Міністерство освіти і науки України

Харківський національний університет імені В.Н. Каразіна

Навчально-науковий інститут комп'ютерних наук та штучного інтелекту

Спеціальність 125 «Кібербезпека та захист інформації»

Освітня програма «Безпека інформаційних та комунікаційних систем»

«Допущено до захисту»

В.о. зав. кафедрою БІСТ

Марина ЄСІНА

_____

« » 2024 р.

Пояснювальна записка

до кваліфікаційної роботи магістра

на тему: «Аналіз безпеки веб-застосунків на основі різних архітектур мікросервісів»

оцінка « »

Голова ЕК

Олександр ЛЕМЕШКО _____

Керівник к. т. н. доцент каф. _____

Мелкозьорова О. М.

Рецензент д.т.н., с.н.с.

Толстолузька О. Г.

Виконавець : студент групи КБ-61

Гріщенко М.Р.

Харків – 2024

РЕФЕРАТ

Пояснювальна записка містить 72 сторінок, 4 таблиці, 55 джерела.

Метою дипломної роботи є аналіз досвіду та методів забезпечення безпеки веб-застосунків, побудованих на основі мікросервісних архітектур, з урахуванням сучасних загроз та інструментів захисту.

Об’єкт дослідження: безпека веб-застосунків, що функціонують на основі мікросервісної архітектури.

Предмет дослідження: методи, підходи та технології забезпечення безпеки веб-застосунків, побудованих на мікросервісних архітектурах.

Основними методами дослідження є аналіз сучасних підходів до забезпечення безпеки в мікросервісних архітектурах, порівняння засобів і стандартів для забезпечення захисту веб-застосунків.

У роботі проведено аналіз основних характеристик мікросервісної архітектури, її переваг та викликів у контексті безпеки. Розглянуто типові загрози та уразливості, характерні для мікросервісних веб-застосунків, такі як атаки на API, DDoS-атаки, SQL-ін’єкції, атаки "man-in-the-middle" тощо. Досліджено існуючі методи захисту мікросервісних систем, включаючи аутентифікацію, авторизацію, шифрування даних, контроль доступу та моніторинг безпеки. Особливу увагу приділено порівнянню підходів до забезпечення безпеки у монолітних та мікросервісних архітектурах. Виявлено ключові відмінності в підходах до захисту даних, взаємодій між сервісами та користувачів. Розглянуто інструменти тестування безпеки, такі як Burp Suite, OWASP ZAP, Kubernetes, Istio, а також сучасні стандарти та протоколи, зокрема HTTPS, TLS, OAuth 2.0, JWT.

Результати дослідження можуть бути використані для вдосконалення процесів розробки веб-застосунків. Отримані рекомендації забезпечують

конкретні інструменти та стратегії для зменшення ризиків уразливостей у мікросервісних системах. Зокрема, їх можуть використовувати розробники для впровадження ефективних механізмів аутентифікації, шифрування та захисту API. Вони також можуть бути корисними для інженерів кібербезпеки при інтеграції інструментів моніторингу та автоматизованого тестування безпеки в процеси CI/CD. Запропоновані стратегії допоможуть організаціям забезпечити більшу стійкість веб-застосунків до сучасних загроз, таких як DDoS-атаки, компрометація API чи міжсервісні ін’єкції.

Ключові слова: ВЕБ-ЗАСТОСУНКИ, API, МІКРОСЕРВІСНА АРХІТЕКТУРА, РОЗПОДІЛЕНІ СИСТЕМИ, SQL-ІН’ЄКЦІ.

ABSTRACT

The explanatory note contains 72 pages, 4 tables, and 55 references.

The purpose of this thesis is to analyze the experience and methods of ensuring the security of web applications built on microservice architectures, considering modern threats and protection tools.

Object of the research: the security of web applications operating on microservice architectures.

Subject of the research: methods, approaches, and technologies for ensuring the security of web applications built on microservice architectures.

The primary research methods include analyzing modern approaches to ensuring security in microservice architectures and comparing tools and standards for protecting web applications.

The study analyzes the main characteristics of microservice architecture, its advantages, and challenges in the context of security. Typical threats and vulnerabilities associated with microservice-based web applications, such as API attacks, DDoS attacks, SQL injections, and man-in-the-middle attacks, are examined. Existing methods for protecting microservice systems, including authentication, authorization, data encryption, access control, and security monitoring, have been investigated. Particular attention is paid to comparing approaches to security in monolithic and microservice architectures. Key differences in approaches to data protection, inter-service interactions, and user security have been identified. The research also reviews security testing tools such as Burp Suite, OWASP ZAP, Kubernetes, and Istio, as well as modern standards and protocols, including HTTPS, TLS, OAuth 2.0, and JWT.

The results of the study can be applied to improve web application development processes. The recommendations provide specific tools and strategies

to reduce vulnerability risks in microservice systems. Developers can use these findings to implement effective authentication mechanisms, encryption methods, and API protection strategies. They may also be useful for cybersecurity engineers in integrating monitoring tools and automated security testing into CI/CD processes. The proposed strategies will help organizations enhance the resilience of web applications to modern threats such as DDoS attacks, API compromise, and inter-service injections.

Keywords: WEB APPLICATIONS, API, MICROSERVICE ARCHITECTURE, DISTRIBUTED SYSTEMS, SQL INJECTIONS.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

API – Application Programming Interface
CMS – Content Management System
DAST – Dynamic Application Security Testing
gRPC – gRPC Remote Procedure Call
HTTPS – HyperText Transfer Protocol Secure
IDS – Intrusion Detection Systems
IPS – Intrusion Prevention Systems
JWT – JSON Web Token
OWASP – Open Web Application Security Project
TLS – Transport Layer Security
XSS – Cross-Site Scripting

ВСТУП

У сучасну цифрову епоху веб-застосунки стали невід'ємною частиною повсякденного життя, забезпечуючи спілкування, торгівлю, освіту та розваги. Ці застосунки базуються на передових архітектурах, які гарантують їхню ефективність і масштабованість. Перехід від монолітних систем до мікросервісних архітектур є одним із таких етапів еволюції, що забезпечує гнучкість, розподілену розробку та відмовостійкість [1, 2]. Однак розподілений характер таких систем створює значні виклики у забезпеченні безпеки на різних рівнях і для численних компонентів. Веб-застосунки часто виступають головною точкою доступу до інформаційних систем організацій, що робить їх привабливими цілями для кіберзагроз [2]. Зловмисники використовують вразливості в логіці застосунків, протоколах комунікації або неправильних налаштуваннях, що призводить до порушення конфіденційності, цілісності або доступності. Мікросервісні архітектури, хоча й потужні, загострюють цю проблему, збільшуючи поверхню для атак і вимагаючи надійних стратегій захисту для міжсервісної взаємодії, обробки даних та автентифікації користувачів [2].

На сьогодні все більше зростає залежність між обсягом даних і веб-застосунками, забезпечення їхньої безпеки стає критично важливим. За даними OWASP, більшість атак на веб-застосунки спрямовані на недоліки в управлінні автентифікацією, доступом і захистом конфіденційних даних. Особливу небезпеку становлять атаки на мікросервісну архітектуру, яка є складнішою для захисту через розподілену природу взаємодії між сервісами. Зловмисники можуть використовувати уразливості внутрішньої комунікації між мікросервісами або недостатній захист API для поширення атак у межах системи [3]. Сучасні кіберзагрози мають також соціально-економічний вплив,

оскільки компрометація веб-застосунків може спричинити зрив критичних послуг, таких як медичні системи, фінансові платформи чи державні сервіси. Крім того, посилюється необхідність відповідності міжнародним стандартам, як-от ISO/IEC 27001, які визначають мінімальні вимоги до безпеки інформаційних систем. У цьому контексті, дослідження механізмів захисту веб-застосунків, побудованих на основі мікросервісів, набуває особливого значення, оскільки дозволяє знаходити ефективні способи попередження атак і підвищення стійкості систем до сучасних загроз [1].

Основною метою цього дослідження є аналіз викликів безпеки, пов’язаних із мікросервісними архітектурами, та розробка стратегій для їхнього подолання. Шляхом вивчення механізмів, таких як безпечні протоколи комунікації, фреймворки автентифікації та авторизації, а також стійкість до розподілених атак, це дослідження спрямоване на підвищення стійкості веб-застосунків до кіберзагроз [2].

Результати цього дослідження можуть бути використані для розробки та підтримки безпечних і масштабованих веб-застосунків. Вони є особливо актуальними для організацій, що впроваджують мікросервіси, щоб отримати їх гнучкість, мінімізуючи ризики безпеки. Крім того, отримані висновки можуть бути корисними для проєктування інструментів і фреймворків, які забезпечують безпечну взаємодію в розподілених системах, сприяючи розвитку галузі

1 ОГЛЯД СУЧАСНИХ ЗАГРОЗ У СФЕРІ БЕЗПЕКИ ВЕБ-ЗАСТОСУНКІВ

1.1 Огляд типових загроз безпеці веб-застосунків

Ми живемо в епоху, де цифрові технології проникають у всі сфери нашого життя. Інтернет став невід'ємною частиною повсякденності, а веб-застосунки стали основним інструментом для роботи, навчання та розваг. Зростання їхньої складності та функціональності призвело до появи нових можливостей, але водночас і нових загроз [1].

Паралельно зі зростанням залежності від цифрових технологій спостерігається стрімке зростання кіберзагроз. Зловмисники постійно розробляють нові методи атак, спрямовані на крадіжку конфіденційної інформації, порушення роботи систем та отримання незаконної вигоди. Веб-застосунки, як найбільш доступна точка входу в інформаційні системи організацій, стають основною мішенню для кібератак [2].

Зважаючи на вищезазначене, питання безпеки веб-застосунків набуває критичної важливості. Захист інформації, забезпечення безперебійної роботи систем та підтримка довіри користувачів є ключовими завданнями для організацій, що розробляють та експлуатують веб-додатки [4-6].

Перехід від монолітних архітектур до мікросервісних відкриває нові можливості для розробки масштабованих та гнучких систем. Однак, розподілена природа мікросервісів створює додаткові складнощі в забезпеченні безпеки. Збільшення кількості компонентів, незалежність їхнього розгортання та динамічність середовища призводять до появи нових векторів атак.

1.1.1 Класифікація загроз для веб-застосунків

Зростання складності веб-застосунків та їх інтеграція в різні сфери життя створюють нові можливості для кібератак. Щоб ефективно протистояти цим загрозам, необхідно глибоко розуміти їх природу та класифікацію [7].

Пасивні атаки є початковою стадією багатьох кіберзагроз, адже вони дозволяють зловмисникам непомітно зібрати велику кількість інформації про потенційну ціль [7]. Однією з ключових технік є розвідка, яка полягає у зборі відкритих даних з публічних джерел. Зловмисники шукають інформацію про компанію, її організаційну структуру, співробітників, технології, які використовуються в системах, а також деталі, що можуть стати корисними для майбутніх атак. Це включає пошук через соціальні мережі, публічні бази даних, форуми чи навіть офіційний веб-сайт компанії. Виявлені дані, наприклад, контактна інформація співробітників чи опис використовуваного програмного забезпечення, можуть стати основою для більш цілеспрямованих атак, таких як фішинг чи експлуатація вразливостей [8-9].

Ще одним популярним методом розвідки є сканування портів, яке дозволяє зловмисникам перевірити, які порти відкриті на сервері, а отже, і які служби працюють. Використовуючи спеціальні інструменти, як-от Nmap або Masscan, зловмисники визначають активні служби, їхні версії та можливі вразливості, пов’язані з ними [10].

Аналіз веб-сайту також відіграє важливу роль. Зловмисники можуть досліджувати структуру сайту, дізнаватись, які технології використовуються (CMS, мови програмування, фреймворки), версії цих компонентів і наявність відомих вразливостей. Виявлені слабкі місця, наприклад, застаріле програмне забезпечення, можуть бути використані для подальшого проникнення.

Окремо варто згадати про методи соціальної інженерії. Це інструменти впливу на співробітників компанії для отримання конфіденційної інформації. Зловмисники можуть обманом виманювати паролі, іншу важливу інформацію

або отримувати доступ до систем, використовуючи довіру співробітників до певних осіб або електронних листів [7-9]. Усе це робить етап розвідки надзвичайно важливим для успішності подальших атак, адже він надає зловмисникам ключові дані для планування наступних дій.

Мета пасивних атак полягає в отриманні максимально детальної початкової інформації про цільову систему або організацію, що дозволяє зловмисникам краще зрозуміти її структуру, слабкі місця та потенційні вразливості. Такий підхід дозволяє ретельно спланувати наступні дії, уникаючи зайвого ризику бути поміченим. Основне завдання пасивної атаки — зібрати всі можливі дані, включаючи технічну інформацію (наприклад, конфігурацію мережі, версії програмного забезпечення, відкриті порти) та організаційні дані (структуру компанії, контакти співробітників, політики безпеки тощо). Це створює базу для подальших активних атак, таких як експлуатація вразливостей або соціальна інженерія [11].

Ще одна важлива складова мети — виявлення вразливостей. Пасивні атаки спрямовані на пошук слабких місць у системі, наприклад, незахищених мережевих сегментів, застарілих версій програмного забезпечення або неправильно налаштованих служб. Ця інформація може бути використана для створення точкових, ефективних атак, спрямованих саме на ці недоліки [12].

Крім того, пасивні атаки сприяють профілюванню цілі, тобто створенню комплексного портрета організації. Це може включати деталі про технологічну інфраструктуру, типові робочі процеси, поведінку співробітників і навіть фінансову інформацію [13]. Такий підхід допомагає зловмисникам адаптувати свої дії, наприклад, створити персоналізовані фішингові листи або розробити специфічні шляхи проникнення.

Небезпека пасивних атак полягає в тому, що вони проводяться без активного втручання в систему, що робить їх практично невидимими для традиційних засобів захисту, таких як антивіруси чи системи виявлення атак.

У результаті організації можуть навіть не підозрювати, що їхні дані були зібрані, поки не станеться активна атака. Це робить пасивні атаки критично важливим етапом у ланцюгу складних кіберзагроз.

Захист від пасивних атак вимагає багаторівневого підходу, оскільки основна мета таких атак — збір інформації, яка може бути використана для подальших цілеспрямованих дій [14-16]. Перш за все, важливо мінімізувати обсяг відкритої інформації, яка може бути доступною у публічному просторі. Це включає ретельний аудит даних, які організація публікує у відкритих джерелах: веб-сайтах, соціальних мережах, пресрелізах тощо. Наприклад, деталі про інфраструктуру, версії програмного забезпечення або контактну інформацію технічного персоналу не повинні бути легкодоступними для зовнішніх осіб.

Не менш важливим є впровадження технічних засобів захисту мережі, таких як міжмережеві екрани (firewalls) та системи виявлення і запобігання вторгнень (IDS/IPS). Ці інструменти допомагають контролювати та аналізувати мережевий трафік, виявляти підозрілу активність і запобігати спробам сканування портів або пасивного перехоплення даних. Для ефективності ці системи повинні бути налаштовані відповідно до специфіки організації та регулярно оновлюватись [7-9].

