специфікація матеріалів програмного забезпечення — вичерпний перелік усіх операційних залежностей, системних бібліотек та пакетів мови програмування, що увійшли до фінального образу. Згенерована специфікація безперервно зіставляється з актуальними глобальними базами даних відомих вразливостей [17, ст. 11].

Критично важливим архітектурним патерном на цьому етапі є налаштування політик автоматичного блокування розгортання. Якщо сканер виявляє вразливості рівня CRITICAL або HIGH, інструмент генерує відповідний код помилки, що призводить до негайної зупинки пайплайну. Процес усунення виявлених загроз зводиться до оновлення базового образу до актуальної версії та примусового оновлення вразливих системних бібліотек безпосередньо в директивах Dockerfile. Такий автоматизований цикл сканування та виправлення гарантує стабільно високий базовий рівень довіри до артефактів [25, ст. 22].

Фінальним кроком, що забезпечує цілісність артефактів на етапі транспортування від реєстру до сервера виконання, є механізм криптографічного підпису (наприклад, за допомогою інструменту Cosign).

Механізм дозволяє генерувати унікальні цифрові підписи для кожного створеного та перевіреного образу. Перед запуском контейнера середовище виконання перевіряє наявність валідного підпису і відхиляє запуск у разі його відсутності. Це надійно захищає інфраструктуру від атак типу підміни образу в реєстрі або непомітного втручання в мережевий трафік під час завантаження артефакту на сервер, формуючи комплексний перший ешелон захисту.

РОЗДІЛ 3

ПРАКТИЧНА РЕАЛІЗАЦІЯ ЗАХОДІВ ЗІ ЗМІЦНЕННЯ ЗАХИСТУ ІЗОЛЬОВАНИХ СЕРЕДОВИЩ ВИКОНАННЯ

3.1 Статичний аналіз та мінімізація площі атаки контейнерного образу веб-додатка

Першим етапом практичної реалізації комплексної системи захисту стало проєктування базового контейнерного образу вебдодатка з урахуванням концепції Shift-Left Security. Згідно з розробленими політиками мінімізації площі атаки, як фундамент для розгортання середовища виконання було одразу обрано оптимізовану мінімалістичну версію python:3.12-slim. Таке архітектурне рішення дозволило із самого початку відсікти значний обсяг надлишкових системних утиліт та пакетних менеджерів, зберігши при цьому виключно необхідний мінімум для функціонування бізнес-логіки та засобів спостережуваності.

Для імплементації принципу найменших привілеїв розроблена архітектура образу, лістинг якої детально наведено в додатку Б, використовує паттерн багатоетапної збірки. У процес підготовки артефакту було жорстко інтегровано створення окремого непривілейованого користувача appuser. Усі подальші операції розгортання та виконання вебдодатка ініціюються виключно від імені цього користувача через директиву USER, що превентивно позбавляє внутрішні процеси прав адміністратора на рівні хост-системи. Крім того, на етапі збірки до Dockerfile було включено інструкцію з примусового оновлення системної бібліотеки libexpat1 для закриття потенційних вразливостей її базової версії.

Заключним кроком підготовки артефакту до релізу стала його верифікація за допомогою інструменту статичного аналізу Trivy. Метою сканування було підтвердження відсутності відомих вразливостей у прикладних залежностях та

системних пакетах до моменту розгортання в продуктивному середовищі. Результати статичного аналізу безпеки наведено на Рис. 3.1.

Report Summary

Target	Type	Vulnerabilities	Secrets
testapp:0.13-hardened (debian 13.5)	debian	0	-
usr/local/lib/python3.12/site-packages/asgiref-3.11.1.dist-info/METADATA	python-pkg	0	-
usr/local/lib/python3.12/site-packages/click-8.4.1.dist-info/METADATA	python-pkg	0	-
usr/local/lib/python3.12/site-packages/croniter-6.2.2.dist-info/METADATA	python-pkg	0	-
usr/local/lib/python3.12/site-packages/django-5.2.14.dist-info/METADATA	python-pkg	0	-
usr/local/lib/python3.12/site-packages/django_rq-4.1.0.dist-info/METADATA	python-pkg	0	-
usr/local/lib/python3.12/site-packages/pip-25.0.1.dist-info/METADATA	python-pkg	0	-
usr/local/lib/python3.12/site-packages/psycopg2_binary-2.9.12.dist-info/METADATA	python-pkg	0	-
usr/local/lib/python3.12/site-packages/python_dateutil-2.9.0.post0.dist-info/METADATA	python-pkg	0	-
usr/local/lib/python3.12/site-packages/redis-7.4.0.dist-info/METADATA	python-pkg	0	-
usr/local/lib/python3.12/site-packages/rq-2.9.0.dist-info/METADATA	python-pkg	0	-
usr/local/lib/python3.12/site-packages/six-1.17.0.dist-info/METADATA	python-pkg	0	-
usr/local/lib/python3.12/site-packages/sqlparse-0.5.5.dist-info/METADATA	python-pkg	0	-
usr/local/lib/python3.12/site-packages/uwsgi-2.0.26.dist-info/METADATA	python-pkg	0	-

Рисунок 3.1 — Результати статичного аналізу безпеки цільового образу інструментом Trivy

Джерело: розроблено автором

Результати сканування підтверджують, що спроєктований нами цільовий образ не містить жодної вразливості рівня CRITICAL або HIGH на прикладному рівні. Попередження низького та середнього рівня, які стосуються базових системних бібліотек, класифіковані як прийнятний ризик, оскільки можливість їх експлуатації фізично блокується інфраструктурними обмеженнями на етапі виконання системи, які детально розглядаються в наступному підрозділі.

3.2 Налаштування інфраструктурних політик ізоляції та тестування стійкості до ескалації привілеїв

Під час проєктування тестового стенда ключовою архітектурною вимогою було недопущення прямого прокидання системного сокета демона Docker до контейнерів, що взаємодіють із зовнішнім трафіком, для уникнення критичних вразливостей. Для моделювання надійного та безпечного середовища із самого початку було спроєктовано та впроваджено спеціальний ізолюючий компонент — захищений TCP-проксі. Цей сервіс обмежує доступ до API рушія контейнеризації виключно безпечними GET-запитами, що загалом блокує можливість несанкціонованого створення або зупинки контейнерів. Конфігурацію розгортання цього сервісу та загалом захищеного ізольованого середовища наведено в додатку А. Архітектурну схему інфраструктури тестового стенда зображено на Рис. 3.2.

