Міністерство освіти і науки України
Київський Національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ДОПУСТИТИ ДО ЗАХИСТУ:

В.о. завідувача кафедри кібербезпеки та захисту інформації

_____ Іван ПАРХОМЕНКО

«17» травня 2024 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи

галузь знань *12 Інформаційні технології*
(шифр і назва галузі знань)

спеціальність *125 Кібербезпека*
(код і назва спеціальності)

освітній ступень *магістр*

освітньо-наукова програма *Кібербезпека*
(назва освітньої програми)

на тему: «Метод захисту компонентів архітектури програмного забезпечення»

Виконавець: студент ІІ курсу, групи КБм-22

_____ **Андрій ХОМИН**
(підпис) (Ім'я, ПРІЗВИЩЕ)

	Ім'я, ПРІЗВИЩЕ	**Підпис**
Науковий керівник	Лариса МИРУТЕНКО	
Нормоконтроль	Сергій ДАКОВ	

Київ 2024

Міністерство освіти і науки України
Київський Національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ЗАТВЕРДЖЕНО:
В.о. завідувача кафедри
кібербезпеки та захисту
інформації
_____ Іван ПАРХОМЕНКО

«17» листопада 2023 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи

спеціальності *125 Кібербезпека*
(код і назва спеціальності)

освітній ступень *магістр*

Здобувача(ки) КБм-22 (група) Хомин Андрію Петровичу (прізвище ім'я по-батькові)

Тема кваліфікаційної роботи Метод захисту компонентів архітектури програмного забезпечення

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Рішення засідання кафедри кібербезпеки та захисту інформації факультету інформаційних технологій протокол № 5 від 15.11.2023 р.

2. МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єкт досліджень процес захисту компонентів архітектури програмного забезпечення

Предмет досліджень методи захисту компонентів архітектури програмного забезпечення

Мета вдосконалення методу захисту компонентів архітектури програмного забезпечення

Вихідні дані для проведення роботи способи побудови архітектури, загрози компонентам архітектури, існуючі методи захисту

3. ОЧІКУВАНІ РЕЗУЛЬТАТИ

Наукова новизна удосконалення методу захисту компонентів архітектури програмного забезпечення від ін'єкцій шкідливого коду на основі використання штучного інтелекту

Практична цінність метод захисту з використанням штучного інтелекту для забезпечення захисту компонентів архітектури програмного забезпечення

4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Робота виконана у повному обсязі відповідно до теми.

5. ЕТАПИ ВИКОНАННЯ РОБОТИ

Найменування етапів робіт	**Строки виконання робіт (початок-кінець)**
Уточнення постановки задачі	17.11.2023 – 28.01.2024
Аналіз літературних джерел	29.01.2024 – 10.02.2024
Ознайомлення з способами побудови архітектури програмного забезпечення	11.02.2024 – 20.02.2024
Розгляд нормативно-правових актів, що стосуються захисту архітектури програмного забезпечення	21.02.2024 – 25.02.2024
Дослідження загроз та вразливостей, що спрямовуються на компоненти архітектури	26.02.2024 – 02.03.2024
Аналіз рекомендацій стосовно розгортання компонентів архітектури програмного забезпечення в хмарних провайдерах	03.03.2024 – 13.03.2024
Дослідження існуючих методів та засобів захисту архітектури програмного забезпечення	14.03.2024 – 19.03.2024
Розгляд відомих атак	20.03.2024 – 17.03.2024
Аналіз існуючих методів захисту від ін'єкцій коду	18.03.2024 – 16.04.2024
Розробка вдосконаленого методу захисту компонентів архітектури програмного забезпечення	17.04.2024 – 24.04.2024

Найменування етапів робіт	Строки виконання робіт (початок-кінець)
Оформлення пояснювальної записки згідно методичних рекомендацій	25.04.2024 – 12.05.2024
Подача пакету документів на розгляд ЕК	13.05.2024 – 17.05.2024

6. РЕАЛІЗАЦІЯ РЕЗУЛЬТАТІВ ТА ЕФЕКТИВНІСТЬ

Економічний ефект зменшення витрат на захист архітектури програмного забезпечення, збільшення захищеності компонентів архітектури від атак

Соціальний ефект покращення здатності додатків впоратися з помилками під час виконання, підвищення загального рівня безпеки та захищеності даних

7. ДОДАТКОВІ ВИМОГИ

Завдання видав _____ Лариса МИРУТЕНКО
(підпис) (Ім'я, ПРІЗВИЩЕ)

Завдання прийняв до виконання _____ Андрій ХОМИН
(підпис) (Ім'я, ПРІЗВИЩЕ)

Дата видачі завдання: 17.11.2023 р.
Термін подання кваліфікаційної роботи до ЕК 17.05.2024 р.