Регулярне оновлення програмного забезпечення є ще одним ключовим заходом. Застарілі версії програм можуть містити відомі вразливості, які легко використовуються зловмисниками. Автоматизація процесів оновлення допоможе своєчасно закривати ці "дірки" в безпеці.

Моніторинг загроз і вразливостей — це безперервний процес, який включає як автоматизоване сканування системи за допомогою спеціалізованих інструментів, так і відстеження нових вразливостей, що з’являються у відкритих джерелах (наприклад, CVE-бази). Регулярні перевірки допоможуть

виявити потенційні ризики до того, як вони будуть використані зловмисниками.

Одним із найбільш недооцінених, але критично важливих заходів є навчання співробітників. Більшість пасивних атак використовують людський фактор, наприклад, недбале поводження з конфіденційною інформацією або слабку обізнаність у методах соціальної інженерії. Регулярні тренінги з інформаційної безпеки допоможуть співробітникам розпізнавати підозрілу активність, уникати фішингових атак і дотримуватись найкращих практик щодо збереження конфіденційності даних [7].

Поєднання технічних і організаційних заходів дозволяє значно знизити ризик успішного проведення пасивних атак і убезпечити систему від подальших загроз.

Активні атаки — це навмисні дії зловмисників, спрямовані на втручання в роботу системи або додатка для отримання несанкціонованого доступу, викрадення даних чи завдання шкоди. Одним із найпоширеніших типів таких атак є ін’єкції, що передбачають введення зловмисних даних у вхідні поля веб-додатків [17]. Наприклад, SQL-ін’єкції дозволяють зловмисникам вставляти спеціальні SQL-команди у форму вводу (наприклад, логін чи пошук), щоб маніпулювати базою даних. Це може призводити до викрадення конфіденційної інформації, видалення даних або навіть повного контролю над системою управління базами даних [17]. Аналогічно працюють NoSQL-ін’єкції, але вони орієнтовані на сучасні бази даних, як-от MongoDB, де використовується інший формат запитів. Ще один вид ін’єкцій — XSS (Cross-Site Scripting), який дозволяє впроваджувати шкідливі скрипти на веб-сторінку. У результаті ці скрипти виконуються в браузері користувача, що може призвести до викрадення сесій, облікових даних чи іншої чутливої інформації [15-17].

Атаки на автентифікацію та авторизацію є ще одним поширеним напрямом активних загроз. Наприклад, зловмисники можуть використовувати підбір паролів, застосовуючи автоматизовані інструменти для перебору можливих комбінацій або словникові атаки. Інший метод — перехоплення сесій, коли зловмисники викрадають автентифікаційні токени або файли cookie, щоб отримати доступ до облікового запису користувача. Фішинг також належить до цієї категорії: шляхом обману зловмисники змушують користувачів надати свої облікові дані або іншу конфіденційну інформацію, використовуючи підроблені веб-сайти чи електронні листи.

Не менш небезпечні атаки, що спрямовані на порушення цілісності даних. У таких випадках зловмисники можуть змінювати або підробляти дані в базі, що призводить до спотворення інформації або несанкціонованого доступу до ресурсів. Наприклад, маніпуляція фінансовими даними може використовуватися для шахрайства, а підробка прав доступу — для отримання привілейованого доступу.

Серед атак на стабільність системи особливу увагу привертає відмова в обслуговуванні (DoS/DDoS). У таких атаках зловмисники спрямовують великий обсяг трафіку на сервери, що призводить до їхнього перевантаження та недоступності для легітимних користувачів. Інший серйозний вид атак — переповнення буфера, коли надмірний обсяг введених даних порушує роботу програми, спричиняючи збій або відкриваючи можливість виконання шкідливого коду. Ці дії не тільки викликають тимчасові проблеми, але й можуть стати основою для подальших складніших атак.

Логічні помилки є однією з основних причин вразливостей у веб-застосунках, оскільки вони виникають через недоліки у проєктуванні або реалізації функціональності. Однією з найпоширеніших проблем є неправильна валідація даних. Це може проявлятися у відсутності перевірки даних, що вводяться користувачами, або у поверхневій перевірці, яка не

охоплює всі можливі сценарії. Наприклад, якщо веб-додаток дозволяє вводити дані без перевірки на спеціальні символи чи довжину, зловмисники можуть використати це для SQL-ін’єкцій, переповнення буфера або інших атак. Відсутність належної валідації також робить систему вразливою до впровадження шкідливого коду чи маніпуляції даними.

Ще однією серйозною проблемою є незахищені директорії, які дозволяють отримати доступ до конфіденційних файлів або ресурсів. Наприклад, якщо у системі відсутні належні обмеження доступу, зловмисники можуть переглядати чи завантажувати файли конфігурації, бази даних або навіть вихідний код додатка. Це створює критичну загрозу, оскільки отримана інформація може бути використана для глибшого аналізу системи, пошуку додаткових вразливостей або навіть для повного компрометування веб-застосунку.

Не менш небезпечним аспектом є відсутність управління помилками. Коли додаток генерує детальні повідомлення про помилки, зловмисники можуть використовувати їх для отримання додаткової інформації про структуру системи, технології, бази даних або сервери. Наприклад, повідомлення про помилку SQL може вказувати на тип бази даних і структуру запиту, що дозволяє краще підготувати атаку. Таким чином, незахищені помилки стають своєрідним «вікном» у внутрішній устрій системи, яке значно спрощує завдання атакуючим.

Логічні помилки часто недооцінюють, однак їхній вплив може бути масштабним. Вони не лише створюють прямі вразливості, але й відкривають зловмисникам шлях для складніших атак, використовуючи отриману інформацію. Це робить їх критичним фактором, на який необхідно звертати увагу під час розробки та тестування веб-додатків.

Атаки на API стають дедалі поширенішими через зростання популярності мікросервісної архітектури, де API є основним способом

взаємодії між сервісами. Однією з ключових загроз є неавторизований доступ, коли зловмисники отримують доступ до API без необхідних прав чи токенів автентифікації. Це може статися через слабкі механізми автентифікації, використання простих паролів або недостатньо захищені ключі доступу. У таких випадках зловмисники можуть взаємодіяти з API, отримуючи конфіденційну інформацію, змінюючи дані чи викликаючи небажані операції. Особливо небезпечним є доступ до адміністративних API, які дозволяють контролювати критичні аспекти системи.

Іншим серйозним видом атак є ін’єкції, коли у запити до API вводяться зловмисні дані. Це може включати SQL-ін’єкції, якщо API безпосередньо взаємодіє з базою даних, або NoSQL-ін’єкції для сучасних баз даних. Ін’єкції можуть також використовуватися для обходу захисних механізмів чи маніпуляції вхідними параметрами, що може призводити до несанкціонованих змін у системі або витоку даних. Наприклад, API для авторизації може бути зламаний шляхом підміни токена доступу або маніпуляції даними у JSON-запитах.

Особливо небезпечними для API є DDoS-атаки, коли зловмисники перевантажують API-сервери величезною кількістю запитів. Це призводить до того, що легітимні користувачі не можуть отримати доступ до сервісу через його недоступність. Оскільки API є ключовим елементом для функціонування багатьох сервісів, їхня недоступність може серйозно вплинути на роботу всієї системи. DDoS-атаки часто використовуються для саботажу або як частина комплексних атак, спрямованих на виведення з ладу бізнес-критичних додатків.

Захист API від таких атак вимагає впровадження багаторівневих механізмів безпеки, які включають строгий контроль доступу, фільтрацію запитів, обмеження швидкості (rate limiting) і регулярний аудит на предмет уразливостей. Однак, навіть із такими заходами, API залишаються

привабливою ціллю для атак через їхню критичну роль у сучасних веб-застосунках.

1.1.2 Загрози, пов’язані з архітектурою мікросервісів

Мікросервісна архітектура, хоча і має численні переваги, такі як гнучкість, масштабованість і можливість швидшого розгортання окремих компонентів, створює нові виклики в плані безпеки веб-застосунків [18]. Оскільки в цій архітектурі застосунок розділений на безліч дрібних, автономних сервісів, кожен з яких виконує певну функцію, це збільшує кількість точок входу для зловмисників, які можуть спробувати отримати доступ до однієї з таких служб. У традиційній монолітній архітектурі більшість компонентів зазвичай інтегровані в один великий блок, що полегшує управління безпекою, оскільки лише кілька точок доступу потребують захисту. В разі мікросервісів же кожен сервіс може мати власні API, базу даних і обмеження доступу, що вимагає більш детального контролю за безпекою на кожному рівні [1].

Зокрема, розподілений характер мікросервісів означає, що взаємодія між ними відбувається через мережу, а це відкриває додаткові можливості для атак. Зловмисники можуть скористатися вразливостями у внутрішній комунікації між сервісами, наприклад, перехоплюючи або підробляючи запити між ними. Якщо система не використовує належного шифрування або механізмів автентифікації для запитів, переданих через мережу, це може стати каналом для атак, таких як man-in-the-middle або повторне відправлення запитів (replay attacks). Іншими словами, зловмисники можуть використовувати слабкі місця у внутрішньому спілкуванні мікросервісів для проникнення в систему, навіть якщо зовнішній доступ до сервісів захищений [16-18].

Також важливим аспектом є управління конфіденційністю даних. У мікросервісній архітектурі кожен сервіс часто має свою окрему базу даних, і доступ до кожної з них може вимагати окремого контролю доступу. Без

централізованого механізму управління ідентифікацією та авторизацією, уразливості в одному з мікросервісів можуть призвести до витоку чи зміни конфіденційних даних. Крім того, якщо один з сервісів не належно обробляє помилки або виконує несанкціоновані запити до інших сервісів, це може стати вразливістю для всієї системи.

Моніторинг і керування безпекою в мікросервісній архітектурі також стає складнішим, оскільки потрібно відстежувати стан кожного сервісу, який може бути розташований на різних серверах чи навіть у різних хмарах. Це потребує інтеграції зібраних даних з усіх компонентів у централізовану систему моніторингу та логування. В іншому випадку, атака може залишитись непоміченою в одному з мікросервісів, а її наслідки поширюватимуться на всю систему.

З огляду на це, для забезпечення безпеки в мікросервісних архітектурах необхідно впроваджувати комплексний підхід до безпеки, який охоплює такі аспекти, як шифрування переданої інформації, централізовану ідентифікацію і авторизацію, детальне моніторингова систему та регулярні аудити безпеки.

Збільшення атакованої поверхні є одним з основних викликів, з якими стикаються організації при впровадженні мікросервісної архітектури. Оскільки кожен мікросервіс є автономною одиницею, яка виконує певну специфічну функцію, це створює безліч окремих точок входу, через які зловмисники можуть спробувати проникнути в систему. Якщо в традиційній монолітній архітектурі є одна основна точка доступу, то в мікросервісній архітектурі їх може бути десятки чи навіть сотні, що значно збільшує кількість можливих каналів для атак. Зловмисники можуть використовувати ці точки входу для намагання отримати доступ до системи, навіть якщо один з мікросервісів захищений належним чином [1].

Кожен мікросервіс має свою специфіку у вигляді інтерфейсів, API, баз даних або навіть різних підсистем. Ці компоненти можуть мати різні рівні

безпеки, а отже, більша кількість компонентів збільшує ймовірність того, що деякі з них матимуть вразливості. Наприклад, один мікросервіс може використовувати застаріле програмне забезпечення або мати некоректно налаштовані параметри безпеки, що надасть зловмиснику можливість знайти і використати ці слабкі місця для проникнення в систему. Більше мікросервісів також означає більше можливостей для помилок при налаштуванні доступу, зокрема, неправильна конфігурація політик авторизації або автентифікації, що дозволяє несанкціонованим користувачам отримати доступ до чутливої інформації.

Крім того, кожен окремий мікросервіс може використовувати різні технології та мови програмування, що створює додаткові виклики для забезпечення безпеки. Наприклад, один сервіс може працювати на базі даних SQL, інший — на NoSQL, один мікросервіс може використовувати REST API, а інший — GraphQL. Ці різноманітні технології мають свої специфічні вразливості, тому важливо ретельно перевіряти кожен компонент на наявність потенційних загроз і забезпечувати узгоджене застосування заходів безпеки для кожного мікросервісу.

Збільшення атакованої поверхні в мікросервісах вимагає більш складного підходу до безпеки. Потрібно впроваджувати багаторівневі заходи захисту для кожного мікросервісу, враховуючи його особливості та взаємодію з іншими компонентами. Такі заходи можуть включати моніторинг трафіку між мікросервісами, використання шифрування для забезпечення конфіденційності даних, а також регулярні аудитування кожного компонента на наявність вразливостей.

Складність управління доступом є однією з основних проблем безпеки, яка виникає при використанні мікросервісної архітектури. Оскільки система складається з великої кількості окремих мікросервісів, кожен з яких виконує свою функцію, забезпечення належного управління доступом стає набагато

складнішим завданням. Взаємодія між мікросервісами часто вимагає ретельного налаштування механізмів автентифікації та авторизації, оскільки кожен мікросервіс може мати свої власні вимоги до доступу та безпеки. Наприклад, один мікросервіс може обробляти чутливі фінансові дані, а інший — обслуговувати менш важливі функції, і тому потрібно чітко визначати, які користувачі або сервіси можуть отримати доступ до кожного з них. У разі неправильного налаштування доступу, зловмисники можуть використати недоліки в конфігурації для того, щоб проникнути в систему, отримати несанкціонований доступ до важливих даних або функціоналу, що створює серйозні загрози для безпеки.

Мікросервісна архітектура передбачає, що кожен мікросервіс має свою базу даних, API або інтерфейс взаємодії з іншими компонентами. Це збільшує кількість точок, де може бути налаштований доступ до ресурсу, і тому кожна точка потребує окремого управління правами доступу. Якщо авторизація та автентифікація не узгоджуються між всіма мікросервісами, існує ризик того, що сервіс отримає доступ до даних або функцій, до яких не має права. Наприклад, неправильно налаштовані дозволи можуть дозволити одному мікросервісу без обмежень звертатися до іншого сервісу або маніпулювати чутливими даними без відповідної перевірки.

Одним із найбільших викликів є забезпечення централізованої автентифікації та авторизації для всіх мікросервісів. У разі наявності великої кількості окремих компонентів важко підтримувати єдиний контроль доступу, що збільшує ймовірність помилок у налаштуваннях дозволів і створює уразливості, які можуть бути використані для атак. Один із підходів до вирішення цієї проблеми — це використання OAuth чи JWT (JSON Web Tokens) для забезпечення єдиного механізму авторизації, який дозволяє централізовано контролювати доступ до різних частин системи.

Важливою частиною є також моніторинг і аудит доступу. Завдяки великій кількості компонентів мікросервісної архітектури, важко відслідковувати кожну спробу доступу до системи. Без належного механізму моніторингу можна пропустити спроби несанкціонованого доступу або зловживання дозволами. Регулярний аудит та логування всіх запитів дозволяє вчасно виявити підозрілі активності і вжити заходів для забезпечення безпеки.

Загалом, складність управління доступом у мікросервісній архітектурі вимагає застосування ретельних і комплексних стратегій безпеки, зокрема чіткої політики доступу, централізованої автентифікації та авторизації, постійного моніторингу і аудитів для уникнення можливих загроз.