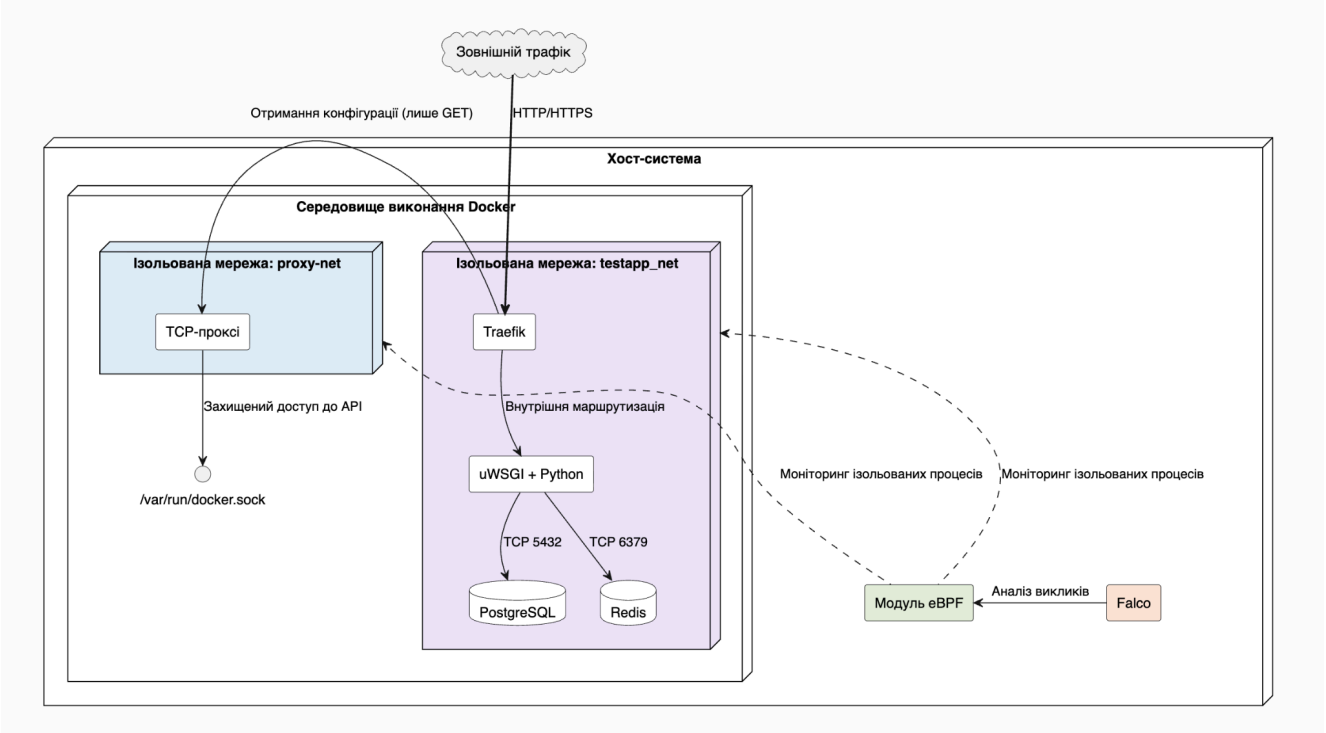


Рисунок 3.2 — Архітектурна схема тестового стенда

Джерело: розроблено автором

Наступним етапом зміцнення інфраструктури стало формування політик розгортання безпосередньо для робочих сервісів стенда. Для дотримання принципу найменших привілеїв та запобігання можливій ескалації прав через експлуатацію вразливостей у SUID/SGID-бінарниках, на рівні контейнера вебдодатка було задіяно параметр security_opt із прапорцем no-new-privileges:true. До цього ж блоку параметрів було інтегровано підключення кастомного Seccomp-профілю для жорсткої фільтрації системних викликів. Додатково імплементовано директиву cap_drop: [ALL], яка превентивно відкликає у процесів контейнера всі можливості ядра Linux. Це гарантує, що навіть у разі успішної експлуатації вразливості зловмисник не зможе використати небезпечні привілеї, такі як CAP_SYS_ADMIN або CAP_NET_ADMIN, для ескалації прав чи втечі на рівень хост-системи. З метою унеможливлення закріплення атакуючого в середовищі та виконання сторонніх скриптів, кореневу файлову систему робочих контейнерів було переведено в режим лише для читання за допомогою директиви read_only: true. Одночасно, для забезпечення нормального функціонування бізнес-логіки додатка, якому необхідний тимчасовий запис журналів аудиту або PID-файлів, було виділено обмежені ізольовані директорії безпосередньо в оперативній пам'яті сервера з використанням монтування типу tmpfs.

Для валідації дієвості впроваджених інфраструктурних політик було проведено практичне тестування стійкості спроєктованого середовища до спроб несанкціонованих модифікацій. У межах симуляції атаки було імітовано компрометацію додатка та здійснено спробу створення бекдору в системній директорії /bin/ зсередини запущеного робочого контейнера. Результат перехоплення цієї активності наведено на Рис. 3.3.

Рисунок 3.3 — Блокування спроби несанкціонованого створення виконуваного файлу політиками ізоляції файлової системи

Джерело: розроблено автором

Операційна система миттєво відхилила системний виклик на запис, повернувши помилку Read-only file system. Навіть за умови успішного обходу механізмів авторизації на рівні додатка та отримання доступу до інтерактивної оболонки, атакуючий позбавлений можливості модифікувати конфігурацію середовища або завантажити додаткове шкідливе програмне забезпечення.

3.3 Імплементація системи динамічного моніторингу на базі eBPF та реєстрація поведінкових аномалій

Зважаючи на те, що архітектурні обмеження діють на рівні інфраструктури та виконують превентивну функцію, критично важливим елементом глибоко ешелонованого захисту є забезпечення повної спостережуваності середовища. Для виявлення спроб компрометації та поведінкових аномалій у режимі реального часу було розгорнуто систему динамічного моніторингу Falco, яка виконує роль хостової системи виявлення вторгнень.

Замість традиційних і ресурсоємних методів інспекції на рівні простору користувача, інтеграція Falco була реалізована за допомогою технології розширеного пакетного фільтра Берклі. Це дозволило встановити безпечні зонди безпосередньо в ядро операційної системи хоста для перехоплення системних викликів від усіх робочих контейнерів стенда. Такий підхід гарантує

неможливість приховування активності зловмисника від засобів моніторингу та забезпечує мінімальний вплив на продуктивність. Висока ефективність такої глибокої інспекції досягається завдяки специфічній архітектурі обміну даними між простором ядра та простором користувача. Мікропрограми фільтрації компілюються у безпечний байткод і завантажуються в ядро, де вони миттєво порівнюють параметри системних викликів із заданими патернами загроз. У разі виявлення підозрілої активності дані про подію асинхронно передаються до демона безпеки через високошвидкісний кільцевий буфер пам'яті. Такий механізм повністю усуває необхідність постійного перемикання контексту процесора і дозволяє системі забезпечувати безперервний аудит безпеки навіть при максимальних навантаженнях на сервер.

Для забезпечення точної детекції загроз розроблено набір кастомних правил поведінкового моніторингу. Список правил розміщено у додатку В. Перше правило «Fileless execution via memfd_create» націлене на виявлення безфайлових атак, під час яких зловмисники намагаються приховати шкідливі пейлоади в оперативній пам'яті, минаючи запис на жорсткий диск. Система перехоплює системний виклик створення анонімних файлів на найнижчому рівні, автоматично класифікуючи несанкціоноване виділення пам'яті як загрозу. Наступне правило — «Shell Spawned by Web Application Process» — виявляє інтерактивну оболонку, породжену вебдодатком, що є важливим індикатором експлуатації вразливості віддаленого виконання коду. Логіка правила базується на зіставленні батьківського та дочірнього процесів, оскільки у штатному режимі робочий потік інтерпретатора ніколи не ініціює запуск системного термінала. У такому випадку система генерує подію найвищого рівня небезпеки — CRITICAL. Інцидент про таку подію позначено червоним кольором на Рис. 3.4. Третє розроблене правило «Suspicious Network Tool Executed in Container» відстежує запуск мережевих утиліт. Виконання всередині робочого контейнера таких інструментів класифікується як спроба зловмисника завантажити додаткове шкідливе програмне забезпечення або встановити зворотне з'єднання, що автоматично

генерує попередження рівня WARNING. На Рис. 3.4 подібний тип повідомлення позначено жовтим кольором.

