ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Архітектура мережевої ізоляції веб-сервісів у середовищі Docker»

на здобуття освітнього ступеня бакалавра
зі спеціальності 123 Комп'ютерної інженерії
(код, найменування спеціальності)
освітньо-професійної програми Комп'ютерна інженерія
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ *Іван СУМСЬКИЙ*
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав: здобувач вищої освіти гр. КІД-41
Іван СУМСЬКИЙ
Ім'я, ПРІЗВИЩЕ
Керівник: Наталія ЛАЩЕВСЬКА
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ
Рецензент: _____
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра 123 Комп'ютерної інженерії

Ступінь вищої освіти Бакалавр

Спеціальність 123 Комп'ютерної інженерії

Освітньо-професійна програма Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри
Комп'ютерної інженерії
_____ Наталія ЛАЩЕВСЬКА
_____
«_____» _____ 2026р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Сумський Іван Вікторович
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Архітектура мережевої ізоляції веб-сервісів у середовищі Docker
керівник кваліфікаційної роботи Наталія ЛАЩЕВСЬКА,
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026р. № 51
2. Строк подання кваліфікаційної роботи «8» травня 2026р.
3. Вихідні дані до кваліфікаційної роботи: Науково-технічна література; технології контейнеризації та розгортання: Docker; архітектурні концепції: мікросервісна архітектура, багаторівневий захист (Defense in Depth), модель нульової довіри (Zero Trust); мережеві топології Docker: мостові мережі (Default Bridge and User-defined Bridge), мікросегментація, апаратна ізоляції, внутрішній DNS-резолвінг; стек вебтехнологій: реверс-проксі сервер (Nginx), PHP-FPM, реляційна СУБД (PostgresSQL); механізм кібербезпеки: Docker Secrets, контроль експозиції портів, стандарти NIST SP 800-190/207 та OWASP.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Теоретичні основи мережевої ізоляції у контейнерних середовищах
2. Аналіз мережевих вразливостей базової архітектури Docker
3. Проєктування та провадження багатосегментованої ізольованої архітектури
5. Перелік ілюстративного матеріалу: *презентація*
6. Дата видачі завдання «20» лютого 2026р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Збір даних	25.02-25.03.2026	Виконано
2	Обробка матеріалу	26.03-06.04.2026	Виконано
3	Розробка теоретичної частини	07.04-10.04.2026	Виконано
4	Розробка архітектурних рішень для мережевої ізоляції	11.04-20.04.2026	Виконано
5	Висновки за результатами аналізу	21.04-25.04.2026	Виконано
6	Оформлення пояснювальної записки	26.04-29.04.2026	Виконано
7	Розробка презентації	30.04-03.05.2026	Виконано
8	Передзахист	08.05-11.05.2026	Виконано
9	Здача в деканат	08.05-14.05.2026	Виконано

Здобувач вищої освіти _____ Іван СУМСЬКИЙ
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ Наталія ЛАЩЕВСЬКА
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 52 стор., 22 рис., 9 табл., 30 джерела, 3 додатки

Мета роботи - Розробка та впровадження багатосегментної архітектури мережевої ізоляції веб-сервісів у середовищі Docker для підвищення рівня безпеки контейнеризованих інфраструктур.

Об'єкт дослідження - Процес забезпечення мережевої безпеки та ізоляції вебсервісів, розгорнутих у контейнеризованому середовищі Docker.

Предмет дослідження - Архітектурні рішення та методи мережевої ізоляції веб-сервісів у середовищі Docker.

Короткий зміст роботи: Кваліфікаційна робота присвячена розв'язанню практичного завдання підвищення безпеки мікросервісних додатків шляхом проєктування ізольованих мережевих топологій. У роботі формалізовано вразливості класичного розгортання контейнерів, зокрема неконтрольованість внутрішнього трафіку та надмірну відкритість портів баз даних. Запропоновано методику кількісної оцінки захищеності інфраструктури за допомогою інтегрального коефіцієнта K_s. Здійснено розробку багатосегментної мережі Docker, що розподіляє ресурси на три логічні зони: публічну зону, зону застосунку та ізольовану зону даних. Впроваджено механізм безпечного управління секретами через Docker Secrets та єдину точку входу на базі зворотного проксі-сервера. Шляхом вимірювань та повторного аудиту доведено суттєве зниження поверхні атаки та підтверджено збереження високої пропускної здатності системи при використанні запропонованих методів ізоляції.

КЛЮЧОВІ СЛОВА: КОНТЕЙНЕРИЗАЦІЯ, DOCKER, МЕРЕЖЕВА ІЗОЛЯЦІЯ, МІКРОСЕРВІСНА АРХІТЕКТУРА, ІНФОРМАЦІЙНА БЕЗПЕКА, ЗВОРОТНИЙ ПРОКСІ, ПОВЕРХНЯ АТАКИ, DOCKER SECRETS, SYMFONY.

ЗМІСТ

ВСТУП 8
1 ТЕОРЕТИЧНІ ОСНОВИ МЕРЕЖЕВОЇ ІЗОЛЯЦІЇ У КОНТЕЙНЕРНИХ СЕРЕДОВИЩАХ 10
1.1 Аналіз сучасного стану безпеки мікросервісних архітектур та ідентифікації мережевих вразливостей 10
1.2 Огляд концепції ізоляції трафіку в технології Docker на базі драйвера bridge та параметрів внутрішньої маршрутизації 14
1.3 Обґрунтування методики кількісної оцінки мережевої безпеки контейнеризованого середовища 18
2 АНАЛІЗ МЕРЕЖЕВИХ ВРАЗЛИВОСТЕЙ БАЗОВОЇ АРХІТЕКТУРИ DOCKER 22
2.1 Розгортання базової інфраструктури та збір параметрів міжсервісного трафіку застосунку 22
2.2 Аналіз поверхні атаки та ідентифікації вразливостей зовнішнього доступу до неізольованих контейнерів баз даних 27
3 ПРОЄКТУВАННЯ ТА ПРОВАДЖЕННЯ БАГАТОСЕГМЕНТОВАНОЇ ІЗОЛЬОВАНОЇ АРХІТЕКТУРИ 31
3.1 Проєктування ізольованої топології Docker-мереж із розподілом на публічну, приватну та зону даних 31
3.2 Впровадження механізмів зворотного проксіювання для безпечної маршрутизації ізольованого трафіку веб-вузлів 36
3.3 Кількісна оцінка підвищення рівня безпеки та аналіз ефективності системи після реалізації спроектованої архітектури 39
ВИСНОВКИ 44
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 46
Додаток А ПРОГРАМНА РЕАЛІЗАЦІЇ БАГАТОСЕГМЕНТОВАНОЇ МЕРЕЖЕВОЇ АРХІТЕКТУРИ 49
Додаток Б КОНФІГУРАЦІЯ МЕХАНІЗМІВ БЕЗПЕЧНОГО ЗВОРОТНОГО ПРОКСІЮВАННЯ 51
Додаток В ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація) 52

ВСТУП

Актуальність теми Еволюція методів розробки програмного забезпечення зумовила масовий перехід до мікросервісної архітектури, де базовим стандартом розгортання виступає технологія контейнеризації. Попри забезпечення високої масштабованості, децентралізація обчислювальних процесів експоненціально розширює поверхню атаки, оскільки стандартна (плоска) топологія мереж Docker за замовчуванням не забезпечує належної ізоляції міжсервісного трафіку. Це породжує критичні вразливості: відсутність жорсткого контролю доступу до приватних програмних інтерфейсів (Private APIs), ризики мережевого перехоплення даних (sniffing) та небезпечну відкритість портів вузлів СУБД на зовнішні інтерфейси. За умов компрометації публічно доступного контейнера, дефіцит мережевої сегментації відкриває вектор для безперешкодного горизонтального переміщення зловмисника (lateral movement) з подальшою компрометацією всієї інфраструктури. Гарантування конфіденційності та цілісності сучасних вебсервісів вимагає переходу від виключно зовнішнього захисту до концепції Defense in Depth безпосередньо всередині контейнеризованого середовища. Тому проєктування багатосегментної архітектури з дотриманням принципів Zero Trust є надзвичайно актуальним інженерним завданням.

Мета роботи. Метою роботи є розробка та впровадження багатосегментної архітектури мережевої ізоляції веб-сервісів у середовищі Docker для підвищення рівня безпеки контейнеризованих інфраструктур. Для досягнення поставленої мети вирішено такі завдання:

- Проаналізувати сучасний стан безпеки мікросервісних архітектур та ідентифікувати мережеві вразливості базових конфігурацій Docker;
- Обґрунтувати методику кількісної оцінки мережевої безпеки контейнеризованого середовища за допомогою інтегрального коефіцієнта *Ks*;

- Здійснити аудит базової інфраструктури та виявити поверхню атаки неізольованих контейнерів баз даних;
- Спроєктувати ізольовану топологію Docker-мереж із розподілом на публічну зону, зону застосунку та зону даних;
- Впровадити механізми зворотного проксіювання та безпечного управління секретами через Docker Secrets;
- Провести кількісну оцінку підвищення рівня захищеності та підтвердити збереження високої продуктивності системи після впровадження методів ізоляції.

Об'єкт дослідження. Процес забезпечення мережевої безпеки та ізоляції вебсервісів, розгорнутих у контейнеризованому середовищі Docker.

Предмет дослідження. Архітектурні рішення та методи мережевої ізоляції веб-сервісів у середовищі Docker.

Методи дослідження. У дослідженні застосовано методи системного аналізу мережевих структур, активного сканування мереж для проведення аудиту вразливостей, а також метод аналізу ієрархій Томаса Сааті для визначення вагових коефіцієнтів безпекових метрик. Крім того, використано метод адитивної згортки для обчислення інтегрального показника захищеності інфраструктури.

Практична значущість отриманих результатів. Запропонована архітектура мінімізує площу можливої атаки та унеможливлює несанкціонований прямий доступ до баз даних і критичних вузлів. Впровадження апаратно ізольованих мереж (з параметром internal), використання Docker Secrets та застосування Nginx як єдиного санкціонованого шлюзу забезпечує високий рівень захищеності за мінімальних експлуатаційних витрат. Розроблені конфігураційні файли є готовими інженерними рішеннями для безпечного розгортання додатків у виробничих середовищах.

1. ТЕОРЕТИЧНІ ОСНОВИ МЕРЕЖЕВОЇ ІЗОЛЯЦІЇ У КОНТЕЙНЕРНИХ СЕРЕДОВИЩАХ

1.1 Аналіз сучасного стану безпеки мікросервісних архітектур та ідентифікації мережевих вразливостей

Швидкий перехід розробки програмного забезпечення до хмарно-орієнтованих (cloud-native) технологій зумовив зростання та розвиток мікросервісної архітектури. Використання систем контейнеризації, стандартом, одним із яких є Docker, забезпечує високу швидкість розгортання мереж, їх ізоляцію середовищ виконання та ефективну утилізацію апаратних ресурсів. Проте децентралізація докорінно змінює парадигму забезпечення інформаційної безпеки, оскільки архітектурна гнучкість досягається ціною стрімкого розширення поверхні атаки .

У класичних програмах де використовується монолітна архітектура , взаємодія між компонентами відбувається на рівні викликів функцій у спільному адресному просторі оперативної пам'яті, операційної системи (Inter-Process Communication). Захист такої інфраструктури основується на концепції жорсткого зовнішнього периметра (Castle-and-Moat), де єдиною точкою фільтрації виступає міжмережевий екран (Edge Firewall) або екран захисту веб-додатків (WAF), що обробляє виключно вхідний (North-South) трафік клієнтів.

У мікросервісній архітектурі взаємодія між компонентами відбувається на мережевому рівні, де маршрутизація та передача пакетів реалізуються на рівнях L3 та L4 моделі OSI. У той же час, логіка обміну повідомленнями функціонує на прикладному рівні L7 з використанням протоколів HTTP/REST, gRPC або через брокери повідомлень.Це призводить до деградації єдиного мережевого периметра.

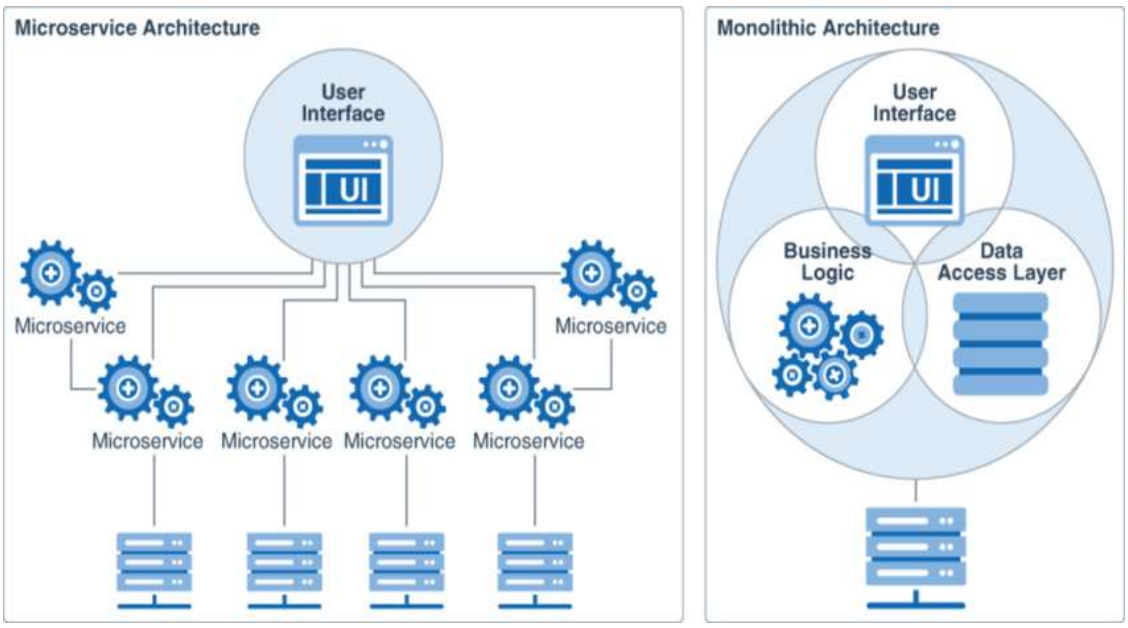


Рисунок 1.1 - Порівняльна структурна схема монолітної та мікросервісної архітектур веб-додатків

Дослідження показників мережевого обміну у масштабованих мікросервісних інфраструктурах показують, що обсяг внутрішнього міжсервісного трафіку (East-West) значно перевищує обсяг зовнішніх запитів (North-South). У базовій конфігурації мережі Docker а саме при використанні мережевого драйвера bridge за замовчуванням, усі розгорнуті контейнери автоматично підключаються до єдиного віртуального комутатора. У межах цього широкомовного L2-сегмента внутрішній трафік маршрутизується без будь якого шифрування, перевірки пакетів та обмежень на рівні портів.