Розподіленість даних є ще однією складністю в контексті безпеки мікросервісної архітектури. Оскільки дані в такій системі можуть бути зберігані та оброблятися в різних мікросервісах, кожен з яких має свою базу даних або інший механізм зберігання, це створює додаткові труднощі у забезпеченні їх цілісності та конфіденційності. У традиційних монолітних системах дані, як правило, зберігаються в одному центральному сховищі, що спрощує застосування заходів безпеки, таких як шифрування або контроль доступу. В мікросервісах же, коли дані зберігаються в різних місцях, кожен мікросервіс має свої власні правила обробки та доступу до інформації. Це вимагає ретельного контролю за тим, як дані передаються між мікросервісами, а також як забезпечується їх захист при кожному доступі або зміні [1].

Один з основних викликів, пов’язаних з розподіленістю даних, — це забезпечення цілісності даних в межах всієї системи. Кожен мікросервіс може використовувати свою власну технологію зберігання даних, і якщо ця інформація буде пошкоджена чи змінена зловмисниками в одному з мікросервісів, це може привести до порушення роботи всієї системи. Наприклад, одна з вразливостей може виникнути через неправильне налаштування системи реплікації даних, що може призвести до втрати або

неповної передачі інформації. В такому випадку важливо використовувати методи верифікації даних, такі як контрольні суми або хешування, для того, щоб у разі потреби можна було перевірити цілісність даних.

Що стосується конфіденційності даних, то в розподіленій системі кожен мікросервіс може мати доступ до різних рівнів чутливої інформації. Це потребує додаткових заходів, таких як шифрування як під час зберігання даних (at rest), так і при їх передачі між сервісами (in transit). Оскільки дані передаються між різними мікросервісами, важливо використовувати захищені протоколи, наприклад, TLS (Transport Layer Security), для забезпечення безпечної передачі інформації між сервісами, щоб уникнути витоків даних під час їх обробки або пересилання.

Окрім того, автентифікація та авторизація в мікросервісах мають великий вплив на захист даних. Кожен мікросервіс має бути налаштований таким чином, щоб тільки авторизовані сервіси або користувачі могли доступати та змінювати певну частину даних [2-5]. Наприклад, важливо правильно налаштувати політики доступу до баз даних або API для кожного мікросервісу, щоб мінімізувати ризик несанкціонованого доступу до чутливих даних.

Враховуючи ці фактори, забезпечення захисту даних у розподіленому середовищі мікросервісів вимагає впровадження багаторівневої безпеки, яка охоплює як фізичний, так і логічний захист, зокрема, шифрування, автентифікацію, контроль доступу та постійний моніторинг [16]. Такий комплексний підхід дозволить мінімізувати ризики, пов'язані з розподіленістю даних, і забезпечити їх захист на всіх етапах обробки та зберігання.

Залежності між сервісами створюють додаткові складнощі для забезпечення безпеки в мікросервісній архітектурі. Оскільки мікросервіси часто взаємодіють один з одним через API або інші механізми, вразливість одного сервісу може стати потенційною загрозою для інших. Це створює ефект

доміно: якщо один з компонентів системи буде скомпрометований, це може призвести до порушення безпеки всіх інших, з якими він взаємодіє. Така ситуація ускладнює контроль за безпекою, оскільки важко виявити, яка конкретно частина системи стала вразливою та як це може вплинути на інші частини. Наприклад, якщо зловмисник знайде уразливість у сервісі, який обробляє аутентифікацію користувачів, він може використати цю уразливість для доступу до інших сервісів, які вимагають автентифікації. Таким чином, зловмисник може отримати доступ до різних частин системи через один вразливий компонент.

Проблеми з аудитом та моніторингом виникають через розподілену природу мікросервісів. Кожен мікросервіс може генерувати окремі логи, що ускладнює їх збір та аналіз для виявлення аномалій чи безпекових інцидентів. Відсутність єдиної точки контролю часто призводить до пропусків критичних подій, оскільки моніторинг кожного компонента здійснюється окремо, і немає можливості отримати повну картину того, що відбувається в системі в цілому. Це може створювати серйозні проблеми при реагуванні на атаки або інші загрози, оскільки важко відстежити, де саме сталася атака або аномалія. Для ефективного моніторингу в мікросервісах необхідно впроваджувати централізовану систему збору логів і використовувати інструменти для аналізу даних, які можуть виявляти аномалії на рівні всієї системи.

Ускладнення тестування на проникнення — це ще один важливий аспект безпеки мікросервісної архітектури [14]. Через велику кількість взаємодіючих компонентів повне тестування на проникнення стає складним завданням. Тестування повинно охоплювати всі шляхи взаємодії між сервісами та їх зовнішніми точками доступу, щоб виявити можливі уразливості на кожному етапі комунікації. Проблема ускладнюється тим, що необхідно враховувати не тільки вразливості на рівні окремих мікросервісів, але й на рівні їх взаємодії між собою. Для ефективного тестування необхідно створювати комплексні

сценарії, що відтворюють реальні умови атаки та враховують взаємозв'язки між всіма компонентами.

Проблеми з управлінням конфігурацією є критичним аспектом безпеки в мікросервісах. Кожен сервіс має свою конфігурацію, і для правильного функціонування системи необхідно забезпечити узгодженість цих конфігурацій. У великих системах, де багато мікросервісів, підтримка правильних налаштувань може бути складною і вимагати постійного моніторингу [13]. Помилки в конфігурації, такі як неправильні налаштування доступу або шляхи зберігання даних, можуть призвести до виникнення серйозних вразливостей, через які зловмисники можуть отримати несанкціонований доступ до системи або її компонентів. Для уникнення таких проблем потрібно впроваджувати автоматизовані засоби управління конфігурацією, що дозволяють відстежувати та підтримувати узгодженість конфігурацій усіх компонентів системи, а також мати чітку стратегію оновлень і перевірок налаштувань.

1.2 Стандарти та рекомендації з безпеки веб-застосунків

Для забезпечення високого рівня безпеки веб-застосунків важливо дотримуватися спеціально розроблених стандартів і рекомендацій, створених експертами в галузі кібербезпеки. Ці документи надають чіткі вказівки для виявлення, усунення та запобігання вразливостям, що можуть бути використані зловмисниками. Вони також забезпечують структурований підхід до організації безпеки, враховуючи всі етапи життєвого циклу веб-додатків: від проєктування до експлуатації [9-11].

Стандарти допомагають розробникам і адміністраторам ідентифікувати ризики, аналізувати їхній вплив на систему та впроваджувати відповідні заходи захисту. Вони сприяють створенню єдиної бази знань і кращих практик, що полегшує інтеграцію безпеки в процес розробки. Окрім технічних аспектів,

такі рекомендації враховують організаційні і процедурні заходи, що дозволяє забезпечити комплексний підхід до захисту.

Дотримання цих стандартів забезпечує узгодженість у впровадженні безпекових заходів та сприяє взаємодії між різними командами, що працюють над захистом веб-додатків. Вони також допомагають встановити мінімальні вимоги до безпеки, що мають виконуватися для захисту користувачів і їхніх даних. Це є основою для зменшення ризиків і підвищення довіри до веб-застосунків.

1.2.1 OWASP: основні ризики та методи їхньої мінімізації

OWASP визначає ключові ризики безпеки для веб-застосунків, які найчастіше стають причиною успішних атак. Серед них особливу увагу привертають вразливості, що виникають через ін'єкції [18]. Це ситуації, коли зловмисник вводить шкідливі команди у вхідні дані, що можуть бути виконані системою. Ін'єкції є критично небезпечними, оскільки дозволяють отримати несанкціонований доступ до баз даних або інших систем [19]. Для їхньої мінімізації необхідно застосовувати підготовлені запити, параметризовані SQL-запити та ретельно перевіряти всі дані, що вводяться.

Ще одним значним ризиком є порушення автентифікації та управління сесіями. Ця проблема виникає, коли механізми авторизації та автентифікації налаштовані неправильно, що дозволяє зловмисникам викрадати облікові дані, перехоплювати сесії або отримувати доступ до конфіденційних даних. Ефективними методами захисту є впровадження багатофакторної автентифікації, шифрування даних сесії та обмеження терміну дії сесій [20].

Витоки конфіденційних даних також входять до основних загроз. Недостатній захист інформації може призвести до її викрадення, що особливо небезпечно для фінансових або особистих даних користувачів. Для зменшення ризиків потрібно використовувати сучасні методи шифрування, захищати

передавання даних через HTTPS та регулярно оновлювати алгоритми шифрування.

Слабкі місця в управлінні доступом є ще одним серйозним ризиком. Відсутність чіткої політики доступу або неправильна її конфігурація дозволяє зловмисникам виконувати несанкціоновані дії, наприклад, отримувати доступ до функцій адміністратора. Мінімізувати ці загрози можна за рахунок принципу мінімальних привілеїв, регулярного перегляду прав доступу та впровадження чітких правил контролю доступу [18-20].

Для кожної з цих проблем OWASP пропонує практичні рекомендації, що допомагають інтегрувати безпеку в процес розробки та експлуатації веб-застосунків. Дотримання цих рекомендацій є критично важливим для забезпечення стійкості системи до сучасних загроз.

1.2.2 ISO/IEC 27001: застосування в контексті веб-безпеки

ISO/IEC 27001 — це міжнародний стандарт управління інформаційною безпекою (Information Security Management System, ISMS), який встановлює вимоги до створення, впровадження, підтримки та постійного вдосконалення системи управління безпекою інформації [21]. У контексті веб-безпеки цей стандарт є потужним інструментом, що допомагає організаціям забезпечувати захист веб-застосунків та даних, з якими вони працюють.

У веб-безпеці стандарт ISO/IEC 27001 використовується для визначення та зниження ризиків, пов’язаних із загрозами інформаційній безпеці. Він починається з оцінки ризиків: ідентифікації активів, які потребують захисту (наприклад, бази даних, веб-сервіси, облікові записи користувачів), виявлення потенційних загроз (таких як кібератаки, витоки даних або технічні збої) та визначення їхнього впливу [21, 22]. Це дозволяє побудувати чіткий план дій для захисту веб-додатків.

Стандарт також вимагає впровадження чітких політик безпеки, які охоплюють усі аспекти роботи з веб-застосунками, включно з управлінням

доступом, моніторингом мережевої активності, захистом даних під час передавання та зберігання, а також реагуванням на інциденти [21]. Наприклад, для захисту веб-додатків можуть бути впроваджені політики багатофакторної автентифікації, регулярного аудиту конфігурації серверів і програмного забезпечення, а також забезпечення шифрування критичних даних.

Однією з ключових вимог ISO/IEC 27001 є безперервне вдосконалення процесів безпеки. У контексті веб-додатків це означає регулярне тестування на проникнення, оновлення програмного забезпечення, моніторинг нових загроз і впровадження сучасних методів захисту [22]. Наприклад, організація може використовувати автоматизовані інструменти для сканування вразливостей у коді веб-додатків або аналізувати журнали подій для виявлення підозрілої активності.

ISO/IEC 27001 також акцентує увагу на важливості навчання співробітників. У сфері веб-безпеки це може включати тренінги для розробників щодо принципів захищеного кодування або навчання адміністраторів роботі з системами виявлення загроз.

Загалом, застосування ISO/IEC 27001 у веб-безпеці дозволяє організаціям створити комплексну систему захисту, яка враховує як технічні аспекти, так і управлінські процеси [22]. Це знижує ризики компрометації веб-додатків, підвищує довіру користувачів і допомагає забезпечити відповідність юридичним та регуляторним вимогам.

1.3. Вплив архітектурних рішень на безпеку веб-застосунків

Архітектура веб-застосунку є ключовим фактором, що визначає його стійкість до кібератак та здатність адаптуватися до нових викликів у сфері безпеки. Вона охоплює спосіб організації компонентів системи, взаємодії між ними та механізми захисту, інтегровані на кожному етапі роботи. Добре спроєктована архітектура забезпечує базу для впровадження передових

методів захисту, мінімізуючи ризик експлуатації вразливостей і знижуючи ймовірність реалізації атак.

Одна з головних переваг ретельно продуманої архітектури — це її здатність ізолювати компоненти системи, що обмежує можливість поширення атак. Наприклад, розподіл веб-застосунку на окремі шари (презентаційний, логічний, даних) дозволяє чітко розмежувати доступ до ресурсів і забезпечує різні рівні захисту для кожного шару [22]. Також важливим аспектом є вибір архітектурного підходу, такого як монолітна структура чи мікросервіси. Хоча мікросервісна архітектура надає гнучкість та масштабованість, вона також ускладнює забезпечення безпеки через більшу кількість точок доступу та необхідність захисту кожного мікросервісу окремо [1].

Неправильні архітектурні рішення, навпаки, можуть створювати значні загрози. Наприклад, централізація критичних компонентів без належного резервування може стати "єдиною точкою відмови", що підвищує ризик збоїв або атак [22]. Відсутність ізоляції компонентів може дозволити зловмисникам, отримавши доступ до одного елемента, впливати на інші частини системи. Неврахування масштабованості може призвести до перевантаження системи під час атак типу DoS/DDoS, якщо архітектура не здатна ефективно розподілити навантаження [4-7].

Правильно обрана архітектура також спрощує процес інтеграції засобів захисту. Вона дозволяє легко впроваджувати шифрування даних, системи автентифікації, моніторинг загроз та інші механізми безпеки. Наприклад, у добре структурованій системі легше забезпечити контроль доступу до ресурсів, проводити регулярне тестування на проникнення та впроваджувати автоматизовані засоби виявлення аномалі [21]. З іншого боку, неструктурована або надмірно складна архітектура може ускладнити ці процеси, створюючи додаткові ризики для безпек. Таким чином, архітектура веб-застосунку відіграє критично важливу роль у забезпеченні його безпеки. Вона визначає, наскільки

легко захищати систему від атак, і впливає на здатність організації ефективно реагувати на сучасні загрози. Інвестування в ретельне проектування архітектури на ранніх етапах розробки допомагає уникнути багатьох проблем у майбутньому, зберігаючи ресурси та підвищуючи рівень довіри користувачів до веб-додатку. Архітектура веб-застосунку має безпосередній вплив на рівень його безпеки, адже визначає, як організовано взаємодію компонентів і які заходи захисту можуть бути ефективно інтегровані. Наприклад, модульність і роз'єднання, що характерні для мікросервісної архітектури, дозволяють ізолювати компоненти, обмежуючи вплив компрометації одного з них на інші частини системи. Це знижує ризик каскадних збоїв, але водночас висуває високі вимоги до налаштування взаємодії між сервісами. Неправильна конфігурація таких зв’язків може створити нові вектори атак, що вимагає додаткової уваги до розробки механізмів авторизації, автентифікації та шифрування. Централізація або децентралізація компонентів також має свої переваги та виклики. Централізовані системи часто простіші у налаштуванні та моніторингу, але вони є більш вразливими до одноточкових відмов, коли компрометація або збій одного центрального вузла можуть паралізувати всю систему [4-6]. Децентралізовані системи складніше атакувати через відсутність єдиної точки доступу, але їхній захист вимагає більш комплексного підходу до управління та синхронізації елементів. Застосування готових фреймворків і бібліотек може спростити розробку та знизити витрати, але також створює додаткові ризики. Якщо такі компоненти не оновлюються регулярно, вони можуть містити відомі вразливості, які стають легким шляхом для атак. Оновлення та аудит цих рішень повинні стати невід’ємною частиною стратегії безпеки.