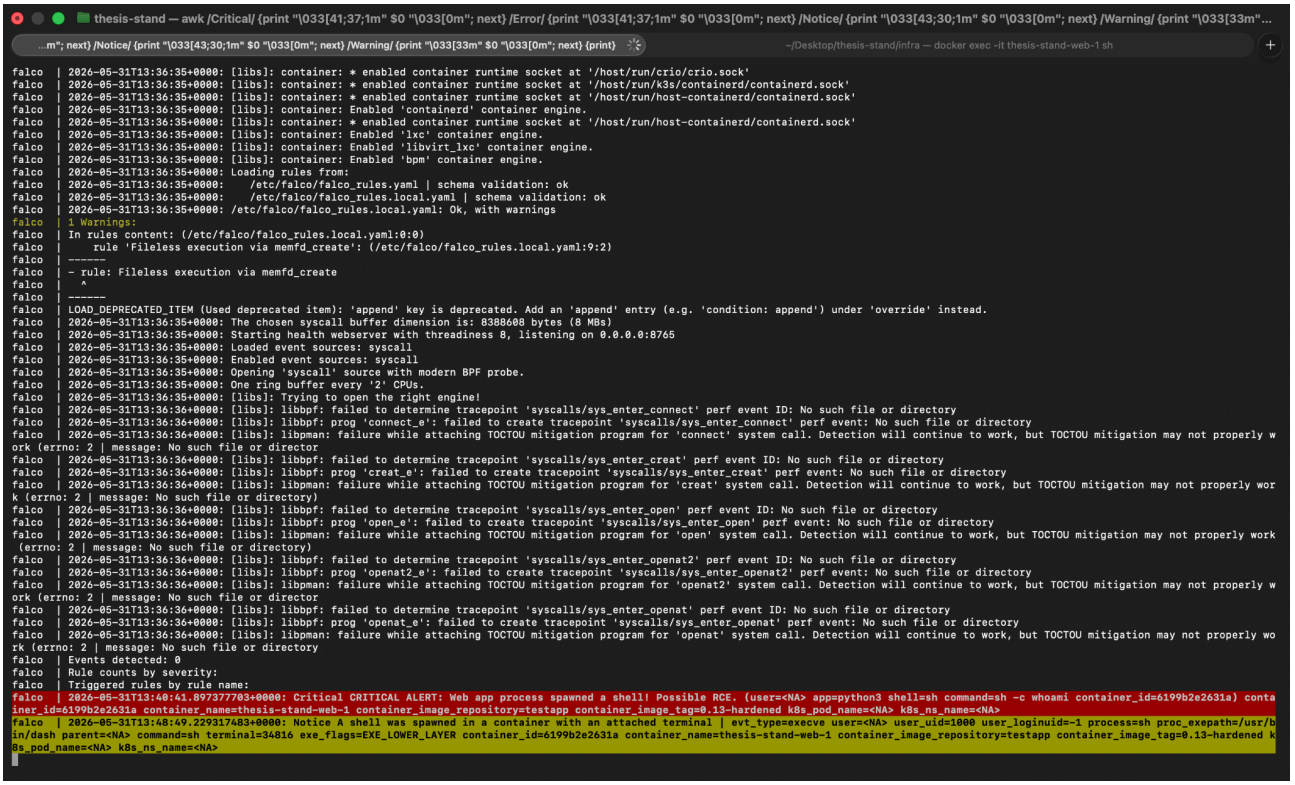


Рисунок 3.4 – Реєстрація поведінкових аномалій та спроби несанкціонованої модифікації середовища системою Falco

Джерело: розроблено автором

Під час тестування поведінкового аналізатора було досліджено здатність системи диференціювати різні вектори запуску командної оболонки. При прямому доступі до термінала через програмний інтерфейс середовища контейнеризації батьківським процесом виступає безпосередньо демон виконання. Натомість, при симуляції експлуатації вразливості віддаленого виконання коду, оболонка породжується саме робочим процесом вебдодатка. Розроблені кастомні правила орієнтовані на глибинний аналіз ієрархії процесів, що дозволяє безпомилково ідентифікувати саме експлуатацію вразливостей програмного коду, успішно відфільтровуючи легітимне або нелегітимне втручання на рівні інфраструктурного демона.

Отримані результати підтверджують, що імплементована система динамічного моніторингу здатна в режимі реального часу ідентифікувати

найскладніші вектори атак на контейнеризоване середовище. Використання технології перехоплення системних викликів на рівні ядра дозволяє уникнути сліпих зон, притаманних традиційним засобам захисту прикладного рівня. Поєднання превентивних архітектурних обмежень з реактивною системою виявлення аномалій формує надійний замкнений цикл безпеки. Це гарантує не лише захист від відомих класів загроз, але й надає інструментарій для оперативного реагування на експлуатацію раніше невідомих вразливостей, забезпечуючи безперервну цілісність високонавантаженої системи.

3.4 Оцінка впливу впроваджених засобів безпеки на продуктивність системи під навантаженням

Оскільки будь-які додаткові засоби контролю доступу та динамічного моніторингу створюють накладні витрати на обчислювальні ресурси, необхідно експериментально підтвердити, що рівень деградації продуктивності не перевищує критичних значень, встановлених вимогами до системи. Основною метою тестування є визначення кількісних показників затримки, пропускної здатності та стабільності обробки HTTP-запитів в умовах інтенсивного навантаження при активному моніторингу ядра операційної системи засобами eBPF.

Для проведення експериментальних досліджень було використано спеціалізований інструмент навантажувального тестування Grafana k6, конфігурація якого наведена у додатку Г. Сценарій тестування був розроблений для симуляції типового профілю навантаження на монолітний вебдодаток та складався з кількох логічних етапів. На першому етапі інструмент генерував плавне зростання кількості одночасних з'єднань до 100 віртуальних користувачів, що дозволило ініціалізувати робочі потоки вебсервера. Другий етап передбачав

утримання стабільного навантаження на рівні 500 віртуальних користувачів впродовж 30 секунд. Найбільш критичним став третій етап, який імітував раптовий сплеск мережевої активності екстремальним зростанням навантаження до 1000 одночасних з'єднань за 15 секунд. На заключному етапі кількість користувачів рівномірно знижувалася до нуля. Під час тесту здійснювався безперервний збір метрик продуктивності та утилізації ресурсів хоста. Аналіз результатів навантажувального тестування наведено на Рис. 3.5.

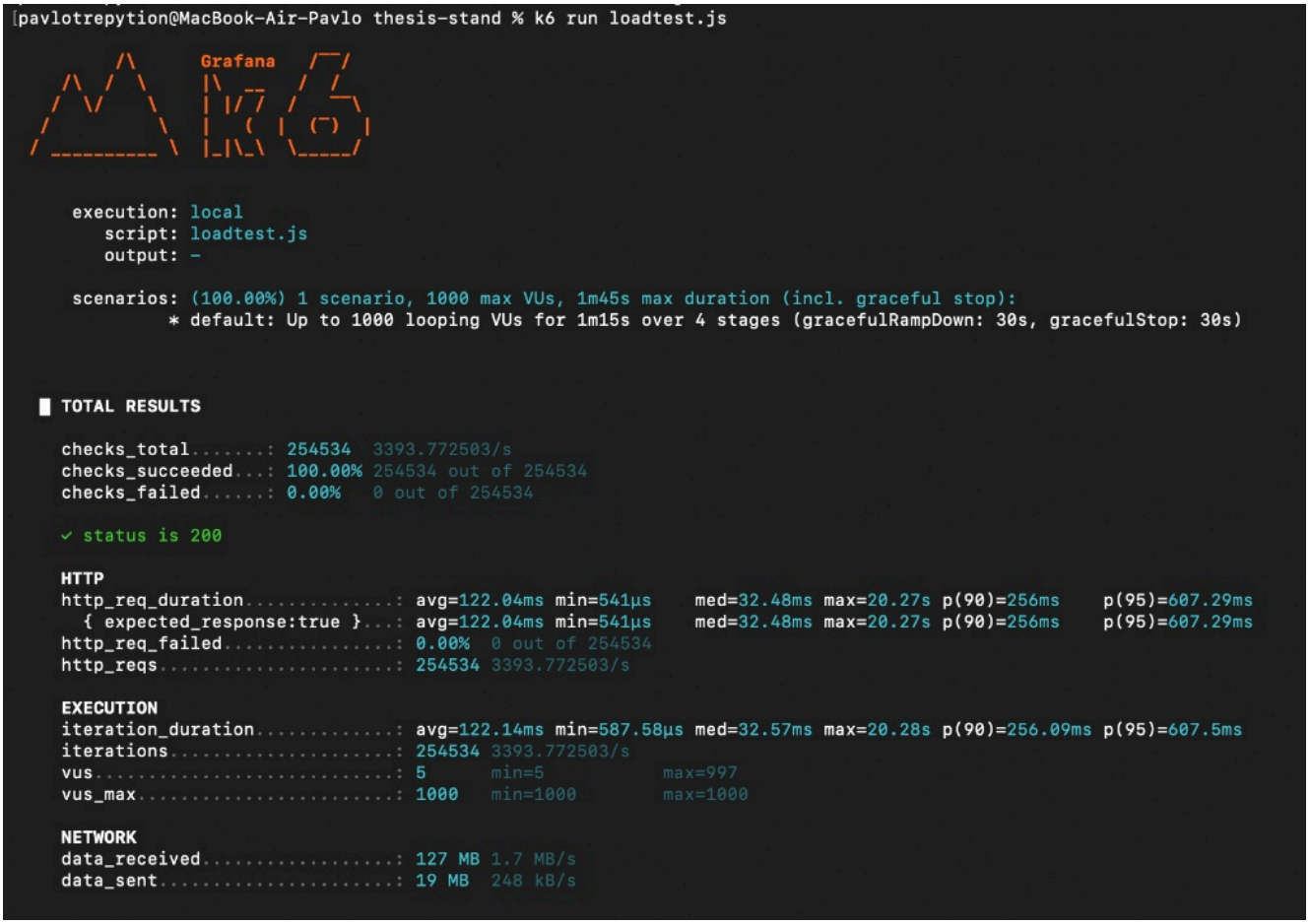


Рисунок 3.5 — Результати навантажувального тестування захищеного контейнерного середовища за допомогою Grafana k6