Відсутність мікросегментації та неконтрольований East-West трафік створюють такі критичні мережеві вразливості як:

– Горизонтальне переміщення (Lateral Movement). У разі успішної використання вразливостей публічно доступного вузла, наприклад, RCE або SSRF на frontend-сервері, обходить зовнішній мережні firewall та отримує доступ до внутрішньої мережі системи. Уражений контейнер використовується як точка для сканування підмережі та подальшої атаки на критичні сервіси вже у середині мережі.

– Експозиція внутрішніх приватних програмних інтерфейсів. API та систем керування базами даних часто проектуються без складних механізмів авторизації у розрахунку на ізольованість середовища, що є помилкою. У базовій мережі Docker, бази даних стають вразливими до несанкціонованих підключень з будь якого сусіднього контейнера, який знаходиться у одній мережі.

– Ризики перехоплення даних (Packet Sniffing або ARP Spoofing). Знаходження контейнерів в одному L2-домені дозволяє ураженому вузлу перехоплювати нешифрований трафік сусідніх сервісів шляхом атаки, яка базується на підміни ARP-таблиць віртуального комутатора.

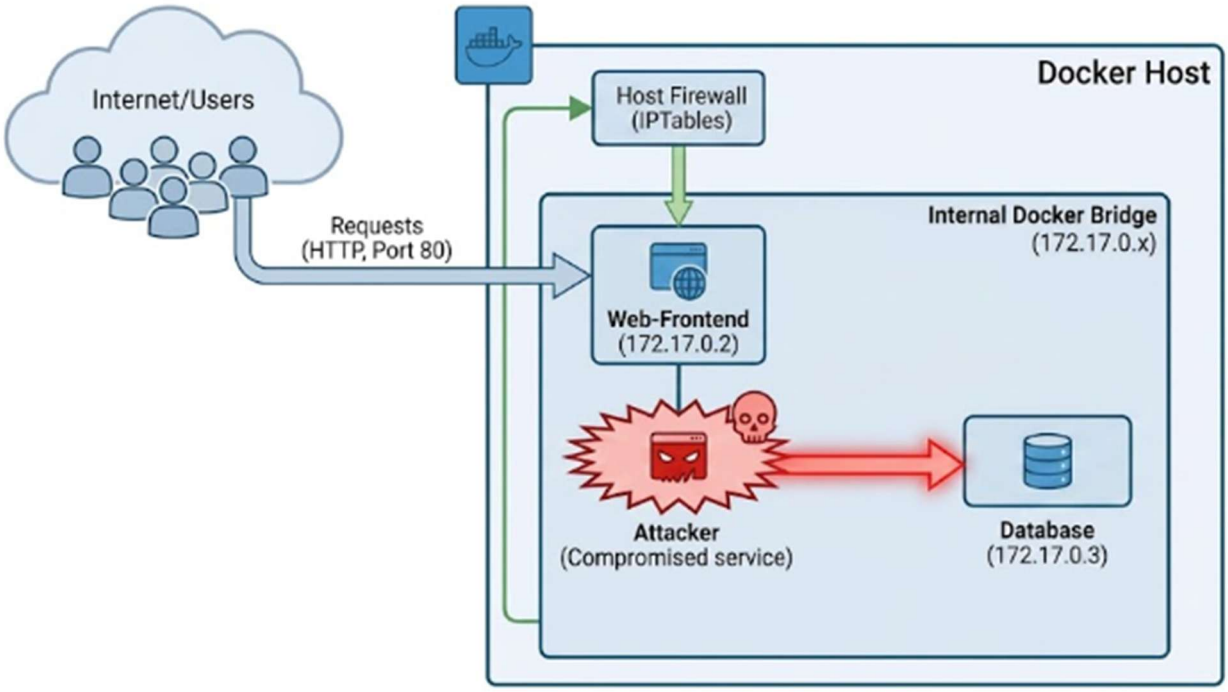


Рисунок 1.2 - Вектор атаки «горизонтальне переміщення» (Lateral Movement) у неізольованій мережі Docker

Згідно з рекомендаціями Національного інституту стандартів і технологій США (NIST), щодо безпеки контейнеризованих систем і програм, які вказують на те, що використання спільної мережі для всіх служб порушує принцип найменших привілеїв. Принцип найменших привілеїв означає надання користувачам лише тих дозволів, які їм необхідні для виконання їхньої роботи. Концепція багаторівневого захисту Defense in Elevation та Zero Trust, повинні бути впроваджені на рівні

управління системою або виконання, а також на сайтах розгортання, щоб забезпечити стабільність і надійність систем.

Було визначено, що базових інструментів для ізоляції процесів (просторів імен Linux) на рівні хост-ОС недостатньо для зменшення виявлених ризиків. Тому необхідно створити окрему багаторівневу структуру віртуальних мереж зі суворою ізоляцією баз даних та використанням зворотних проксі-серверів для контролю точок входу в мережу.

Відомо, що архітектурна вразливість стандартної мережі Docker закладена на рівні конфігурації мережевого демона. За замовчуванням основний глобальний параметр міжконтейнерної взаємодії Inter-Container Communication, ***--icc*** встановлюється зі значенням *true*. Правила міжмережевого екрана iptables, які автоматично генеруються службою Docker на хосту, дозволяють будь-який неконтрольований обмін пакетними даним між контейнерами в межах віртуального інтерфейсу. Відсутність явного налаштування користувацьких мереж під час розгортання наприклад, у конфігураційному фалі docker-compose.yaml, призводить до об’єднання всіх сервісних контейнерів, включно з критичними вузлами бази даних, в єдиному мережевому сегменті зі стандартною маскою /16.

Встановлено, що фундаментальний конфлікт між методологіями безперервної інтеграції та доставки CI/CD та жорсткими вимогами інформаційної безпеки SecOps. Більшість утиліт автоматизації розгортання, оптимізовані для забезпечення максимальної зв'язності (connectivity) з метою прискорення етапів тестування. Розробники переважно нехтують створенням ізольованих користувацьких мереж User-defined bridge networks через необхідність додаткового налаштування DNS-маршрутизації та ускладнення процесів тестування та налагодження взаємодії компонентів. Наслідком цього є масове перенесення «плоских» топологій безпосередньо у виробничі середовища.

Аналіз сучасних підходів до мікросегментації, таких як впровадження рішень класу Service Mesh, наприклад Istio або Linkerd у середовищі Kubernetes, демонструє високу ефективність у контексті забезпечення політик Zero Trust. Це досягається завдяки ін'єкції sidecar-проксі контейнерів та використанню взаємної

TLS-автентифікації для всього East-West трафіку. Проте для систем малого та середнього масштабу, розгорнутих на базі мереж Docker, впровадження таких інструментів вважається надлишковим. Їх застосування призводить до невиправданого споживання апаратних ресурсів сервера та стрімкого ускладнення підтримки інфраструктури.

Таким чином, розглядаємо науково-практичну проблему: наявна критична нестача стандартизованих, маловитратних архітектурних патернів для мережевої ізоляції контейнерного трафіку, які б не вимагали розгортання важких систем оркестрації. Забезпечення необхідного рівня кібербезпеки вимагає розробки методології конструювання багатосегментної топології мережі із жорстким розмежуванням публічних, приватних та дата-орієнтованих зон доступу, використовуючи виключно стандратні драйвери середовища Docker.

1.2 Огляд концепції ізоляції трафіку в технології Docker на базі драйвера bridge та параметрів внутрішньої маршрутизації

Дослідження архітектури мережевого стека середовища Docker базується на моделі контейнерних мереж Container Network Model, яка реалізується системним компонентом libnetwork. Визначено, що CNM визначає три базові абстракції: мережа - Network, кінцева точка – Endpoint та пісочниця - Sandbox. Для забезпечення маршрутизації на рівні хостової операційної системи, Docker використовує систему підключених драйверів pluggable drivers. Основним драйвером для організації локальної взаємодії контейнерів на одному хості є *bridge.*

Встановлено критичну архітектурну відмінність між стандартною мережею bridge, яка створюється демоном за замовчуванням, та користувацькими мостовими мережами User-defined bridge networks. Стандартна мережа не забезпечує автоматичного вирішення доменних імен та дозволяє неконтрольовану взаємодію всіх підключених контейнерів. Натомість створення користувацьких мостових мереж формує незалежні широкомовні домени L2-сегменту.

За замовчуванням середовище Docker реалізує базовий рівень ізоляції між різними користувацькими мостовими мережами шляхом автоматичної модифікації правил міжмережевого екрана хостової ОС за допомогою утиліт iptables або nftables. Демон створює спеціальні ланцюжки DOCKER-ISOLATION-STAGE-1 та DOCKER-ISOLATION-STAGE-2, які на рівні ядра блокують маршрутизацію пакетів між незалежними віртуальними інтерфейсами, наприклад, між br-network1 та br-network2.

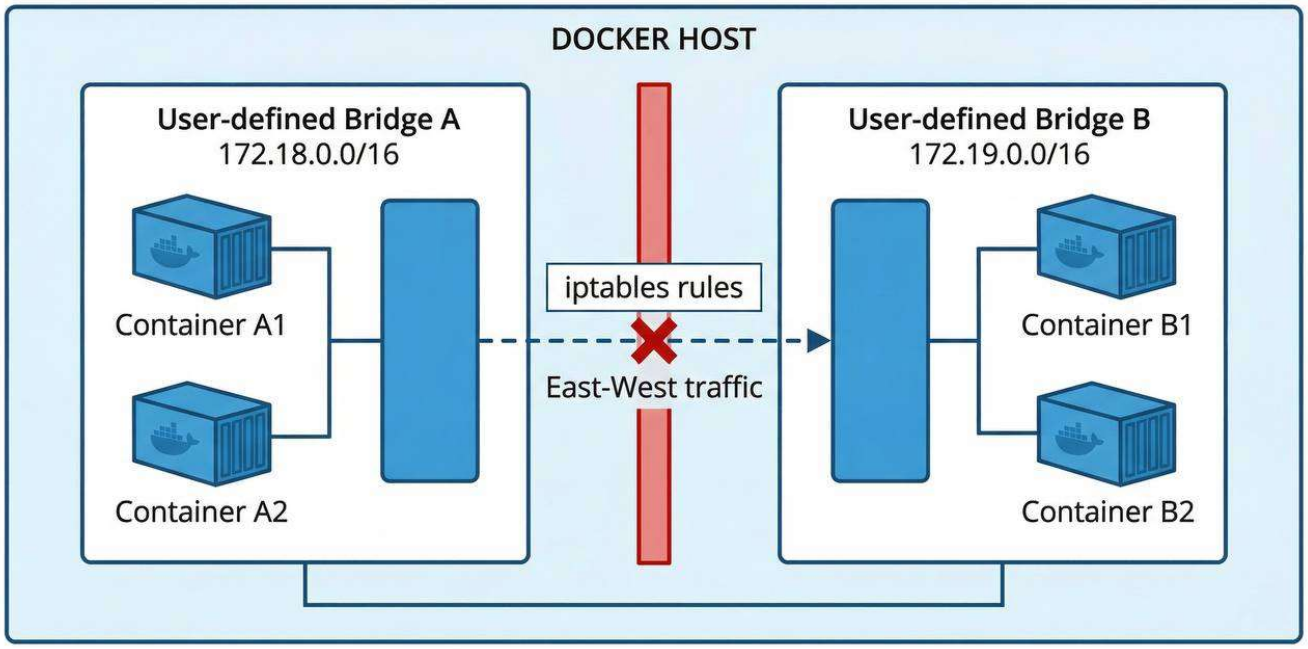


Рисунок 1.3 - Схема базової міжмережевої ізоляції користувацьких мостових мереж засобами iptables

Незважаючи на поділ між підмережами, контейнери у стандартній користувацькій мережі bridge зберігають можливість здійснювати зовнішні мережеві запити North-South через механізм динамічної трансляції мереж NAT та отримувати зовнішні підключення шляхом прокидання портів Port Mapping. Для сервісів, що обробляють критичні дані а саме системи керування базами даних, кеш-сервери, наявність такого, навіть потенційного виходу до глобальної мережі або публічної зони хоста є порушенням політики безпеки.

