КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 4 (32), 2026
ISSN 2663 – 4023
DOI 10.28925/2663-4023.2026.32.1192
УДК 004.056.5

Молнар Віталій Васильович
асистент кафедри захисту інформації
Національний Університет «Львівська Політехніка», Львів, Україна
: 0009-0001-3183-0117

Грушковський Олексій Олегович
студент кафедри захисту інформації
Національний Університет «Львівська Політехніка», Львів, Україна
: 0009-0007-3626-9780

ОЦІНЮВАННЯ ВПЛИВУ ZERO TRUST НА ЛОКАЛІЗАЦІЮ ІНЦИДЕНТІВ У КОНТЕЙНЕРИЗОВАНИХ МІКРОСЕРВІСНИХ СЕРЕДОВИЩАХ

Анотація. У контейнеризованих мікросервісних архітектурах компрометація одного сервісу може стати відправною точкою для бічного руху зловмисника вглиб інфраструктури, оскільки внутрішні взаємодії між сервісами часто зберігають надлишковий рівень довіри. Підхід Zero Trust вимагає взаємної автентифікації сервіс-сервіс, мінімально необхідних привілеїв, примусової мікросегментації та явного контролю кожного запиту, однак у публічно доступних рекомендаціях і практичних настановах усе ще бракує формалізованої методики кількісного порівняння стану системи «до» і «після» впровадження таких контролів. У роботі запропоновано модельне середовище з трьох сервісів, сценарій загрози типу «скомпрометований app-service» та три метрики оцінювання: радіус ураження R як частку досяжних об'єктів доступу, глибину бічного руху D як максимальну кількість послідовних переходів та операційну складність політик C як кількість артефактів безпеки, які потрібно підтримувати в актуальному стані. Обчислення на базовій моделі показали, що впровадження принципів Zero Trust знижує R з 0,93 до 0,21, скорочує D з 2 до 1 та підвищує C з 2 до 13. Для розширеної моделі з п'яти сервісів і нелінійною топологією встановлено аналогічну тенденцію: R знижується з 0,83 до 0,17, D з 2 до 1, а C зростає до 28, що свідчить про відтворюваність ефекту локалізації інциденту в складнішій архітектурі. Емпіричну валідацію проведено на локальному Kubernetes-кластері з Calico CNI: у базовому стані 17 з 17 тестів доступу виявилися успішними, тоді як після впровадження Zero Trust лише 6 з 17. Це підтверджує модельні передбачення та одночасно показує, що для повної реалізації обмежень на рівні окремих операцій потрібна L7-авторизація, яку зазвичай забезпечує service mesh. Отримані результати демонструють, що примусові механізми Zero Trust істотно звужують можливості поширення атаки, підвищують здатність архітектури до локалізації наслідків компрометації та можуть бути використані як основа для подальшого проєктування кількісних методів оцінювання безпеки хмарно-нативних систем. Водночас посилення захисту супроводжується відчутним зростанням операційних витрат на підтримку політик, сертифікатів, правил сегментації та суміжних безпекових артефактів.

Ключові слова: мікросервісна архітектура, Kubernetes, бічний рух, мікросегментація, DevSecOps, безпека контейнерів, емпірична валідація, NetworkPolicy.

ВСТУП

Сучасні хмарні обчислювальні середовища дедалі частіше будуються як сукупність контейнеризованих мікросервісів, які взаємодіють через стандартизовані інтерфейси та автоматизовані конвеєри постачання програмного коду [1]. Така архітектура забезпечує високу еластичність і спрощує масштабування, водночас

радикально змінюючи модель загроз: компрометація одного сервісу перестає бути локальною подією і може стати відправною точкою для подальшого розширення контролю зловмисника вглиб інфраструктури [2]. Це означає, що завдання безпеки не може обмежуватися виключно запобіганням первинному проникненню; необхідно також обмежувати масштаб наслідків уже здійсненого інциденту.

У традиційній периметровій парадигмі безпеки внутрішній трафік у межах одного кластера розглядається як довірений «за замовчуванням» [3]. Припускається, що якщо компонент розгорнуто «всередині» інфраструктури, то йому дозволено напряму викликати інші сервіси, ділитися службовими секретами доступу та ініціювати внутрішні з'єднання без додаткової перевірки [4]. Це погано узгоджується з реальністю контейнеризованих застосунків, де кожен мікросервіс часто має власні технічні облікові дані з доступом до чутливих ресурсів [5]. У такому режимі компрометація одного сервісу може одразу надати зловмиснику можливість виконувати виклики до інших внутрішніх компонентів від імені легітимного сервісного облікового запису [4].

Цей механізм безпосередньо пов'язаний із явищем бічного руху (lateral movement) – здатністю зловмисника, який уже захопив один сервіс, послідовно розширювати свою присутність у межах середовища [2]. Актуальні огляди загроз показують, що типовий розвиток інциденту полягає не в миттєвому повному руйнуванні всієї системи, а саме у поступовому просуванні всередину через довірені канали між сервісами [6]. Огляд безпеки хмарно-нативних архітектур додатково підтверджує, що саме внутрішня комунікація між мікросервісами є пріоритетним вектором атак після початкового проникнення [7].

Відповіддю на цю категорію ризиків стала концепція Zero Trust. Її базовий принцип полягає в тому, що жоден компонент не повинен вважатися довіреним лише через своє розташування у «внутрішньому домені», і що кожен запит між сервісами має бути явно автентифікований і авторизований [8]. У прикладних термінах це означає: ізоляцію секретів, суворо визначені канали взаємодії, взаємну автентифікацію сервіс – сервіс та надання кожному сервісу лише мінімально необхідних привілеїв [9]. У галузевих моделях зрілості безпеки це трактується вже не як «опціональне посилення», а як вимога до операційно придатної архітектури [8].

Попри формальне закріплення Zero Trust на рівні урядових стратегій і галузевих настанов, у більшості практичних матеріалів ця модель все ще подається як перелік найкращих практик без формалізованої методики кількісного порівняння станів «до» і «після» впровадження контролів [9]. Проблема полягає в тому, що навіть за наявності вимог практично відсутня відтворювана процедура, яка б дозволяла вимірювати зміну конфігураційної стійкості після компрометації одного сервісу [9]. У більшості публікацій бракує усталеної схеми, яка б кількісно описувала: наскільки стискається простір дій зловмисника після первинного доступу; наскільки скорочується можливість бічного руху; наскільки зростає операційна складність підтримки політик [10].

У цій роботі пропонується саме така формалізація. Розглядається модельне середовище з трьома сервісами і аналізуються два стани архітектури: початковий стан без примусових обмежень довіри та стан із впровадженими принципами Zero Trust [9]. Вводяться три метрики: радіус ураження, глибина бічного руху та операційна складність політик. На відміну від існуючих якісних рекомендацій, ці метрики обчислюються на конкретній моделі й подаються у вигляді числових значень, що дозволяє перейти від декларативної тези «Zero Trust підвищує безпеку» до кількісно підкріпленого порівняння станів архітектури. Для верифікації модельних передбачень додатково проведено емпіричну валідацію на локальному Kubernetes-кластері з

відтворенням обох станів архітектури та виконанням тестів доступу зі скомпрометованого контейнера.