Мережева архітектура відіграє ключову роль у захисті системи від атак на рівні комунікацій. Використання фаєрволів, систем виявлення вторгнень (IDS/IPS) та правильна сегментація мережі можуть значно підвищити безпеку,

зменшуючи ризик перехоплення трафіку або несанкціонованого доступу до даних. Однак неправильна конфігурація цих механізмів може залишити систему вразливою [7].

Обрані технології також впливають на загальну безпеку архітектури. Використання застарілих рішень або програмного забезпечення без підтримки оновлень створює значні ризики для системи. Сучасні технології, які підтримують безпечні методи автентифікації, шифрування та управління доступом, допомагають мінімізувати ці ризики та забезпечити надійний захист веб-застосунку.

Побудова безпечної архітектури веб-застосунку є одним із ключових аспектів захисту від кібератак. Один із найважливіших принципів, який слід враховувати, — це принцип найменших привілеїв. Він передбачає, що кожному користувачу чи компоненту надаються лише ті права доступу, які потрібні для виконання їхніх завдань. Це зменшує ризик несанкціонованого доступу та зловживань, якщо одна з частин системи буде скомпрометована. Дотримання цього принципу особливо важливе в мікросервісних архітектурах, де різноманітні компоненти взаємодіють один з одним [9-15].

Регулярне оновлення програмного забезпечення є ще одним важливим заходом. Багато атак здійснюються через відомі вразливості в застарілих компонентах, тому всі елементи системи повинні бути оновлені до останніх стабільних версій. Це стосується не лише основного програмного забезпечення, але й бібліотек, фреймворків та інструментів, які використовуються під час розробки і підтримки веб-застосунку.

Необхідно також регулярно проводити аудити безпеки для оцінки загального стану системи. Такі аудити допомагають виявляти потенційні вразливості та виправляти їх до того, як вони будуть використані зловмисниками [16, 17]. Результати аудитів дозволяють створювати та оновлювати плани безпеки, які адаптуються до нових загроз та вимог системи.

Ще одним ключовим кроком є впровадження механізмів виявлення вторгнень. Системи IDS/IPS (Intrusion Detection/Prevention Systems) можуть моніторити активність у системі та виявляти підозрілу поведінку, яка може свідчити про спробу злому або несанкціонованого доступу. Це дозволяє швидко реагувати на інциденти і мінімізувати їхній вплив на систему.

Шифрування відіграє важливу роль у забезпеченні безпеки даних як у стані спокою (наприклад, у базах даних), так і під час їх передачі мережею. Надійне шифрування захищає інформацію навіть у разі її перехоплення, забезпечуючи конфіденційність і цілісність даних [17].

Якщо веб-застосунок використовує API для взаємодії з іншими сервісами, ці API мають бути ретельно захищені. Це включає використання безпечних протоколів (наприклад, HTTPS), аутентифікацію запитів, обмеження доступу за IP-адресами та моніторинг активності. Забезпечення безпеки API допомагає зменшити ризик несанкціонованого доступу до внутрішніх даних і функцій веб-застосунку. Поєднання цих підходів створює багаторівневу систему захисту, яка забезпечує високу стійкість веб-застосунку до потенційних атак і мінімізує наслідки можливих інцидентів.

Висновки до розділу: таким чином, ми провели систематичний аналіз сучасних загроз для безпеки веб-застосунків, зокрема тих, що впливають на мікросервісні архітектури [22]. Основними викликами є атаки на автентифікацію, ін’єкції, перехоплення даних і вразливості API, які посилюються через розподілену природу мікросервісів. Особливу увагу слід приділити проблемам відсутності централізованого контролю безпеки та множинності точок доступу, які створюють нові вектори атак. Стандарти, такі як OWASP і ISO/IEC 27001, є ключовими стандартами безпеки: перший пропонує практичні рекомендації для мінімізації ризиків, тоді як другий забезпечує системний підхід до управління інформаційною безпекою. Аналіз показав, що архітектурні рішення відіграють вирішальну роль у забезпеченні

безпеки, оскільки недостатньо продумане проєктування може сприяти виникненню нових загроз. Водночас впровадження принципів найменших привілеїв, шифрування даних і захисту API дозволяє суттєво підвищити стійкість систем до атак .

2 БЕЗПЕКА МІКРОСЕРВІСНИХ АРХІТЕКТУР

2.1 Основи мікросервісної архітектури.

Мікросервісна архітектура — це підхід до розробки програмного забезпечення, в якому система складається з невеликих, незалежних сервісів, кожен з яких виконує чітко визначену функцію. Ці сервіси працюють автономно, взаємодіючи один з одним через стандартизовані інтерфейси, зазвичай API. На відміну від монолітних систем, де всі компоненти інтегровані в єдину структуру, мікросервіси забезпечують гнучкість, масштабованість і легкість у розробці та обслуговуванні [23].

Ключовою характеристикою мікросервісної архітектури є децентралізація. Кожен мікросервіс має власну базу даних і бізнес-логіку, що дозволяє йому бути незалежним від інших частин системи. Такий підхід дає можливість розробникам зосереджуватися на окремих функціях, спрощуючи процес розробки та дозволяючи кільком командам працювати паралельно над різними сервісами. Це також сприяє швидшому впровадженню змін і оновлень, оскільки кожен сервіс можна змінювати або масштабувати незалежно від інших.

Мікросервіси спілкуються між собою через легковагові протоколи, такі як HTTP, gRPC чи повідомлення через черги. Це сприяє інтеграції різних технологій і мов програмування [23]. Наприклад, один сервіс може бути написаний на Python, а інший — на Java, що дає змогу використовувати найефективніші інструменти для кожного конкретного завдання.

Архітектура мікросервісів спрямована на підвищення стійкості до збоїв. У разі несправності одного сервісу решта системи зазвичай продовжує працювати. Це досягається завдяки ретельному проектуванню залежностей і використанню механізмів резервування або повторних спроб у разі помилок.

Однак мікросервісна архітектура також ускладнює управління системою [24]. Зростання кількості сервісів потребує впровадження оркестраційних інструментів, таких як Kubernetes, для управління розгортанням, масштабуванням і моніторингом [25]. Збільшення кількості точок взаємодії між сервісами також підвищує важливість забезпечення безпеки, моніторингу та узгодженості в розподіленому середовищі.

Отже, мікросервісна архітектура — це сучасний і потужний підхід до розробки веб-застосунків, який поєднує гнучкість і масштабованість із новими викликами в управлінні та безпеці. Вона є основою багатьох сучасних високонавантажених систем, але вимагає ретельного планування, щоб реалізувати всі її переваги без створення додаткових ризиків.

2.1.1 Переваги та виклики мікросервісів у контексті безпеки

Однією з головних переваг мікросервісної архітектури є ізоляція вразливостей. Кожен мікросервіс працює як окремий компонент з чіткими межами, що дозволяє обмежити вплив вразливості в одному з них на інші частини системи. Якщо зловмисник зуміє проникнути в один з мікросервісів, система в цілому не буде повністю скомпрометована, і інші сервіси продовжать працювати без значних порушень. Така ізоляція дає змогу зменшити потенційну шкоду від атак, оскільки зловмисник не зможе швидко поширити свою атаку на всі компоненти системи [26].

Швидке виявлення та усунення вразливостей є ще однією важливою перевагою. Мікросервіси зазвичай є невеликими, спеціалізованими компонентами, тому знайти і виправити вразливості в них набагато простіше, ніж у монолітних системах. Малі розміри і чітка спеціалізація кожного сервісу дозволяють проводити більш точкові перевірки безпеки, що значно підвищує ефективність виявлення слабких місць [23].

Модульне оновлення є ще однією важливою перевагою мікросервісів з точки зору безпеки. У разі необхідності оновлення або виправлення

уразливості можна оновлювати тільки конкретний мікросервіс, не впливаючи на всю систему. Це зменшує час і ризик, пов’язаний з оновленнями, оскільки можна поетапно вносити зміни і швидко усувати проблеми без необхідності перезапуску чи переписування всієї системи.

Зменшення атакованої поверхні є ще одним ключовим аспектом. Кожен мікросервіс, завдяки своїй невеликій функціональності, має відносно невелику атаковану поверхню. Це ускладнює зловмисникам пошук вразливостей, оскільки кожен сервіс має менше точок входу та обмежену кількість можливих векторів атак. У результаті, навіть при великій кількості мікросервісів в системі, кожен з них є окремим блоком, що зменшує загальний обсяг потенційних вразливостей, які можуть бути використані для атаки [25, 26].

Усі ці переваги мікросервісів сприяють більш ефективному управлінню безпекою, полегшуючи як виявлення, так і усунення вразливостей, знижуючи ймовірність серйозних атак і надаючи системі більшу гнучкість у випадку необхідності оновлень.

Одним із основних викликів мікросервісної архітектури є збільшення кількості атакованих точок. Кожен окремий мікросервіс створює потенційну точку входу для зловмисників, що значно розширює атаковану поверхню. Кількість сервісів у системі може бути великою, і кожен з них має свої вхідні порти, API та інтерфейси. Це збільшує кількість векторів атак, що робить систему вразливою до більшої кількості потенційних атак.

Складність управління доступом також є важливою проблемою. Кожен мікросервіс має свій набір дозволів і політик доступу, що ускладнює централізоване управління доступом до всієї системи. Коли зростає кількість сервісів, потрібно контролювати доступ до кожного з них, що створює ризики помилок у налаштуваннях доступу і в результаті може призвести до несанкціонованого доступу до важливих даних або функцій.

Розподіленість даних є ще одним викликом. У мікросервісній архітектурі дані можуть зберігатися в різних сервісах або навіть на різних фізичних серверів. Це ускладнює забезпечення цілісності та конфіденційності даних, оскільки для кожного сервісу потрібно реалізувати окремі засоби захисту, шифрування і механізми резервного копіювання [22-24]. Зрозуміло, що чим більше сервісів, тим складніше управляти даними і забезпечити їх безпеку.

Складність моніторингу та логування є ще одним важливим аспектом. Враховуючи велику кількість мікросервісів, кожен з яких генерує свої власні логи та метрики, збір, обробка та аналіз цих даних стає складним завданням. Важливо забезпечити централізовану систему моніторингу та логування для ефективного виявлення атак і аномалій, однак у мікросервісних середовищах це може бути технічно складно реалізувати.

Залежності між сервісами створюють додаткові труднощі у забезпеченні безпеки. Взаємодія між різними сервісами може призвести до ефекту "доміно", коли вразливість одного сервісу може вплинути на безпеку інших компонентів. Це збільшує загальний ризик для всієї системи, оскільки кожен сервіс стає не лише окремим елементом, а й частиною більшої взаємозалежної екосистеми.

Ускладнення тестування на проникнення також є значним викликом. Тестування безпеки стає більш складним і трудомістким, коли система складається з великої кількості взаємодіючих компонентів. Перевірити кожен компонент на наявність вразливостей, зокрема в межах складних взаємозв'язків між сервісами, потребує багато часу і ресурсів. Це може призвести до того, що деякі уразливості залишаться непоміченими.

Усі ці виклики створюють серйозні труднощі при впровадженні та підтримці безпеки в мікросервісних архітектурах, що вимагає розробки спеціальних стратегій і засобів захисту для кожного компонента системи.

Для мінімізації ризиків у мікросервісній архітектурі важливо впроваджувати комплексний підхід до безпеки на всіх етапах розробки та експлуатації системи.

Одним із перших кроків є ретельне планування безпеки з самого початку розробки. Безпека має бути інтегрована в процес створення системи, а не додана пізніше. Це включає визначення загальних вимог до безпеки, вибір відповідних технологій і методів захисту, а також розробку планів для реагування на інциденти та відновлення після атак.

Мікросегментація мережі є ефективним методом для обмеження поширення атак [27]. Розділення мережі на менші сегменти дозволяє ізолювати кожен мікросервіс від інших, зменшуючи таким чином можливість атаки на всю систему. Це допомагає мінімізувати масштаби пошкоджень у разі компрометації одного сервісу і знижує ймовірність поширення шкідливих програм чи зловмисного трафіку по мережі [27].

Строгі правила доступу є важливим елементом безпеки в мікросервісах. Для кожного сервісу потрібно чітко визначити, хто має доступ до яких ресурсів, і встановити відповідні політики авторизації. Це означає обмеження доступу до мінімально необхідних для виконання задач компонентів, що допомагає зменшити кількість потенційних точок атаки.

Шифрування даних — це критично важливий захід для захисту конфіденційної інформації. Дані повинні бути зашифровані як у стані спокою, так і під час транзиту між сервісами, щоб захистити їх від перехоплення чи несанкціонованого доступу. Шифрування зменшує ризик витоку даних навіть у випадку успішної атаки на окремі компоненти.

Регулярне оновлення програмного забезпечення є ще одним ключовим заходом для забезпечення безпеки. Всі компоненти системи повинні регулярно оновлюватися, щоб усунути відомі вразливості. Застаріле програмне

забезпечення може стати мішенню для зловмисників, тому автоматизоване оновлення і моніторинг версій допомагає знизити ці ризики.

Моніторинг та аналіз логів дають змогу виявляти аномалії та інциденти в реальному часі. Оскільки мікросервісна архітектура складається з багатьох компонентів, моніторинг кожного з них і збір логів з усіх точок доступу дозволяє оперативно реагувати на можливі загрози. Система моніторингу повинна бути налаштована так, щоб вчасно виявляти аномальну активність і дозволяти ефективно реагувати на інциденти.

Тестування на проникнення допомагає виявити вразливості на ранніх етапах. Регулярне проведення тестів на проникнення дає змогу перевірити безпеку всієї системи, виявити слабкі місця та своєчасно їх усунути. Це особливо важливо для мікросервісних систем, де кожен компонент має потенційні вразливості, які можуть бути використані для атаки.

Автоматизація процесів значно знижує ймовірність людської помилки, яка є однією з основних причин вразливостей у системах. Автоматизація завдань, таких як моніторинг, оновлення компонентів і управління доступом, допомагає знизити навантаження на адміністраторів і забезпечує більш стабільну та безпечну роботу системи.

Всі ці заходи у комплексі дозволяють значно знизити ризики, пов'язані з безпекою в мікросервісній архітектурі, та забезпечити більш ефективне захист системи від потенційних атак.

2.1.2 Розподілені системи: нові загрози для веб-застосунків

Перехід до розподілених систем, таких як мікросервіси та хмарні обчислення, безсумнівно, приніс безліч переваг для розробки веб-застосунків. Вони забезпечують гнучкість, масштабованість та ефективність, дозволяючи організаціям швидше адаптуватися до змін і знижувати витрати на інфраструктуру [28]. Однак одночасно з цими перевагами виникають нові

загрози для безпеки, які необхідно враховувати на кожному етапі розробки та експлуатації таких систем.