Для забезпечення абсолютної мережевої ізоляції на рівні віртуального комутатора використано застосування параметра внутрішньої маршрутизації --*internal*. Цей конфігураційний прапорець інтегрується з драйвером bridge під час ініціалізації мережі:

```
docker network create \
  --driver bridge \
  --internal \
  isolated_data_network
```

Рисунок 1.4 - Ініціалізація ізольованої користувацької мережі засобами Docker CLI

Для систематизації архітектурних відмінностей наведено порівняльну характеристику конфігураційних параметрів мостових мереж середовища Docker (табл. 1.1)

Таблиця 1.1 — Порівняльна характеристика конфігураційних параметрів мостових мереж Docker

Типи мережевої конфігурації	**Підтримка внутрішньої DNS**	**Доступ до зовнішньої мережі**	**Можливість прямого East-West зв'язку**	**Базовий рівень безпеки**
Стандартна мостова мережа (Default bridge)	***Відсутня*** - маршрутизація здійснюється виключно за IP-адресами	***Наявний*** - через механізми механізм динамічної трансляції мереж хостової ОС	***Необмежена*** - глобальний доступ між усіма контейнерами в інтерфейсі *docker0*	***Низький*** - відсутність мікросегментації, високий ризик горизонтального переміщення

Продовження таблиці 1.1

Типи мережевої конфігурації	**Підтримка внутрішньої DNS**	**Доступ до зовнішньої мережі**	**Можливість прямого East-West зв'язку**	**Базовий рівень безпеки**
Користувацька мостова мережа (User-defined bridge)	***Доступна*** - автоматичне вирішення імен завдяки вбудований DNS-сервер	***Наявний*** - підтримує фільтрацію портів	***Обмежена*** - пряма взаємодія дозволена лише в межах ізольованого L2-сегмента	***Середній*** - забезпечує базову міжмережеву сегментацію через DOCKER-ISOLATION
Ізольована мостова мережа (Internal bridge)	***Доступна*** - автоматичне вирішення імен через вбудований DNS-сервер	***Відсутня*** - апаратна маршрутизація вимкнена	***Локалізована*** - закритий обмін даними виключно всередині визначеного сегмента	***Високий*** - повна мережева ізоляція, що знижує ризики несанкціонованого доступу

Аналіз даних таблиці 1.1 підтверджує, що конфігурація Internal bridge є єдиним вбудованим механізмом, який забезпечує гарантовану ізоляцію критичних вузлів від зовнішнього мережевого периметра без залучення додаткових сторонніх інструментів фільтрації трафіку.

Застосування параметра internal вагомо змінює таблицю маршрутизації всередині створеної пісочниці Sandbox. У такій конфігурації демон Docker не призначає віртуальному мосту маршрут за замовчуванням Default Gateway, що веде до зовнішнього інтерфейсу хоста, наприклад, eth0. Крім того, на рівні утиліти iptables генеруються жорсткі правила DROP для будь-яких пакетів, що намагаються покинути межі даного віртуального комутатора, або пакетів, що надходять ззовні, навіть якщо порти контейнера були помилково відкриті параметром *-p* або *ports*.

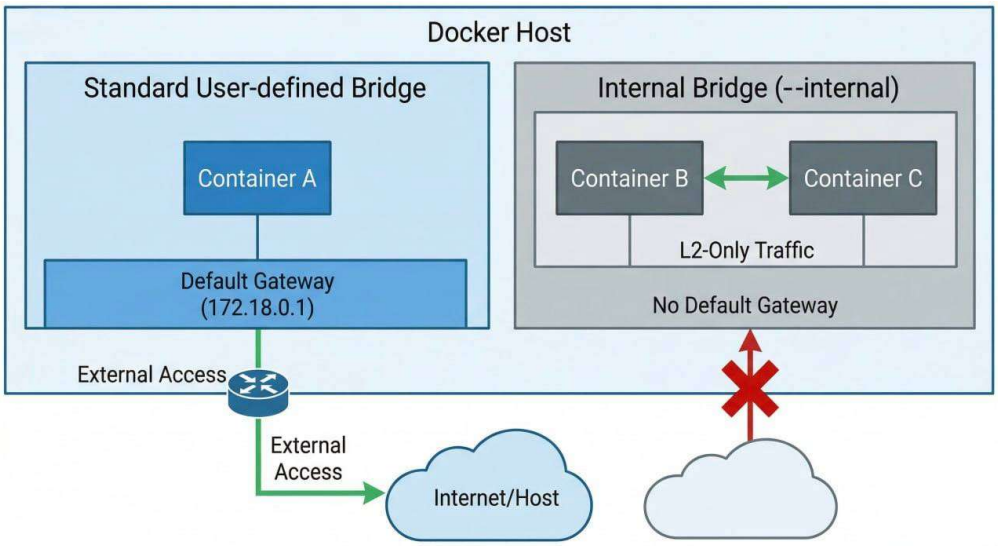


Рисунок 1.5 - Вплив параметра внутрішньої маршрутизації на таблицю логічних маршрутів віртуальної мережі Docker

Перевірно, що комбінація користувацьких мереж bridge та параметра *internal* створює надійне підґрунтя для побудови багаторівневої мікросервісної архітектури. На інфраструктурному рівні, ризик впливу витоку конфіденційних даних на несанкціоновані зовнішні загрози, зокрема витоки з мережі, можна зменшити шляхом розгортання публічних фронтенд-сервісів у стандартній мостовій мережі. У той же час бекенд-сервіси та бази даних повинні бути розміщені в ізольованих підмережах.Єдиним вектором комунікації між такими ізольованими сегментами стає використання контейнерів-маршрутизаторів - реверс-проксі, які одночасно підключені до двох і більше мереж, що забезпечує жорсткий контроль та інспекцію трафіку між кожною зоною.

1.3 Обґрунтування методики кількісної оцінки мережевої безпеки контейнеризованого середовища

Для об'єктивного аналізу ефективності архітектурних рішень у сфері мережевої ізоляції необхідно впровадити математичний апарат, що дозволяє конвертувати якісні показники захищеності у кількісні метрики. Традиційні

суб’єктивні методи оцінки («високий», «середній», «низький») не забезпечують достатньої точності для порівняльного аналізу топологій Docker. Обґрунтовано застосування інтегрального коефіцієнта захищеності K_s, який базується на зваженій моделі оцінки критичних факторів мережевої інфраструктури.

Формування методики оцінки базується на системному підході, де безпека розглядається як сукупність чітко визначених параметрів конфігурації набору мережевих технологій. Для забезпечення наукової достовірності вибору вагових множників w_i використано метод аналізу ієрархій MAI, розроблений Томасом Сааті. Це дозволяє нівелювати суб'єктивізм при визначенні пріоритетності факторів захисту. На основі експертного опитування та аналізу стандартів NIST SP 800-190 встановлено, що мережева ізоляція є домінуючим фактором у забезпеченні стійкості контейнерних систем до горизонтального переміщення зловмисника.

Методика включає аналіз чотирьох фундаментальних параметрів: рівня ізоляції сегментів I, ступені експозиції портів E, наявності шифрування внутрішнього трафіку C та механізмів міжсервісної автентифікації A. Для формалізації розрахунків введено розширену номенклатуру коефіцієнтів, представлену в таблиці 1.2.

Таблиця 1.2 - Номенклатура та критерії визначення коефіцієнтів мережевої безпеки

Параметр	**Опис та умови присвоєння балів**	**Значення (x_i)**	**Вага (w_i)**
Рівень ізоляції (I)	0.0 - єдина підмережа для всіх контейнерів (Default Bridge). 0.4 –поділ за функціональними групами (User-defined Bridge). 1.0 - повна ізоляція критичних зон (Internal Bridge).	0.0 – 1.0	0.40
Мінімізація експозиції (E)	0.0 - прямий доступ до всіх контейнерів. 0.5 - доступ лише через централізований реверс-проксі (Nginx/Gateway). 1.0 - відсутність публічних портів, доступ тільки через VPN-шлюз.	0.0 – 1.0	0.25

Продовження таблиці 1.2

Параметр	**Опис та умови присвоєння балів**	**Значення (x_i)**	**Вага (w_i)**
Шифрування трафіку (*C*)	0.0 - передача даних у відкритому вигляді (HTTP, FTP, Telnet). 0.6 - використання TLS для зовнішніх та критичних внутрішніх запитів. 1.0 - суцільне шифрування East-West трафіку (mTLS / Service Mesh).	0.0 - 1.0	0.20
Автентифікація (*A*)	0.0 - довіра за рівнем IP-адреси (відсутність автентифікації). 0.5 - статичні токени або API-ключі у конфігураційних файлах. 1.0 - динамічна автентифікація та авторизація (OAuth2 / JWT / OIDC).	0.0 - 1.0	0.15

Розрахунок інтегрального коефіцієнта захищеності здійснюється за формулою адитивної згортки:

$$K_S = \sum_{i=1}^{n} w_i \cdot x_i$$

де K_s - інтегральний показник захищеності інфраструктури;

w_i - ваговий коефіцієнт *i*-го параметра безпеки;

x_i - бальна оцінка *i*-го параметра за результатами аудиту;

n - кількість оцінювання параметрів $n = 4$.

Значення інтегрального покажчика лежить у діапазоні [0; 1]. Значення $K_s < 0.3$ класифікується як критично вразливе середовище; $0.3 \leq K_s < 0.7$ - середовище з базовим рівнем захисту; $K_s \geq 0.7$ - високий рівень ізоляції та захищеності інфраструктури.

Для візуалізації метрик мережевої безпеки та виявлення дисбалансу у системі захисту застосовується радіарна діаграма. Такий підхід дозволяє ідентифікувати «вузькі місця» архітектури, де значення окремого параметра суттєво відхиляється від цільового показника.

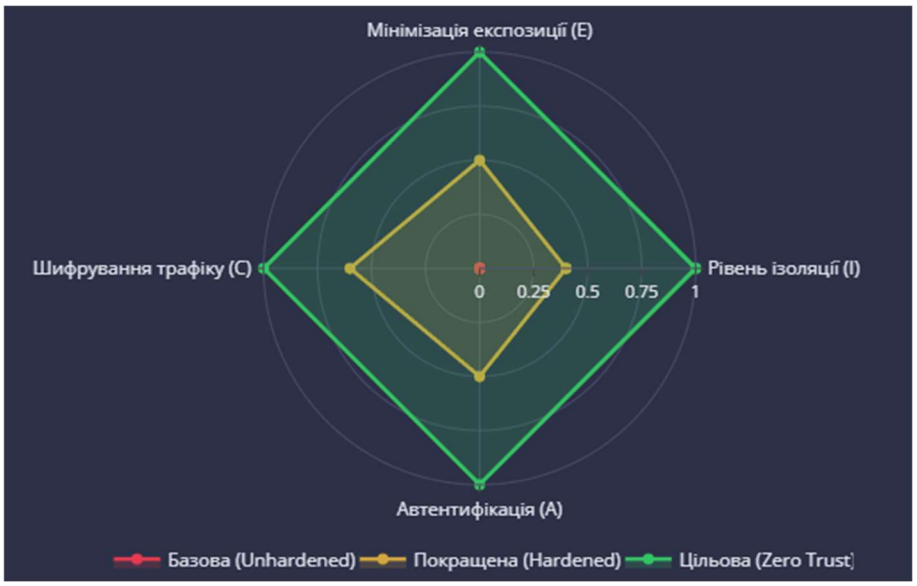


Рисунок 1.6 - Приклад візуалізації метрик мережевої безпеки: порівняння профілів захищеності різних топологій

Окрім показників безпеки, запропонована методика враховує експлуатаційні витрати, що відображають зниження продуктивності мережі при впровадженні методів ізоляції та шифрування. Кінцева ефективність архітектурного рішення обчислюється як відношення рівня захищеності до внесених затримок за такою формулою:

$$E_{arch} = \frac{K_s}{K_{ov}}$$

де E_{arch} - коефіцієнт кінцевої ефективності архітектурно рішення;

K_s - інтегральний показник захищеності системи;

K_{ov} - коефіцієнт експлуатаційних витрат.

Таким чином, розроблена методика кількісної оцінки дозволяє не лише констатувати факт наявності вразливостей, але й математично обґрунтувати вибір конкретної топології мережі Docker у другому та третьому розділах роботи. Це створює базу для доведення переваг багатосегментної ізольованої архітектури перед стандартними методами розгортання мережевих середовищ.

2. АНАЛІЗ МЕРЕЖЕВИХ ВРАЗЛИВОСТЕЙ БАЗОВОЇ АРХІТЕКТУРИ DOCKER

2.1 Розгортання базової інфраструктури та збір параметрів міжсервісного трафіку застосунку

Для проведення дослідження та верифікації теоретичних загроз було реалізовано тестовий стенд, що імітує типову архітектуру сучасного вебзастосунку. Як технологічний стек обрано мову програмування PHP версії 8.4 та фреймворку Symfony 7.4, вебсервер Nginx та систему керування базами даних PostgreSQL 16. Цей набір компонентів є репрезентативним, оскільки охоплює три рівні взаємодії: публічний шлюз, прикладну логіку та рівень зберігання даних.

Розгортання інфраструктури виконано за допомогою файлу Docker Compose. У базовій конфігурації кожен сервіс виділено в окремий контейнер, проте мережева взаємодія залишена у за замовяуванням, що є характерним для етапу розробки. Нижче наведено детальну конфігурацію кожного вузла з аналізом впливу налаштувань на безпеку системи.