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Метод захисту компонентів архітектури програмного забезпечення» складається зі списку скорочень, вступу, основної частини, що містить 4 розділи, висновків і списку використаних джерел. Загальний обсяг роботи – 93 сторінки. Робота містить 16 рисунків. Список використаної літератури включає 53 джерел.

Об’єктом дослідження є процес захисту компонентів архітектури.

Мета роботи – вдосконалення методу захисту компонентів архітектури програмного.

Предмет дослідження – методи захисту компонентів архітектури програмного забезпечення.

Метод дослідження – аналіз можливих способів побудови архітектури програмного забезпечення, визначення вразливих місць в компонентах архітектури.

В роботі проведено аналіз особливостей побудови архітектури програмного забезпечення, типових загроз та методів протидії.

Практичне значення роботи полягає у створенні вдосконаленого методу захисту з використанням штучного інтелекту для захисту компонентів архітектури програмного забезпечення.

Результати здійснених у дипломній роботі досліджень можуть бути використані спеціалістами із захисту та розробки програмних продуктів.

Напрямки подальших досліджень: вдосконалення інших методів захисту з використання штучного інтелекту.

Ключові слова: архітектура програмного забезпечення, компоненти архітектури, безпека, загроза, ін'єкція коду, методи захисту, штучний інтелект.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

AWS	-	Amazon Web Services;
AI	-	Artificial intelligence;
ML	-	Machine Learning;
HTTP	-	HyperText Transfer Protocol;
SOA	-	Service-oriented Architecture;
API	-	Application programming interface;
SRPP	-	Secure Remote Password Protocol;
SOAP	-	Simple Object Access Protocol;
DDoS	-	Distributed Denial of Service;
IT	-	Information Technology;
DNS	-	Domain Name System;
DoS	-	Denial of Service;
HTML	-	HyperText Markup Language;

ВСТУП

В сучасному цифровому світі, де програмне забезпечення проникає в усі сфери нашого життя, забезпечення його безпеки та стійкості стає вирішальною задачею. Методи захисту компонентів архітектури програмного забезпечення відіграють критичну роль у гарантуванні надійності та безпеки системи в цілому.

Справжня важливість цієї теми стає зрозумілою, коли ми розглядаємо масштаби можливих загроз: від класичних атак злому, до сучасних форм кіберзагроз, які можуть походити від кіберзлочинців, конкурентів чи навіть держав. Незахищені компоненти архітектури можуть стати вразливими точками, які використовуються для доступу, витоку даних, або навіть збоїв у системі.

Розуміння принципів та методів захисту стає ключовим як для розробників, які мають створити програмне забезпечення з високим рівнем захисту, так і для кінцевих користувачів, які сподіваються на безпечну та надійну роботу програмних продуктів. Основні аспекти цієї теми включають в себе визначення потенційних загроз, розробку стратегій захисту, використання криптографічних засобів, моніторинг та аналіз системи на предмет вразливостей, а також вдосконалення методів захисту відповідно до змінюваних умов і загроз.

З виникненням нових технологій та появою нових загроз кібербезпеці, методи захисту компонентів архітектури програмного забезпечення постійно еволюціонують. Розвиток штучного інтелекту, Інтернету речей, обчислювального хмарного середовища та інших інноваційних технологій створює нові вектори атак та вразливості, на які необхідно реагувати. Тому розробники програмного забезпечення постійно вдосконалюють і адаптують свої методи захисту, щоб відповідати сучасним вимогам безпеки.