Постановка проблеми. Ключова наукова проблема цієї роботи полягає у відсутності формалізованих кількісних показників, які дозволяють об'єктивно оцінити здатність архітектури локалізувати наслідки одиничної компрометації сервісу. Типова заява «ми впровадили Zero Trust» не дає відповіді на питання, наскільки саме обмежено простір дій зловмисника після захоплення конкретного компонента.

У контексті цієї проблеми виокремлюються три параметри. Перший – радіус ураження – сукупність сервісів, категорій даних та технічних повноважень, до яких зловмисник отримує доступ з точки компрометації [12]. Другий – глибина бічного руху – максимальна кількість послідовних кроків просування зловмисника без отримання адміністративного контролю над платформою [15]. Третій – операційна складність політик – вартість підтримки режиму мінімальної довіри: кількість мережевих правил, винятків доступу, ізольованих секретів та контурів авторизації [22]. Цей параметр визначає, чи є модель безпеки не лише теоретично бажаною, а й операційно утримуваною [23].

Аналіз останніх досліджень і публікацій. Перед формулюванням власного методичного апарату доцільно проаналізувати існуючі підходи до кількісної оцінки безпеки та пояснити, чому вони не вирішують поставленої задачі повною мірою. Найбільш поширеною системою кількісної оцінки вразливостей є CVSS (Common Vulnerability Scoring System) [18]. CVSS оцінює серйозність окремих вразливостей за шкалою від 0 до 10, враховуючи вектор атаки, складність експлуатації та вплив на конфіденційність, цілісність і доступність. Проте CVSS оцінює саму вразливість ізольовано, а не наслідки її експлуатації у контексті конкретної архітектурної конфігурації. Оцінка 9.8 для вразливості в app-service нічого не говорить про те, чи може зловмисник після її експлуатації дістатися до data-service або інших компонентів.

Методологія моделювання загроз STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege) забезпечує систематичну класифікацію типів загроз для окремих компонентів системи [19]. Однак STRIDE є якісним фреймворком, що ідентифікує категорії загроз, але не дає числових показників для порівняння двох конфігурацій одного й того самого середовища.

Підходи на основі графів атак (attack graphs) є найбільш близькими до запропонованого методу. У цих моделях вузли представляють стани системи або вразливості, а ребра – можливі переходи між ними; аналіз шляхів у графі дозволяє оцінити досяжність цілей зловмисником [20]. Проте класичні графи атак зазвичай моделюють мережеву інфраструктуру на рівні хостів і вразливостей CVE, а не на рівні мікросервісних взаємодій із сервісними ідентичностями, mTLS-автентифікацією та дрібнозернистою авторизацією запитів. Крім того, вони рідко враховують операційну складність підтримки контролів як окремий вимірюваний параметр.

Фреймворк BeyondCorp від Google [21] є відомою промисловою реалізацією Zero Trust, яка усуває поняття внутрішньої довіреної мережі та переносить контроль доступу на рівень ідентичності й контексту запиту. BeyondCorp описує архітектурні принципи й інженерний досвід впровадження, але не формалізує метрики для кількісного порівняння станів системи «до» і «після» впровадження.

Таким чином, існуючі підходи або оцінюють вразливості ізольовано (CVSS), або класифікують загрози якісно (STRIDE), або моделюють атаки на рівні мережевих хостів без урахування специфіки мікросервісних середовищ (attack graphs), або описують впровадження без формалізації ефекту (BeyondCorp). Запропонований у цій

роботі підхід адаптує ідею графової моделі доступу до контексту мікросервісної архітектури з mTLS, service mesh та політиками авторизації рівня окремого запиту, при цьому вводячи показник операційної складності як необхідний компонент оцінки.

Мета статті. Метою дослідження є розробка та обґрунтування методики кількісної оцінки впливу принципів Zero Trust на локалізацію наслідків компрометації одного мікросервісу [9], [11]. Завдання полягає у формуванні відтворюваної моделі оцінювання, яка дозволяє дати відповідь на конкретне питання: наскільки впровадження примусових політик доступу, сегментації трафіку та ізоляції секретів звужує простір дії зловмисника після захоплення одного сервісу. Для цього пропонується модельне середовище з трьох сервісів: frontend (сервіс зовнішнього інтерфейсу), app-service (сервіс бізнес-логіки) та data-service (сервіс доступу до даних). Використовується сценарій, у якому саме app-service вважається захопленим зловмисником. Далі для одного й того самого сценарію порівнюються два стани середовища з опорою на уніфіковані метрики.

Робоча гіпотеза: якщо принципи Zero Trust реалізовані примусово (взаємна автентифікація, жорстко обмежені мережеві шляхи, мінімальні привілеї, ізоляція секретів, контроль постачання артефактів), тоді навіть після повної компрометації одного сервісу зловмисник залишається локально небезпечним, але не здатним системно масштабувати атаку [9], [11]. Ця гіпотеза перевіряється через формальне визначення та порівняння метрик для двох контрольованих станів середовища.

Для досягнення мети дослідження виконуються такі завдання:

1. Визначити систему метрик оцінювання впливу принципів Zero Trust на локалізацію наслідків компрометації одного мікросервісу.
2. Обчислити та порівняти значення метрик для базового стану середовища і стану після впровадження Zero Trust.
3. Провести емпіричну валідацію модельних передбачень у контрольованому контейнеризованому середовищі.
4. Оцінити практичний компроміс між рівнем безпеки, керованістю політик і вартістю їх підтримки.

Теоретичні основи дослідження. Сучасні підходи до безпеки хмарно-нативних мікросервісних систем виходять із припущення, що компрометація окремого сервісу є штатним сценарієм, а не винятковою подією [3]. Такий підхід зумовлює зміщення акценту від суто превентивної моделі («не допустити проникнення») до моделі стримування («не дозволити вже наявному проникненню перерости у втрату цілісного контролю над середовищем») [6]. У стратегічних документах підкреслюється, що локальна компрометація окремого сервісу не повинна автоматично означати компрометацію всієї системи [11]. Технічні звіти з інцидентів демонструють, що типовий сценарій розвитку атаки полягає у послідовному бічному русі за рахунок використання вже виданих сервісу-жертві легітимних облікових даних [12].

У сучасній літературі безпека контейнеризованих мікросервісів трактується як властивість цілісного середовища виконання, а не як атрибут окремого контейнера [1]. Це середовище охоплює контроль ланцюга постачання програмного забезпечення, верифікацію походження артефактів, керування міжсервісним доступом, сегментацію мережевої взаємодії та дисципліну поводження із секретами [13]. Від розгортання очікується відповідність заздалегідь визначеним політикам ідентифікації, авторизації та розмежування доступу, які застосовуються автоматично [14].

Практика показує, що контейнеризований кластер більше не вважається «внутрішньо довіреним доменом», а розглядається як пріоритетна ціль [4]. Серед

найчастіших векторів компрометації описуються викрадення службових облікових даних, використання вразливих контейнерних образів, надмірно широкі ролі RBAC та розгортання привілейованих контейнерів [12]. CIS Kubernetes Benchmark фіксує конкретні вимоги: заборонити привілейовані контейнери, забезпечити незмінювану кореневу файлову систему, мати явну мережеву політику в кожному просторі імен та мінімізувати глобальні адміністраторські ролі [10].