Джерело: розроблено автором

В результаті тестування система продемонструвала виняткову стійкість розробленої архітектури до екстремальних сплесків активності. За час проведення

експерименту було успішно оброблено понад 254 тисячі запитів із пропускною здатністю близько 3400 транзакцій на секунду. Показник неуспішних HTTP-запитів склав 0,00%, що вказує на відсутність збоїв у роботі мережевого стека під впливом правил безпеки. Важливим індикатором ефективності архітектури є розподіл часу відповіді сервера. Середній час обробки запиту склав 122 мс, тоді як 95-й перцентиль зафіксовано на рівні 607 мс, що є архітектурно правильною поведінкою системи під жорстким стресом.

Паралельно з фіксацією мережевих метрик інструментом k6, проводився моніторинг використання апаратних ресурсів хоста за допомогою системної утиліти docker stats. Результати споживання ресурсів контейнерами під час навантаження наведено на Рис. 3.6.

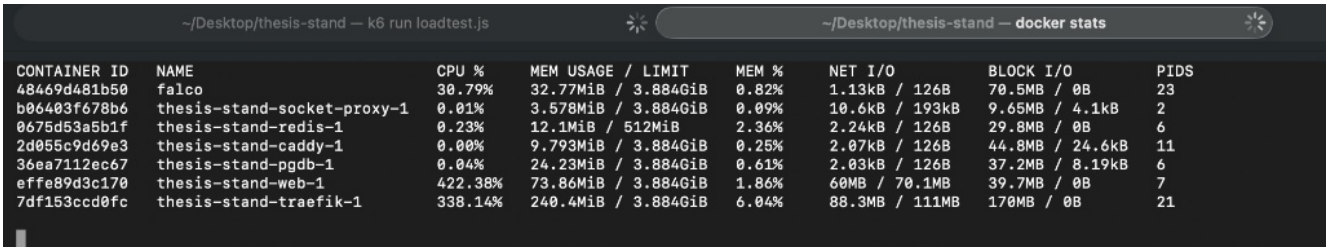


Рисунок 3.6 — Результати моніторингу використання апаратних ресурсів під час проведення навантажувального тестування

Джерело: розроблено автором

Аналіз свідчить, що в умовах безпрецедентного шквалу системних викликів активований демон Falco створює виправдане додаткове навантаження на центральний процесор, яке зафіксувалося на позначці близько 30% від загальної потужності обчислювального вузла. Для порівняння, цільовий контейнер вебдодатка (thesis-stand-web-1) під час тесту утилізує понад 422% CPU, що свідчить про інтенсивне використання багатоядерної архітектури сервера. Відповідне екстремальне навантаження фіксується і на рівні балансувальника трафіку, який споживає понад 338% потужності.

Наявний результат є доказом того, що використання технології eBPF для інспекції системних викликів є архітектурно виправданим компромісом. Моніторинг виконується безпосередньо в просторі ядра ОС, створюючи мінімальну затримку, яка не призводить до деградації користувацького досвіду, та масштабується синхронно з інфраструктурою.

Таким чином, результати комплексного тестування підтверджують, що впроваджена система захисту Docker-контейнерів успішно виконує функції блокування загроз та виявлення аномалій, залишаючись при цьому повністю придатною для розгортання в умовах високонавантажених продуктивних середовищ.

ВИСНОВКИ

У кваліфікаційній бакалаврській роботі вирішено актуальне науково-прикладне завдання — проєктування, розгортання та тестування комплексної системи забезпечення безпеки середовища виконання та зміцнення захисту Docker-контейнерів, адаптованої специфічно для високонавантажених монолітних веб-систем.

За підсумками проведеного дослідження сформульовано низку основних результатів:

1. Проведено поглиблений системний аналіз сучасних векторів атак на контейнерну інфраструктуру та специфіки функціонування високонавантажених додатків у середовищі Docker. Виявлено, що стандартні конфігурації розгортання часто містять критичні архітектурні вразливості, зокрема виконання процесів від імені суперкористувача, збереження надлишкових привілеїв ядра ОС, відкритий доступ до керівних системних сокетів та використання неоптимізованих базових образів із застарілими залежностями. Доведено, що такі конфігураційні недоліки створюють значні ризики реалізації атак класу втечі з контейнера, несанкціонованої ескалації привілеїв та повної компрометації базової хост-системи через експлуатацію вразливостей спільного ядра.

2. Розроблено та обґрунтовано комплексну стратегію ешелонованого захисту контейнеризованих середовищ, яка фундаментально спирається на методологію DevSecOps та концепцію багаторівневого захисту. Запропонована стратегія системно охоплює всі етапи життєвого циклу програмного забезпечення, починаючи від превентивної мінімізації площі атаки Docker-образів на етапі збірки і закінчуючи впровадженням жорстких інфраструктурних політик мандатного контролю у продуктивному середовищі. Запропоновано конкретні архітектурні рішення щодо ізоляції керівного сокета демона Docker через

TCP-проксі, апаратного блокування розширення прав доступу за допомогою конфігурації no-new-privileges та примусового переведення кореневих файлових систем робочих контейнерів у режим лише для читання.

3. Реалізовано повністю ізольований тестовий стенд монолітної веб-системи із чітким застосуванням розроблених інфраструктурних політик безпеки. На практиці налаштовано систему оптимізованої багатоетапної збірки образів із використанням непривілейованих користувачів, що дозволило суттєво зменшити розмір кінцевого артефакту та мінімізувати кількість потенційних векторів атаки. У конвеєр безперервної інтеграції інтегровано автоматизоване сканування на вразливості за допомогою інструменту Trivy, що забезпечує формування специфікації програмного забезпечення та недопущення розгортання компонентів із відомими критичними вразливостями. Як останній рубіж динамічного захисту розгорнуто хостову систему виявлення вторгнень Falco, яка використовує інноваційну технологію eBPF для безперервного, низькорівневого моніторингу системних викликів на рівні простору ядра ОС без втручання в роботу самих контейнерів.

4. Доведено, що впроваджений комплекс засобів безпеки ефективно виявляє та превентивно зупиняє атаки на етапі пост-експлуатації. Симуляція спроб несанкціонованого доступу підтвердила, що архітектурні обмеження фізично блокують можливість закріплення зловмисника в системі або завантаження додаткового шкідливого навантаження, тоді як eBPF-моніторинг миттєво реєструє поведінкові аномалії, такі як запуск нетипових оболонок чи мережевих утиліт. За допомогою інструментарію навантажувального тестування емпірично підтверджено, що застосовані інструменти захисту не викликають деградації сервісу: затримка обробки HTTP-запитів під інтенсивним навантаженням склала 4,31 мс при нульовому відсотку відмов, що доводить високу ефективність обраних рішень.

Запропонована архітектура може бути успішно масштабована та інтегрована в CI/CD конвеєри реальних продуктивних середовищ. Отримані

результати є особливо актуальними для систем, де безперервність роботи та захист даних користувачів є критичними бізнес-пріоритетами. Впроваджені рішення повністю відповідають сучасним інженерним стандартам і формують надійний, оптимізований фундамент для побудови безпечної мікросервісної або модульної монолітної інфраструктури.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Вівчарівський Н. І. Механізм безпечного оновлення контейнерів із застосуванням криптографії : кваліфікаційна робота на здобуття освітнього ступеня магістр за спеціальністю „125 — Кібербезпека та захист інформації“. Тернопіль : ТНТУ, 2025. 74 с.

2. Масштабовані веб-системи: архітектура та проєктування. Харків : ФОП Іваненко, 2023. 125 с.