Однією з головних причин, чому розподілені системи створюють нові загрози, є збільшення атакованої поверхні. У розподілених системах кожен окремий компонент або мікросервіс стає потенційною точкою входу для зловмисника. Кожен додатковий сервіс чи система відкриває нові можливості для атаки, що ускладнює захист [28]. Якщо одна з точок доступу в системі вразлива, це може дати хакерам доступ до всього середовища, включаючи критичні дані та функціонал.

Додатковою складністю є управління великою кількістю компонентів, що є характерним для розподілених систем. Кожен мікросервіс або компонент може мати свої специфічні вимоги до безпеки, а їх взаємодія вимагає ретельного налаштування і моніторингу [28]. Координація безпеки між усіма компонентами стає складним завданням, оскільки кожен із них може мати різні рівні доступу, обмеження та загрози.

Ще одним важливим аспектом є розподіленість даних. У розподілених системах дані часто зберігаються в різних місцях, таких як різні мікросервіси або хмарні сервіси. Це створює додаткові труднощі у забезпеченні цілісності та конфіденційності даних. Наприклад, якщо дані переміщуються між різними точками системи, вони можуть стати вразливими для атак, таких як перехоплення або модифікація під час передачі.

Загрози, пов'язані з мережею, є ще одним важливим аспектом безпеки в розподілених системах. Оскільки різні компоненти системи часто взаємодіють через мережу, це збільшує ризик перехоплення даних під час їх передачі. Мережеві атаки, такі як атаки типу man-in-the-middle або DoS, можуть стати серйозною загрозою для системи, якщо не вжито належних заходів захисту [29].

Останнім викликом є складність виявлення інцидентів в таких системах. Розподілені системи генерують величезну кількість логів та подій, що робить виявлення аномалій і атак значно складнішим. Коли дані про події і активність зберігаються в різних місцях, важко побудувати єдину картину безпеки системи, що ускладнює оперативне реагування на інциденти та виявлення потенційних загроз [29].

Таким чином, розподілені системи, хоча й мають численні переваги, також створюють нові унікальні виклики для забезпечення безпеки веб-застосунків. Ці системи вимагають більш ретельного планування, комплексних заходів захисту та постійного моніторингу для виявлення та нейтралізації потенційних загроз [27-29].

Розподілені системи, завдяки своїй складній та багатокомпонентній природі, піддаються ряду специфічних загроз, які можуть серйозно вплинути на їхню безпеку. Ось деякі з типових загроз для таких систем:

Атаки на API: Однією з найбільших загроз для розподілених систем є атаки на інтерфейси програмування додатків (API). API є основними точками входу для взаємодії між різними мікросервісами, тому зловмисники можуть експлуатувати вразливості в API для несанкціонованого доступу до даних або функціоналу системи. Наприклад, API може бути недостатньо захищеним від SQL-ін'єкцій або міжсервісних атак, що дозволяють зловмисникам отримати контроль над додатками або витягнути конфіденційну інформацію [30-31].

Ін'єкції: Ін'єкції, такі як SQL-ін'єкції, XSS (Cross-Site Scripting) та інші типи, залишаються актуальними загрозами навіть у контексті розподілених систем [28]. Мікросервіси часто використовують різні бази даних і комунікаційні протоколи, що робить їх вразливими до подібних атак. Зловмисники можуть вставляти шкідливі запити або код, що маніпулює даними або змінює поведінку системи, що в свою чергу може привести до компрометації цілісності або конфіденційності даних.

Порушення автентифікації: Оскільки кожен мікросервіс часто має свою систему автентифікації, виникає ризик порушень цієї автентифікації. Якщо механізми автентифікації або авторизації є незахищеними або некоректно налаштованими, зловмисники можуть отримати несанкціонований доступ до ресурсів системи [28]. Наприклад, використання слабких паролів або незахищених токенів автентифікації може призвести до компрометації важливих частин системи.

Відмова в обслуговуванні (DoS/DDoS): Розподілені системи особливо вразливі до атак типу "відмова в обслуговуванні" (DoS) або "розподілена відмова в обслуговуванні" (DDoS), оскільки зловмисники можуть атакувати кілька компонентів одночасно, перевантажуючи їх і роблячи систему недоступною для легітимних користувачів [8, 32]. Зокрема, в умовах мікросервісної архітектури одна зламані точка входу може порушити роботу всього середовища, що значно ускладнює виявлення та усунення атак.

Бокові канали: Бокові канали - це техніки, за допомогою яких зловмисники можуть отримувати конфіденційну інформацію, не маючи прямого доступу до системи. У розподілених системах ці атаки можуть включати аналіз часу виконання операцій, споживання ресурсів, мережеві трафіки та інші непрямі канали. Через ці канали зловмисники можуть отримати чутливі дані або виявити вразливості, які важко зафіксувати традиційними методами моніторингу.

Внутрішні загрози: Не менш серйозними є внутрішні загрози, які виникають через співробітників або інсайдерів, що мають доступ до критичних компонентів системи. Внутрішні загрози можуть включати зловмисне використання своїх прав доступу для крадіжки або маніпуляції з даними, а також випадкові помилки через неуважність або недостатню кваліфікацію персоналу. Враховуючи величезну кількість взаємодіючих

компонентів у розподілених системах, внутрішні загрози можуть бути важко виявлені без належного моніторингу та аудиту.

Таким чином, розподілені системи створюють унікальні виклики в сфері безпеки, і для їх ефективного захисту необхідно впроваджувати комплексний підхід, що враховує всі типи загроз, починаючи від атак на API і закінчуючи внутрішніми загрозами.

Стратегії захисту розподілених систем спрямовані на зменшення ризиків та забезпечення безпеки через комплексний підхід до захисту даних і компонентів системи. Ось деякі основні стратегії:

Мікросегментація: Один із найбільш ефективних підходів для обмеження поширення атак у розподілених системах — це мікросегментація. Вона передбачає розділення мережі на менші, ізольовані сегменти. Це дозволяє обмежити рух зловмисників по мережі, навіть якщо вони отримають доступ до одного з компонентів. Кожен сегмент має власні правила доступу, що знижує ймовірність масштабного порушення безпеки.

Строгі правила доступу: Визначення чітких і обмежених правил доступу є важливим для контролю над тим, хто і до яких частин системи має доступ. Це забезпечує мінімізацію можливостей зловмисників використовувати уразливості для доступу до критичних даних чи функціоналу. Стратегії, такі як "найменші привілеї" (least privilege), коли кожен компонент або користувач має лише мінімально необхідний доступ, значно знижують ризики.

Шифрування даних: Захист даних є ключовим аспектом для розподілених систем. Шифрування даних, як в спокої, так і в транзиті, забезпечує захист від несанкціонованого доступу. Дані, які передаються між мікросервісами або зберігаються в різних частинах системи, повинні бути зашифровані, щоб уникнути їхнього перехоплення або модифікації під час руху через мережу.

Регулярне оновлення програмного забезпечення: Однією з найважливіших практик для забезпечення безпеки є регулярне оновлення всього програмного забезпечення. Це дозволяє своєчасно усувати відомі вразливості, які зловмисники можуть експлуатувати для атаки на систему. Розподілені системи з численними компонентами потребують чіткої стратегії для централізованого оновлення, щоб усі мікросервіси та інші частини системи залишалися захищеними.

Моніторинг та аналіз логів: Виявлення атак та аномальної активності у великих, розподілених системах може бути складним завданням, тому моніторинг усіх компонентів і ретельний аналіз логів є важливими. Використання сучасних інструментів для збору та аналізу логів допомагає виявити потенційні інциденти на ранніх стадіях і швидко реагувати на загрози.

Тестування на проникнення: Регулярні перевірки системи на наявність вразливостей за допомогою тестування на проникнення дозволяють імітувати реальні атаки та перевірити її стійкість до потенційних загроз. Такі перевірки допомагають виявити слабкі місця, що можуть бути використані зловмисниками, і своєчасно їх усунути.

Автоматизація процесів: Для зменшення людського фактора та мінімізації помилок, що можуть призвести до безпекових інцидентів, важливо автоматизувати рутинні завдання, такі як оновлення програмного забезпечення, моніторинг мережі, а також реагування на інциденти. Це дозволяє підвищити ефективність і точність операцій безпеки.

Безапеляційна верифікація: Важливо регулярно перевіряти відповідність усіх компонентів системи встановленим стандартам безпеки. Безапеляційна верифікація передбачає, що кожен елемент системи, будь то новий або вже наявний, проходить ретельну перевірку на відповідність безпековим вимогам перед тим, як бути інтегрованим чи оновленим.

Застосування цих стратегій допомагає не лише підвищити рівень безпеки розподілених систем, але й зменшити потенційний вплив загроз, роблячи систему більш стійкою до атак.

2.3 Використання протоколів безпеки (HTTPS, TLS, gRPC)

Протоколи безпеки є основою для забезпечення захищеної комунікації між сервісами у розподілених системах. Вони надають механізми для збереження конфіденційності даних, перевірки їхньої цілісності та забезпечення автентичності учасників взаємодії. Серед найпоширеніших протоколів, що використовуються в цих цілях, слід виділити HTTPS, T LS і gRPC.

HTTPS (HyperText Transfer Protocol Secure) є розширенням стандартного HTTP-протоколу з використанням шифрування через TLS (Transport Layer Security). HTTPS захищає ко мунікацію від перехоплення і підміни, гарантуючи, що дані залишаються недоступними для зловмисників під час передачі між серверами. У контексті мікросервісів HTTPS використовується для безпечної передачі даних через зовнішні і внутрішні API.

TLS (Transport Layer Security) є фундаментальним протоколом для забезпечення захищеного з'єднання. Він гарантує конфіденційність, застосовуючи шифрування, і перевіряє цілісність даних за допомогою алгоритмів хешування [33]. TLS також забезпечує автентифікацію обох сторін комунікації через цифрові сертифікати, що робить його ідеальним вибором для захисту взаємодії між сервісами [35]. Оновлені версії TLS мають вдосконалені механізми безпеки, зокрема захист від атак типу "людина посередині" (MITM) [29].

gRPC (gRPC Remote Procedure Call) — це сучасний фреймворк для організації комунікації між сервісами, який використовує HTTP/2 та підтримує вбудовану інтеграцію з TLS[35]. Він дозволяє передавати дані у стисненому форматі, що підвищує ефективність роботи системи. Вбудована підтримка

шифрування забезпечує конфіденційність і безпеку переданої інформації, що робить gRPC популярним вибором для масштабованих розподілених систем [35].

Використання цих протоколів у мікросервісних архітектурах дозволяє створювати захищену та надійну інфраструктуру для обміну даними. Вони не тільки мінімізують ризики перехоплення та маніпуляції даними, але й забезпечують високу продуктивність та сумісність у складних системах (див. табл. 2.1).

Таблиця 2.1 – Порівняння протоколів безпеки в мікросервісах

Характеристика	HTTPS	TLS	gRPC
Основне призначення	Захист веб-трафіку	Забезпечення безпечного з'єднання між двома пристроями	Високопродуктивний RPC фреймворк
Протокол транспорту	HTTP	TCP	HTTP/2
Формат даних	Текстовий (HTTP)	Двійковий	Двійковий (Protobuf)
Типізація даних	Слабка	Немає	Сильна
Використання	Веб-застосунки	Різноманітні мережеві протоколи	Мікросервісні архітектури

Вибір протоколу безпеки для захисту комунікації між сервісами залежить від особливостей архітектури системи, її масштабів і вимог до продуктивності. Кожен протокол має свої переваги та найкращі сценарії використання.

HTTPS є оптимальним вибором для захисту веб-трафіку, особливо у випадках, коли сервіси взаємодіють через зовнішні API. Цей протокол забезпечує простоту інтеграції та високу сумісність із сучасними веб-технологіями, що робить його стандартом для веб-застосунків.

TLS виступає універсальним рішенням для захисту різних типів мережевих з'єднань. Він не лише використовується для захисту веб-трафіку (у

складі HTTPS), але й може застосовуватися в інших сценаріях, включаючи прямі з'єднання між сервісами, базами даних та іншими компонентами системи.

gRPC є найбільш підходящим для мікросервісних архітектур, де критично важлива продуктивність і ефективність передачі даних. Використовуючи протокол HTTP/2 і вбудоване шифрування через TLS, gRPC дозволяє зменшити затримки та підвищити швидкість обміну даними між сервісами, що робить його ідеальним для складних і масштабованих систем.

Для забезпечення максимальної безпеки важливо використовувати сучасні версії протоколів, які включають вдосконалення для протидії відомим загрозам. Регулярне оновлення програмного забезпечення і перевірка на відповідність безпековим стандартам дозволяють мінімізувати ризики компрометації системи.

2.4 Ідентифікація та авторизація між мікросервісами (OAuth 2.0, JWT)

У розподілених системах мікросервісної архітектури ідентифікація та авторизація є основою для забезпечення безпеки. Ці процеси гарантують, що тільки уповноважені сервіси мають доступ до ресурсів, зменшуючи ризик несанкціонованих дій і забезпечуючи цілісність усієї системи. Для цього найчастіше використовуються OAuth 2.0 та JWT (JSON Web Token), які дозволяють реалізувати гнучкі та ефективні механізми управління доступом [36].

OAuth 2.0 є широко використовуваним протоколом авторизації, який дозволяє сервісам отримувати обмежений доступ до ресурсів від імені користувачів або інших сервісів [36]. Його основна перевага полягає в модульному підході, який підтримує різні сценарії аутентифікації та авторизації. OAuth 2.0 забезпечує використання токенів доступу, які виступають тимчасовими ключами для отримання доступу до ресурсів. Ці

токени можуть бути відкликані, налаштовуватися для різних рівнів доступу та легко інтегруються в розподілені системи [36].

JWT (JSON Web Token) часто використовується в рамках OAuth 2.0 або самостійно як легкий спосіб передачі інформації про ідентифікацію [37]. JWT є закодованим токеном у форматі JSON, який містить інформацію про користувача або сервіс, а також підписаний для гарантії цілісності. Його перевага в тому, що токен є автономним, тобто містить усю необхідну інформацію для перевірки доступу без необхідності звертатися до центрального сервера. Це значно знижує навантаження на систему та підвищує швидкість обробки запитів [37].

У поєднанні OAuth 2.0 і JWT створюють потужну систему управління доступом для мікросервісів. OAuth 2.0 забезпечує динамічний і гнучкий механізм отримання доступу, тоді як JWT дозволяє швидко й ефективно передавати та перевіряти інформацію про авторизацію [38]. Однак для ефективного використання цих інструментів важливо впроваджувати такі практики, як використання короткоживучих токенів, регулярна ротація ключів підпису та забезпечення захищеної передачі токенів через шифровані канали, такі як TLS [37, 38].

2.5 Механізми забезпечення стійкості до атак у мікросервісах

У сучасних мікросервісних архітектурах питання стійкості до атак є одним із ключових викликів у забезпеченні безпеки систем. Розподілений характер мікросервісів створює нові можливості для зловмисників, які можуть використовувати вразливості кожного окремого сервісу для проникнення або порушення роботи всієї системи. Водночас, така архітектура надає унікальні можливості для впровадження гнучких і надійних механізмів захисту.