Паралельно хмарні платформи впроваджують політики, які накладаються безпосередньо під час розгортання: деплоймент, що порушує правила доступу або сегментації, може бути заблоковане інфраструктурою без участі оператора [5]. У стратегічних документах така поведінка вже визначається як операційна норма [11].

Zero Trust у мікросервісних середовищах виходить із відмови від припущення про апріорну внутрішню довіру: сам факт розміщення двох сервісів в одному кластері не є підставою вважати їх взаємодію безпечною [3]. Кожен внутрішній виклик розглядається як потенційно ворожий, доки не буде встановлено достовірну ідентичність ініціатора та не буде підтверджено право на конкретну операцію [8]. Такий підхід прямо спрямований на обмеження бічного руху [15].

Одним з ключових механізмів є сервісна сітка (service mesh), яка забезпечує взаємну автентифікацію сервіс – сервіс, шифрування трафіку та примусове виконання політик доступу на рівні кожного запиту [16]. Другим базовим елементом є інтеграція контролю безпеки в конвеєр постачання (DevSecOps): забезпечення цілісності контейнерних образів, перевірка залежностей і підписування артефактів до моменту розгортання [13]. У державних і корпоративних моделях зрілості Zero Trust позиціонується як очікувана операційна властивість хмарної інфраструктури загального призначення [17]. Мінімізація довіри між сервісами, дрібнозерниста авторизація, недопущення спільних секретів і сегментація трафіку вважаються базовим станом архітектури [11].

Межі застосовності та обмеження. Робота має архітектурний характер і зосереджена на безпековій поведінці системи після компрометації одного сервісу, а не на продуктивності або надійності системи в нормальному режимі. Розглядається сценарій одиничної компрометації: зловмисник контролює app-service, але не має початкового адміністративного контролю над платформою оркестрації, не може змінювати мережеві політики або політики авторизації, не може підмінювати артефакти через CI/CD. Аналізується типовий випадок прикладного злому (application-level breach), а не повний прорив контрольної площини (control plane takeover) [5], [11].

Середовище моделюється як мінімальна конфігурація з трьох сервісів та однієї бази даних як окремого інфраструктурного вузла. У промислових середовищах абсолютні значення метрик будуть іншими, тому для демонстрації масштабованості методики у розділі 5 додатково наведено приклад із розширеною топологією з п’яти сервісів. Також вважається, що мережеві обмеження, взаємна автентифікація та політики авторизації виконуються так, як задекларовано [6], [22]. Дослідження не охоплює випадок контролю зловмисником контрольної площини або конвеєра постачання [11], [15], і не містить вимірів впливу політик на продуктивність [9], [13].

Методика дослідження. Модельне середовище складається з чотирьох компонентів: frontend, app-service, data-service та database. Frontend обробляє зовнішні запити і передає їх до app-service, який реалізує бізнес-логіку та в межах сценарію вважається скомпрометованим. Data-service виконує доступ до даних від імені прикладного рівня, а database є кінцевим сховищем. Така спрощена архітектура дає

змогу оцінити, як компрометація одного сервісу впливає на подальшу досяжність компонентів системи.

Сценарій загрози «скомпрометований app-service». У сценарії загрози припускається, що зловмисник отримав контроль над app-service. Це означає можливість виконувати довільний код у контейнері, читати доступні секрети та ініціювати мережеві запити до інших компонентів середовища. Оцінюється не первинне проникнення, а те, наскільки далі зловмисник може поширити контроль у межах системи.

Стан базової конфігурації («до Zero Trust»). У базовій конфігурації середовище побудоване на припущенні внутрішньої довіри: міжсервісні взаємодії не сегментовані жорстко, секрети ізольовані неповністю, а сервісні повноваження є ширшими, ніж це потрібно мінімально. За таких умов компрометація app-service потенційно відкриває доступ до суміжних сервісів і пов’язаних ресурсів.

Схематично взаємодію у базовому стані показано на рис. 1.

Рис. 1. Модель архітектури у стані «до Zero Trust», що демонструє системний характер ураження при компрометації одного сервісу

Конфігурація з примусовими контролями («після Zero Trust»). У конфігурації з примусовими контролями застосовано принципи Zero Trust: міжсервісна взаємодія обмежена явними політиками, секрети ізольовані, доступ зведено до мінімально необхідного, а мережеві шляхи сегментовані. У результаті навіть після компрометації app-service можливості подальшого просування зловмисника суттєво звужуються. Конфігурацію після застосування принципів Zero Trust наведено на рис. 2.

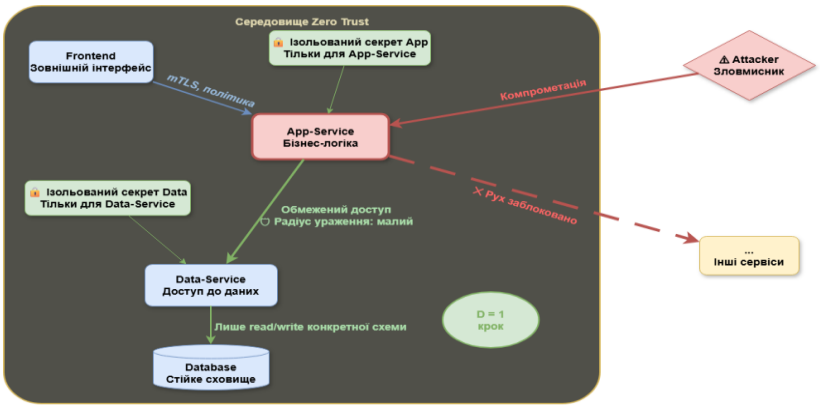

Рис. 2. Архітектурна модель стану «після Zero Trust», що демонструє локалізацію інциденту за допомогою примусових контролів

Методика обчислення метрик. Для кількісного порівняння станів вводяться три метрики: радіус ураження R, глибина бічного руху D та операційна складність політик C.

Модель доступу та базові означення. Нехай система складається з множини компонентів V та множини потенційних взаємодій E, що формують орієнтований граф G = (V,E). Вузол s ∈ V позначає скомпрометований компонент, у цій роботі – app-service. Доступ описується множиною об'єктів доступу A = {(r,o,c)}, де r – ресурс або ціль доступу, o – тип операції (read, write, delete, admin, call, exec, scan), c – категорія даних або контекст. Нехай A_allowed(s) – множина об'єктів доступу, досяжних і дозволених зі скомпрометованого вузла s з урахуванням чинних політик, а A_total – множина всіх об'єктів доступу, визначених у моделі системи.

Примітка щодо тривіальних об'єктів доступу. До A_total включено об'єкт (app-service, read, власні-секрети), який є тривіально досяжним у будь-якому стані, оскільки зловмисник контролює контейнер. Це означає, що R(s) > 0 за будь-яких умов ізоляції – нульовий радіус ураження неможливий за побудовою. Така властивість є свідомою: вона відображає реальність, де повна компрометація контейнера завжди дає доступ принаймні до його власних секретів. Альтернативний варіант – виключити тривіальні об'єкти з A_total – зменшив би знаменник, але не вплинув би на відносне порівняння R між станами.