У світі, де кіберзагрози стають все більш складними та вишуканими, ключовою вимогою до методів захисту компонентів архітектури програмного забезпечення є їх гнучкість та адаптивність. Інновації в цій області включають в себе розробку інтелектуальних систем виявлення вразливостей, використання

аналізу поведінки для виявлення незвичних атак, а також застосування технологій машинного навчання та штучного інтелекту для автоматизації процесів виявлення та реагування на загрози. Такий підхід дозволяє забезпечити високий рівень захисту програмного забезпечення.

Метою даної кваліфікаційної роботи є вдосконалення методу захисту компонентів архітектури програмного забезпечення.

Для досягнення зазначеної мети поставлені наступні завдання:

- ознайомлення з способами побудови архітектури програмного забезпечення;
- розгляд нормативно-правових актів, що стосуються захисту архітектури програмного забезпечення;
- дослідження загроз та вразливостей, що спрямовуються на компоненти архітектури;
- аналіз рекомендацій стосовно розгортання компонентів архітектури програмного забезпечення в хмарних провайдерах;
- дослідження існуючих методів та засобів захисту архітектури програмного забезпечення;
- розгляд відомих атак;
- аналіз існуючих методів захисту від ін'єкцій коду;
- розробка вдосконаленого методу захисту компонентів архітектури програмного забезпечення.

Об’єктом дослідження є процес захисту компонентів архітектури.

Предмет дослідження – методи захисту компонентів архітектури програмного забезпечення.

Метод дослідження – аналіз можливих способів побудови архітектури програмного забезпечення, визначення вразливих місць в компонентах архітектури.

Практична цінність дипломної роботи полягає у вдосконаленому методу захисту архітектури програмного забезпечення від ін'єкцій коду з використанням штучного інтелекту.

РОЗДІЛ 1
СПОСОБИ ПОБУДОВИ АРХІТЕКТУРИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Архітектура програмного забезпечення — спосіб структурування програмної або обчислювальної системи, абстракція елементів системи на певній фазі її роботи. Система може складатись з кількох рівнів абстракції і мати багато фаз роботи, кожна з яких може мати окрему архітектуру. Дослідження архітектури програмного забезпечення намагається визначити як найкраще розбити систему на частини, як ці частини визначають та взаємодіють одна з одною, як між ними передається інформація, як ці частини розвиваються поодинці і як все вищеописане найкраще записати використовуючи формальну чи неформальну нотацію [1].

1.1 Клієнт-серверна архітектура

Архітектура клієнт-сервер — це обчислювальна модель, у якій сервер розміщує, надає та керує більшістю ресурсів і послуг, які споживає клієнт. Архітектура цього типу передбачає один або кілька клієнтських комп’ютерів, підключених до центрального сервера через мережу або підключення до Інтернету. Архітектура клієнт-сервер також відома як модель мережевих обчислень або мережа клієнт-сервер, оскільки всі запити та послуги доставляються через мережу [2].

Клієнти – це такі пристрої, як комп’ютер, смартфон або планшет. Вони ініціюють зв'язок і запитують послуги від сервера.

Сервери — це потужні комп’ютери або пристрої, призначені для надання послуг або ресурсів клієнтам. Вони чекають вхідних запитів від клієнтів і відповідають відповідним чином.

Клієнти та сервери спілкуються через мережу, як Інтернет, запит надсилається через мережу на сервер, який обробляє запит та надсилає відповідь.

Зв'язок між клієнтами та серверами зазвичай відбувається за моделлю запит-відповідь. Клієнт надсилає запит на певну послугу, а сервер обробляє запит і надсилає відповідь із необхідною інформацією.

Зв'язок клієнт-сервер зазвичай не має стану, що означає, що кожен запит обробляється незалежно. Сервер не пам’ятає минулі взаємодії з конкретним клієнтом.

Архітектура клієнт-сервер забезпечує легку масштабованість. Якщо більше клієнтів потребують одночасного доступу до сервера, він може обробити ці запити шляхом ефективного розподілу робочого навантаження.