Стійкість до атак у мікросервісах передбачає не лише захист від загальних загроз, таких як DDoS-атаки, SQL-ін’єкції чи злом API, але й забезпечення безперервності роботи навіть у разі часткової компрометації. Для

цього застосовуються сучасні методи автентифікації, шифрування даних, управління доступом, а також інструменти моніторингу та виявлення аномалій.

Механізми забезпечення стійкості повинні враховувати унікальні особливості мікросервісної архітектури: децентралізацію, незалежність компонентів і активну взаємодію через API. Головна мета цих механізмів — створення багаторівневої системи захисту, яка не тільки запобігає атакам, але й ефективно реагує на інциденти, мінімізуючи їх вплив на систему в цілому.

2.5.1 Rate limiting та Throttling

Rate limiting та throttling — це механізми, які допомагають захистити мікросервісні системи від перевантажень і атак шляхом контролю обсягу та швидкості трафіку. Вони дозволяють обмежувати кількість запитів, які клієнт може надіслати до сервісу за визначений період часу, і забезпечують стабільність роботи системи навіть під час пікових навантажень[39.

Rate limiting обмежує максимальну кількість запитів, які клієнт може виконати в певний проміжок часу. Наприклад, API може дозволяти до 1000 запитів на годину з однієї IP-адреси. Якщо цей ліміт перевищено, сервер починає відхиляти нові запити, зазвичай із кодом відповіді 429 Too Many Requests. Такий підхід не лише запобігає зловживанню ресурсами, а й захищає систему від DDoS-атак, одночасно зберігаючи ресурси рівномірно доступними для всіх користувачів [39].

Throttling, на відміну від rate limiting, зосереджується на керуванні швидкістю обробки запитів у реальному часі. У разі перевищення допустимої швидкості сервер може сповільнювати обробку запитів клієнта замість їхнього повного блокування. Це дозволяє системі залишатися стабільною під час раптових сплесків трафіку, зменшуючи ризик недоступності.

Застосування цих механізмів разом дозволяє створити баланс між обмеженнями використання ресурсів та стабільністю роботи системи. Rate

limiting встановлює довгострокові обмеження, тоді як throttling допомагає впоратися з короткочасними піками навантаження. Це робить їх незамінними для забезпечення ефективності, безпеки та надійності сучасних веб-додатків.

2.5.2 Circuit Breaker та Fault Tolerance

Circuit Breaker та Fault Tolerance є ключовими концепціями для забезпечення надійності розподілених систем, таких як мікросервісні архітектури. Вони дозволяють зменшити вплив помилок та збоїв у системі, забезпечуючи її стабільну роботу навіть у разі виникнення проблем [40].

Circuit Breaker (Запобіжник) працює як механізм захисту системи від перевантажень і повторних спроб взаємодії з несправними сервісами. Цей підхід імітує роботу електричного запобіжника, який розриває ланцюг, щоб уникнути пошкодження через надмірне навантаження. У розподілених системах circuit breaker контролює запити до залежного сервісу. Якщо кількість помилок або таймаутів перевищує встановлений поріг, він переходить у "відкритий" стан, блокуючи нові запити до цього сервісу на певний період часу. Після цього запобіжник може перейти до "напіввідкритого" стану, тестуючи, чи сервіс відновився, і тільки тоді повертається до "закритого" стану, якщо помилки більше не повторюються [41, 42]. Це зменшує додаткове навантаження на несправні компоненти та запобігає поширенню збоїв.

Fault Tolerance (Толерантність до збоїв) — це здатність системи продовжувати функціонувати навіть у разі часткових збоїв. Для досягнення цього застосовуються різні підходи, такі як дублювання компонентів, автоматичне переключення на резервні ресурси, механізми самовідновлення або альтернативні сценарії виконання [42]. Толерантність до збоїв дозволяє зберігати доступність і надійність системи, навіть якщо один із її компонентів не працює належним чином.

Обидва механізми часто використовуються разом у мікросервісних архітектурах для створення стійких до збоїв систем. Circuit Breaker допомагає зменшити миттєве навантаження на несправний компонент, тоді як Fault Tolerance забезпечує резервні стратегії для виконання критично важливих функцій. Наприклад, якщо один сервіс не доступний, система може автоматично перенаправити запити до його резервної копії або надати спрощену відповідь замість повного відмовлення.

Поєднання цих підходів сприяє стабільності та безперервній роботі сучасних веб-додатків, зменшуючи ризик великих відмов і підвищуючи довіру до системи з боку користувачів.

Висновки до розділу: Загалом, можна стверджувати, що безпека мікросервісних архітектур значною мірою залежить від правильного впровадження оркестраційних інструментів та забезпечення координації між численними компонентами системи. Використання таких платформ, як Kubernetes, дозволяє ефективно управляти розгортанням, масштабуванням і моніторингом, що є критично важливим для стабільної роботи мікросервісів. Проте зростання кількості точок взаємодії між сервісами значно ускладнює забезпечення безпеки і вимагає ретельної координації політик доступу та управління. Мікросегментація є одним із найбільш ефективних підходів до зниження ризиків, оскільки вона ізолює мікросервіси один від одного, що обмежує поширення атак. Дотримання принципу найменших привілеїв і автоматизація управління доступом значно знижують ймовірність несанкціонованого доступу та атак з боку внутрішніх загроз. Водночас застосування протоколів безпеки, таких як gRPC, HTTPS і TLS, забезпечує надійний захист даних під час їхнього обміну між сервісами, що є основою для створення стійкої інфраструктури. Механізми захисту, такі як rate limiting, throttling, Circuit Breaker і Fault Tolerance, дозволяють системі залишатися стабільною навіть під час пікових навантажень або збоїв окремих компонентів.

Це свідчить про важливість інтеграції таких методів для забезпечення надійності та відмовостійкості мікросервісних систем. Отже, я вважаю, що правильне проєктування архітектури, впровадження сучасних інструментів безпеки та постійна автоматизація процесів є основою для забезпечення надійного захисту мікросервісних архітектур і протистояння сучасним кіберзагрозам.

3 АНАЛІЗ ТА ДОСЛІДЖЕННЯ БЕЗПЕКИ НА ОСНОВІ РІЗНИХ МІКРОСЕРВІСНИХ АРХІТЕКТУР

3.1 Огляд популярних платформ та інструментів

Огляд популярних платформ та інструментів дозволяє оцінити їхній функціонал, переваги та недоліки з погляду безпеки. Це дослідження включає аналіз рішень для розгортання мікросервісів, керування трафіком, забезпечення ідентифікації та авторизації, а також протидії поширеним загрозам. Особлива увага приділяється таким аспектам, як інтеграція безпечних комунікацій, впровадження політик доступу, виявлення аномалій та управління ризиками в розподілених системах.

Цей розділ слугує основою для подальшого дослідження взаємозв'язків між різними архітектурними підходами та їхнім впливом на безпеку, що сприятиме вибору оптимальних рішень для забезпечення надійності та захисту мікросервісних веб-додатків.

3.1.1 Kubernetes: засоби для забезпечення безпеки

Kubernetes, як потужний оркестратор контейнерів, надає широкий набір інструментів і механізмів для забезпечення безпеки в розподілених системах. Одним із ключових компонентів його екосистеми є роль-базований контроль доступу (RBAC) [43], який дозволяє гнучко управляти правами користувачів і сервісних акаунтів у межах кластерів. Це забезпечує чітке розмежування обов’язків і зменшує ризики несанкціонованого доступу до критично важливих ресурсів.

NetworkPolicies є ще одним важливим засобом, який дозволяє налаштовувати мережеву ізоляцію між підмережами та контейнерами. Цей механізм забезпечує контроль трафіку, що входить і виходить із певних

ресурсів, обмежуючи можливість горизонтального переміщення зловмисників у разі компрометації одного з компонентів [44].

Namespaces надають можливість логічно розділяти ресурси кластера на ізольовані області. Це корисно для створення багатокористувацьких середовищ або розмежування різних етапів розробки, таких як тестування, розгортання та продуктивність [45]. Завдяки цьому забезпечується ізоляція середовищ і зменшується ризик випадкового або зловмисного впливу на ресурси.

Механізм Admission Controllers дає змогу інтерпретувати запити на створення чи зміну об'єктів у кластері, додаючи додатковий рівень захисту. Цей інструмент може застосовувати політики безпеки, перевіряти дотримання стандартів або відхиляти потенційно небезпечні запити [46].

Для захисту конфіденційної інформації Kubernetes використовує Secrets. Цей інструмент дозволяє безпечно зберігати й управляти чутливими даними, такими як паролі, ключі доступу або сертифікати, без необхідності їхнього збереження у відкритому вигляді в коді або конфігураційних файлах.

Загалом, засоби безпеки Kubernetes дозволяють створювати надійні та захищені мікросервісні архітектури, адаптуючи механізми захисту до потреб сучасних розподілених систем.

3.1.2 Istio: управління безпечним трафіком між мікросервісами

Istio — це потужна сервісна сітка, яка надає велику кількість функцій для забезпечення безпеки комунікації між мікросервісами в розподілених системах [47]. Однією з основних переваг Istio є підтримка двостороннього шифрування через протокол Mutual TLS (mTLS). Цей механізм забезпечує шифрування даних, що передаються між сервісами, а також дозволяє аутентифікацію як клієнта, так і сервера, створюючи довіру між ними. Встановлення двостороннього шифрування запобігає ризикам перехоплення

або маніпулювання даними під час їх передачі через мережу, що особливо важливо для захисту конфіденційної інформації.

Важливою функцією Istio є авторизація, що дозволяє контролювати доступ до мікросервісів на основі визначених політик безпеки. Це дозволяє гарантувати, що тільки авторизовані користувачі або сервіси можуть отримати доступ до конкретних ресурсів або виконувати певні дії, таким чином обмежуючи можливості для зловмисників [47].

Іншою ключовою функцією є аутентифікація, яка в Istio здійснюється за допомогою таких методів, як сертифікати та токени JWT. Це дозволяє гарантувати, що лише легітимні сервіси можуть взаємодіяти один з одним, і що доступ до системи не отримають сторонні особи або шкідливі програми.

Політики безпеки в Istio дозволяють створювати детальні інструкції для управління трафіком між сервісами. Наприклад, можна визначати правила, які обмежують або дозволяють доступ до певних API або сервісів, або застосовувати обмеження на частоту запитів, що дозволяє захистити систему від атак типу DoS [47].

Таким чином, Istio є потужним інструментом для безпеки мікросервісних архітектур, забезпечуючи надійне управління трафіком, аутентифікацією, авторизацією та політиками безпеки, що дозволяє створити безпечну та стійку систему.

3.1.3 Service Mesh як метод забезпечення безпеки

Service Mesh є архітектурним шаблоном, який забезпечує абстракцію мережевих функцій у розподілених системах, особливо у мікросервісних архітектурах. Він дозволяє централізовано управляти безпекою та взаємодією між мікросервісами без необхідності змінювати їхній код [48, 49]. Це особливо важливо для великих систем, де кількість сервісів може досягати десятків або навіть сотень, і потреби в безпеці можуть швидко стати надто складними для управління вручну (див. табл. 3.1).

Однією з основних переваг Service Mesh є централізоване управління безпекою. Всі політики безпеки, такі як шифрування трафіку, аутентифікація та авторизація, можуть бути конфігуровані глобально для всіх мікросервісів. Це дозволяє забезпечити узгодженість у всій системі та зменшити ризики, пов'язані з розрізненими політиками безпеки, які могли б виникнути при розподіленому управлінні [48].

Service Mesh також дозволяє проводити A/B тестування політик безпеки, що дає змогу експериментувати з різними варіантами захисту для перевірки їх ефективності перед впровадженням на всю систему. Це корисно для швидкого реагування на нові загрози та для тестування нових технологій чи протоколів безпеки, не порушуючи роботу всіх сервісів.

Іншою важливою перевагою є прозорість. Service Mesh надає детальну видимість всього трафіку між сервісами, що дозволяє краще розуміти, як дані переміщуються в межах системи та виявляти аномалії або спроби атак [49]. Це дозволяє значно покращити процес моніторингу та аналізу безпеки.

Окрім того, Service Mesh володіє розширюваністю, що дозволяє легко інтегрувати його з іншими інструментами безпеки, такими як системи виявлення вторгнень, інструменти для моніторингу трафіку або рішення для управління політиками безпеки.

Таким чином, Service Mesh забезпечує потужні можливості для централізованого управління безпекою мікросервісних архітектур, дозволяючи значно спростити адміністрування та посилити захист системи без необхідності втручатися в код окремих сервісів.

Таблиця 3.1 – Порівняння характеристик Kubernetes і Istio

Характеристика	Kubernetes	Istio
Основна функціональність	Оркестрація контейнерів	Управління трафіком між сервісами
Безпека	Базові механізми безпеки (RBAC, NetworkPolicies)	Додаткові функції для забезпечення безпеки (Mutual TLS, авторизація)
Інтеграція	Може працювати з різними сервісними сітками	Спеціалізований для управління трафіком

3.2 Аналіз ризиків для різних архітектур мікросервісів

Незалежно від того, чи є система однорідною, з використанням єдиної технології, чи неоднорідною, з використанням різних платформ для кожного окремого сервісу, кожна архітектура має свої унікальні переваги та виклики з точки зору безпеки. Процес аналізу ризиків включає в себе вивчення потенційних вразливостей, які можуть виникнути через особливості архітектурних рішень, а також оцінку впливу цих ризиків на цілісність, конфіденційність і доступність даних у системі. Вивчення різних архітектур допомагає визначити ефективні стратегії захисту та вибір інструментів для забезпечення належного рівня безпеки у мікросервісних середовищах.

3.2.1 Однорідні та неоднорідні мікросервіси: порівняння безпеки

Однорідні та неоднорідні мікросервіси мають суттєві відмінності, які впливають на їхню безпеку. Однорідні мікросервіси використовують одну технологічну платформу для всіх компонентів, наприклад, всю систему можна побудувати на Java Spring Boot або Node.js. Це дає певні переваги в управлінні та підтримці, оскільки всі сервіси мають спільні стандарти і можуть використовувати однакові бібліотеки та інструменти безпеки. Така однорідність також забезпечує кращу сумісність між компонентами і зменшує витрати на навчання та підтримку команди.

Проте є й ризики. Однорідні мікросервіси мають схильність до однотипних вразливостей. Якщо одна з технологій має уразливість, вона може бути присутньою в усіх компонентах системи, що підвищує ризик масових атак. Крім того, така архітектура обмежує гнучкість у виборі технологій, оскільки всі сервіси повинні працювати на одній платформі.

Неоднорідні мікросервіси, з іншого боку, базуються на різних технологіях для кожного окремого сервісу. Це дає більше гнучкості у виборі найкращих інструментів для конкретних задач, дозволяє вибирати оптимальні технології для кожного компоненту та знижує ризик поширення вразливостей. Кожен мікросервіс може бути захищений окремими інструментами безпеки, зменшуючи вплив потенційної загрози на всю систему.

Однак така архітектура має і свої недоліки. Вона ускладнює управління та підтримку, оскільки кожен компонент потребує свого налаштування та моніторингу. Крім того, забезпечення сумісності між різними технологіями потребує додаткових зусиль. Крім того, використання різних технологій збільшує кількість точок входу, що може створити більше векторів для атак.