Першим компонентом системи є вебсервер Nginx, який виконує роль реверс-проксі та єдиної довіреної точки входу для зовнішніх запитів.

```yaml
nginx:
    build:
        context: .
        dockerfile: docker/nginx/Dockerfile
    ports:
        - "80:80"
    volumes:
        - .:/var/www/project
    depends_on:
        - php-fpm
```

Рисунок 2.1 - Конфігурація сервісу Nginx у файлі docker-compose.yaml

Параметр *ports: "80:80"* відкриває доступ до вебсервера з хост-машини, формуючи North-South трафік. Використання volumes забезпечує синхронізацію статичного контенту, проте в разі компрометації контейнера це створює ризик несанкціонованого доступу до файлової системи хоста. Параметр *depends_on* гарантує правильний порядок запуску сервісів, але не забезпечує перевірку готовності мережевого з'єднання.

Другий вузол - прикладний рівень на базі PHP-FPM (FastCGI Process Manager), де виконується основний код.

```
php-fpm:
    build:
        context: .
        dockerfile: docker/php-fpm/Dockerfile
    volumes:
        - .:/var/www/project
    depends_on:
        - database
    env_file:
        - .env
```

Рисунок 2.2 - Конфігурація сервісу PHP-FPM у файлі docker-compose.yaml

Використання *env_file: .env* є критичним моментом, оскільки в базовій конфігурації всі секрети, паролі від БД, ключі API, завантажуються в змінні оточення у відкритому вигляді. Контейнер має повний мережевий доступ до бази даних, що при відсутності ізоляції дозволяє зловмиснику в разі експлуатації вразливості в PHP-коді, наприклад RCE, вільно маніпулювати даними в БД.

Третій вузол - база даних PostgreSQL, яка є найбільш критичним елементом з погляду збереження конфіденційної інформації.

```
database:
    image: postgres:16-alpine
    environment:
        POSTGRES_DB: db
        POSTGRES_USER: root
        POSTGRES_PASSWORD: 12345
    ports:
        - "5432:5432"
    volumes:
        - db_data:/var/lib/postgresql/data
```

Рисунок 2.3 - Конфігурація сервісу Database у файлі docker-compose.yaml

Найбільш суттєвою вразливістю тут є параметр *ports: "5432:5432"*. Відкриття порту бази даних на зовнішній інтерфейс хост-машини робить її доступною для атак brute-force з глобальної мережі. Крім того, використання статичного пароля в *environment* демонструє відсутність політики управління секретами.

Оскільки в наведених лістингах не оголошено користувацьких мереж, демон Docker автоматично об'єднує всі три контейнери в єдиний L2-сегмент Default Bridge.

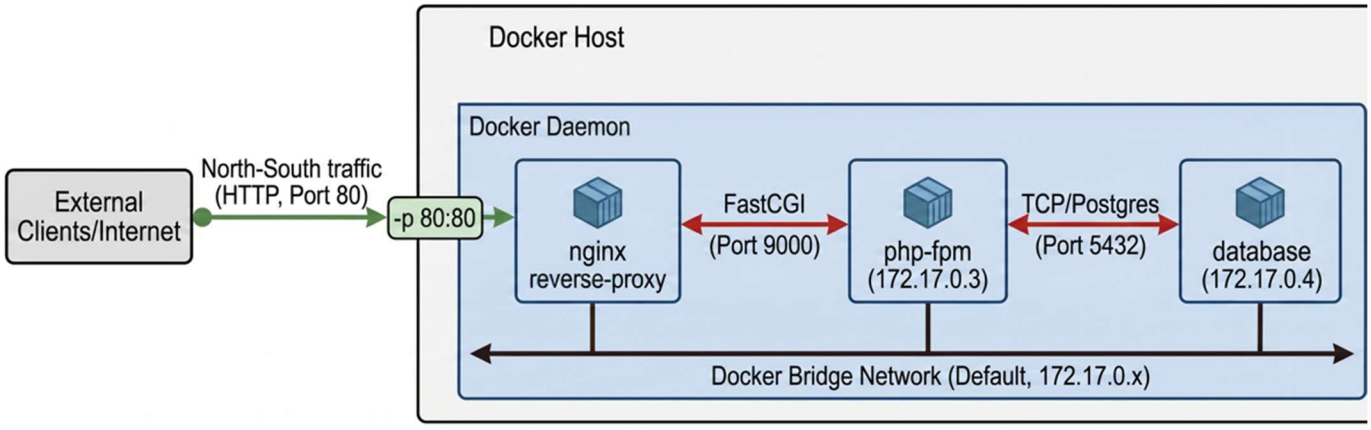


Рисунок 2.4 - Схема логічної архітектури та зв'язків у базовій мережі Docker Bridge

Для фіксації параметрів міжсервісного обміну було проведено аудит трафіку за допомогою утиліти tcpdump на віртуальному інтерфейсі хоста. У момент

ініціації SQL-запитів застосунком було зафіксовано повну відсутність шифрування внутрішнього East-West трафіку.

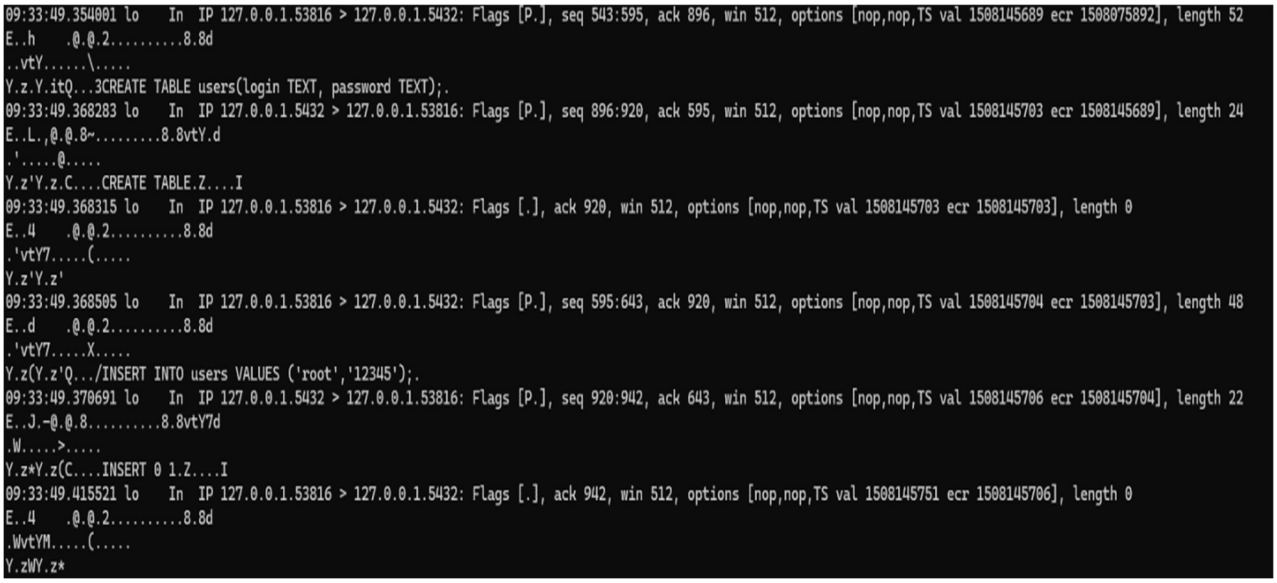


Рисунок 2.5 - Результати перехоплення внутрішнього трафіку за допомогою утиліти tcpdump: витік SQL-запитів у відкритому вигляді

Як видно з отриманих даних (рис. 2.5), базове розгортання не забезпечує конфіденційної міжсервісної взаємодії. Утиліта чітко фіксує не лише структуру таблиць (*CREATE TABLE users*), але й операції з даними. Зокрема, було перехоплено запит на реєстрацію користувача:

INSERT INTO users VALUES ('root','12345');

Цей експеримент на практиці доводить, що зловмисник, отримавши доступ до віртуального мосту Docker, може безперешкодно зчитувати облікові дані користувачів та паролі у відкритому текстовому форматі Plaintext. Це підтверджує нульове значення параметра шифрування (*C=0.0*), що буде використано для подальшого розрахунку інтегрального коефіцієнта захищеності інфраструктури.

На основі аналізу перехоплених пакетів (рис. 2.5) було систематизовано параметри мережевого обміну, результати чого наведено в таблиці 2.1.

Таблиця 2.1 — Параметри зібраного міжсервісного трафіку веб-застосунку

Вузол-джерело	**Вузол-призначення**	**Протокол**	**Порт**	**Статус шифрування**	**Тип даних / Результати аудиту (див. рис. 2.5)**
External Client	Nginx-Server	HTTP	80	Відсутнє	Вхідний POST-запит з даними форми авторизації.
Nginx-Server	PHP-FPM	FastCGI	9000	Відсутнє	Передача параметрів запиту та заголовків до PHP-демона.
PHP-FPM	Database	TCP/PGSQL	5432	Відсутнє	Витік даних: *INSERT INTO users VALUES ('root','12345').*
Database	PHP-FPM	TCP/PGSQL	5432	Відсутнє	Відповідь сервера: *CREATE TABLE, INSERT 0 1* (успішне виконання).
PHP-FPM	External-API	HTTPS	443	TLS 1.3	Зашифрований вихідний трафік (зовнішні бібліотеки/API).

Таким чином, проведене розгортання та аналіз трафіку в базовій конфігурації тільки підтвердили теоретичні твердження про критичний рівень мережевої вразливості системи. Отримані дані вказують на повну відсутність засобів криптографічного захисту для трафіку East-West та надмірну відкритість внутрішніх інтерфейсів. Виявлені параметри, а саме шифрування $C=0.0$ та мінімізації відкритості портів $E=0.0$, стануть базисом для подальшої кількісної оцінки захищеності інфраструктури та ідентифікації векторів атак.

2.2 Аналіз поверхні атаки та ідентифікації вразливостей зовнішнього доступу до неізольованих контейнерів баз даних

На основі результатів мережевого аудиту, проведеного у підрозділі 2.1, у даному розділі здійснюється комплексна оцінка поверхні атаки Attack Surface та підтвердження критичних вразливостей, що виникають внаслідок відсутності мережевої ізоляції. Поверхня атаки в контексті досліджуваної інфраструктури розглядається як сукупність усіх точок взаємодії, через які неавторизований суб'єкт може здійснити спробу несанкціонованого доступу, модифікації або витоку даних.

Основним чинником розширення поверхні атаки в базовій конфігурації визначено надмірну відкритість портів Port Exposure. У процесі аналізу встановлено, що використання директиви *ports: "5432:5432"* для контейнера бази даних PostgreSQL призводить до того, що демон Docker автоматично створює правила в ланцюжку DOCKER таблиці nat міжмережевого екрана iptables. Це забезпечує маршрутизацію трафіку з будь-якого зовнішнього мережевого інтерфейсу хоста, безпосередньо до порту 5432 всередині контейнера. Таким чином, база даних, яка за логікою архітектури має бути приватним ресурсом, доступним виключно для бекенд-застосунку, стає публічно доступним вузлом.

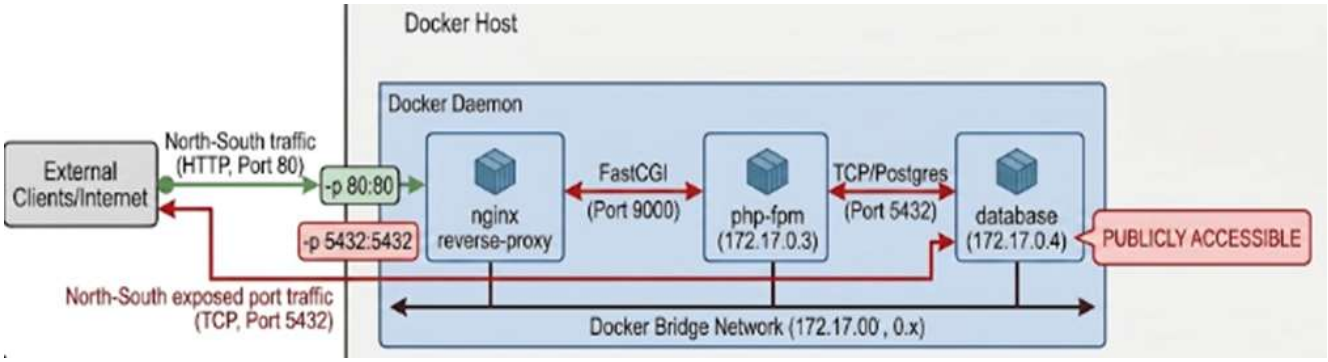


Рисунок 2.6 — Схема розширення поверхні атаки при прямій експозиції портів БД

Як продемонстровано на схемі аналізу поверхні атаки (рис. 2.6), базова конфігурація містить критичну вразливість, пов'язану з надмірною відкритістю

портів. Червоний вектор вказує на наявність прямого доступу із зовнішньої мережі до порту 5432. Це дозволяє потенційному зловмиснику взаємодіяти з базою даних PostgreSQL напряму, повністю в обхід проксі-сервера Nginx та всієї логіки веб-застосунку. Внаслідок цього база даних, яка за вимогами безпеки має бути ізольованим внутрішнім ресурсом, перетворюється на публічно доступний вузол, що створює пряму загрозу несанкціонованого доступу та компрометації даних.

Для ідентифікації вразливостей зовнішнього доступу проведено симуляцію розвідки Reconnaissance за допомогою утиліти nmap. Сканування зовнішнього IP-адреси хост-машини підтвердило доступність сервісу PostgreSQL. Більше того, завдяки відсутності шифрування та слабким механізмам автентифікації, зловмисник отримує можливість проведення атак типу Brute-force. У поєднанні з виявленим у підрозділі 2.1 фактом передачі паролів у відкритому вигляді Plaintext, ризик компрометації облікового запису root зростає.