Коли клієнт хоче отримати дані від сервера, відбуваються такі кроки:

- Клієнт повинен знати IP-адресу конкретного сервера, з яким він хоче спілкуватися. Щоб отримати цю інформацію, він надсилає запит на сервер системи доменних імен (DNS), надаючи доменне ім’я цільового сервера.
- DNS-сервер відповідає IP-адресою, пов’язаною із запитаним ім’ям домену.
- Знаючи IP-адресу, клієнт надсилає запит на сервер. Запит містить IP-адресу призначення та конкретний номер порту, який відповідає певній програмі, запущеній на сервері, з яким клієнт хоче спілкуватися.
- Сервер приймає запит клієнта і обробляє його.
- Після обробки запиту сервер формує відповідне повідомлення.
- Відповідь надсилається назад клієнту за допомогою IP-адреси та номера порту, вказаних у запиті клієнта.
- Клієнт отримує відповідний пакет і витягує необхідну інформацію на основі номера порту.

На високому рівні зв’язок між клієнтом і сервером відбувається за допомогою пакетів HTTP. HTTP — це протокол, який дозволяє клієнту та серверу спілкуватися через Інтернет і обмінюватися даними. Це

стандартизований набір правил, які визначають, як потрібно форматувати та обробляти запити та відповіді.

Архітектуру клієнт-сервер можна розділити на різні типи залежно від того, як зв’язок і обов’язки розподіляються між клієнтами та серверами. Існують такі поширені типи клієнт-серверних архітектур:

- Дворівнева архітектура. У цьому типі клієнт безпосередньо зв’язується з сервером для запиту послуг і отримання даних. Сервер обробляє як логіку інтерфейсу користувача, так і бізнес-логіку. Ця архітектура підходить для невеликих програм, але може стати менш масштабованою в міру зростання програми. Приклад: традиційні клієнт-серверні програми, де настільна програма спілкується з центральним сервером для доступу до бази даних.

- Трирівнева архітектура. У цьому типі клієнт взаємодіє з проміжним рівнем, званим «проміжним програмним забезпеченням», який діє як міст між клієнтом і сервером бази даних. Рівень презентації: відповідає за користувальницький інтерфейс і фіксує дані користувача. Рівень програми: керує бізнес-логікою та обробкою запитів, він взаємодіє з сервером бази даних для отримання та зберігання даних. Рівень даних: це сервер бази даних, який зберігає та керує даними. Ця архітектура покращує масштабованість, оскільки робоче навантаження може бути розподіленою між сервером додатків і сервером бази даних. Приклад: веб-програми, де веб-браузер є клієнтом, веб-сервер є сервером програм, а сервер бази даних зберігає дані.

- N-рівнева архітектура. Ця архітектура є розширенням трирівневої архітектури та дозволяє додавати більше рівнів або рівнів для певних цілей. Додаткові рівні можуть включати сервери безпеки, сервери кешування, балансувальники навантаження, залежно від складності та вимог програми. Ця архітектура забезпечує кращу модульність, гнучкість і оптимізацію продуктивності. Приклад: великомасштабні корпоративні програми з кількома рівнями безпеки, кешування та балансування навантаження.

- Однорангова архітектура. У цьому типі немає чіткої різниці між клієнтами та серверами. Кожен пристрій (одноранговий пристрій) може

виступати як клієнтом, так і сервером, обмінюючись ресурсами та послугами безпосередньо з іншими одноранговими вузлами. Рівні спілкуються та співпрацюють один з одним у децентралізований спосіб. Ця архітектура зазвичай використовується в програмах обміну файлами та системах спільної роботи. Приклад: BitTorrent, одноранговий протокол обміну файлами.

- Хмарні обчислення (SOA). Хмарні обчислення часто використовують сервіс-орієнтовану архітектуру, де клієнти отримують доступ до послуг, які надають віддалені сервери через Інтернет. Клієнти взаємодіють із хмарними службами через API, а не через пряме спілкування. Хмарні послуги можуть включати різні функції, як-от зберігання, обчислення, бази даних тощо. Приклад: хмарні платформи, такі як Amazon Web Services або Microsoft Azure.