У підсумку, вибір між однорідними та неоднорідними мікросервісами залежить від конкретних потреб бізнесу, бажаної гнучкості, а також здатності ефективно управляти складністю та ризиками безпеки (див. табл. 3.2).

Таблиця 3.2 – Порівняння характеристик однорідних і неоднорідних мікросервісів

Характеристика	Однорідні мікросервіси	Неоднорідні мікросервіси
Ризик поширення вразливостей	Вищий	Нижчий
Складність управління безпекою	Нижчий	Вищий
Гнучкість	Нижча	Вища

3.2.2 Вплив інтеграції сторонніх API на безпеку

Інтеграція сторонніх API є важливим елементом мікросервісної архітектури, оскільки дозволяє розширити функціональність системи без необхідності розробляти нові компоненти з нуля. Однак така інтеграція несе із собою додаткові ризики для безпеки.

Одним з основних ризиків є вразливості сторонніх API. Якщо сторонній API має невиявлені або експлуатовані вразливості, зловмисники можуть скористатися ними для атак на вашу систему, що може призвести до компрометації даних або відмови в обслуговуванні. Існує також проблема неконтрольованого доступу: якщо сторонній API надає занадто великий доступ до вашої системи або даних, це може спричинити витік конфіденційної інформації або зміну даних без вашого відома. Інтеграція сторонніх API також підвищує залежність від зовнішніх постачальників, і ви не можете повністю контролювати їхні процеси розробки, оновлення або реагування на вразливості, що виникають.

Для мінімізації цих ризиків важливо ретельно вибирати сторонніх постачальників API. Перед інтеграцією варто перевірити їхню репутацію, а також відповідність стандартам безпеки та наявність механізмів для своєчасного виправлення вразливостей. Крім того, необхідно обмежувати доступ сторонніх API до тільки необхідних для їх функціонування даних. Це дозволяє мінімізувати масштаби потенційного витоку інформації, якщо API стане об'єктом атаки. Важливо також регулярно моніторити активність сторонніх API, щоб виявити підозрілі дії або спроби несанкціонованого доступу.

Одним з корисних підходів є використання проксі-серверів або API Gateway, що дозволяє централізовано керувати доступом до сторонніх API. Це дає змогу застосовувати додаткові заходи безпеки, такі як автентифікація, моніторинг і обмеження запитів.

Інтеграція сторонніх API може значно полегшити розширення функціональності мікросервісної архітектури, однак вона також створює додаткові виклики щодо безпеки. Тому для зниження ризиків необхідно застосовувати додаткові заходи, щоб захистити систему від можливих загроз, пов'язаних із використанням зовнішніх ресурсів.

3.3 Практичне тестування безпеки мікросервісів

Практичне тестування безпеки мікросервісів є критично важливою частиною процесу забезпечення безпеки в розподілених системах. Це тестування дозволяє виявити вразливості в системі на ранніх етапах розробки, що допомагає уникнути серйозних порушень безпеки на етапі експлуатації. Тестування безпеки мікросервісів включає в себе різноманітні методи та інструменти, які забезпечують перевірку всіх аспектів безпеки, від шифрування та аутентифікації до виявлення вразливостей і атак.

3.3.1 Використання статичних та динамічних аналізаторів

Статичний та динамічний аналіз є двома основними підходами до тестування безпеки мікросервісів, кожен з яких має свої переваги та обмеження. Обидва ці методи використовуються для виявлення вразливостей та неправильних конфігурацій у програмному забезпеченні на різних етапах розробки та експлуатації.

Статичний аналіз (Static Application Security Testing, SAST) полягає в аналізі коду без його виконання [50]. Цей метод дозволяє знаходити потенційні вразливості на рівні програмного коду, зокрема, такі як SQL-ін'єкції, XSS, помилки в аутентифікації та інші логічні вразливості. Статичний аналіз дає можливість знаходити помилки на ранніх етапах розробки, до того, як код буде запущений в середовищі [50]. Це дозволяє зменшити витрати на виправлення помилок, оскільки виявлені проблеми можуть бути виправлені до того, як система буде запущена в експлуатацію.

Інструменти для статичного аналізу сканують вихідний код або байт-код програми, порівнюючи його з відомими патернами вразливостей. Вони можуть також перевіряти конфігураційні файли, бібліотеки та інші ресурси, що використовуються в мікросервісах, на предмет наявності застарілих або небезпечних залежностей.

Динамічний аналіз (Dynamic Application Security Testing, DAST) здійснюється під час виконання програми в реальному середовищі. Цей метод дозволяє виявити вразливості в робочому середовищі, коли програма вже працює і взаємодіє з іншими компонентами системи. Динамічний аналіз включає тестування системи в процесі її роботи, зокрема перевірку поведінки програмного забезпечення під час взаємодії з базами даних, API, зовнішніми сервісами тощо [51].

Динамічні аналізатори можуть перевіряти, як система реагує на несанкціоновані запити, як поводиться при спробах атаки через API, чи є відкриті порти чи інші точки доступу до конфіденційних даних. Вони можуть також автоматично перевіряти уразливості в мережевих з’єднаннях між сервісами, що особливо важливо для мікросервісних архітектур.

Використання обох підходів є необхідним для забезпечення комплексної безпеки мікросервісів. Статичний аналіз дозволяє виявити проблеми ще на етапі розробки, тоді як динамічний аналіз фокусується на виявленні проблем у роботі системи в реальних умовах. Тому комбінація цих двох методів дає найкращий результат у забезпеченні безпеки на всіх етапах життєвого циклу мікросервісів.

3.3.2 Інструменти для перевірки безпеки (Burp Suite, OWASP ZAP)

Інструменти для перевірки безпеки є важливим елементом для виявлення вразливостей у мікросервісних архітектурах. Два з найбільш популярних і ефективних інструментів для тестування безпеки — це Burp Suite та OWASP ZAP. Обидва інструменти використовуються для виявлення

уразливостей у веб-додатках та мікросервісах, таких як SQL-ін'єкції, XSS-атаки та інші небезпечні вразливості.

Burp Suite є одним з найпоширеніших та найпотужніших інструментів для тестування безпеки веб-додатків, здобувши популярність завдяки своєму багатофункціональному підходу до виявлення вразливостей у веб-додатках [52, 53]. Цей інструмент охоплює весь спектр перевірок, від перехоплення трафіку до автоматизованого сканування вразливостей, що робить його ідеальним для професійних тестувальників і спеціалістів з безпеки.

Основною характеристикою Burp Suite є можливість перехоплення та модифікації HTTP(S)-трафіку через його Proxy. Ця функція дозволяє аналізувати трафік між клієнтом і сервером в реальному часі, досліджувати запити та відповіді, що надходять, та маніпулювати ними для виявлення потенційних уразливостей. Це дозволяє тестувальникам не тільки перевірити існуючі уразливості, але й коригувати запити, щоб випробувати систему на стійкість до різних атак.

Scanner є ще однією потужною функцією Burp Suite, яка автоматично сканує веб-додатки на наявність вразливостей, таких як SQL-ін'єкції, Cross-Site Scripting (XSS) та інші типи атак. Цей інструмент значно спрощує процес виявлення проблем безпеки та економить час, автоматизуючи багато з робіт, що раніше виконувались вручну [53].

Завдяки Intruder, користувачі можуть здійснювати брутфорсінг для тестування додатків на наявність вразливостей через спроби введення різних параметрів, таких як логіни, паролі, або інші типи введення даних, що дозволяє знайти уразливі точки у системах автентифікації та авторизації. Крім того, Repeater дає можливість вручну повторювати HTTP-запити для тестування різних сценаріїв взаємодії з додатком, що є важливим для більш детального аналізу уразливостей або відтворення специфічних ситуацій.

Burp Suite також надає функцію Extender, яка дозволяє додавати нові функції через плагіни для специфічних типів атак або перевірок безпеки. Це забезпечує гнучкість та розширюваність інструменту, що робить його ще більш ефективним при роботі з різними типами веб-додатків.

Серед переваг Burp Suite варто зазначити наявність багатої документації та активної спільноти користувачів, що робить процес освоєння інструменту швидким і доступним навіть для новачків [52]. Інструмент підтримує автоматизовані тести, що дає можливість значно прискорити процес тестування безпеки, а також має можливість інтеграції з іншими інструментами, такими як системи CI/CD або платформи для моніторингу безпеки.

Проте, Burp Suite має і певні недоліки. Безкоштовна версія інструменту має обмежену функціональність, що стосується автоматизованого сканування вразливостей. Крім того, для великих та складних проектів Burp Suite може вимагати значних ресурсів системи, оскільки під час використання великої кількості тестів інструмент може навантажувати комп'ютер, що потребує високих апаратних характеристик [52, 53].

OWASP ZAP — це потужний інструмент для тестування безпеки веб-додатків, який є відкритим та безкоштовним. Як і Burp Suite, ZAP пропонує ряд інструментів для перехоплення трафіку, аналізу вразливостей та тестування на наявність слабких місць у безпеці. Його основною метою є допомога виявлення проблем безпеки на ранніх етапах розробки та експлуатації веб-додатків [54].

Основною функцією OWASP ZAP є Proxy, який дозволяє перехоплювати і модифікувати HTTP(S)-трафік, що проходить між браузером і сервером. Це дає можливість в реальному часі тестувати взаємодію між клієнтами та серверами, а також виявляти уразливості під час фактичних запитів. Завдяки

такій функціональності, користувачі можуть досліджувати взаємодії між компонентами і перевіряти їх на безпеку [54].

Однією з найбільш корисних функцій ZAP є Active Scan, що дозволяє автоматизовано сканувати веб-додатки на наявність вразливостей. Цей інструмент активно взаємодіє з сервером і дозволяє знаходити різноманітні проблеми безпеки, включаючи SQL-ін'єкції, XSS та інші типи атак. Крім того, Passive Scan дозволяє здійснювати неактивний аналіз трафіку без взаємодії з сервером, що корисно для виявлення відомих вразливостей, не навантажуючи систему. Spider в ZAP використовується для автоматичного сканування і мапування веб-застосунку [55]. Це дозволяє знаходити всі доступні точки входу та шляхи до ресурсів додатка, що є корисним для тестування всіх елементів додатку на вразливості. Інструмент Fuzzer в свою чергу допомагає тестувати на введення некоректних або шкідливих даних, що може виявити уразливі місця додатка, які можуть бути використані для атак. Перевагою OWASP ZAP є те, що він повністю безкоштовний і має відкритий вихідний код, що дозволяє користувачам адаптувати інструмент під свої потреби. Крім того, ZAP є легким у використанні, що робить його ідеальним вибором для початківців. Він має активну спільноту та отримує регулярні оновлення, що підтримує актуальність інструменту в умовах швидко змінюваної сфери безпеки. Однак, незважаючи на численні переваги, ZAP має деякі недоліки [54]. Порівняно з Burp Suite, функціональність ZAP може бути обмеженою в складних сценаріях тестування безпеки, що робить його менш придатним для професіоналів, які потребують більш складних інструментів для глибокого аналізу. Також деякі користувачі відзначають, що документація ZAP може бути менш детальною для проведення глибоких і специфічних тестів [55].

Порівняння Burp Suite та OWASP ZAP:

Функціональність [52]: Обидва інструменти мають схожі основні функції, такі як перехоплення трафіку, сканування вразливостей, та інші

інструменти для тестування безпеки веб-додатків. Однак Burp Suite пропонує більш розвинені можливості для автоматизованого тестування, включаючи більш потужні інструменти для брутфорсінгу (Intruder) і повторного тестування (Repeater). Вона також має більш розширену можливість інтеграції з іншими інструментами, що є перевагою для більш складних або корпоративних середовищ. OWASP ZAP орієнтований на спрощену автоматизацію і надає функціональність для початкового та середнього рівня тестування, зокрема завдяки активному та пасивному скануванню [52]. Це робить ZAP кращим вибором для початківців і тих, хто потребує менш складних налаштувань.

Простота використання: Зазвичай OWASP ZAP вважається більш доступним для початківців завдяки зрозумілому інтерфейсу та простим налаштуванням. Він підходить для тих, хто тільки починає тестування безпеки і хоче зосередитися на базових аспектах безпеки веб-додатків. Burp Suite, хоча й не складний для використання, має більше налаштувань і функцій, що може вимагати більш глибокого розуміння інструментів для досягнення оптимальних результатів.

Ціна: OWASP ZAP є повністю безкоштовним, що робить його чудовим вибором для невеликих команд, індивідуальних тестувальників або стартапів, які не мають бюджету на платні інструменти [55]. Burp Suite, у свою чергу, пропонує безкоштовну версію з обмеженою функціональністю та платні варіанти, які включають розширену підтримку автоматизованих тестів, інтеграцію з іншими інструментами та додаткові функції для професіоналів і корпоративних користувачів [52]. Інтеграція: Burp Suite має кращу підтримку для інтеграції з іншими інструментами тестування, що робить його більш підходящим для комплексних тестувальних середовищ або командних проєктів. Вона підтримує інтеграцію з CI/CD конвеєрами та іншими інструментами, що дозволяє автоматизувати більшість тестів і спростити

процес безпеки в рамках більшої організації. Водночас OWASP ZAP більше підходить для індивідуальних тестувальників або невеликих команд, які працюють з відкритим кодом і мають обмежені потреби в інтеграції з іншими системами (див. табл. 3.13).

Таблиця 3.3 – Порівняння особливості інструментів тестування безпеки Burp Suite та OWASP ZAP

Характеристика	Burp Suite	OWASP ZAP
Ціна	Комерційна (є безкоштовна версія)	Безкоштовна
Функціональність	Широкий спектр функцій, глибокий аналіз	Основні функції для сканування вразливостей
Простота використання	Вимагає певного досвіду	Проста у використанні
Спільнота	Активна, але більш закрита	Велика і активна спільнота з відкритим кодом
Інтеграція	Широкі можливості інтеграції	Гнучкі можливості інтеграції

Висновки до розділу: Загалом, можна стверджувати, що забезпечення безпеки мікросервісних архітектур значною мірою залежить від правильного вибору сучасних платформ, методів аналізу та інструментів тестування. Використання Kubernetes і Istio дозволяє впроваджувати багаторівневі механізми захисту, такі як мікросегментація, роль-базований контроль доступу (RBAC) і Mutual TLS, які підвищують захищеність взаємодії між сервісами. Service Mesh, у свою чергу, спрощує управління масштабуванням і безпекою політик, що є важливим для складних систем. Однорідні архітектури,

незважаючи на легкість управління, демонструють більшу вразливість до атак через єдину технологічну основу. Натомість неоднорідні системи, хоч і створюють труднощі в моніторингу та забезпеченні сумісності, знижують ризик масштабних атак за рахунок технологічної різноманітності. Це підкреслює важливість вибору архітектурного підходу з урахуванням потреб і загроз. Ретельна перевірка сторонніх API, інтеграція статичного (SAST) та динамічного (DAST) аналізів є критичними для виявлення вразливостей на всіх етапах життєвого циклу. Використання таких інструментів, як Burp Suite і OWASP ZAP, дозволяє забезпечити високий рівень автоматизації тестування та точність виявлення загроз. Таким чином, ефективність безпеки мікросервісних архітектур базується на застосуванні сучасних технологій захисту, збалансованому виборі архітектурного підходу та постійному моніторингу ризиків, що дозволяє мінімізувати вплив кіберзагроз.