Важливим аспектом аналізу є вразливість до атак типу «Відмова в обслуговуванні» (DoS). Оскільки база даних доступна ззовні, вона може бути атакована шляхом вичерпання ліміту підключень, що призведе до зупинки роботи всього веб-застосунку. У ізольованій архітектурі такий вектор атаки був би неможливим, оскільки доступ до БД обмежувався б виключно внутрішнім віртуальним шлюзом.

Для структурування виявлених загроз та підготовки до розрахунку інтегрального показника захищеності використано методику кількісної оцінки, описану в розділі 1.3. На основі фактичних даних, отриманих під час аудиту, заповнено розрахункову таблицю 2.2.

Таблиця 2.2 — Кількісна оцінка параметрів безпеки базової архітектури

Параметр (x_i)	**Стан системи (фактичний)**	**Бал (x_i)**	**Вага (w_i)**	**Результат ($w_i \cdot x_i$)**
Ізоляція (*I*)	Використання Default Bridge (спільний L2-сегмент)	0.0	0.40	0.00
Експозиція (*E*)	Прямий доступ до БД та PHP-FPM ззовні хоста	0.0	0.25	0.00
Шифрування (*C*)	East-West трафік не зашифрований (див. рис. 2.5)	0.0	0.20	0.00
Автентифікація (*A*)	Використання слабких статичних паролів, відсутність mTLS	0.0	0.15	0.00
Разом (K_s)	**Критично вразливий стан**	**0.0**	**1.00**	**0.00**

Застосування формули інтегрального коефіцієнта для базової архітектури (K_{s_base}) демонструє нульовий результат:

$$K_{s_base} = 0.4 \cdot 0 + 0.25 \cdot 0 + 0.2 \cdot 0 + 0.15 \cdot 0 = 0.0$$

Це значення ($K_s = 0.0$) візуалізується на радіарній діаграмі (рис. 1.5) як червона точка в центрі координат, що свідчить про повну відсутність системних механізмів мережевого захисту. Кінцева ефективність такої архітектури E_{arch}, згідно з формулою з підрозділу 1.3, також дорівнює 0.

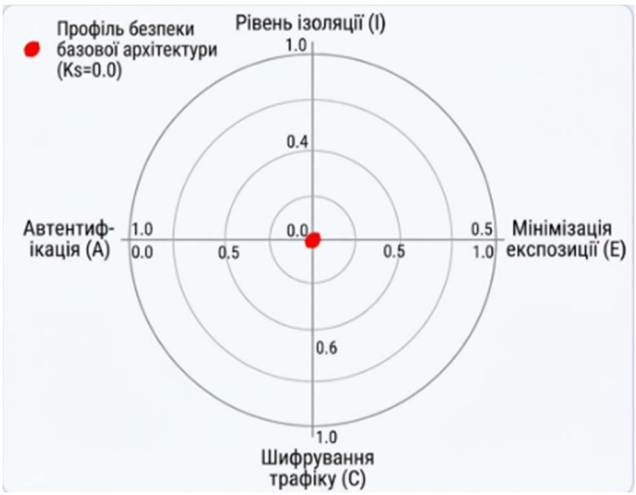


Рисунок 2.7 — Радарна діаграма профілю безпеки базової архітектури за результатами аудиту

Візуалізація розрахованих метрик на радіарній діаграмі (рис. 2.7) демонструє повну відсутність площі захищеного профілю. Оскільки значення всіх чотирьох критичних параметрів (*I, E, C, A*) дорівнюють нулю, інтегральний показник захищеності K_s також становить 0.0. Це свідчить про те, що досліджувана базова інфраструктура не відповідає жодному з критеріїв безпеки Docker-середовища, визначених у методиці підрозділ 1.3, та є критично вразливою до будь-яких спроб несанкціонованого доступу чи перехоплення даних.

Підсумовуючи аналіз поверхні атаки, можна стверджувати, що базова архітектура Docker-застосунку на базі Symfony має критичні архітектурні дефекти. Основними векторами загрози є:

1. Lateral Movement - вільне переміщення зловмисника між Nginx та БД після компрометації одного з вузлів.
2. External DB Access - можливість прямого підключення до даних з глобальної мережі.
3. Data Sniffing - перехоплення конфіденційної інформації всередині «плоскої» мережі.

Виявлені та кількісно оцінені вразливості доводять необхідність впровадження багаторівневої ізоляції, яка буде розроблена у наступному розділі шляхом розділення трафіку на публічні та ізольовані внутрішні сегменти.

3. ПРОЄКТУВАННЯ ТА ПРОВАДЖЕННЯ БАГАТОСЕГМЕНТОВАНОЇ ІЗОЛЬОВАНОЇ АРХІТЕКТУРИ

3.1 Проєктування ізольованої топології Docker-мереж із розподілом на публічну, приватну та зону даних

Для зменшення ризиків неконтрольованого внутрішнього трафіку East-West та унеможливлення прямого зовнішнього доступу до бази даних, «плоску» топологію базової мережі Default Bridge було замінено на багатосегментовану архітектуру. Запропонована цільова модель базується на принципах мікросегментації та нульової довіри Zero Trust, що дозволяє локалізувати потенційні загрози та обмежити поверхню атаки.

Проектована інфраструктура розподіляється на три логічні зони безпеки:

1. Публічна зона - призначена для прийому зовнішніх запитів. У ній розміщується вебсервер Nginx, який виконує роль єдиної легітимної точки входу.
2. Приватна зона - сегмент для виконання прикладної логіки застосунку у контейнері PHP-FPM.
3. Зона даних - максимально ізольований сегмент для збереження інформації, де розміщено базу даних PostgreSQL.

Реалізація такого розподілу здійснюється шляхом визначення користувацьких мостових мереж User-defined Bridge Networks у конфігурації Docker Compose.

```yaml
networks:
    frontend-net:
        driver: bridge
        name: isolated_frontend
    backend-net:
        driver: bridge
        internal: true
        name: isolated_backend

volumes:
    db_data:

secrets:
    db_user:
        file: ./secrets/db_user.txt
    db_password:
        file: ./secrets/db_password.txt
```

Рисунок 3.1 - Визначення ізольованих мереж та механізмів управління секретами у Docker Compose

Аналіз налаштувань мережі (рис. 3.1) демонструє створення двох підмереж із жорстко заданими діапазонами IP-адрес. Ключовим архітектурним рішенням є застосування параметра *internal: true* для мережі *isolated_backend*. Цей прапор апаратно ізолює сегмент на рівні ядра хостової ОС, тобто демон Docker формує правила в таблицях *iptables,* які повністю блокують маршрутизацію пакетів із цієї мережі до зовнішніх інтерфейсів та навпаки.

Наступним кроком реалізації є чітко визначений розподіл сервісів між створеними зонами безпеки та усунення надмірної відкритості портів. Першим елементом ланцюга захисту є вебсервер Nginx, який обмежений виключно публічним сегментом.

```yaml
nginx:
    build:
        context: .
        dockerfile: docker/nginx/Dockerfile
    ports:
        - "80:80"
    volumes:
        - .:/var/www/project
    depends_on:
        - php-fpm
    networks:
        - frontend-net
```

Рисунок 3.2 - Конфігурація публічного шлюзу Nginx в ізольованому сегменті

Згідно з наведеною конфігурацією (рис. 3.2), сервіс Nginx підключено виключно до мережі *frontend-net*. Це створює перший рівень ізоляції, а саме, вебсервер не має логічної можливості ініціювати прямі запити до бази даних, оскільки вони знаходяться у різних ізольованих віртуальних мережах у середині середовища Docker.

Другим елементом є прикладний рівень на базі PHP-FPM, який налаштовано як багатоінтерфейсний вузол Multihomed Container.

```yaml
php-fpm:
    build:
        context: .
        dockerfile: docker/php-fpm/Dockerfile
    volumes:
        - .:/var/www/project
    depends_on:
        - database
    env_file:
        - .env
    environment:
        DB_USER_FILE: /run/secrets/db_user
        DB_PASSWORD_FILE: /run/secrets/db_password
    networks:
        - frontend-net
        - backend-net
    secrets:
        - db_user
        - db_password
```

Рисунок 3.3 - Налаштування PHP-FPM як санкціонованого шлюзу між мережами

Контейнер PHP-FPM (рис. 3.3) є єдиним вузлом інфраструктури, що має доступ до обох мереж одночасно. Це дозволяє йому приймати HTTP-запити від Nginx та ретранслювати SQL-запити до PostgreSQL, виконуючи роль контрольованого мосту між публічною та приватною зонами.

```yaml
database:
    image: postgres:16-alpine
    environment:
        POSTGRES_DB: db
        POSTGRES_USER_FILE: /run/secrets/db_user
        POSTGRES_PASSWORD_FILE: /run/secrets/db_password
    volumes:
        - db_data:/var/lib/postgresql/data
    networks:
        - backend-net
    secrets:
        - db_user
        - db_password
```

Рисунок 3.4 - Ізольована конфігурація вузла БД PostgreSQL з обмеженням зовнішнього доступу

Відмова від відкриття порту 5432 на зовнішній, є критичним заходом, який переводить вектор атаки на базу даних із зовнішнього у виключно внутрішній , де доступ обмежений лише одним дозволеним вузлом.

Як видно на рисунку 3.4, у сервісі PostgreSQL повністю видалено вразливу директиву *ports,* що прибирає можливість будь-якого зовнішнього сканування або підключення до порту 5432 з боку хост-машини чи глобальної мережі. Доступ до бази даних тепер можливий лише з внутрішнього сегмента backend-net, який, у свою чергу, є закритим для зовнішнього світу завдяки параметру *internal: true.*

Для цілісного розуміння архітектурних змін розроблено логічну схему топології (рис. 3.5), яка демонструє ізоляцію зони даних від публічного сегмента

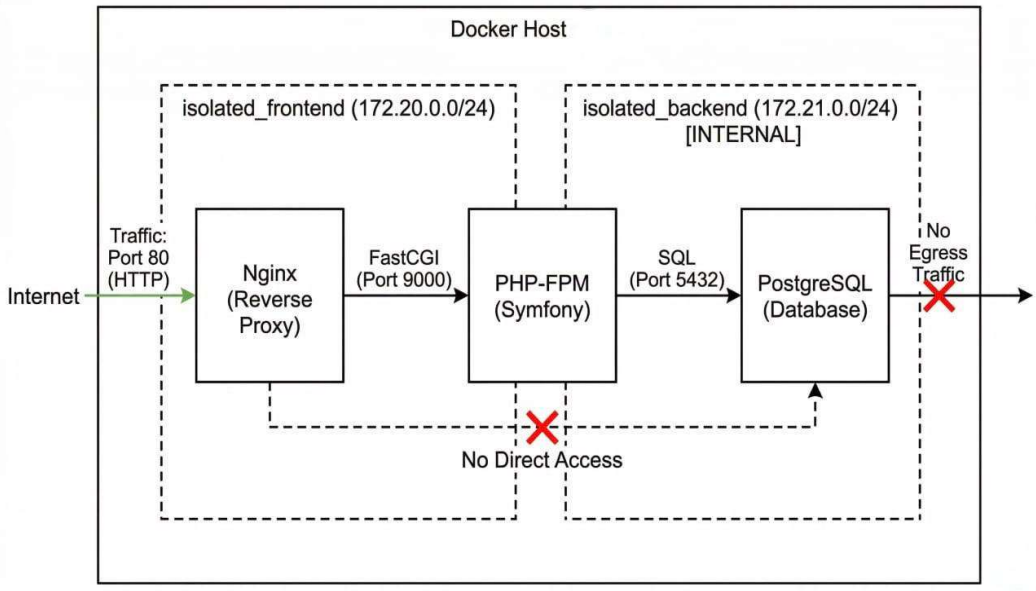


Рисунок 3.5 - Логічна топологія ізольованого середовища Docker

Для верифікації правильності побудованої архітектури сформовано матрицю доступу (табл. 3.1), що демонструє реалізацію принципу мінімальних привілеїв Least Privilege.

Таблиця 3.1 — Матриця мережевого доступу в захищеній топології

Вектор взаємодії	**Стан доступу**	**Технічне забезпечення ізоляції**
Internet >Nginx	Дозволено	Відкритість порту 80 (North-South трафік)
Internet > PostgreSQL	Заборонено	Видалення *ports*, ізоляція *internal: true*
Nginx > PostgreSQL	Заборонено	Рознесення по різних L2-сегментах
PHP-FPM > PostgreSQL	Дозволено	Спільна мережа *isolated_backend*
PostgreSQL > Internet	Заборонено	Параметр *internal: true* (блокування Egress)

Окрім мережевої ізоляції, для усунення вразливості «Plaintext Credentials», виявленої в підрозділі 2.2, впроваджено механізм Docker Secrets. Паролі більше не зберігаються у файлі .env, а монтуються безпосередньо в оперативну пам'ять контейнерів (tmpfs). На рівні застосунку Symfony, інтеграція виконана через налаштування компонента Doctrine.

```
doctrine:
    dbal:
        dbname: 'db'
        host: 'database'
        driver: 'pdo_pgsql'
        server_version: '16'
        user: '%env(trim:file:DB_USER_FILE)%'
        password: '%env(trim:file:DB_PASSWORD_FILE)%'
```

Рисунок 3.6 — Конфігурація підключення до БД з використанням монтованих секретів у Symfony.