Особливості та переваги 2-рівневої архітектури:

- Складається з двох основних рівнів: клієнт і сервер.
- Клієнт відповідає за інтерфейс користувача та взаємодіє безпосередньо з сервером, щоб запитувати послуги та отримувати дані.
- Сервер обробляє як логіку презентації (інтерфейс користувача), так і бізнес-логіку (обробка запиту та даних).
- Зазвичай використовується для невеликих додатків з обмеженою кількістю користувачів.
- 2-рівневу архітектуру відносно просто реалізувати, оскільки вона включає лише два рівні.
- Оскільки між клієнтом і сервером існує пряме з’єднання, час відповіді може бути швидшим порівняно зі складнішими архітектурами.
- Низький мережевий трафік: зв’язок відбувається безпосередньо між клієнтом і сервером, що зменшує мережевий трафік.
- Просте обслуговування: завдяки меншій кількості компонентів технічне обслуговування.

Недоліки 2-рівневої архітектури:

- Масштабованість: із зростанням програми та збільшенням кількості клієнтів сервер може стати перевантаженим, що вплине на продуктивність.

- Обмежена модульність: відсутність поділу між презентацією та бізнес-логікою може зробити програму менш модульною та її складніше підтримувати.
- Безпека: заходи безпеки можуть бути обмеженими, оскільки клієнт безпосередньо взаємодіє з сервером і може бути чутливим до різноманітних атак.
- Цілісність даних: відсутність проміжного рівня може призвести до проблем із цілісністю даних, якщо клієнти безпосередньо маніпулюють даними без належної перевірки.

Особливості та переваги 3-рівневої архітектури:

- Складається з трьох основних рівнів: рівень презентації, рівень програми (проміжне програмне забезпечення) і рівень даних (сервер).
- Рівень презентації (клієнт): відповідає за користувальницький інтерфейс і фіксує дані користувача.
- Рівень програми (проміжне програмне забезпечення): діє як посередник між клієнтом і сервером бази даних, обробляючи бізнес-логіку та обробляючи запити.
- Рівень даних (сервер): зберігає та керує даними, а рівень програми зв’язується з ними, щоб отримати або зберегти інформацію.
- Пропонує кращу масштабованість і модульність, ніж 2-рівнева архітектура.
- Масштабованість: поділ завдань дозволяє розподіляти робоче навантаження між рівнем додатків і рівнем даних, підвищуючи масштабованість.
- Модульність: кожен рівень можна розробляти незалежно, що полегшує підтримку та оновлення програми.
- Покращена безпека: Рівень програми може самостійно впроваджувати заходи безпеки, зменшуючи потенційну вразливість.
- Цілісність даних: рівень даних забезпечує послідовність і цілісність даних, запобігаючи прямим маніпуляціям з боку клієнтів.

Недоліки 3-рівневої архітектури:

- Складність: додатковий рівень (рівень програми) створює більшу складність, що може ускладнити розробку та налагодження.
- Додаткові витрати на продуктивність: залучення додаткового рівня може спричинити певні накладні витрати на продуктивність через збільшення зв’язку між рівнями.
- Вищі витрати: Впровадження та керування трирівневою архітектурою може бути дорожчим, ніж дворівнева, особливо для невеликих програм із обмеженою базою користувачів.
- Проблеми з обслуговуванням: оскільки програма стає складнішою, технічне обслуговування та усунення несправностей може вимагати більше зусиль і досвіду.

N-рівнева архітектура — це шаблон проектування програмного забезпечення, який організовує програму на кілька рівнів або рівнів, причому кожен рівень відповідає за певні функції та має певну роль у загальній системі. На відміну від 3-рівневої архітектури, яка має три основні рівні (презентація, додаток і дані), N-рівнева архітектура може мати більше трьох рівнів, що робить її придатною для складних і великомасштабних програм. Типові рівні в багаторівневій архітектурі такі:

- Рівень презентації — це частина програми, де користувачі взаємодіють із системою. Він керує інтерфейсом користувача та фіксує дані користувача. Клієнтом може бути веб-браузер, мобільний додаток або будь-який інший інтерфейс користувача, який взаємодіє з рештою програми.
- Рівень програми діє як посередник між рівнем презентації та рівнем даних. Він містить основну бізнес-логіку та обробляє запити та дії користувачів. Рівень програми обробляє специфічні для програми функції, такі як автентифікація користувачів, керування сеансами та бізнес-правила.
- Рівень даних відповідає за зберігання та керування даними програми. Він керує зберіганням, пошуком і маніпулюванням даними. Рівень даних взаємодіє з базами даних або іншими системами зберігання даних, щоб зберігати та отримувати дані, необхідні програмі.