Радіус ураження R. Нормована форма:

$$R(s) = |A\_allowed(s)| / |A\_total|, 0 \le R(s) \le 1 \quad (1)$$

Об'єкт доступу (r, o, c) включається до $A\_allowed(s)$ тоді й лише тоді, коли одночасно: існує шлях у графі від s до компонента, що надає ресурс r; мережеві політики дозволяють трафік; політики автентифікації/авторизації дозволяють операцію o.

Глибина бічного руху D. $Reach(s) = \{ v \in V : \text{існує дозволений шлях } s \to v \}$. Нехай $dist(s, v)$ – мінімальна кількість переходів у дозволеному шляху від s до v.

$$D(s) = \max dist(s,v) \text{ для всіх } v \in Reach(s) \quad (2)$$

$D(s) = 0$ означає повну ізоляцію; $D(s) = 1$ – лише прямі дозволені звернення; $D(s) \ge 2$ – багатокрокові траєкторії поширення. Під «переходом» розуміється переміщення точки контролю зловмисника до нового вузла графа (наприклад, від скомпрометованого app-service до data-service, а потім до database), а не просто розширення набору облікових даних без зміни вузла виконання.

Операційна складність політик C. Метрика C оцінює операційну складність підтримки політик безпеки через кількість незалежних артефактів, які необхідно створювати, перевіряти та супроводжувати. Вводяться такі величини: N_net – кількість мережевих політик або еквівалентних правил, N_auth – кількість артефактів автентифікації та авторизації, $N\_secrets$ – кількість унікальних секретів, $N\_exceptions$ – кількість винятків поза стандартною моделлю мінімальних привілеїв. Базова форма метрики:

$$C = N\_net + N\_auth + N\_secrets + N\_exceptions \quad (3)$$

У цій роботі використано рівні ваги для всіх складових, що забезпечує прозорість і відтворюваність базового порівняння.

Процедура порівняння. Для кожного стану (baseline та Zero Trust) за однаковими правилами обчислюються R(s), D(s) та C. Процедура обчислення A_allowed(s) для заданого стану виконується так: (1) побудова графа дозволених мережевих шляхів із вузла s з урахуванням чинних NetworkPolicy; (2) для кожного досяжного вузла – перевірка наявності автентифікаційних повноважень (mTLS-ідентичність, токени) та авторизаційних дозволів (AuthorizationPolicy, RBAC); (3) формування множини дозволених об'єктів (r, o, c) на основі досяжних вузлів та перевірених повноважень. Ефективність Zero Trust інтерпретується як зменшення R та D при допустимому зростанні C.

РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

Визначення множини об'єктів доступу A_total

Для модельного середовища з трьох сервісів та однієї бази даних формується вичерпний перелік об'єктів доступу $A\_total = \{(r, o, c)\}$, що існують у системі. Для кожного ресурсу визначено семантично можливі операції та категорії даних.

1. (data-service/API, read, бізнес-дані) – читання операційних бізнес-даних через API data-service.
2. (data-service/API, write, бізнес-дані) – запис операційних бізнес-даних через API data-service.
3. (data-service/API, read, усі-записи) – масове зчитування всіх записів (bulk read), що виходить за рамки штатного бізнес-запиту.
4. (data-service/API, write, усі-записи) – масова модифікація або перезапис усіх записів.
5. (data-service/API, delete, усі-записи) – деструктивне видалення даних.
6. (database, admin, конфігурація-БД) – адміністративний доступ до конфігурації бази даних як окремого інфраструктурного вузла.
7. (frontend, call, внутрішній-API) – виклик внутрішніх кінцевих точок frontend.
8. (frontend, read, секрети-середовища) – читання секретів, змонтованих у frontend.
9. (k8s-API, read, метадані-кластера) – читання списку подів, сервісів, конфігурацій.
10. (k8s-API, exec, робочі-навантаження) – виконання команд в інших подах кластера.
11. (мережа, scan, внутрішні-сервіси) – сканування внутрішньої мережі для виявлення доступних ендпоінтів.
12. (спільні-секрети, read, міжсервісні-токени) – читання повторно використаних (shared) секретів або токенів.
13. (інші-сервіси, call, довільні-ендпоінти) – виклик будь-яких інших внутрішніх сервісів поза визначеною архітектурою.
14. (app-service, read, власні-секрети) – читання секретів, легітимно змонтованих у app-service (тривіально доступний об'єкт, див. примітку у розділі 4.5.1).

Отже, $|A\_total| = 14$.

Стан до застосування принципів Zero Trust: обчислення метрик.

Радіус ураження R. Після компрометації app-service зловмисник має доступ до об'єктів, наведених у табл. 1.

Таблиця 1

Доступність об'єктів для скомпрометованого app-service у стані до Zero Trust

№	Об’єкт доступу (r, o, c)	Доступний	Обґрунтування
1	(data-service/API, read, бізнес-дані)	Так	Штатний доступ через наявні облікові дані
2	(data-service/API, write, бізнес-дані)	Так	Облікові дані не розрізняють типи операцій
3	(data-service/API, read, усі-записи)	Так	Відсутня дрібнозерниста авторизація
4	(data-service/API, write, усі-записи)	Так	Те саме
5	(data-service/API, delete, усі-записи)	Так	Широкі привілеї облікового запису
6	(database, admin, конфігурація-БД)	Так	Надмірно широкі облікові дані data-service містять admin-права до БД; після компрометації data-service зловмисник отримує прямий доступ до database
7	(frontend, call, внутрішній-API)	Так	Відсутня мережева сегментація
8	(frontend, read, секрети-середовища)	Так	Спільні секрети дублюються між сервісами
9	(k8s-API, read, метадані-кластера)	Так	Токен ServiceAccount за замовчуванням змонтовано
10	(k8s-API, exec, робочі-навантаження)	Ні	Потребує окремих привілеїв RBAC
11	(мережа, scan, внутрішні-сервіси)	Так	Відсутні NetworkPolicy
12	(спільні-секрети, read, міжсервісні-токени)	Так	Повторно використані секрети доступні в контейнері
13	(інші-сервіси, call, довільні-ендпоінти)	Так	Відсутня мережева сегментація
14	(app-service, read, власні-секрети)	Так	Тривіально – це власний контейнер

$A\_allowed(app-service) = \{1,2,3,4,5,6,7,8,9,11,12,13,14\}$, $|A\_allowed| = 13$. Отже, $R\_baseline = 13/14 \approx 0{,}93$.

Глибина бічного руху D. У базовому стані app-service має прямі дозволені шляхи до data-service, frontend, k8s-API та інших внутрішніх сервісів. Після компрометації data-service зловмисник може досягти database з використанням його облікових даних. Тому максимальна довжина ланцюга становить $D\_baseline = 2$ (app-service → data-service → database).

Операційна складність політик C. Для базового стану маємо $N\_net = 0$, $N\_auth = 1$, $N\_secrets = 1$, $N\_exceptions = 0$. Отже, $C\_baseline = 2$.

Стан після застосування принципів Zero Trust: обчислення метрик. Результати оцінки доступності об'єктів у стані після Zero Trust наведено в табл. 2.