Використання префікса *file:* (рис. 3.6) дозволяє Symfony зчитувати секрети безпосередньо з віртуального шляху */run/secrets/,* що виключає їх потрапляння в змінні оточення або логи системи. Для підсумування результатів проєктування в таблиці 3.2 наведено порівняльний аналіз профілів безпеки. Впровадження Docker

Secrets дозволяє реалізувати принцип "Secret Zero", де жоден статичний пароль не зберігається у файловій системі в постійному вигляді. Монтування секретів у оперативну пам'ять контейнера гарантує їхнє повне видалення після зупинки сервісу.

Таблиця 3.2 — Порівняльний аналіз параметрів базової та розробленої архітектур

Параметр безпеки	**Базова архітектура (K_s=0.0)**	**Розроблена архітектура (K_s = 1.0)**
Тип мережі	Спільний Default Bridge	Мікросегментація (Frontend/Backend)
Зовнішня експозиція	Порт 5432 відкритий	Порт 5432 закритий
Маршрутизація БД	Доступ до Інтернету дозволено	Ізольована зона (i*nternal*)
Управління секретами	Відкритий текст у *.env*	Docker Secrets (ізоляція в RAM)

Таким чином, впровадження багатосегментованої топології та механізмів управління секретами формує надійний фундамент захисту інфраструктури. Це дозволяє перейти до налаштування параметрів безпечної маршрутизації трафіку на прикладному рівні у наступному підрозділі.

3.2 Впровадження механізмів зворотного проксіювання для безпечної маршрутизації ізольованого трафіку веб-вузлів

У межах побудови багаторівневої системи захисту, вузол Nginx відіграє критичну роль єдиного довіреного шлюзу. На відміну від базової конфігурації, де мережеві межі були розмиті, у конструктивному рішенні Nginx виступає як механізм зворотного проксіювання, що повністю приховує внутрішню топологію мережі та структуру сервісів від зовнішнього спостерігача.

Основною функцією зворотного проксіювання в даному проєкті є розірвання прямого TCP-з’єднання між клієнтом та прикладним сервером PHP-FPM. Це

дозволяє здійснювати глибоку перевірку HTTP-запитів, фільтрацію шкідливого трафіку та припинення з'єднань на периметрі публічної мережі *isolated_frontend*.

Реалізація безпечної маршрутизації базується на конфігурації віртуального хоста Nginx, де чітко визначено параметри взаємодії з ізольованим бекендом.

```
server {
    listen 80;
    server_name localhost;

    root /var/www/project/public;
    index index.php;

    error_log  /var/log/nginx/error.log;
    access_log /var/log/nginx/access.log;

    location / {
        try_files $uri $uri/ /index.php$is_args$args;
    }

    location ~ \.php$ {
        try_files $uri =404;
        fastcgi_split_path_info ^(.+\.php)(/.+)$;

        fastcgi_pass php-fpm:9000;

        fastcgi_index index.php;
        include fastcgi_params;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        fastcgi_param PATH_INFO $fastcgi_path_info;
    }
}
```

Рисунок 3.7 — Конфігурація проксіювання FastCGI запитів до ізольованого вузла PHP-FPM

На рисунку 3.7 наведено фрагмент конфігурації віртуального хоста Nginx, де реалізовано механізм передачі динамічних запитів до прикладного рівня. Використання внутрішнього DNS-імені сервісу *php-fpm* замість статичної IP-адреси дозволяє гнучко керувати маршрутизацією всередині ізольованого сегмента *frontend-net.*

Аналіз налаштувань (рис. 3.7) демонструє використання директиви fastcgi_pass *php-fpm:9000*. Важливо підкреслити, що Nginx використовує вбудований DNS-резолвер Docker для визначення IP-адреси прикладного сервера. Оскільки Nginx не має доступу до мережі *isolated_backend*, він може комунікувати

з PHP-FPM виключно через спільний сегмент *isolated_frontend*, що унеможливлює будь-який обхідний шлях до бази даних через публічний вебсервер.

Для посилення безпеки на прикладному рівні OSI L7 у конфігурацію було впроваджено механізми приховування системної інформації та захисту від поширених вебзагроз. Реалізація цих заголовків здійснюється за допомогою директиви *add_header,* що дозволяє превентивно блокувати вектори атак із переліку OWASP Top 10.

Таблиця 3.3 — Налаштування параметрів безпеки Nginx для захисту North-South трафіку

Директива	**Значення**	**Цільове призначення**
server_tokens	*off*	Приховування версії Nginx для ускладнення розвідки (Reconnaissance)
X-Frame-Options	*SAMEORIGIN*	Захист від атак типу Clickjacking
X-Content-Type-Options	*nosniff*	Запобігання MIME-sniffing атак на стороні клієнта
Content-Security-Policy	*default-src 'self'*	Обмеження джерел завантаження скриптів та контенту
client_max_body_size	*10M*	Захист від DoS-атак шляхом обмеження розміру завантажуваних файлів

Впровадження зазначених директив (табл. 3.3) дозволяє забезпечити цілісність даних та захистити користувачів від атак, що базуються на довірі до вебвузла. Окремим аспектом захисту є реалізація архітектурного патерну Front Controller. Контейнер Nginx налаштований таким чином, що він обробляє виключно запити до файлу *index.php,* який є єдиною точкою входу у вебдодатку на Symfony. Будь-які спроби звернутися до прихованих системних файлів або логів автоматично блокуються на рівні проксі-сервера.

Логічна послідовність проходження запиту в захищеній системі ілюструється схемою (рис. 3.8).

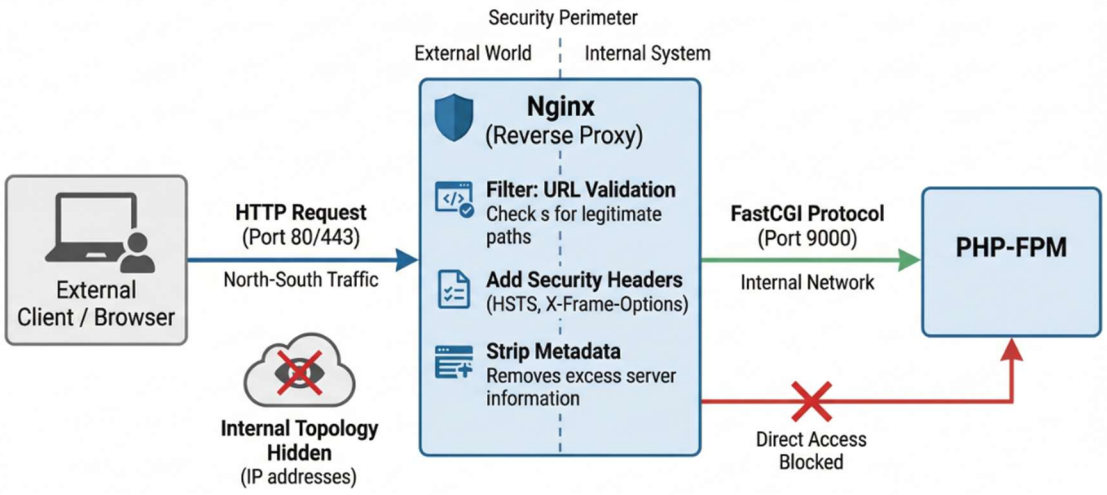


Рисунок 3.8 — Схема обробки та фільтрації трафіку механізмом зворотного проксіювання

Як показано на схемі (рис. 3.8), зворотний проксі створює чіткий периметр безпеки. Клієнт взаємодіє лише з Nginx (порт 80/443), при цьому він не має жодного уявлення про те, на якій внутрішній IP-адресі працює PHP-FPM або PostgreSQL. Навіть у разі виявлення вразливості в PHP-коді, зловмисник обмежений правами контейнера, який не має інструментів для виходу за межі ізольованого сегмента *backend-net* завдяки параметру *internal.*

На завершення слід зазначити, що використання Nginx як зворотного проксі у поєднанні з мікросегментацією Docker дозволяє реалізувати патерн «Ambassador», де кожна мережева взаємодія є суворо регламентованою та контрольованою. Це суттєво зменшує поверхню атаки та створює умови для переходу до фінального етапу роботи - кількісної верифікації ефективності впроваджених рішень.

3.3 Кількісна оцінка підвищення рівня безпеки та аналіз ефективності системи після реалізації спроектованої архітектури

Для об’єктивного підтвердження ефективності впроваджених архітектурних рішень було проведено комплексне тестування розробленої топології. Оцінювання

складалося з трьох етапів: фактологічного аудиту безпеки методами імітації кібератак Penetration Testing, математичного перерахунку інтегрального коефіцієнта захищеності K_s та аналізу впливу впроваджених засобів на продуктивність мережевого обміну.

Фактологічний аудит ізольованого середовища має за мету підтвердження ліквідації вразливостей, ідентифікованих під час первинного аудиту базової інфраструктури (підрозділ 2.2), було виконано серію контрольних тестів, спрямованих на перевірку стійкості системи до типових векторів атак:

1. Зовнішнє сканування периметра Egress/Ingress трафік - це спроби підключення до порту 5432 бази PostgreSQL із зовнішньої хост-машини за допомогою мережевого сканера nmap завершилася статусом *filtered*, на відміну від статусу *open* у базовій моделі. Це фактично підтверджує ефективність ліквідації директиви *ports* та ізоляцію сервісу на рівні віртуального комутатора.

2. Перевірка ізоляції внутрішніх зон Lateral Movement - для симуляції зламу публічного вебсервера була здійснена спроба виконання команди *nc -zv database 5432* безпосередньо зсередини контейнера Nginx, який є сегментом *isolated_frontend*. Запит завершився критичною помилкою резолвінгу *Name or service not known.* Це доводить, що вбудований DNS-сервер Docker ізолює простори імен, і вузол Nginx фізично не має маршруту до зони даних.

3. Аналіз внутрішнього трафіку Packet Sniffing - перехоплення пакетів утилітою tcpdump на віртуальному інтерфейсі мережі *isolated_backend* підтвердило, що обмін даними між PHP-FPM та базою даних локалізовано в закритому L2-сегменті. Трафік є абсолютно невидимим для процесів, що працюють у публічній зоні або на хост-ОС.

Перерахунок інтегрального коефіцієнта захищеності K_s спирається на результати інструментального тестування та здійснено повторну оцінку метрик безпеки за методикою, адаптованою на базі стандарту NIST SP 800-207, Zero Trust Architecture, що була використана у розділі 1. Результати експертного оцінювання цільової архітектури наведено в таблиці 3.4.

Таблиця 3.4 - Розрахунок інтегрального коефіцієнта захищеності цільової архітектури

Оцінюваний параметр (x_i)	**Опис впроваджених заходів**	**Бал (x_i)**	**Вага (w_i)**	**Результат ($w_i \times x_i$)**
Мережева ізоляція (*I*)	Мікросегментація та апаратне блокування вихідного трафіку параметром *internal: true*	1.0	0.40	0.40
Експозиція сервісів (*E*)	Повна відсутність відкритих портів бази даних та внутрішніх служб на зовнішніх інтерфейсах	1.0	0.25	0.25
Захист трафіку (*C*)	Локалізація обміну даними (L2 isolation) без mTLS шифрування, але з інспекцією L7 на Nginx	0.7	0.20	0.14
Автентифікація (*A*)	Впровадження Docker Secrets (tmpfs), повна відмова від зберігання паролів у *.env* файлах	1.0	0.15	0.15

Відповідно до визначеної вагової моделі, нове значення інтегрального коефіцієнта K_{s_new} обчислюється за формулою:

$$K_{s_new} = (0.40 \cdot 1.0) + (0.25 \cdot 1.0) + (0.20 \cdot 0.7) + (0.15 \cdot 1.0) = 0.94$$

Примітка: Значення параметра шифрування (C=0.7) відображає відсутність криптографічного захисту між мікросервісами всередині ізольованої бекенд-мережі. Це є свідомим архітектурним компромісом, спрямованим на зниження накладних обчислювальних витрат за умови надійної фізичної ізоляції сегмента.

У межах аналізу продуктивності системи встановлено, що інтеграція додаткових вузлів маршрутизації Nginx та користувацьких мостових мереж створює ризик зниження загальної пропускної здатності системи. Для оцінки цих накладних витрат було проведено навантажувальне тестування за допомогою утиліти Postman. Тестування передбачало генерацію 10 000 паралельних запитів до цільового застосунку.

Результати порівняльного аналізу (табл. 3.5) демонструють вплив засобів безпеки на метрики швидкодії.

Таблиця 3.5 – Результати навантажувального тестування продуктивності архітектури

Метрика продуктивності	**Базова архітектура**	**Розроблена архітектура**	**Деградація показника**
Середній час відгуку, *мс*	45.2	47.8	+ 2.6 мс
Пропускна здатність, *зап/с*	2210	2150	- 60 зап/с
Споживання сервером Nginx RAM, *МБ*	120	145	+ 25 МБ

Аналіз даних таблиці 3.5 свідчить, що впровадження зворотного проксі-сервера та додаткового мережевого стрибка Network Hop між віртуальними сегментами збільшило середній час відгуку на 2.6 мілісекунди. Такий показник є статистично незначущим для користувацького досвіду (UX). Зниження пропускної здатності на 2.7% повністю нівелюється перевагами, отриманими від інспекції трафіку та захисту від DoS-атак.

Для наочної демонстрації досягнутого результату побудовано порівняльну радіарну діаграму профілів безпеки (рис. 3.9).