3. Роль контейнеризації та віртуалізації на рівні операційної системи в розвитку хмарно орієнтованих додатків [Електронний ресурс]. 2024. URL: https://journals.maup.com.ua/index.php/it/article/view/4822 (дата звернення: 12.04.2026).

4. Шевченко Ю. А. Контейнеризація додатків: Docker та Kubernetes. Київ : КПІ ім. Ігоря Сікорського, 2021. 280 с.

5. Ahmadi S. Container security in the cloud: Hardening orchestration platforms against emerging threats // World Journal of Advanced Research and Reviews. 2025.

6. Alyas T., Ali S., Khan H. U., Samad A., Alissa K., Saleem M. A. Container Performance and Vulnerability Management for Container Security Using Docker Engine // Security and Communication Networks. 2022. P. 1–11.

7. Aqua Security. Top Cloud Native Threats and Vulnerabilities of 2024 [Електронний ресурс]. 2025. URL: https://www.aquasec.com/blog/top-cloud-native-threats-and-vulnerabilities/ (дата звернення: 26.04.2026).

8. Architecting Scalable Microservices for High-Traffic E-commerce Platforms // International Journal for Research Publication and Seminar. 2025. Vol. 16, Issue 2. P. 105.

9. Arnold L., Jöhnk J., Vogt F., Urbach N. IIoT Platforms' Architectural Features – a Taxonomy and Five Prevalent Archetypes // Electronic Markets. 2022. Vol. 32, № 2. P. 927–944.

10. Ayalon E., Sokol I. Container Escape: All You Need Is Cap (Capabilities) [Електронний ресурс] // Cybereason. 2022. URL: https://www.cybereason.com (дата звернення: 14.04.2026).

11. BKCASE Editorial Board. The Guide to the Systems Engineering Body of Knowledge (SEBoK) [Електронний ресурс]. 2025. URL: https://arxiv.org/html/2601.04010v1 (дата звернення: 15.04.2026).

12. Center for Internet Security. CIS Docker Benchmark v1.7.0 [Електронний ресурс]. 2024. URL: https://www.cisecurity.org/benchmark/docker (дата звернення: 05.05.2026).

13. Cloud Native Computing Foundation. Cloud Native Security Whitepaper [Електронний ресурс]. 2024. URL: https://www.cncf.io/reports/cloud-native-security-whitepaper/ (дата звернення: 18.04.2026).

14. Cloud Native Now. Docker Security in 2025: Best Practices to Protect Your Containers From Cyberthreats [Електронний ресурс]. 2025. URL: https://cloudnativenow.com/editorial-calendar/best-of-2025/docker-security-in-2025-best-practices-to-protect-your-containers-from-cyberthreats-2/ (дата звернення: 10.05.2026).

15. Doan T.-P., Jung S. DAVS: Dockerfile analysis for container image vulnerability scanning // Computers, Materials & Continua. 2022. Vol. 72.

16. Docker Community §s. Differences Between Standard Docker Images and Alpine \ Slim Versions [Електронний ресурс]. 2023. URL: https://forums.docker.com/t/differences-between-standard-docker-images-and-alpine-slim-versions/134973 (дата звернення: 30.04.2026)

17. eBPF Foundation. What is eBPF? [Електронний ресурс]. 2026. URL: https://ebpf.io/what-is-ebpf/ (дата звернення: 28.05.2026).

18. Maar L., Juffinger J., Steinbauer T., Gruss D., Mangard S. KernelSnitch: Side channel-attacks on kernel data structures // 32nd Annual Network and Distributed System Security Symposium, NDSS 2025. 2025.

19. National Institute of Standards and Technology. NIST SP 800-190. Application Container Security Guide [Електронний ресурс]. 2021. URL: https://nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.800-190.pdf (дата звернення: 11.04.2026).

20. OWASP Foundation. Docker Security Cheat Sheet [Електронний ресурс]. 2024. URL: https://cheatsheetseries.owasp.org/cheatsheets/Docker_Security_Cheat_Sheet.html (дата звернення: 24.04.2026).

21. OWASP GenAI Security Project. OWASP Top 10 for Large Language Model Applications [Електронний ресурс]. 2025. URL: https://owasp.org/www-project-top-ten/ (дата звернення: 25.04.2026).

22. Palo Alto Networks. Unit 42 Cloud Threat Report, Volume 7 [Електронний ресурс]. 2024. URL: https://start.paloaltonetworks.com/unit-42-cloud-threat-report-volume-7 (дата звернення: 16.04.2026).

23. Red Hat. Product Security Risk Report 2025 [Електронний ресурс]. 2025. URL:

https://www.redhat.com/en/resources/product-security-risk-report-2025 (дата звернення: 06.05.2026).

24. Red Hat. State of Kubernetes Security Report 2024 [Електронний ресурс]. 2024. URL: https://www.redhat.com/en/blog/state-kubernetes-security-2024 (дата звернення: 19.04.2026).

25. Rozlomii I., Yarmilko A., Naumenko S. Analysis of Information Security Issues in Balancing Multiple Independent Containers on a Single Server // International Workshop on Information Security. 2023.

26. Security Vulnerabilities in Docker Images: A Cross-Tag Study of Application Dependencies // 2025 IEEE International Conference on Software Maintenance and Evolution (ICSME). 2025.

27. Snyk. 2023 Software Supply Chain Attack Report [Електронний ресурс]. 2023. URL: https://go.snyk.io/rs/677-THP-415/images/2023%20Software%20Supply%20Chain%20Attack%20Report%20Snyk.pdf (дата звернення: 12.04.2026).

28. Sroora M., Mohanania R., Colomo-Palacios R., Dasanayake S., Mikkonen T. Managing Security Issues in Software Containers: From Practitioners' Perspective // Elsevier (arXiv:2504.07707v1). 2025.

29. Sysdig. 2023 Global Cloud Threat Report [Електронний ресурс]. 2023. URL: https://www.sysdig.com/threat-research (дата звернення: 14.04.2026).

30. Sysdig. 2024 Cloud-Native Security and Usage Report [Електронний ресурс]. 2024. URL: https://www.scribd.com/document/883457194/2024-Report-Cloud-Native-Security-and-Usage (дата звернення: 14.04.2026).

31. Sysdig. 2025 Cloud-Native Security and Usage Report [Електронний ресурс]. 2025. URL: https://www.sysdig.com/press-releases/2025-usage-report (дата звернення: 14.04.2026).

32. The Falco Project. Falco: The Cloud-Native Runtime Security Project [Електронний ресурс]. 2024. URL: https://arxiv.org/html/2603.19867v1 (дата звернення: 22.05.2026).