- Дозволяє включати додаткові рівні за потреби для певних функцій або вимог. Ці рівні можуть включати спеціалізовані компоненти, такі як сервери кешування, балансувальники навантаження, пошукові системи або зовнішні API.

Переваги N-рівневої архітектури:

- Масштабованість: Модульний характер дозволяє горизонтальне масштабування, де додаткові екземпляри кожного рівня можуть бути додані для обробки збільшеного трафіку користувачів і потреби в ресурсах.

- Гнучкість. Розподіл завдань архітектури дозволяє розробникам змінювати або додавати рівні відповідно до вимог програми, дозволяючи інтегрувати спеціалізовані компоненти для покращення продуктивності та функціональності.

- Високий рівень безпеки: розділяючи конфіденційні функції та дані на окремі рівні, покращує безпеку, обмежуючи доступ до критичних компонентів і зменшуючи поверхню для атак.

- Модульність і багаторазове використання: кожен рівень можна розробляти незалежно та повторно використовувати в кількох програмах, сприяючи супроводження коду та скорочуючи час розробки для майбутніх проектів.

Недоліки N-рівневої архітектури:

- Складність: наявність кількох рівнів і взаємодії між ними може зробити архітектуру складнішою для проектування, реалізації та налагодження, що вимагає кваліфікованих розробників і ретельного планування.

- Додаткові витрати на продуктивність: зв’язок між рівнями може спричинити затримку та накладні витрати на продуктивність через додаткові мережеві запити та передачу даних, що потенційно може вплинути на час відповіді.

- Вартість і споживання ресурсів. Впровадження кількох рівнів і керування ними може збільшити витрати на розробку та інфраструктуру, особливо для менших проектів із меншим навантаженням на користувачів.

- Складність розгортання: розгортання, моніторинг і підтримка кількох рівнів потребують складніших процесів та інструментів, що може бути складним для менш досвідчених команд.

1.2 Компонентна архітектура

Компонентна архітектура описує підхід до проектування і розробки систем з використанням методів проектування програмного забезпечення. Основна увага в цьому випадку приділяється розкладанню дизайну на окремі функціональні або логічні компоненти, певні інтерфейси, що надають чітко, містять методи, події і властивості. В даному випадку забезпечується вищий рівень абстракції, чим при об'єктно-орієнтованій розробці, і не відбувається концентрації уваги на таких питаннях, як протоколи зв'язку або загальний стан [3].

Розробка зовнішнього програмного забезпечення прогресує та розвивається, щоб задовольнити потреби користувачів і стати максимально гнучким. Ось чому багато розробників використовують компонентну архітектуру.

Це підхід до проектування та розробки, який зосереджується на перетворенні елементів на окремі багаторазові компоненти. Наприклад, мобільний додаток має панель пошуку, заголовок і зображення. У компонентній архітектурі ці компоненти незалежні. Uber, Spotify і PayPal використовували компонентну архітектуру при створенні своїх систем. Останнім часом на зміну монолітній архітектурі прийшла компонентна архітектура, яка є надто громіздкою та не допускає повторного використання, оскільки всі елементи зібрані в одному блоці.

Компонент — це багаторазовий елемент, який надає додаткові функції, одночасно прискорюючи розгортання та розробку програми. Незважаючи на гнучкість і модульність, їх можна багаторазово використовувати в кількох проектах. Ви можете використовувати компоненти послідовно в кількох

інтерфейсах і модулях, не ставлячи під загрозу безпеку свого коду чи взаємодію з користувачем. Різні компоненти мають різні типи, і розробники та компанії можуть використовувати їх для різних цілей у своїх програмах. Компанії можуть продавати їх у готовому вигляді на ринках, щоб інші підприємства використовували їх у своїх проектах через API.

Оскільки компоненти легко розкладаються та налаштовуються, вони повинні мати такі характеристики:

- Компоненти — це незалежні модулі, які можуть обмінюватися даними один з одним через порти своїх інтерфейсів.
- Система на основі компонентів може бути розібрана та повторно зібрана з багаторазовими, цілісними та незалежними, зібраними з різних модулів.
- Ці модулі можна легко підтримувати та масштабувати без коригувань.