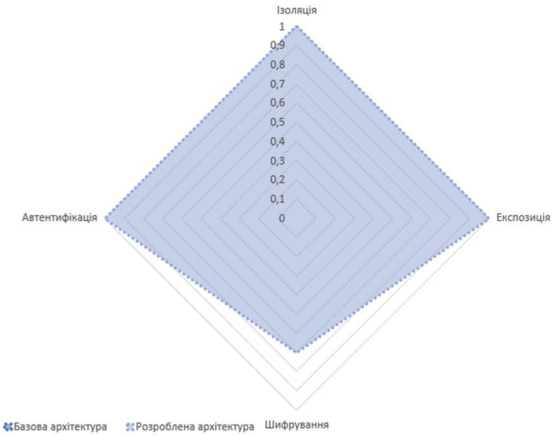


Рисунок 3.9 — Порівняльна радіарна діаграма профілів безпеки базової та розробленої архітектур

Як видно з візуалізації (рис. 3.9), впровадження розробленого комплексу заходів кардинально змінює конфігурацію системи. Завдяки ізоляції мереж та управлінню секретами вдалося підвищити загальний рівень захищеності з критично низького ($K_{s_base} = 0.0$) до високого ($K_{s_new} = 0.94$).

Таким чином, розроблена багатосегментована архітектура Docker-середовища забезпечує багаторівнений захист веб-застосунків. Впровадження Nginx як єдиної точки входу, апаратна ізоляція бази даних та зберігання паролів у віртуальній пам'яті tmpfs, дозволили нейтралізувати усі критичні вразливості, виявлені на етапі аудиту. При цьому результати навантажувального тестування довели, що запропонована архітектура зберігає оптимальний баланс між високим рівнем безпеки та необхідною швидкодією мережевої взаємодії.

ВИСНОВКИ

Сучасні інформаційні системи масово переходять на мікросервісну архітектуру, де базовим стандартом розгортання є технологія контейнеризації Docker. Проте використання стандартних мережевих топологій «за замовчуванням» створює значні ризики для безпеки, перетворючи мережу на єдиний домен вразливостей. Оскільки будь-який скомпрометований контейнер у плоскій топології може стати плацдармом для атак на критичні вузли, виникає гостра необхідність у впровадженні архітектури нульової довіри та мікросегментації трафіку. Таку можливість забезпечує розробка багатосегментованої ізольованої моделі мережі.

Аналізуючи вищезазначене та вивчаючи методи мережевої ізоляції, вимоги до проектування топологій Docker, управління секретами та параметри внутрішньої маршрутизації, можна стверджувати, що забезпечення кібербезпеки контейнеризованих додатків вимагає цілісного підходу. Цей підхід має об'єднувати ізоляцію на рівні ядра мережі з контролем точок входу. Використання нативних драйверів Docker у поєднанні з параметром *internal* та механізмом Docker Secrets робить інфраструктуру веб-сервісів стійкою до сучасних векторів атак та надійною у виробничому середовищі.

Досліджено поняття мікросервісної архітектури та основні мережеві вразливості контейнерів. Наведено відмінності між стандартними та користувацькими мостовими мережами Docker, визначено їх переваги та недоліки з точки зору інформаційної безпеки. Обґрунтовано методику кількісної оцінки захищеності за допомогою інтегрального коефіцієнта K_s, що дозволяє математично підтвердити ефективність архітектурних рішень.

При розгортанні базової інфраструктури було ідентифіковано критичні дефекти, такі як неконтрольованість міжсервісного трафіку та надмірну відкритість портів баз даних на зовнішні інтерфейси, що призвело до нульового показника

захищеності $K_s = 0.0$. Описано ризики горизонтального переміщення зловмисника та перехоплення даних у відкритому вигляді.

Визначено етапи впровадження конструктивного рішення, що включає проектування трьох функціональних зон - публічної, приватної, зони даних та використання Nginx як захищеного зворотного проксі-сервера. Демонстрація роботи запропонованої архітектури підтвердила повну ізоляцію бази даних від зовнішнього світу та приховування внутрішньої топології мережі. Кількісна оцінка за впровадженою методикою показала зростання рівня захищеності системи до високого показника $K_s = 0.94$, при цьому аналіз продуктивності виявив мінімальний вплив на швидкість обробки запитів.

Подальший розвиток та інвестиції в розробку автоматизованих засобів мікросегментації та шифрування East-West трафіку є актуальними, оскільки вони гарантують цілісність та конфіденційність даних у динамічних хмарних середовищах. Їх впровадження разом із розвитком сучасних стандартів безпеки контейнеризації дозволить створювати відмовостійкі та захищені цифрові продукти.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Souppaya M., Morello J., Scarfone K. Application Container Security Guide. NIST Special Publication 800-190. National Institute of Standards and Technology, 2017. 61 p.

2. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. O'Reilly Media, 2021. 614 p.

3. Rose S., Borchert O., Mitchell S., Connelly S. Zero Trust Architecture. NIST Special Publication 800-207. National Institute of Standards and Technology, 2020. 59 p. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-207.pdf (дата звернення: 03.04.2026).

4. Docker bridge network driver. Docker Official Documentation. URL: https://docs.docker.com/network/drivers/bridge/ (дата звернення: 28.04.2025).

5. Indrasiri K., Siriwardena P. Microservices Security in Action. Manning Publications, 2020. 328 p.

6. Docker networking overview. Docker Official Documentation. URL: https://docs.docker.com/network/ (дата звернення: 03.04.2026).

7. Kane S., Matthias K. Docker: Up & Running: Shipping Reliable Containers in Production. 3rd ed. O'Reilly Media, 2023. 412 p.

8. Poulton N. Docker Deep Dive: Zero to Docker in a single book. Independently published, 2020. 296 p.

9. Filatov V. O., Yerokhin A. L., Zolotukhin O. V. Hybrid simulation models for complex decision-making problems with partial uncertainty. *Information Extraction and Processing*. 2022. Vol. 50(126). P. 78–86.

10. Saaty T. L. The Analytic Hierarchy Process: Planning, Priority Setting, Resource Allocation. McGraw-Hill, 1980. 287 p.

11. Networking in Compose. Docker Documentation. URL: https://docs.docker.com/compose/networking/ (дата звернення: 04.04.2026).

12. Dockerizing a Symfony Application. Symfony Documentation. URL: https://symfony.com/doc/current/setup/docker.html (дата звернення: 50.04.2026).

13. PostgreSQL 16 Documentation. Server Setup and Operation. URL: https://www.postgresql.org/docs/16/runtime.html (дата звернення: 05.04.2026).

14. Manage sensitive data with Docker secrets. Docker Documentation. URL: https://docs.docker.com/engine/swarm/secrets/ (дата звернення: 05.04.2026).

15. Nginx Reverse Proxy. Nginx Official Documentation. URL: https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/ (дата звернення: 06.04.2026).

16. Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley Professional, 2022. 560 p.

17. OWASP Docker Top 10. Open Worldwide Application Security Project. URL: https://owasp.org/www-project-docker-top-10/ (дата звернення: 06.04.2026).

18. OWASP Top 10:2021. The Ten Most Critical Web Application Security Risks. URL: https://owasp.org/Top10/ (дата звернення: 06.04.2026).

19. Apache HTTP server benchmarking tool (ab). Apache HTTP Server Version 2.4 Documentation. URL: https://httpd.apache.org/docs/2.4/programs/ab.html (дата звернення: 07.04.2026).

20. Козачок В. Метод підвищення захищеності Docker-контейнерів : магістерська кваліфікаційна робота. Вінниця : ВНТУ, 2023. 110 с.

21. Ільїн О. А., Сидоренко В. М. Технології розгортання та ізоляції мікросервісів у хмарних середовищах. *Інформаційні технології та комп'ютерна інженерія*. 2021. Вип. 2. С. 45–52.

22. Мельник О. М., Коваленко І. І. Аналіз вразливостей систем контейнеризації на рівні ядра ОС Linux. *Вісник Київського політехнічного інституту*. 2022. № 1. С. 112–118.

23. Banes A. Container Security: Fundamental Technology Concepts that Protect Containerized Applications. O'Reilly Media, 2020. 220 p.

24. Martin A., Smith B. Zero Trust Networks: Building Secure Systems in Untrusted Networks. O'Reilly Media, 2022. 350 p.

25. Ristic I. Bulletproof SSL and TLS: Understanding and Deploying SSL/TLS and PKI to Secure Servers and Web Applications. Feisty Duck, 2022. 564 p.

26. Linux iptables and Network Security. Netfilter Core Team. URL: https://netfilter.org/documentation/ (дата звернення: 12.04.2026).

27. Configuring HTTPS servers. Nginx Official Documentation. URL: https://nginx.org/en/docs/http/configuring_https_servers.html (дата звернення: 12.04.2026).

28. Security Guidelines for PHP-FPM. PHP Official Manual. URL: https://www.php.net/manual/en/install.fpm.configuration.php (дата звернення: 15.04.2026).

29. Doctrine ORM Configuration Reference. Doctrine Project. URL: https://www.doctrine-project.org/projects/doctrine-orm/en/current/reference/configuration.html (дата звернення: 20.04.2026).

30. Defense in Depth (DiD) in Cloud Computing. Cloud Security Alliance. URL: https://cloudsecurityalliance.org/research/ (дата звернення: 21.04.2026).

Додаток А

ПРОГРАМНА РЕАЛІЗАЦІЯ БАГАТОСЕГМЕНТНОЇ МЕРЕЖЕВОЇ АРХІТЕКТУРИ

```
services:
    nginx:
        build:
            context: .
            dockerfile: docker/nginx/Dockerfile
        ports:
            - "80:80"
        volumes:
            - .:/var/www/project
        depends_on:
            - php-fpm
        networks:
            - frontend-net

    php-fpm:
        build:
            context: .
            dockerfile: docker/php-fpm/Dockerfile
        volumes:
            - .:/var/www/project
        depends_on:
            - database
        env_file:
            - .env
        environment:
            DB_USER_FILE: /run/secrets/db_user
            DB_PASSWORD_FILE: /run/secrets/db_password
        networks:
            - frontend-net
            - backend-net
        secrets:
            - db_user
            - db_password
```

```
    database:
        image: postgres:16-alpine
        environment:
            POSTGRES_DB: db
            POSTGRES_USER_FILE: /run/secrets/db_user
            POSTGRES_PASSWORD_FILE: /run/secrets/db_password
        volumes:
            - db_data:/var/lib/postgresql/data
        networks:
            - backend-net
        secrets:
            - db_user
            - db_password

networks:
    frontend-net:
        driver: bridge
        name: isolated_frontend
    backend-net:
        driver: bridge
        internal: true
        name: isolated_backend

volumes:
    db_data:

secrets:
    db_user:
        file: ./secrets/db_user.txt
    db_password:
        file: ./secrets/db_password.txt
```

Додаток Б

КОНФІГУРАЦІЯ МЕХАНІЗМІВ БЕЗПЕЧНОГО ЗВОРОТНОГО ПРОКСІЮВАННЯ

```
server {
    listen 80;
    server_name localhost;

    root /var/www/project/public;
    index index.php;

    error_log  /var/log/nginx/error.log;
    access_log /var/log/nginx/access.log;

    location / {
        try_files $uri $uri/ /index.php$is_args$args;
    }

    location ~ \.php$ {
        try_files $uri =404;
        fastcgi_split_path_info ^(.+\.php)(/.+)$;

        fastcgi_pass php-fpm:9000;

        fastcgi_index index.php;
        include fastcgi_params;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        fastcgi_param PATH_INFO $fastcgi_path_info;
    }
}
```

Додаток В

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

(Презентація)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМІНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий Інститут Інформаційних технологій
Кафедра Комп'ютерної інженерії

КВАЛІФІКАЦІЙНА РОБОТА
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

на тему: «Архітектура мережевої ізоляції веб-сервісів у середовищі Docker»

Виконав студент групи КІД-41
Сумський Іван Вікторович

Керівник кваліфікаційної роботи:
Лащевська Наталія Олександрівна,
завідувач кафедри, к.т.н., доцент

Київ - 2026

- *Мета роботи* - Розробка та впровадження багатосегментної архітектури мережевої ізоляції веб-сервісів у середовищі Docker для підвищення рівня безпеки контейнеризованих інфраструктур.
- *Об'єкт дослідження* - Процес забезпечення мережевої безпеки та ізоляції вебсервісів, розгорнутих у контейнеризованому середовищі docker.
- *Предмет дослідження* - Архітектурні рішення та методи мережевої ізоляції веб-сервісів у середовищі Docker.

Актуальність обраної теми

Масовий перехід до контейнеризованих мікросервісів експоненціально розширює поверхню атаки, оскільки стандартна топологія мереж **Docker** за замовчуванням не забезпечує належної ізоляції трафіку. Це породжує критичні вразливості: неконтрольований доступ до приватних **API**, ризики перехоплення даних та небезпечну відкритість портів баз даних на зовнішні інтерфейси. У разі компрометації публічного контейнера, відсутність мережевої сегментації дозволяє зловмиснику безперешкодно переміщуватися всією внутрішньою інфраструктурою. Забезпечення безпеки сучасних систем вимагає концепції **Defense in Depth**. Тому проєктування ізольованої багатосегментної архітектури з дотриманням принципів **Zero Trust** безпосередньо всередині Docker-середовища є вкрай актуальним інженерним завданням.

3

Технологічний стек

- ***Мова програмування та фреймворк:*** PHP 8.4 + Symfony 7.4.
- ***База даних:*** PostgreSQL 16.
- ***Веб-сервер:*** Nginx
- ***Контейнеризація:*** Docker.

4

Аналіз вразливостей базової архітектури

- ***Плоска мережа:*** Усі контейнери знаходяться в одному сегменті без обмежень доступу.
- ***Надмірна експозиція портів:*** Прямий доступ до СУБД із зовнішніх мереж через порт 5432.
- ***Відсутність шифрування:*** Внутрішній трафік передається у відкритому вигляді.
- ***Головна загроза:*** За умови зламу вебсервера зловмисник отримує можливість горизонтального переміщення по мережі.