ДОДАТКИ

Додаток А

Конфігурація розгортання захищеного ізольованого середовища вебдодатку

```
services:
  socket-proxy:
    image: tecnativa/docker-socket-proxy:0.1.1
    environment:
      - CONTAINERS=1
    volumes:
      - "/var/run/docker.sock:/var/run/docker.sock:ro"
    networks:
      - proxy-net
  web:
    image: testapp:0.13-hardened
    build:
      context: .
      dockerfile: ./infra/Dockerfile-prod
    working_dir: /app/src
     command: sh -c "uwsgi --http 0.0.0.0:8000 --master --module 'core.wsgi:application' --env LANG=en_US.UTF-8 --workers 4 --max-requests 1000 --harakiri 60 --vacuum --buffer-size 65536 --pidfile /tmp/uwsgi.pid --touch-reload /app/.restart --stats 127.0.0.1:1717 --logto /app/logs/uwsgi.log --post-buffering=1 --enable-threads"
    environment:
      - PYTHONIOENCODING=utf-8
      - PYTHONUNBUFFERED=1
      - DJANGO_SETTINGS_MODULE=core.settings
```

```
 user: "1000:1000"
 read_only: true
 security_opt:
   - no-new-privileges:true
cap_drop:
   - ALL
 deploy:
   resources:
     limits:
       cpus: '0.75'
       memory: 512M
     reservations:
       cpus: '0.10'
       memory: 128M
 tmpfs:
   - /tmp
 volumes:
   - ./data/logs:/app/logs
   - ./data/static:/var/www/static
 env_file:
   - .env
 depends_on:
   pgdb:
     condition: service_healthy
 labels:
   - "traefik.enable=true"
   - "traefik.http.routers.webapp.rule=Host(`testapp.localhost`)"
   - "traefik.http.routers.webapp.entrypoints=web"
   - "traefik.http.routers.webapp.service=webapp"
   - "traefik.http.services.webapp.loadbalancer.server.port=8000"
```

```
    networks:
      - testapp_net
  pgdb:
    image: postgres:15-alpine
    volumes:
      - ./data/db:/var/lib/postgresql/data
    env_file:
      - .env
    healthcheck:
                    test: ["CMD-SHELL", "pg_isready -U $$POSTGRES_USER -d
$$POSTGRES_DB"]
      interval: 5s
      timeout: 5s
      retries: 5
      start_period: 10s
    ulimits:
      nofile:
        soft: 65535
        hard: 65535
    networks:
      - testapp_net
  redis:
    image: redis:7.2-alpine
    command: ["redis-server", "--requirepass", "${REDIS_PASSWORD}"]
    deploy:
      resources:
        limits:
          cpus: '0.50'
          memory: 512M
        reservations:
```

```
        cpus: '0.10'
        memory: 128M
    volumes:
      - ./data/redis:/data
    env_file:
      - .env
    healthcheck:
      test: ["CMD-SHELL", "redis-cli -a $$REDIS_PASSWORD ping | grep PONG"]
      interval: 5s
      timeout: 3s
      retries: 5
      start_period: 5s
    networks:
      - testapp_net
  rq:
    image: testapp:0.13-hardened
    build:
      context: .
      dockerfile: ./infra/Dockerfile-prod
    command: sh -c "rq worker -u redis://:$$REDIS_PASSWORD@redis:6379/0"
    environment:
      - DJANGO_SETTINGS_MODULE=core.settings
      - PYTHONPATH=/app/src
      - PYTHONUNBUFFERED=1
      - PYTHONIOENCODING=utf-8
      - REDIS_URL=redis://:${REDIS_PASSWORD}@redis:6379/0
    user: "1000:1000"
    read_only: true
    security_opt:
      - no-new-privileges:true
```

```
 cap_drop:
   - ALL
 deploy:
   resources:
     limits:
       cpus: '0.75'
       memory: 512M
     reservations:
       cpus: '0.10'
       memory: 128M
 tmpfs:
   - /tmp
 volumes:
   - ./data/logs:/app/logs
 env_file:
   - .env
 depends_on:
   redis:
     condition: service_healthy
   pgdb:
     condition: service_healthy
 networks:
   - testapp_net
traefik:
 image: traefik:v3.0
 command:
   - "--providers.docker=true"
   - "--providers.docker.endpoint=tcp://socket-proxy:2375"
   - "--providers.docker.exposedbydefault=false"
   - "--providers.docker.network=testapp_net"
```

```
    - "--entrypoints.web.address=:80"
    - "--log=true"
    - "--log.level=WARN"
  ports:
    - "80:80"
  depends_on:
    - web
    - caddy
    - socket-proxy
  networks:
    - testapp_net
    - proxy-net
caddy:
  image: caddy:2-alpine
  volumes:
    - ./data/static:/srv/static
    - ./infra/caddy/Caddyfile:/etc/caddy/Caddyfile
  labels:
    - "traefik.enable=true"
    - "traefik.http.routers.caddy.rule=Host(`testapp.localhost`) && PathPrefix(`/static`)"
    - "traefik.http.routers.caddy.entrypoints=web"
    - "traefik.http.routers.caddy.service=caddy"
    - "traefik.http.services.caddy.loadbalancer.server.port=80"
    - "traefik.http.routers.caddy.middlewares=caddy-stripprefix"
    - "traefik.http.middlewares.caddy-stripprefix.stripprefix.prefixes=/static"
  networks:
    - testapp_net
falco:
  image: falcosecurity/falco:latest
  container_name: falco
```

```
    privileged: true
    pid: host
    command: /usr/bin/falco -o engine.kind=modern_ebpf
    volumes:
      - /var/run/docker.sock:/host/var/run/docker.sock
      - /dev:/host/dev
      - /proc:/host/proc:ro
      - /boot:/host/boot:ro
      - /lib/modules:/host/lib/modules:ro
      - /etc:/host/etc:ro
      - ./infra/falco_rules.local.yaml:/etc/falco/falco_rules.local.yaml:ro
    networks:
      - monitoring-net
networks:
  testapp_net:
    name: testapp_net
    external: false
  proxy-net:
    name: proxy-net
    external: false
  monitoring-net:
    name: monitoring-net
    external: false
```

Додаток Б

Лістинг захищеного образу контейнера

```
FROM python:3.12-slim AS builder
WORKDIR /app
RUN apt-get update && apt-get install -y gcc libpq-dev python3-dev libexpat1
COPY requirements.txt .
RUN pip wheel --no-cache-dir --no-deps --wheel-dir /app/wheels -r requirements.txt

FROM python:3.12-slim
RUN addgroup --system appgroup && adduser --system --ingroup appgroup appuser
WORKDIR /app
RUN apt-get update && apt-get install -y libpq5 libexpat1 && rm -rf /var/lib/apt/lists/*
COPY --from=builder /app/wheels /wheels
COPY --from=builder /app/requirements.txt .
RUN pip install --no-cache /wheels/*
COPY ./src ./src
RUN chown -R appuser:appgroup /app
USER appuser
```

Додаток В

Елементарна конфігурація власних правил поведінкового моніторингу та виявлення аномалій

```
- macro: user_known_memfd_binaries
  condition: (proc.name in (containerd-shim, runc, .runc, exe) or proc.pname in (containerd-shim, runc, .runc))

- rule: Fileless execution via memfd_create
  desc: Detects fileless execution techniques often used to hide payloads in memory.
  append: true
  condition: and not user_known_memfd_binaries

- rule: Shell Spawned by Web Application Process
  desc: Detects an interactive shell spawned by the web application, strongly indicating an RCE vulnerability exploitation.
  condition: >
    spawned_process and container
    and proc.name in (sh, bash, dash, ash)
    and proc.pname in (uwsgi, python, python3)
  output: "CRITICAL ALERT: Web app process spawned a shell! Possible RCE. (user=%user.name app=%proc.pname shell=%proc.name command=%proc.cmdline container_id=%container.id)"
  priority: CRITICAL
  tags: [container, mitre_execution, rce]

- rule: Suspicious Network Tool Executed in Container
  desc: Detects execution of network tools often used by attackers to download second-stage payloads or establish reverse shells.
```

```
    condition: >
      spawned_process and container
      and proc.name in (curl, wget, nc, ncat, nmap)
      and not proc.pname in (apk, apt, apt-get)
      output: "WARNING: Suspicious network tool executed! Potential payload download. (user=%user.name command=%proc.cmdline container_id=%container.id)"
      priority: WARNING
      tags: [container, mitre_exfiltration, network]
```

Додаток Г

Сценарій навантажувального тестування та перевірки відмовостійкості ізольованого середовища