ВИСНОВКИ

1) Проведене дослідження надає всебічний аналіз безпеки веб-застосунків у контексті мікросервісних архітектур, зосереджуючись на сучасних загрозах, методах захисту та інструментах. У роботі підкреслюється, що мікросервісні архітектури забезпечують як гнучкість, так і підвищену масштабованість веб-застосунків, але водночас створюють унікальні виклики в забезпеченні безпеки, зокрема в аспектах захищеності комунікації між сервісами, контролю доступу та захисту від зовнішніх загроз.

2) Ключові результати демонструють, що технології, такі як Kubernetes, Istio та Service Mesh, відіграють вирішальну роль у підвищенні безпеки. Kubernetes забезпечує ізоляцію ресурсів та зменшує ризики несанкціонованого доступу завдяки використанню просторів імен. Istio надає можливість централізованого управління політиками безпеки та шифруванням. Service Mesh спрощує захищену взаємодію між сервісами, пропонуючи механізми, як-от A/B-тестування політик безпеки. Порівняння однорідних і неоднорідних мікросервісів виявило, що однорідні системи легше управляти, однак вони є більш вразливими до масштабних атак. У свою чергу, неоднорідні архітектури знижують ці ризики завдяки різноманітності, але водночас ускладнюють моніторинг і забезпечення сумісності.

3) Інтеграція методів статичного та динамічного аналізу виявилася ефективною для ідентифікації вразливостей на різних етапах життєвого циклу розробки. Інструменти, такі як SAST і DAST, забезпечують взаємодоповнюючий підхід: SAST фокусується на вразливостях у коді, а DAST виявляє загрози під час виконання. Крім того, дослідження підкреслює корисність інструментів тестування безпеки, таких як Burp Suite та OWASP ZAP. Хоча Burp Suite забезпечує потужну автоматизацію та інтеграцію з CI/CD, його недоліком є високе споживання ресурсів для складних проєктів. OWASP

ZAP, будучи відкритим та простим у використанні, пропонує економічно ефективне рішення для невеликих команд або початкових етапів тестування.

4) Незважаючи на досягнення, дослідження акцентує увагу на критичних викликах у сфері. Складність управління мікросервісними архітектурами, потенційні вразливості, пов'язані зі сторонніми API, а також зростаюча складність кіберзагроз вимагають подальших інновацій у практиках безпеки. Ця робота успішно відповіла на дослідницьке питання, надаючи практичні рекомендації щодо ефективних стратегій захисту мікросервісних архітектур та підтверджуючи їхню застосовність у реальних умовах.

5) Подальші дослідження повинні розглянути інтеграцію штучного інтелекту для виявлення загроз у середовищах мікросервісів і оцінити вплив новітніх технологій, таких як архітектура нульової довіри (Zero-Trust Architecture). Крім того, необхідно приділити увагу оптимізації рішень для забезпечення безпеки в неоднорідних системах з метою збалансування підвищеного рівня захисту та зменшення складності операцій. У результаті дослідження підтверджено важливість проактивного, багаторівневого підходу до безпеки, що створює основу для подальших досягнень у захисті веб-застосунків.

СПИСОК ДЖЕРЕЛ ПОСИЛАННЯ

1. Anisimov, V., & Kunanets, N. (2024). Перехід від монолітної до мікросервісної архітектури: методологія та досвід впровадження. COMPUTER-INTEGRATED TECHNOLOGIES: EDUCATION, SCIENCE, PRODUCTION, (55), 30-41. Режим доступу: https://doi.org/10.36910/6775-2524-0560-2024-55-03
2. Гнатишин, М. А. (2023). Аналіз стану прогресивних технологій для створення веб-застосунків (Bachelor's thesis). Режим доступу: https://elartu.tntu.edu.ua/bitstream/lib/41679/1/2023_KRB_SN-41_Hnatyshyn_MA.pdf
3. Архітектура мікросервісів: Особливості, переваги, реальні приклади Режим доступу: https://www.hostzealot.com.ua/blog/about-solutions/arxitektura-mikroservisiv-osoblivosti-perevagi-realni-prikladi
4. Про мікросервісну архітектуру. Режим длоступу: https://foxminded.ua/mikroservisna-arkhitektura/
5. Що краще моноліт чи мікросервіси? Як обрати архітектуру проєкту? Режим доступу: https://iampm.club/ua/blog/shho-krashhe-monolit-chi-mikroservisi-yak-obrati-arhitekturu-projektu/
6. Пригода, А. (2024). МІКРОСЕРВІСНА АРХІТЕКТУРА, ЯК ОСНОВА ДЛЯ СТВОРЕННЯ CRM-СИСТЕМИ. Наука і техніка сьогодні, (8 (36)). Режим доступу: http://perspectives.pp.ua/index.php/nts/article/download/14403/14473
7. Лаврєнов, В. А., & Сіренко, О. І. (2021). Класифікація загроз веб-застосунків. Режим доступу: https://card-file.ontu.edu.ua/server/api/core/bitstreams/969f9b76-0d27-4637-95f4-b3055d8f48e2/content

8. Королевич, Є. М. (2021). DDoS-атаки на освітні веб-ресурси. Режим доступу: https://card-file.ontu.edu.ua/server/api/core/bitstreams/4921cdc5-fedf-4641-9890-f5547c07dcf1/content
9. Петрів, П., & Опірський, І. (2023). АНАЛІЗ ПРОБЛЕМАТИКИ ВИКОРИСТАННЯ ІСНУЮЧИХ СТАНДАРТІВ З ВЕБ-ВРАЗЛИВОСТЕЙ. Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка», 2(22), 96-112. Режим доступу: https://doi.org/10.28925/2663-4023.2023.22.96112
10. Яремчук, К., Воскобойников, Д., & Мелкозьорова, О. (2022). Сучасні загрози та способи забезпечення безпеки веб-застосунків. Комп'ютерні науки та кібербезпека, (2), 28-34. Режим доступу: https://doi.org/10.26565/2519-2310-2023-1-02
11. Корицький, В. І., Майгур, О. І., Дідківський, О. О., & Заіченко, Т. С. АНАЛІЗ ЗАГРОЗ ІНФОРМАЦІЙНІЙ БЕЗПЕЦІ WEB-ЗАСТОСУНКУ З МІКРОСЕРВІСНОЮ АРХІТЕКТУРОЮ. ОРГАНІЗАТОРИ, 44. Режим доступу: https://rep.nuos.edu.ua/server/api/core/bitstreams/69a859ad-28a6-41de-b086-5e5b377033c1/content#page=44
12. Єпік, О. О. (2024). Удосконалення засобів захисту веб-застосунків від несанкціонованого впливу. Режим доступу: https://ela.kpi.ua/server/api/core/bitstreams/e72ae542-914e-44e7-8eda-7c37ee1f7c26/content
13. Sadqi, Y., & Maleh, Y. (2022). A systematic review and taxonomy of web applications threats. Information Security Journal: A Global Perspective, 31(1), 1-27. Режим доступу: https://doi.org/10.1080/19393555.2020.1853855
14. Hoffman, A. (2024). Web application security. " O'Reilly Media, Inc.". Режим доступу: https://tinyurl.com/ye27a7r5

15. Bach-Nutman, M. (2020). Understanding the top 10 owasp vulnerabilities. arXiv preprint arXiv:2012.09960. Режим доступу: https://arxiv.org/pdf/2012.09960
16. Idris, M., Syarif, I., & Winarno, I. (2022). Web application security education platform based on OWASP API security project. EMITTER international journal of engineering technology, 246-261. Режим доступу: https://emitter.pens.ac.id/index.php/emitter/article/download/705/254
17. Vallabhaneni, R., Pillai, S. E. V. S., Vaddadi, S. A., Addula, S. R., & Ananthan, B. (2024). Secured web application based on CapsuleNet and OWASP in the cloud. Indonesian Journal of Electrical Engineering and Computer Science, 35(3), 1924-1932. Режим доступу: https://doi.org/10.11591/ijeecs.v35.i3.pp1924-1932
18. Nedeljković, N., Vugdelija, N., & Kojić, N. (2020, October). Use of "OWASP Top 10" in web application security. In Fourth International Scientific Conference on Recent Advances in Information Technology, Tourism, Economics, Management and Agriculture (p. 25). Режим доступу: https://doi.org/10.52783/jes.2288
19. Culot, G., Nassimbeni, G., Podrecca, M., & Sartor, M. (2021). The ISO/IEC 27001 information security management standard: literature review and theory-based research agenda. The TQM Journal, 33(7), 76-105. Режим доступу: https://www.emerald.com/insight/content/doi/10.1108/tqm-09-2020-0202/full/pdf
20. Qusef A, Alkilani H. 2022. The effect of ISO/IEC 27001 standard over open-source intelligence. PeerJ Computer Science 8:e810. Режим доступу: https://doi.org/10.7717/peerj-cs.810
21. Amuthadevi, C., Srivastava, S., Khatoria, R., & Sangwan, V. (2022). A study on web application vulnerabilities to find an optimal security

architecture. arXiv preprint arXiv:2204.07107. Режим доступу: https://doi.org/10.48550/arXiv.2204.07107

22. Watch a Free Demo - Intrusion Detection for MSPs. Режим доступу: https://tinyurl.com/ru32tm68
23. Pryhoda, A. (2024). Оцінка ефективності проєкту розробки та впровадження crm-системи на основі мікросервісної архітектури. COMPUTER-INTEGRATED TECHNOLOGIES: EDUCATION, SCIENCE, PRODUCTION, (55), 172-180. Режим доступу: https://doi.org/0.36910/6775-2524-0560-2024-55-22
24. Shabani, I., Mëziu, E., Berisha, B., & Biba, T. (2021). Design of modern distributed systems based on microservices architecture. International Journal of Advanced Computer Science and Applications, 12(2). Режим доступу: https://doi.org/10.14569/IJACSA.2021.0120220
25. Сусла, М. В., Папа, О. А., & Сигінь, М. С. (2023). Інструмент для оцінки автоматичного масштабування у Kubernetes кластерах. Режим доступу: https://tinyurl.com/ycye5bk5
26. Дарвеш, Г., Хаммуд, Д., & ВОРОБЬЕВА, А. А. (2022). Security in kubernetes: best practices and security analysis. Вестник УрФО. Безопасность в информационной сфере, (2 (44)), 63-69. Режим доступу: https://doi.org/10.14529/secur220209
27. Про мікросегментацію мережі - Network Security. Режим доступу: https://rubydevelopers.org/t/topic/500
28. Найвідоміші вразливості веб застосунків. XSS та SQL ін’єкції, вразливості автентифікації. Режим доступу: https://dou.ua/forums/topic/40613/
29. What Is a Man-in-the-Middle Attack (MitM)? Режим доступу: https://www.techtarget.com/iotagenda/definition/man-in-the-middle-attack-MitM

30. Madden, N. (2020). API security in action. Simon and Schuster. Режим доступу: https://tinyurl.com/2pprrv54
31. Understanding API Attacks. Режим доступу; https://shorturl.at/k0WNM
32. DoS Attack vs DDoS Attack: Key Differences? Режим доступу: https://www.fortinet.com/resources/cyberglossary/dos-vs-ddos
33. What is HTTPS? Режим доступу: https://www.cloudflare.com/learning/ssl/what-is-https/
34. What is Transport Layer Security (TLS)? Режим доступу: https://www.cloudflare.com/learning/ssl/transport-layer-security-tls/
35. What is gRPC (Google Remote Procedure Call). Режим доступу: https://gizemcifguvercin.medium.com/whats-grpc-google-remote-procedure-call-8d7c56dd9cb2
36. What is OAuth 2.0 and what does it do for you? Режим доступу: https://auth0.com/intro-to-iam/what-is-oauth-2
37. What is a JWT? Understanding JSON Web Tokens. Режим доступу: https://supertokens.com/blog/what-is-jwt
38. Dimitrijević, N., Zdravković, N., Bogdanović, M., & Mesterovic, A. (2024). Advanced Security Mechanisms in the Spring Framework: JWT, OAuth, LDAP and Keycloak. Режим доступу: https://ceur-ws.org/Vol-3676/short_09.pdf
39. About Throttling and Rate Limiting Policie. Режим доступу: https://tinyurl.com/dud6vzwx
40. Mohammadi, F., Rouzbehi, K., Hajian, M., Niayesh, K., Gharehpetian, G. B., Saad, H., ... & Sood, V. K. (2021). HVDC circuit breakers: A comprehensive review. IEEE Transactions on Power Electronics, 36(12), 13726-13739. Режим доступу: https://doi.org/10.1109/TPEL.2021.3073895

41. Leveraging Circuit Breakers & Retry Patterns | by Sanjay Singh. Режим доступу: https://saannjaay.medium.com/mastering-fault-tolerant-microservices-leveraging-circuit-breakers-retry-patterns-99268599b785
42. Efficient Fault Tolerance with Circuit Breaker Pattern. Режим доступу: https://aerospike.com/blog/circuit-breaker-pattern/
43. Серверний диригент: що таке Kubernetes і як він працює. Режим доступу: https://robotdreams.cc/uk/blog/397-serverniy-dirigent-shcho-take-kubernetes-i-yak-vin-pracyuye
44. Network Policies. Режим доступу: https://kubernetes.io/docs/concepts/services-networking/network-policies/
45. Namespaces. Режим доступу: https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/
46. Admission Controllers Reference. Режим доступу: https://kubernetes.io/docs/reference/access-authn-authz/admission-controllers/
47. Впровадження та переваги Istio Ambient Mesh. Оптимізація ресурсів та підвищення безпеки у мікросервісних середовищах. Режим доступу: https://dou.ua/forums/topic/49912/
48. What is service mesh and why do we need it? Режим доступу: https://www.dynatrace.com/news/blog/what-is-a-service-mesh/
49. What is a service mesh? Service mesh benefits and how to overcome their challenges. Режим доступу: https://www.dynatrace.com/news/blog/what-is-a-service-mesh/
50. Li, K., Xue, Y., Chen, S., Liu, H., Sun, K., Hu, M., ... & Chen, Y. (2024). Static Application Security Testing (SAST) Tools for Smart Contracts: How Far Are We?. Proceedings of the ACM on Software Engineering, 1(FSE), 1447-1470. Режим доступу: https://doi.org/10.1145/3660772

51. Dynamic application security testing (DAST) - Black Duck. Режим доступу: https://www.blackduck.com/glossary/what-is-dast.html
52. What Is the OWASP API Security Top 10? Режим доступу: https://www.pynt.io/learning-hub/owasp-top-10-guide/owasp-top-10-api-security-risks-and-how-to-mitigate-them
53. Burp Suite. Режим доступу: https://www.sciencedirect.com/topics/computer-science/burp-suite
54. Best Practices Guide: Web Application Firewalls – OWASP. Режим доступу: https://wiki.owasp.org/images/a/a4/AppSecEU08-BPWAF.pdf :
55. Driving OWASP ZAP with Selenium. Режим доступу: https://owasp.org/www-chapter-london/assets/slides/OWASPLondon-OWASP-ZAP-Selenium-20180830-PDF.pdf