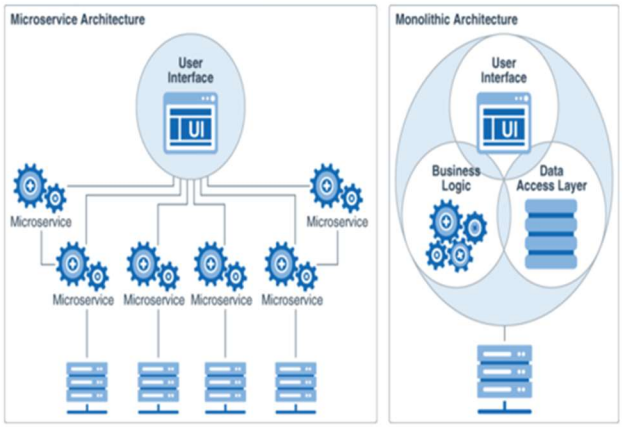


Рисунок 1.1 - Порівняльна структурна схема монолітної та мікросервісної архітектур веб-додатків

5

Результати аудиту

Інструменти:

- Nmap – сканує мережу на наявність відкритих портів.
- Tcpdump – перехоплювач мережевих пакетів у реальному часі.

Результат: Перехоплено SQL-запит авторизації у відкритому вигляді: *INSERT INTO users VALUES ('root','12345')*

Показник захищеності базової мережі *Ks = 0.0*.

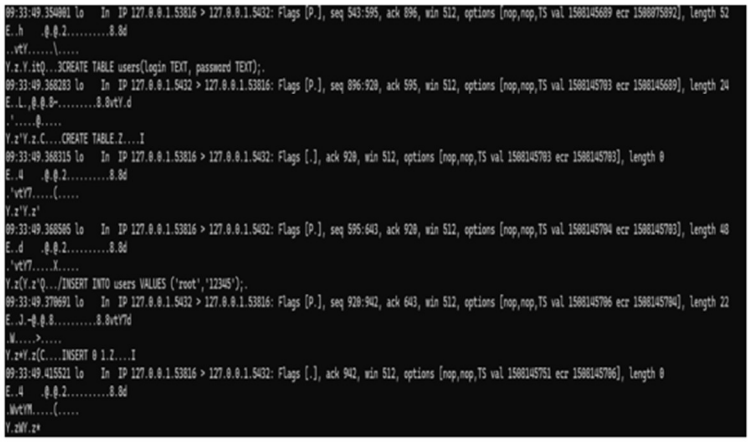


Рисунок 2.5 - Результати перехоплення внутрішнього трафіку за допомогою утиліти tcpdump: витік SQL-запитів у відкритому вигляді

6

Методика кількісної оцінки захищеності

Розрахунок інтегрального коефіцієнта захищеності здійснюється за формулою адитивної згортки:

$$K_s = \sum_{i=1}^{n} w_i \cdot x_i$$

K_s - інтегральний показник захищеності інфраструктури;
w_i - ваговий коефіцієнт і-го параметра безпеки;
x_i - бальна оцінка і-го параметра за результатами аудиту;
n - кількість оцінювання параметрів $n = 4$.

$K_s \leq 0.3$ – критично вразливе середовище.
$K_s \geq 0.7$ - високий рівень ізоляції захищеності.

Таблиця 1.2 - Номенклатура та критерії визначення коефіцієнтів мережевої безпеки

Параметр	Опис та умови присвоєння балів	Значення (x_i)	Вага (w_i)
Рівень ізоляції (I)	0.0 - єдина підмережа для всіх контейнерів (Default Bridge). 0.4 –поділ за функціональними групами (User-defined Bridge). 1.0 - повна ізоляція критичних зон (Internal Bridge).	0.0 - 1.0	0.40
Мінімізація експозиції (E)	0.0 - прямий доступ до всіх контейнерів. 0.5 - доступ лише через централізований реверс-проксі сервер Nginx. 1.0 - відсутність публічних портів, доступ тільки через VPN-шлюз.	0.0 - 1.0	0.25
Шифрування трафіку (C)	0.0 - передача даних у відкритому вигляді (HTTP, FTP, Telnet). 0.6 - використання TLS для зовнішніх та критичних внутрішніх запитів. 1.0 - суцільне шифрування East-West трафіку (mTLS / Service Mesh).	0.0 - 1.0	0.20
Автентифікація (A)	0.0 - довіра за рівнем IP-адреси (відсутність автентифікації). 0.5 - статичні токени або API-ключі у конфігураційних файлах. 1.0 - динамічна автентифікація та авторизація (OAuth2 / JWT / OIDC).	0.0 - 1.0	0.15

7

Візуалізація метрик захищеності

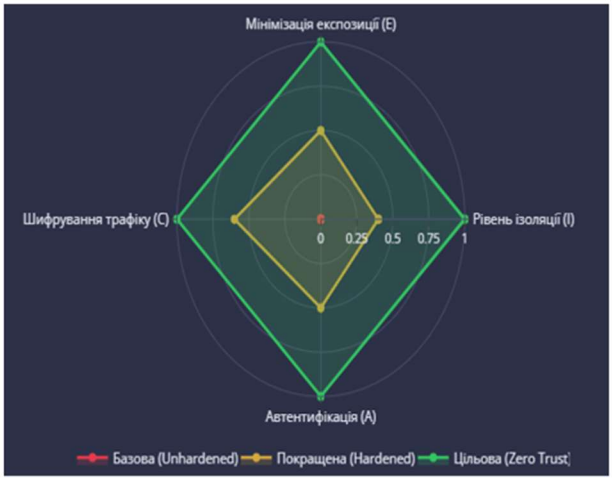


Рисунок 1.6 - Приклад візуалізації метрик мережевої безпеки: порівняння профілів захищеності різних топологій

- **Базова топологія :** *Ks = 0.0 (Критична вразивість)*
- **Покращена топологія:** *Ks= 0.48(Базовий захист)*
- **Цільова топологія:** *Ks = 1.0 (Максимальний захист)*

8

Цільова багатосегментна архітектура

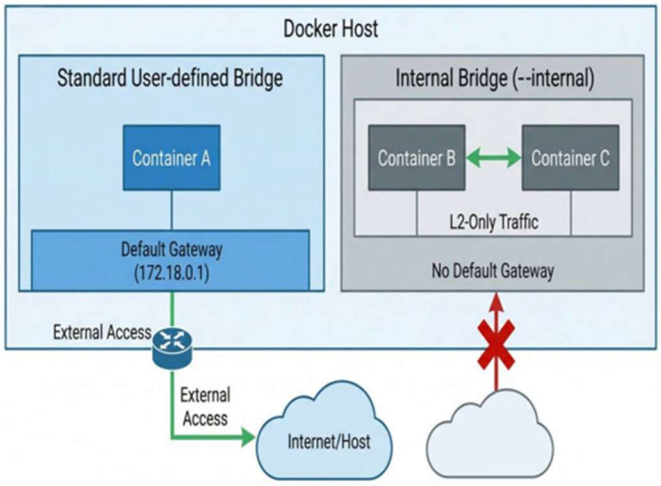


Рисунок 1.5 - Вплив параметра внутрішньої маршрутизації на таблицю логічних маршрутів віртуальної мережі Docker

- **Концепція Zero Trust:** Повна відмова від довіри за замовчуванням.
- **Розподіл на 3 ізольовані зони:** Public zone, application zone, data zone.
- **Ізоляція БД:** використання параметра internal: true для блокування зовнішнього трафіку.

9

Програмна реалізація: Docker Secrets

- Відмова від зберігання паролів у відкритих змінних середовища (*environment variables*).
- Впровадження механізму **docker secrets** для монтування конфіденційних даних безпосередньо в оперативну пам'ять контейнера.
- Використання ізольованих користувацьких мереж (user-defined bridge).
- Секрет монтується у /**run**/**secrets**/ через **tmpfs** та зберігається в оперативній пам'яті, не потрапляє на диск і не відображається в шарах образу
- Недоступний через docker inspect . Значення секрету приховане від інструментів інспекції, на відміну від environment variables
- Область дії - **лише призначений контейнер.** Навіть сусідні сервіси в одній мережі не мають до нього доступу.

```
php-fpm:
    build:
        context: .
        dockerfile: docker/php-fpm/Dockerfile
    volumes:
        - .:/var/www/project
    depends_on:
        - database
    env_file:
        - .env
    environment:
        DB_USER_FILE: /run/secrets/db_user
        DB_PASSWORD_FILE: /run/secrets/db_password
    networks:
        - frontend-net
        - backend-net
    secrets:
        - db_user
        - db_password
```

Рисунок 3.3 - Налаштування PHP-FPM як санкціонованого шлюзу між мережами

10

Програмна реалізація : Реверс-проксіювання сервера Nginx

- Nginx виступає як єдина санкціонована точка входу (Gateway).
- Приховування внутрішньої ip-топології від кінцевого користувача.
- Централізована фільтрація запитів до програми.
- Директива fastcgi_pass php-fpm:9000 - передача запитів до PHP-FPM через внутрішній DNS Docker, без прямого TCP-з'єднання з клієнтом
- Nginx підключений лише до **isolated_frontend**, що фізично унеможливлює маршрут до **isolated_backend** та бази даних, навіть у разі компрометації
- Захисні HTTP-заголовки (OWASP Top 10) - X-Frame-Options, X-Content-Type-Options, server_tokens off для приховування версії сервера

```
server {
    listen 80;
    server_name localhost;

    root /var/www/project/public;
    index index.php;

    error_log  /var/log/nginx/error.log;
    access_log /var/log/nginx/access.log;

    location / {
        try_files $uri $uri/ /index.php$is_args$args;
    }

    location ~ \.php$ {
        try_files $uri =404;
        fastcgi_split_path_info ^(.+\.php)(/.+)$;

        fastcgi_pass php-fpm:9000;

        fastcgi_index index.php;
        include fastcgi_params;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        fastcgi_param PATH_INFO $fastcgi_path_info;
    }
}
```

Рисунок 3.7 — Конфігурація проксіювання FastCGI запитів до ізольованого вузла PHP-FPM

11

Аналіз ефективності архітектури

Таблиця 3.4 - Розрахунок інтегрального коефіцієнта захищеності цільової архітектури

Оцінюваний параметр (x_i)	Опис впроваджених заходів	Бал (x_i)	Вага (w_i)	Результат ($w_i \times x_i$)
Мережева ізоляція (I)	Мікросегментація та апаратне блокування вихідного трафіку параметром internal: true	1.0	0.40	0.40
Експозиція сервісів (E)	Повна відсутність відкритих портів бази даних та внутрішніх служб на зовнішніх інтерфейсах	1.0	0.25	0.25
Захист трафіку (C)	Локалізація обміну даними (L2 isolation) без mTLS шифрування, але з інспекцією L7 на Nginx	0.7	0.20	0.14
Автентифікація (A)	Впровадження Docker Secrets (tmpfs), повна відмова від зберігання паролів у .env файлах	1.0	0.15	0.15

Таблиця 3.5 – Результати навантажувального тестування продуктивності архітектури

Метрика	Базова мережа	Цільова мережа	Зміна
Час відгуку (мс)	45.2	47.8	+2,6 мс
Пропускна можливість (зап/с)	2210	2150	-60 зап/с
Споживання RAM сервером Nginx (МБ)	120	145	+25 МБ

Кінцева ефективність архітектурного рішення обчислюється як відношення рівня захищеності до внесених затримок за такою формулою:

$$E_{arch} = \frac{K_s}{K_{ov}} = 0.89$$

Розрахунок інтегрального коефіцієнта захищеності здійснюється за формулою адитивної згортки:

$$K_s = \sum_{i=1}^{n} w_i \cdot x_i = 0.94$$

12

Практична значущість

- **Готові розроблені рішення:** Розроблені конфігураційні файли *docker-compose.yaml* та *default.conf* для Nginx є готовими інструментами для безпечного розгортання додатків у виробничих, розробницьких та тестованих середовищах.
- **Мінімізація поверхні атаки:** Впровадження апаратно ізольованих мереж та контроль точок входу, унеможливлюють несанкціонований прямий доступ до СУБД та інших критичних вузлів мережі.
- **Універсальна методика оцінки:** Запропонований математичний формат розрахунку інтегрального коефіцієнта ***Ks*** може бути використаний для аудиту мережевої безпеки інших контейнеризованих інфраструктур.
- **Оптимальний баланс між швидкість та безпекою:** Доведено можливість досягнення високого рівня захищеності ***Ks = 0.94*** за мінімальних експлуатаційних витрат і втрат 6% продуктивності.
- **Захист конфіденційних даних:** Використання механізму Docker Secrets забезпечує високий рівень захищеності облікових даних за рахунок їх монтування безпосередньо в оперативну пам'ять контейнера.

Висновки

Впровадження тризонної архітектури: Public, Application, Data та принципів Zero Trust дозволило радикально підвищити безпеку інфраструктури, усунувши ризики горизонтального переміщення зловмисника. Математична оцінка за розробленою методикою підтвердила зростання інтегрального коефіцієнта захищеності з критичного значення $Ks = 0.0$ до високого показника $Ks = 0.94$ при збереженні стабільної продуктивності системи.

Використання нативних драйверів Docker з параметром *internal:true*, механізму Docker Secrets та реверс-проксі Nginx забезпечує повну мережеву ізоляцію бази даних та приховування внутрішньої топології. Запропонований комплексний підхід робить контейнеризовані вебсервіси стійкими до сучасних векторів атак і повністю готовими до експлуатації у реальних виробничих середовищах.