Таблиця 2

Доступність об'єктів для скомпрометованого app-service у стані після Zero Trust

№	Об’єкт доступу (r, o, c)	Доступний?	Обґрунтування
1	(data-service/API, read, бізнес-дані)	Так	Дозволено AuthorizationPolicy для ролі app-service
2	(data-service/API, write, бізнес-дані)	Так	Дозволено в межах бізнес-функції
3	(data-service/API, read, усі-записи)	Ні	AuthorizationPolicy обмежує до конкретних ендпоінтів
4	(data-service/API, write, усі-записи)	Ні	Bulk-операції заблоковано
5	(data-service/API, delete, усі-записи)	Ні	Деструктивні операції заблоковано
6	(database, admin, конфігурація-БД)	Ні	Облікові дані data-service мають лише read/write на конкретну схему без admin-привілеїв; ескалація через database неможлива
7	(frontend, call, внутрішній-API)	Ні	NetworkPolicy блокує app-service → frontend

№	Об’єкт доступу (r, o, c)	Доступний?	Обґрунтування
8	(frontend, read, секрети-середовища)	Ні	Секрети ізольовано, спільних немає
9	(k8s-API, read, метадані-кластера)	Ні	Токен ServiceAccount не монтується (automountServiceAccountToken: false)
10	(k8s-API, exec, робочі-навантаження)	Ні	Те саме
11	(мережа, scan, внутрішні-сервіси)	Ні	NetworkPolicy default-deny обмежує вихідний трафік
12	(спільні-секрети, read, міжсервісні-токени)	Ні	Спільних секретів не існує
13	(інші-сервіси, call, довільні-ендпоінти)	Ні	NetworkPolicy блокує
14	(app-service, read, власні-секрети)	Так	Зловмисник контролює контейнер

$A\_allowed(app-service) = \{1, 2, 14\}$, $|A\_allowed| = 3$. Отже, $R\_zt = 3 / 14 \approx 0{,}21$

Глибина бічного руху D. Після впровадження Zero Trust з app-service дозволено лише звернення до data-service. Доступ до frontend, k8s-API, database та інших сервісів заблоковано мережевими політиками або обмеженням повноважень. Тому максимальна довжина ланцюга становить $D\_zt = 1$.

Операційна складність політик C. Для стану Zero Trust маємо $N_{net} = 3$, $N_{auth} = 6$, $N_{secrets} = 3$, $N_{exceptions} = 1$. Отже, $C_{zt} = 13$

Аналіз чутливості метрики R до гранулярності A_total. Результат R суттєво залежить від того, як саме побудовано A_total, яку гранулярність операцій та категорій даних обрано. Для перевірки стійкості висновку виконано аналіз чутливості за двома альтернативними сценаріями гранулярності.

У сценарії зниженої гранулярності операції з data-service/API об’єднано в два укрупнені об’єкти: (data-service/API, read-any, дані) та (data-service/API, write-any, дані), а scan і call об’єднано в один об’єкт (мережа, explore, внутрішні-ресурси). Це дає $|A\_total| = 9$. У базовому стані $|A\_allowed| = 8$, отже $R\_baseline = 0{,}89$. У стані Zero Trust $|A\_allowed| = 2$, отже $R\_zt = 0{,}22$. Відношення $R\_baseline / R\_zt \approx 4{,}0$.

У сценарії підвищеної гранулярності додано окремі об’єкти для кожної таблиці бази даних ($users, orders, logs — 3 \times \{read, write, delete\} = 9$ нових об’єктів), а також окремий об’єкт для кожного внутрішнього сервісу замість узагальненого «інші-сервіси». Це дає $|A\_total| = 24$. У базовому стані $|A\_allowed| = 22$, отже $R\_baseline = 0{,}92$. У стані Zero Trust $|A\_allowed| = 4$, отже $R\_zt = 0{,}17$. Відношення $R\_baseline / R\_zt \approx 5{,}4$.

Таким чином, при зміні $|A\_total|$ від 9 до 24 відношення $R\_baseline / R\_zt$ зберігається в діапазоні 4,0-5,4. Це підтверджує, що кратне зменшення R при впровадженні Zero Trust є стійким висновком, не залежним від конкретної гранулярності моделі.

Порівняльна таблиця метрик. Кількісне порівняння обох станів архітектури за трьома метриками наведено в табл. 3.

Таблиця 3

Порівняння метрик для двох станів архітектури

Метрика	До Zero Trust	Після Zero Trust	Зміна
Радіус ураження R	0,93 (13 з 14 об’єктів)	0,21 (3 з 14 об’єктів)	Зменшення у 4,4 рази
Глибина бічного руху D	2 кроки	1 крок	Зменшення вдвічі
Операційна складність C	2 артефакти	13 артефактів	Зростання у 6,5 разів

Емпірична валідація на локальному Kubernetes-кластері. Для перевірки адекватності модельних обчислень, представлених у розділах 5.2-5.3, було проведено емпіричну валідацію на локальному Kubernetes-кластері. Мета цього етапу – підтвердити або спростувати, що результати доступу зі скомпрометованого app-service у реальному середовищі відповідають передбаченням теоретичної моделі.

Середовище експерименту. Експеримент проведено на локальному Kubernetes-кластері, розгорнутому за допомогою kind (Kubernetes IN Docker) із підтримкою NetworkPolicy через Calico.

Архітектура модельного середовища відтворена з чотирьох компонентів: frontend (nginx:alpine), app-service (nginx:alpine з інструментами тестування curl, nmap, ncat), data-service (nginx:alpine з кінцевими точками /api/data, /api/data/bulk, /api/admin, /healthz) та database (postgres:15-alpine). Обидва стани конфігурації розгорнуто в окремих просторах імен – zt-baseline та zt-hardened – на тому самому кластері, що забезпечує ідентичність інфраструктурних умов.

Методика тестування. Тестування виконувалося з pod-а app-service за допомогою kubectl exec, що моделює повний контроль зловмисника над контейнером. Для кожного стану конфігурації виконано 17 тестів, згрупованих у шість категорій: міжсервісний API-доступ, доступ до бази даних, доступ до Kubernetes API, мережеве сканування, читання секретів і DNS-розвідка. Результат кожного тесту фіксувався як ACCESSIBLE або BLOCKED.

Усі маніфести Kubernetes, скрипти тестування та журнали виконання опубліковано у відкритому репозиторії [24].

Результати: стан до Zero Trust (baseline). У базовому стані (namespace zt-baseline) конфігурація містила: 0 об’єктів NetworkPolicy, 1 ServiceAccount за замовчуванням із автоматично змонтованим токеном, 1 спільний Secret (shared-credentials), змонтований у всі три сервіси як файл та як змінні середовища.

Результати наведено в табл. 4.