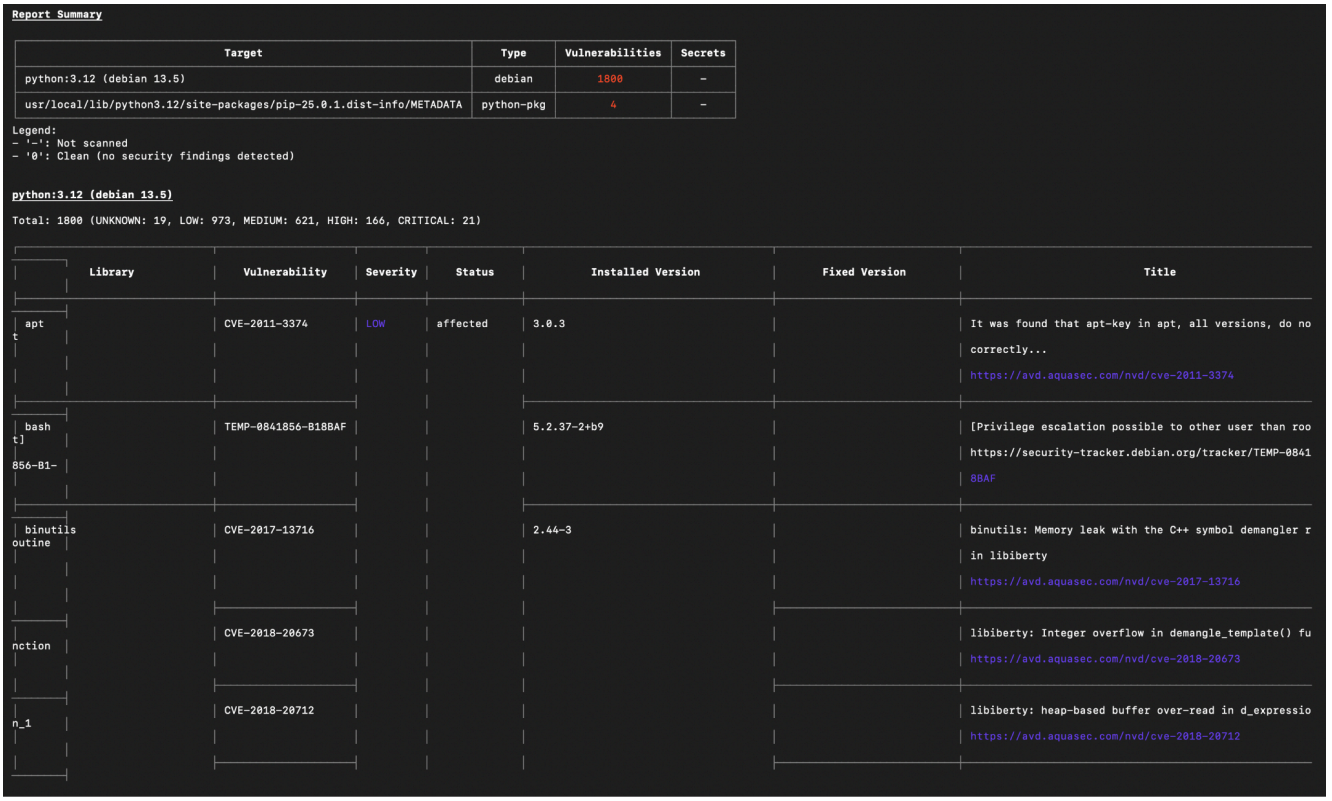
```
import http from 'k6/http';
import { check } from 'k6';
export const options = {
  stages: [
    { duration: '10s', target: 100 },
    { duration: '30s', target: 500 },
    { duration: '15s', target: 1000 },
    { duration: '20s', target: 0 },
  ],
};
export default function () {
  const params = {
    headers: { 'Host': 'testapp.localhost' },
  };

  const res = http.get('http://127.0.0.1/', params);

  check(res, {
    'status is 200': (r) => r.status === 200,
  });
}
```

Додаток Д

Звіт результатів сканування базового образу python:3.12



Report Summary

Target	Type	Vulnerabilities	Secrets
python:3.12 (debian 13.5)	debian	1800	-
usr/local/lib/python3.12/site-packages/pip-25.0.1.dist-info/METADATA	python-pkg	4	-

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

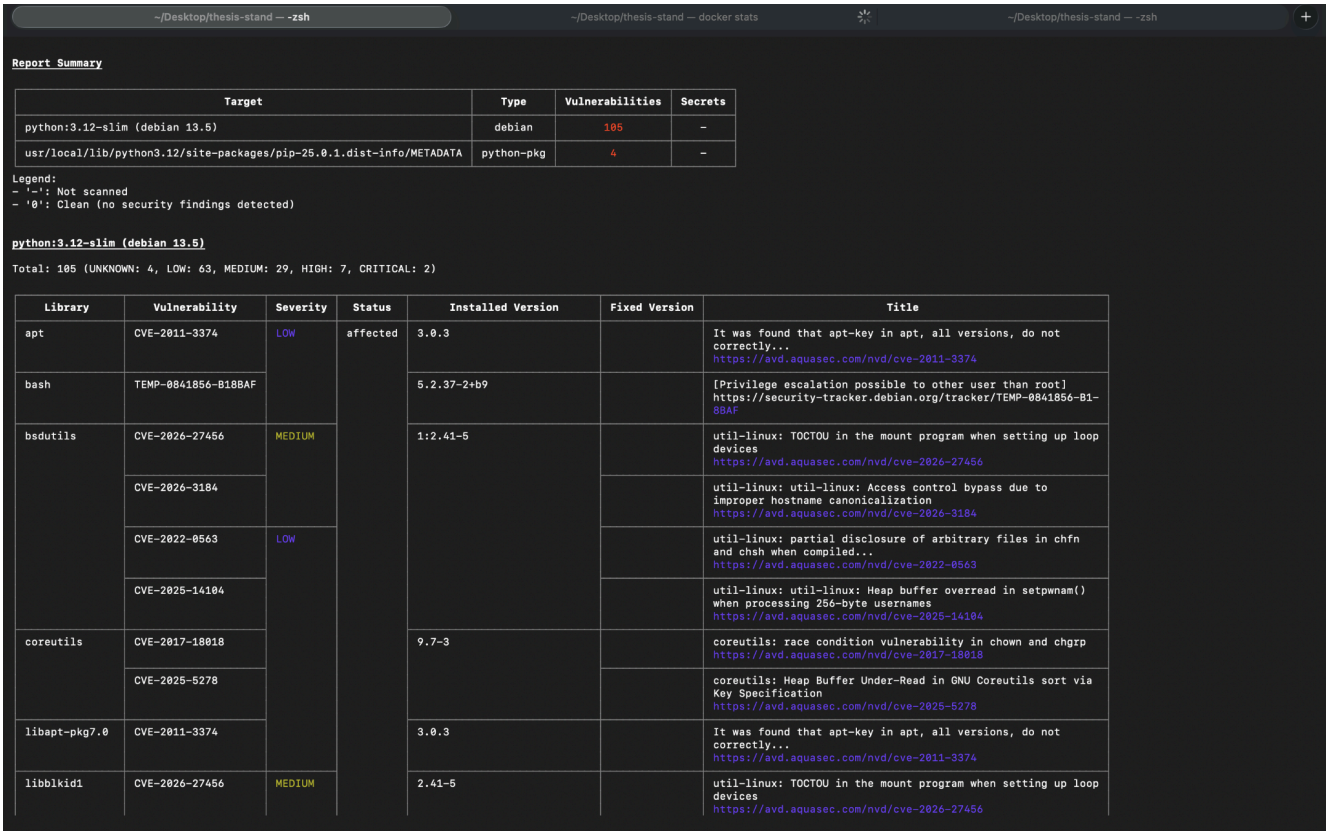
python:3.12 (debian 13.5)

Total: 1800 (UNKNOWN: 19, LOW: 973, MEDIUM: 621, HIGH: 166, CRITICAL: 21)

Library	Vulnerability	Severity	Status	Installed Version	Fixed Version	Title
apt	CVE-2011-3374	LOW	affected	3.0.3		It was found that apt-key in apt, all versions, do not correctly... https://avd.aquasec.com/nvd/cve-2011-3374
bash	TEMP-0841856-B18BAF			5.2.37-2+b9		[Privilege escalation possible to other user than root] https://security-tracker.debian.org/tracker/TEMP-0841856-B1-8BAF
binutils	CVE-2017-13716			2.44-3		binutils: Memory leak with the C++ symbol demangler routine in libiberty https://avd.aquasec.com/nvd/cve-2017-13716
	CVE-2018-20673					libiberty: Integer overflow in demangle_template() function https://avd.aquasec.com/nvd/cve-2018-20673
	CVE-2018-20712					libiberty: heap-based buffer over-read in d_expression_1 https://avd.aquasec.com/nvd/cve-2018-20712

Додаток Е

Звіт результатів сканування образу python:3.12-slim



~/Desktop/thesis-stand — -zsh ~/Desktop/thesis-stand — docker stats ~/Desktop/thesis-stand — -zsh

Report Summary

Target	Type	Vulnerabilities	Secrets
python:3.12-slim (debian 13.5)	debian	105	-
usr/local/lib/python3.12/site-packages/pip-25.0.1.dist-info/METADATA	python-pkg	4	-