Як правило, компоненти функціонують лише на одному рівні програми, наприклад, інтерфейс і серверна частина. Однак різні архітектури компонентів можна повторно використовувати для різних рівнів програми. Існують такі типи компонентів:

- Теми виглядають як правила таблиці стилів і визначення сітки, де дизайнери можуть розміщувати та розмірувати елементи на екрані. Їх мета — надати користувачам узгоджений досвід роботи на всіх платформах і уніфікувати брендинг.
- Віджети є компонентами нижчого рівня. Вони надають функцію багаторазового використання, яка доповнює функціональність програми, і їх можна вважати компонентами, якщо вони мають власний набір визначень, наприклад параметрів і змінних.
- Бібліотеки роблять систему ще кращою у більшому масштабі. Вони забезпечують простий у користуванні інтерфейс, якщо їх обернути навколо віджетів або блоків.
- Завдяки з’єднувачам вам не потрібен спеціальний код для інтеграції з іншими програмами, що зменшує час, зусилля та кількість помилок.

- Плагіни дозволяють інтегрувати інші платформи без написання спеціального коду.

Переваги компонентної архітектури:

- Цей підхід допомагає командам створювати програми на 60% швидше порівняно з іншими методологіями. Це тому, що розробники можуть вибирати компоненти безпосередньо з бібліотек, не турбуючись про те, що ці компоненти вплинуть на безпеку, зручність використання чи продуктивність. У монолітній системі вони будували елементи з нуля.
- При такому підході технічне обслуговування простіше. Замість написання коду для налаштування окремих компонентів ви оновлюєте один компонент лише один раз, а потім додаєте його до моделі під час випуску оновлення програмного забезпечення.
- Команди можуть працювати автономно та незалежно, створюючи компоненти без сторонньої допомоги чи втручання. Це забезпечує плавний робочий процес із свободою, гнучкістю та підзвітністю.
- Цей підхід пропонує багаторазове використання, що має багато переваг. Замість того, щоб витрачати час на написання тих самих рядків коду знову і знову, розробники можуть зосередитися на основній функціональності. Крім того, компоненти можна застосовувати різними способами для різних цілей на кількох платформах.
- Усі компоненти, створені за допомогою одного документа дизайну, матимуть узгоджений інтерфейс користувача.
- Масштабованість має вирішальне значення для будь-якої нової програми. Розробники повинні бути готові до різкого зростання. Використовуючи архітектуру на основі компонентів, масштабування елементів було б схоже на складання частин головоломки, які працюють разом.
- Компонентна архітектура полегшує створення надійної, складної програми. Розробники можуть додавати багато блоків і компонентів, які вже були протестовані, без необхідності писати багато рядків коду, залишаючи місце для помилок.

Недоліки компонентної архітектури:

- Оскільки компоненти ізольовані, IT-адміністратори тестують їх окремо та разом. Це важливий процес, щоб керувати ними та впорядковувати їх, але він все одно виснажливий і трудомісткий.
- Повторне використання компонентів у різних програмах робить їх менш гнучкими.
- Компонентний підхід може потребувати значного обслуговування. Першою проблемою може бути пошук компонента, який відповідає потребам програми. Крім того, оновлювати та підтримувати бібліотеки компонентів непросто, оскільки вони потребують частого моніторингу.
- Використання занадто великої кількості компонентів у додатку чи веб-сайті може погіршити читабельність, ускладнюючи читачеві розуміння тексту.

Особливості компонентів:

- Компонент може працювати з іншими компонентами для створення нової функції.
- Розробники можуть легко замінити їх іншими компонентами.
- Вони інкапсульовані, тобто виглядають як інтерфейси, але не показують внутрішні процеси їх створення.
- Вони незалежні і не залежать від контексту.
- Вони не потребують інших компонентів, що робить їх придатними для функціонування в різних сценаріях.
- Підприємства та розробники можуть використовувати їх повторно, не змінюючи чи коригуючи.