Таблиця 4

Результати тестування доступу зі скомпрометованого app-service у стані до Zero Trust (baseline)

№	Тест	Категорія	Результат
A1	curl data-service:8080/api/data	API-доступ	ACCESSIBLE
A2	curl data-service:8080/api/data/ bulk	API-доступ	ACCESSIBLE
A3	curl data-service:8080/api/admin	API-доступ	ACCESSIBLE
A4	curl frontend:8080/internal/config	API-доступ	ACCESSIBLE
A5	curl frontend:8080/	API-доступ	ACCESSIBLE
B1	nc -zv database 5432	БД-доступ	ACCESSIBLE
C1	cat SA token	K8s API	ACCESSIBLE
C2	curl k8s API /pods	K8s API	ACCESSIBLE
D1	nmap data-service	Сканування	ACCESSIBLE
D2	nmap frontend	Сканування	ACCESSIBLE
D3	nmap database	Сканування	ACCESSIBLE
E1	printenv API_KEY	Секрети	ACCESSIBLE
E2	printenv INTER_SERVICE_TOKEN	Секрети	ACCESSIBLE
E3	printenv DB_PASSWORD	Секрети	ACCESSIBLE
E4	ls /etc/secrets/	Секрети	ACCESSIBLE
E5	printenv DATA_SERVICE_TOKEN	Секрети	ACCESSIBLE
F1	nslookup всіх сервісів	DNS	ACCESSIBLE

Усі 17 тестів у базовому стані дали результат ACCESSIBLE. Це означає, що зі скомпрометованого app-service були доступні міжсервісні API, база даних, Kubernetes API, мережеве сканування та спільні секрети, тобто простір подальших дій зловмисника практично не був обмежений.

Результати: стан після Zero Trust. У стані Zero Trust (zt-hardened) було застосовано явні мережеві політики, окремі ServiceAccount без автоматичного монтування токенів та ізольовані секрети для кожного сервісу. Результати наведено в табл. 5.

Таблиця 5

Результати тестування доступу зі скомпрометованого app-service у стані після Zero Trust

№	Тест	Категорія	Результат
A1	curl data-service:8080/api/data	API-доступ	ACCESSIBLE
A2	curl data-service:8080/api/data/ bulk	API-доступ	ACCESSIBLE
A3	curl data-service:8080/api/admin	API-доступ	ACCESSIBLE
A4	curl frontend:8080/internal/config	API-доступ	BLOCKED
A5	curl frontend:8080/	API-доступ	BLOCKED
B1	nc -zv database 5432	БД-доступ	BLOCKED
C1	cat SA token	K8s API	BLOCKED
C2	curl k8s API /pods	K8s API	BLOCKED
D1	nmap data-service	Сканування	ACCESSIBLE
D2	nmap frontend	Сканування	BLOCKED
D3	nmap database	Сканування	BLOCKED
E1	printenv API_KEY	Секрети	BLOCKED
E2	printenv INTER_SERVICE_TOKEN	Секрети	BLOCKED
E3	printenv DB_PASSWORD	Секрети	BLOCKED
E4	ls /etc/secrets/	Секрети	BLOCKED
E5	printenv DATA_SERVICE_TOKEN	Секрети	ACCESSIBLE
F1	nslookup всіх сервісів	DNS	ACCESSIBLE

6 з 17 тестів показали результат ACCESSIBLE. Доступними залишилися: три ендпоінти data-service (A1-A3), оскільки NetworkPolicy дозволяє трафік від app-service до data-service на порту 8080; сканування data-service (D1), оскільки дозволений TCP-порт 8080 виявляється як відкритий; власний секрет app-service (E5); DNS-розв'язання (F1), оскільки egress-правило явно дозволяє порт 53 для коректної роботи сервісів.

Порівняння з модельними передбаченнями. Емпіричні тести загалом підтверджують напрям модельних висновків: після впровадження Zero Trust простір доступних дій суттєво зменшується. Водночас експериментальне значення $R\_zt = 0{,}35$ виявилося вищим за модельне $R\_zt = 0{,}21$, оскільки в експерименті не реалізовано повноцінну L7-авторизацію. NetworkPolicy на рівні Kubernetes обмежує трафік на L3/L4, але не розрізняє окремі HTTP-операції на одному порту. Отже, різниця між модельним та експериментальним результатами відображає внесок прикладного рівня авторизації у загальну ефективність Zero Trust.

Обговорення. Емпіричні результати підтверджують якісні передбачення теоретичної моделі: впровадження примусових контролів Zero Trust радикально скорочує простір дій зловмисника після компрометації app-service. У базовому стані

зловмисник мав доступ до всіх тестованих ресурсів (17 з 17), тоді як у стані Zero Trust – лише до 6 з 17, що відповідає зменшенню у 2,9 рази.

Водночас виявлено систематичне розходження з модельними передбаченнями: експериментальне $R\_zt = 0{,}35$ є вищим за модельне $R\_zt = 0{,}21$. Причина цього розходження є методологічно значущою. Модель (розділ 5.3) припускає, що об'єкти доступу (data-service/API, read, усі-записи), (data-service/API, write, усі-записи) та (data-service/API, delete, усі-записи) блокуються AuthorizationPolicy. В експерименті ці ендпоінти залишаються доступними, оскільки NetworkPolicy у Kubernetes працює виключно на рівні L3/L4 і не розрізняє HTTP-шляхи (/api/data та /api/admin є одним і тим самим потоком на порту 8080). Для обмеження на рівні окремих операцій необхідна L7-авторизація, наприклад, Istio AuthorizationPolicy.

Цей результат уточнює модель: без service mesh реальна ефективність Zero Trust є нижчою, ніж прогнозує модель, яка імпліцитно припускає наявність L7-контролю. Якщо з A_allowed виключити три ендпоінти data-service, що потребують L7-авторизації (A2, A3), та сканування (D1, що виявляє відкритий порт), експериментальне R_zt зменшується до 3/17 ≈ 0,18, що наближається до модельного значення 0,21. Це підтверджує, що модель коректно описує повністю реалізовану архітектуру Zero Trust (із service mesh), а різниця між модельним та експериментальним R кількісно характеризує внесок L7-контролю у загальну ефективність захисту.

Приклад масштабування на розширену топологію. Для перевірки застосовності методики розглянуто також розширену конфігурацію з п'яти сервісів, у якій A_total збільшується до 24 об'єктів доступу. У такій моделі базовий стан характеризується вищою досяжністю ресурсів ($R \approx 0{,}83$), тоді як після впровадження Zero Trust вона знижується до $R \approx 0{,}17$; глибина бічного руху зменшується з 2 до 1, а операційна складність зростає до 28. Це показує, що зі зростанням архітектурної складності методика зберігає придатність, а різниця між станами стає ще помітнішою.

Загрози валідності результатів. Для коректної інтерпретації результатів необхідно проаналізувати загрози їх валідності за трьома вимірами.

Конструктна валідність (construct validity). Метрика R залежить від визначення A_total, зокрема від обраної гранулярності операцій та категорій даних. Аналіз чутливості (розділ 5.4) показав, що відношення $R\_baseline / R\_zt$ залишається стійким при зміні $|A\_total|$ від 9 до 24, що знижує цей ризик. Метрика C як проксі-показник поверхні адміністрування не еквівалентна прямій вартості супроводу; калібрування зважених коефіцієнтів на емпіричних даних залишається відкритим питанням.

Внутрішня валідність (internal validity). Модель припускає, що політики доступу виконуються строго і без обхідних шляхів. У реальних середовищах тимчасово відкриті доступи «для налагодження», ручні обхідні шляхи або розширення прав «на час інциденту» можуть порушити цю умову [6, 22]. Результати коректні лише за умови дисциплінованого застосування політик.