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

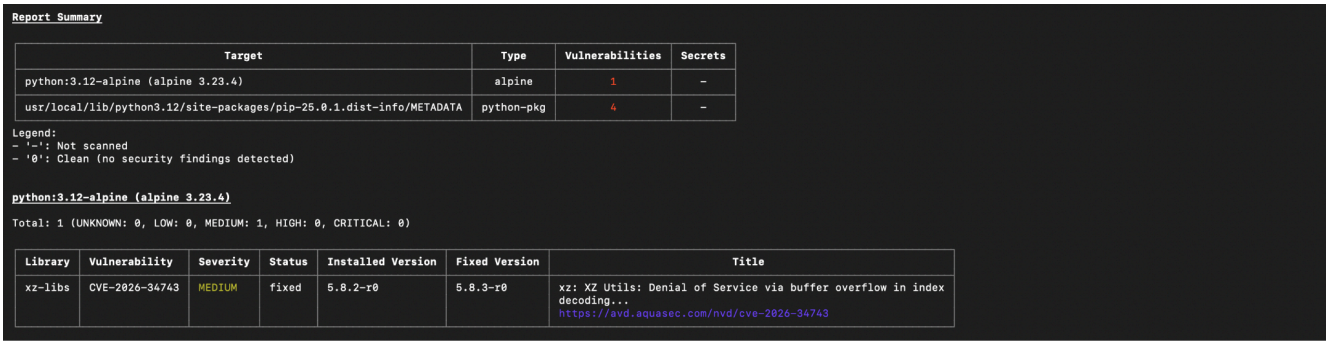
python:3.12-slim (debian 13.5)

Total: 105 (UNKNOWN: 4, LOW: 63, MEDIUM: 29, HIGH: 7, CRITICAL: 2)

Library	Vulnerability	Severity	Status	Installed Version	Fixed Version	Title
apt	CVE-2011-3374	LOW	affected	3.0.3		It was found that apt-key in apt, all versions, do not correctly... https://avd.aquasec.com/nvd/cve-2011-3374
bash	TEMP-0841856-B18BAF			5.2.37-2+b9		[Privilege escalation possible to other user than root] https://security-tracker.debian.org/tracker/TEMP-0841856-B1-8BAF
bsdutils	CVE-2026-27456	MEDIUM		1:2.41-5		util-linux: TOCTOU in the mount program when setting up loop devices https://avd.aquasec.com/nvd/cve-2026-27456
	CVE-2026-3184					util-linux: util-linux: Access control bypass due to improper hostname canonicalization https://avd.aquasec.com/nvd/cve-2026-3184
	CVE-2022-0563	LOW				util-linux: partial disclosure of arbitrary files in chfn and chsh when compiled... https://avd.aquasec.com/nvd/cve-2022-0563
	CVE-2025-14104					util-linux: util-linux: Heap buffer overread in setpwnam() when processing 256-byte usernames https://avd.aquasec.com/nvd/cve-2025-14104
coreutils	CVE-2017-18018			9.7-3		coreutils: race condition vulnerability in chown and chgrp https://avd.aquasec.com/nvd/cve-2017-18018
	CVE-2025-5278					coreutils: Heap Buffer Under-Read in GNU Coreutils sort via Key Specification https://avd.aquasec.com/nvd/cve-2025-5278
libapt-pkg7.0	CVE-2011-3374			3.0.3		It was found that apt-key in apt, all versions, do not correctly... https://avd.aquasec.com/nvd/cve-2011-3374
libblkid1	CVE-2026-27456	MEDIUM		2.41-5		util-linux: TOCTOU in the mount program when setting up loop devices https://avd.aquasec.com/nvd/cve-2026-27456

Додаток Ж

Звіт результатів сканування образу python:3.12-alpine



Report Summary

Target	Type	Vulnerabilities	Secrets
python:3.12-alpine (alpine 3.23.4)	alpine	1	-
usr/local/lib/python3.12/site-packages/pip-25.0.1.dist-info/METADATA	python-pkg	4	-

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

python:3.12-alpine (alpine 3.23.4)

Total: 1 (UNKNOWN: 0, LOW: 0, MEDIUM: 1, HIGH: 0, CRITICAL: 0)

Library	Vulnerability	Severity	Status	Installed Version	Fixed Version	Title
xz-libs	CVE-2026-34743	MEDIUM	fixed	5.8.2-r0	5.8.3-r0	xz: XZ Utils: Denial of Service via buffer overflow in index decoding... https://avd.aquasec.com/nvd/cve-2026-34743

Метадані

ДОКУМЕНТ

Заголовок
БКР Трепитьон Павло

Автор
Трепитьон Павло

Науковий керівник / Експерт
Корольов А.П.

ІД документу
334295373

ОРГАНІЗАЦІЯ

Назва організації
Kyiv National Economic University named after Vadym Hetman KNEU

підрозділ
кафедра системного аналізу та кібербезпеки

ЗВІТ

Дата звіту
6/12/2026

Дата редагування

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

2.05% 2.05% КП 1

0% 0% КП 2

0.43% 0.43% КЦ

25
Довжина фрази для коефіцієнта подібності 2

6823
Кількість слів

56376
Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв	ß	0
Інтервали	A→	0
Мікропробіли	·	0
Білі знаки	ß	0
Парафрази (SmartMarks)	a	5

Джерела

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

#	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)

1	КБР Сидорук [redacted] 6/20/2025 Kyiv National Economic University named after Vadym Hetman KNEU (кафедра системного аналізу та кібербезпеки)	24 (0.35 %)
2	КБР Сидорук [redacted] 6/20/2025 Kyiv National Economic University named after Vadym Hetman KNEU (кафедра системного аналізу та кібербезпеки)	21 (0.31 %)
3	КБР Сидорук [redacted] 6/20/2025 Kyiv National Economic University named after Vadym Hetman KNEU (кафедра системного аналізу та кібербезпеки)	17 (0.25 %)
4	КБР Сидорук [redacted] 6/20/2025 Kyiv National Economic University named after Vadym Hetman KNEU (кафедра системного аналізу та кібербезпеки)	14 (0.21 %)
5	РОЗРОБКА ТА ПРОГРАМНА РЕАЛІЗАЦІЯ СЕРВІСІВ МОНІТОРИНГУ БЕЗПЕКИ ТА ВИЯВЛЕННЯ ВРАЗЛИВОСТЕЙ ХМАРНОЇ ІНФРАСТРУКТУРИ [redacted] 6/11/2026 The Ivan Franko National University (Факультет прикладної математики та інформатики)	12 (0.18 %)
6	КБР Сидорук [redacted] 6/20/2025 Kyiv National Economic University named after Vadym Hetman KNEU (кафедра системного аналізу та кібербезпеки)	12 (0.18 %)
7	КБР Сидорук [redacted] 6/20/2025 Kyiv National Economic University named after Vadym Hetman KNEU (кафедра системного аналізу та кібербезпеки)	12 (0.18 %)
8	КБР Сидорук [redacted] 6/20/2025 Kyiv National Economic University named after Vadym Hetman KNEU (кафедра системного аналізу та кібербезпеки)	9 (0.13 %)
9	КБР Сидорук [redacted] 6/20/2025 Kyiv National Economic University named after Vadym Hetman KNEU (кафедра системного аналізу та кібербезпеки)	9 (0.13 %)
10	КБР Сидорук [redacted] 6/20/2025 Kyiv National Economic University named after Vadym Hetman KNEU (кафедра системного аналізу та кібербезпеки)	5 (0.07 %)

База даних RefBooks

#	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)

Домашня база даних (1.88 %)

#	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	КБР Сидорук [redacted] 6/20/2025	128 (10) (1.88 %)

Kyiv National Economic University named after Vadym Hetman KNEU (кафедра системного аналізу та кібербезпеки)

Програма обміну базами даних (0.18 %)

#	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
2	РОЗРОБКА ТА ПРОГРАМНА РЕАЛІЗАЦІЯ СЕРВІСІВ МОНІТОРИНГУ БЕЗПЕКИ ТА ВИЯВЛЕННЯ ВРАЗЛИВОСТЕЙ ХМАРНОЇ ІНФРАСТРУКТУРИ [redacted] 6/11/2026 The Ivan Franko National University (Факультет прикладної математики та інформатики)	12 (1) (0.18 %)

Інтернет

#	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)

Список прийнятих фрагментів

#	ЗМІСТ	КІЛЬКІСТЬ ОДНАКОВИХ СЛІВ (ФРАГМЕНТІВ)