Зовнішня валідність (external validity). Емпірична валідація (розділ 5.6) проведена на локальному kind-кластері з двома вузлами, що не відтворює повною мірою поведінку промислових кластерів із десятками вузлів, мережевою затримкою та навантаженням. Модель побудована на конфігурації з 3 (базова) та 5 (розширена) сервісів. У промислових середовищах із десятками або сотнями мікросервісів абсолютні значення R та C будуть іншими. Однак методика порівняння (формування A_total за фіксованим правилом, обчислення R та D для двох станів) залишається застосовною незалежно від кількості сервісів. Також модель перевірена лише для Kubernetes-специфічних механізмів (NetworkPolicy, ServiceAccount, Istio

AuthorizationPolicy); перенесення на інші оркестратори або service mesh (Nomad, Linkerd, Consul Connect) потребує адаптації словників ресурсів та артефактів.

ВИСНОВКИ ТА ПЕРСПЕКТИВИ ПОДАЛЬШИХ ДОСЛІДЖЕНЬ

У роботі запропоновано формалізований підхід до оцінювання стійкості хмарно-нативної мікросервісної архітектури у випадку локальної компрометації одного сервісу. На відміну від підходів, що зосереджуються переважно на переліку технічних заходів або ізольованій оцінці вразливостей [5, 6, 9, 13, 18-20], запропонований підхід орієнтований на кількісне порівняння наслідків компрометації між різними станами архітектури.

Для цього використано три метрики: радіус ураження R, глибина бічного руху D та операційна складність політик C. Їх застосування до модельного середовища з трьох сервісів показало, що після впровадження принципів Zero Trust значення R зменшується з 0,93 до 0,21, а D – з 2 до 1, тоді як C зростає з 2 до 13. Це підтверджує робочу гіпотезу: примусове розмежування доступу переводить інцидент із потенційно системного у локалізований, технічно обмежений стан [11].

Емпірична валідація на локальному Kubernetes-кластері підтвердила якісний напрям модельних передбачень: у базовому стані всі 17 тестів доступу були успішними, тоді як після впровадження контролів Zero Trust – лише 6 з 17. Водночас експеримент показав, що без L7-авторизації фактична ефективність захисту є нижчою, ніж прогнозує модель, що підкреслює важливість service mesh-рівня для повної реалізації Zero Trust.

Отримані результати показують, що підвищення стійкості архітектури досягається ціною зростання операційної складності. Отже, практичне впровадження Zero Trust слід розглядати як компроміс між локалізацією наслідків компрометації та витратами на супровід політик безпеки [6, 22].

До обмежень роботи належать орієнтація на сценарій прикладного злому без компрометації контрольної площини [5], неврахування впливу політик на продуктивність [9, 13], а також використання метрики C у незваженій формі. Крім того, результати перевірено на Kubernetes-специфічних механізмах, що потребує адаптації методики для інших оркестраторів.

Перспективами подальших досліджень є: експериментальна валідація моделі з L7-авторизацією; калібрування зваженої форми метрики C на емпіричних даних; оцінювання впливу політик Zero Trust на продуктивність; а також перевірка застосовності методики до складніших промислових конфігурацій і альтернативних платформ оркестрації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Souppaya, M., & Scarfone, K. (2017). *Application container security guide* (NIST Special Publication 800-190). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-190
2. MITRE Corporation. (2022). *MITRE ATT&CK for containers*. https://attack.mitre.org/matrices/enterprise/containers/
3. Rose, S., Borchert, O., Mitchell, S., & Connelly, S. (2020). *Zero trust architecture* (NIST Special Publication 800-207). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-207

4. National Security Agency, & Cybersecurity and Infrastructure Security Agency. (2022). *Kubernetes hardening guidance* (Version 1.2). https://media.defense.gov/2022/Aug/29/2003066362/-1/-1/0/CTR_KUBERNETES_HARDENING_GUIDANCE_1.2_20220829.PDF
5. Amazon Web Services. (2024). *Amazon EKS best practices guide for security*. https://docs.aws.amazon.com/eks/latest/best-practices/security.html
6. European Union Agency for Cybersecurity. (2023). *ENISA threat landscape 2023*. https://doi.org/10.2824/782573
7. Cloud Native Computing Foundation. (2022). *Cloud-native security whitepaper* (Version 2). https://github.com/cncf/tag-security/blob/main/community/resources/security-whitepaper/v2/cloud-native-security-whitepaper.md
8. Cybersecurity and Infrastructure Security Agency. (2023). *Zero trust maturity model* (Version 2.0). https://www.cisa.gov/resources-tools/resources/zero-trust-maturity-model
9. Chandramouli, R. (2022). *Implementation of DevSecOps for a microservices-based application with service mesh* (NIST Special Publication 800-204C). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-204C
10. Center for Internet Security. (2024). *CIS Kubernetes benchmark* (Version 1.8). https://www.cisecurity.org/benchmark/kubernetes
11. U.S. Department of Defense. (2022). *Department of Defense zero trust reference architecture* (Version 2.0). Office of the DoD Chief Information Officer. https://dodcio.defense.gov/Portals/0/Documents/Library/(U)ZT_RA_v2.0(U)_Sep22.pdf
12. Palo Alto Networks. (2024). *Unit 42 cloud threat report: Volume 7—Navigating the expanding attack surface*. https://www.paloaltonetworks.com/resources/research/unit-42-cloud-threat-report-volume-7
13. Chandramouli, R., Kautz, F., & Torres-Arias, S. (2023). *Strategies for the integration of software supply chain security in DevSecOps CI/CD pipelines* (NIST Special Publication 800-204D). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-204D
14. Chandramouli, R. (2019). *Security strategies for microservices-based application systems* (NIST Special Publication 800-204). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-204
15. IBM Security. (2024). *X-Force threat intelligence index 2024*. IBM Corporation. https://www.ibm.com/reports/threat-intelligence
16. Istio Project Authors. (2024). *Security best practices* (Version 1.22). https://istio.io/latest/docs/ops/best-practices/security/
17. U.S. Department of Defense Chief Information Officer. (2023). *DoD zero trust strategy and roadmap*. https://dodcio.defense.gov/Portals/0/Documents/Library/DoD-ZTStrategy.pdf
18. FIRST.Org, Inc. (2023). *Common vulnerability scoring system v4.0: Specification document*. https://www.first.org/cvss/v4-0/
19. Shostack, A. (2014). *Threat modeling: Designing for security*. John Wiley & Sons.
20. Jha, S., Sheyner, O., & Wing, J. (2002). Two formal analyses of attack graphs. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop* (pp. 49–63). IEEE. https://doi.org/10.1109/CSFW.2002.1021806
21. Ward, R., & Beyer, B. (2014). BeyondCorp: A new approach to enterprise security. *;login: The USENIX Magazine, 39*(6), 6–11. https://research.google/pubs/pub43231/
22. Red Hat, Inc. (2023). Understanding compliance. In *OpenShift container platform security and compliance documentation*. https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html/security_and_compliance/index
23. Cloud Security Alliance. (2023). *Cloud-native application protection platforms (CNAPP): Survey results and analysis*. https://cloudsecurityalliance.org/research/topics/cnapp
24. Molnar, V. (2026). *Zero trust empirical validation: Kubernetes manifests and test scripts* [Software repository]. GitHub. https://github.com/j9mbo/zero-trust-empirical-validation

КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 4 (32), 2026
ISSN 2663 – 4023

Vitalii Molnar
Postgraduate student, Assistant Professor, Department of Information Security
Lviv Polytechnic National University, Lviv, Ukraine
ORCID: 0009-0001-3183-0117
vitalii.v.molnar@lpnu.ua

Oleksii Hrushkovskyi
Student, Department of Information Security
Lviv Polytechnic National University, Lviv, Ukraine
ORCID: 0009-0007-3626-9780
oleksii.hrushkovskyi.kb.2022@lpnu.ua

ASSESSING THE IMPACT OF ZERO TRUST ON INCIDENT LOCALIZATION IN CONTAINERIZED MICROSERVICE ENVIRONMENTS

Abstract. In containerized microservice architectures, the compromise of a single service can become a starting point for lateral movement deeper into the infrastructure because internal service interactions often preserve an excessive level of implicit trust. The Zero Trust approach requires service-to-service mutual authentication, least privilege, enforced microsegmentation, and explicit verification of each request; however, publicly available guidance still lacks a formalized method for quantitative comparison of system states before and after such controls are introduced. This paper proposes a model environment with three services, a threat scenario based on a compromised app-service, and three evaluation metrics: impact radius R, defined as the share of reachable access objects; lateral movement depth D, defined as the maximum number of sequential transitions; and policy operational complexity C, defined as the number of security artifacts that must be maintained in an up-to-date state. Model-based calculations show that implementing Zero Trust reduces R from 0.93 to 0.21, decreases D from 2 to 1, and increases C from 2 to 13. For an extended five-service nonlinear topology, the same trend is observed: R decreases from 0.83 to 0.17, D again decreases from 2 to 1, and C rises to 28, indicating that the localization effect remains stable in a more complex architecture. Empirical validation on a local Kubernetes cluster with Calico CNI showed that 17 of 17 access tests were successful in the baseline state, whereas only 6 of 17 remained successful after Zero Trust hardening. These results confirm the model predictions and also demonstrate that complete restriction at the level of individual operations requires L7 authorization, typically provided through a service mesh. Overall, the findings indicate that enforced Zero Trust controls substantially reduce the propagation potential of an attack, improve the architecture's ability to localize the consequences of compromise, and provide a useful basis for further quantitative security assessment of cloud-native systems. At the same time, this security improvement is accompanied by a noticeable increase in the operational burden associated with maintaining policies, certificates, segmentation rules, and related security artifacts.

Keywords: microservice architecture; Kubernetes; lateral movement; microsegmentation; DevSecOps; container security; empirical validation; NetworkPolicy.

REFERENCES (TRANSLATED AND TRANSLITERATED)

1. Souppaya, M., & Scarfone, K. (2017). *Application container security guide* (NIST Special Publication 800-190). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-190
2. MITRE Corporation. (2022). *MITRE ATT&CK for containers*. https://attack.mitre.org/matrices/enterprise/containers/
3. Rose, S., Borchert, O., Mitchell, S., & Connelly, S. (2020). *Zero trust architecture* (NIST Special Publication 800-207). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-207
4. National Security Agency, & Cybersecurity and Infrastructure Security Agency. (2022). *Kubernetes hardening guidance* (Version 1.2). https://media.defense.gov/2022/Aug/29/2003066362/-1/-1/0/CTR_KUBERNETES_HARDENING_GUIDANCE_1.2_20220829.PDF

5. Amazon Web Services. (2024). *Amazon EKS best practices guide for security*. https://docs.aws.amazon.com/eks/latest/best-practices/security.html
6. European Union Agency for Cybersecurity. (2023). *ENISA threat landscape 2023*. https://doi.org/10.2824/782573
7. Cloud Native Computing Foundation. (2022). *Cloud-native security whitepaper* (Version 2). https://github.com/cncf/tag-security/blob/main/community/resources/security-whitepaper/v2/cloud-native-security-whitepaper.md
8. Cybersecurity and Infrastructure Security Agency. (2023). *Zero trust maturity model* (Version 2.0). https://www.cisa.gov/resources-tools/resources/zero-trust-maturity-model
9. Chandramouli, R. (2022). *Implementation of DevSecOps for a microservices-based application with service mesh* (NIST Special Publication 800-204C). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-204C
10. Center for Internet Security. (2024). *CIS Kubernetes benchmark* (Version 1.8). https://www.cisecurity.org/benchmark/kubernetes
11. U.S. Department of Defense. (2022). *Department of Defense zero trust reference architecture* (Version 2.0). Office of the DoD Chief Information Officer. https://dodcio.defense.gov/Portals/0/Documents/Library/(U)ZT_RA_v2.0(U)_Sep22.pdf
12. Palo Alto Networks. (2024). *Unit 42 cloud threat report: Volume 7—Navigating the expanding attack surface*. https://www.paloaltonetworks.com/resources/research/unit-42-cloud-threat-report-volume-7
13. Chandramouli, R., Kautz, F., & Torres-Arias, S. (2023). *Strategies for the integration of software supply chain security in DevSecOps CI/CD pipelines* (NIST Special Publication 800-204D). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-204D
14. Chandramouli, R. (2019). *Security strategies for microservices-based application systems* (NIST Special Publication 800-204). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-204
15. IBM Security. (2024). *X-Force threat intelligence index 2024*. IBM Corporation. https://www.ibm.com/reports/threat-intelligence
16. Istio Project Authors. (2024). *Security best practices* (Version 1.22). https://istio.io/latest/docs/ops/best-practices/security/
17. U.S. Department of Defense Chief Information Officer. (2023). *DoD zero trust strategy and roadmap*. https://dodcio.defense.gov/Portals/0/Documents/Library/DoD-ZTStrategy.pdf
18. FIRST.Org, Inc. (2023). *Common vulnerability scoring system v4.0: Specification document*. https://www.first.org/cvss/v4-0/
19. Shostack, A. (2014). *Threat modeling: Designing for security*. John Wiley & Sons.
20. Jha, S., Sheyner, O., & Wing, J. (2002). Two formal analyses of attack graphs. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop* (pp. 49–63). IEEE. https://doi.org/10.1109/CSFW.2002.1021806
21. Ward, R., & Beyer, B. (2014). BeyondCorp: A new approach to enterprise security. *;login: The USENIX Magazine, 39*(6), 6–11. https://research.google/pubs/pub43231/
22. Red Hat, Inc. (2023). Understanding compliance. In *OpenShift container platform security and compliance documentation*. https://docs.redhat.com/en/documentation/openshift_container_platform/4.14/html/security_and_compliance/index
23. Cloud Security Alliance. (2023). *Cloud-native application protection platforms (CNAPP): Survey results and analysis*. https://cloudsecurityalliance.org/research/topics/cnapp
24. Molnar, V. (2026). *Zero trust empirical validation: Kubernetes manifests and test scripts* [Software repository]. GitHub. https://github.com/j9mbo/zero-trust-empirical-validation

Отримано редакцією журналу / Received: 14.01.26
Прорецензовано / Revised: 30.01.26
Схвалено до друку / Accepted: 26.03.26