1.3 Мікросервісна архітектура

Термін Мікросервісна архітектура, також відомий як мікросервіси, з’явився в середині 2010-х, щоб описати особливий архітектурний стиль розробки програмних додатків. Цей стиль розробки отримав розповсюдження в зв’язку з розвитком практик гнучкої розробки та DevOps. На даний момент

мікросервісній архітектурі приділяється багато уваги: статті, блоги, дискусії в соціальних мережах і презентації на конференціях. Коли йдеться про забезпечення гнучкої розробки і доставки складних корпоративних додатків — такий спосіб розробки має значні переваги [4].

Мікросервісна архітектура – це підхід до розробки програмного забезпечення, в якому додаток складається з невеликих незалежних компонентів, які називаються мікросервісами. Кожен сервіс відповідає за обробку одного або кількох пов’язаних запитів або функцій, і може бути розроблений, випробуваний та розгорнутий незалежно від інших мікросервісів [5].

Архітектура мікросервісів — це особливий метод розробки програмних систем, який намагається зосередитися на створенні одно-функціональних модулів із чітко визначеними інтерфейсами та операціями. Ця тенденція зросла в останні роки, оскільки підприємства прагнуть стати більш гнучкими та рухаються до DevOps і постійного тестування. Мікросервіси мають багато переваг для команд Agile та DevOps.

На відміну від мікросервісів, монолітна програма будується як єдина автономна одиниця. Тому будь-які зміни в додатку відбуваються повільно, оскільки впливають на всю систему. Модифікація невеликої частини коду може вимагати створення та розгортання абсолютно нової версії програмного забезпечення. Отже, масштабування певної функції програми також означає масштабування всієї програми.

Мікросервіси вирішують проблеми монолітних систем, будучи максимально модульними. У найпростішому вигляді вони допомагають створювати програму як набір невеликих служб, кожна з яких працює у своєму власному процесі та розгортається незалежно. Ці служби можуть бути написані на різних мовах програмування і можуть навіть використовувати різні методи зберігання даних.

Хоча це робить речі більш масштабованими та гнучкими, це потребує динамічного перетворення. Мікросервіси часто підключаються через API і можуть використовувати багато тих самих інструментів і рішень, які виросли в

екосистемі REST і веб-сервісів. Тестування цих API тепер не підлягає обговоренню, щоб забезпечити якісне розгортання програмного забезпечення. Він працює, перевіряючи шляхи зв’язку та потік даних у розгортанні мікросервісу.

Переваги мікросервісів:

- Простіше в розгортанні. Розгортання частинами, не впливаючи на інші служби.
- Простіше для розуміння. Простіше слідувати коду, оскільки функція є ізольованою та менш залежною.
- Багаторазове використання в бізнесі. Можна ділитися невеликими послугами, такими як платіжні системи або системи входу, у всьому бізнесі.
- Мінімізований ризик від змін. Уникнення блокування технологій або мов, можливість змінювати їх на льоту з мінімальним ризиком.

Так само, як немає офіційного визначення терміну мікросервіси, немає стандартної моделі, яку ви побачите в кожній системі, заснованій на цьому архітектурному стилі. Але більшість мікросервісних систем мають деякі спільні риси:

- Кілька компонентів. Є можливість розбити на кілька компонентів, таким чином, кожну службу можна розгортати, налаштовувати та повторно розгортати незалежно без шкоди для цілісності програми. Таким чином може знадобитися лише змінити одну окрему службу замість повторного розгортання цілих програм. Але цей підхід має свої недоліки, включаючи дорогі віддалені виклики та більшу складність під час перерозподілу обов’язків між компонентами.
- Стиль мікросервісів зазвичай організований навколо бізнес-можливостей і пріоритетів. На відміну від традиційного монолітного підходу до розробки, коли кожна з різних команд зосереджена на, скажімо, інтерфейсах користувача, базах даних, технологічних рівнях на стороні сервера, архітектура мікросервісу використовує багатофункціональні команди. Кожна команда відповідає за створення конкретних продуктів на основі окремих служб, які спілкуються через шину повідомлень.

- Мікросервіси діють дещо подібно до класичної системи UNIX: вони отримують запити, обробляють їх і відповідно генерують відповідь. Це протилежно тому, як працюють багато інших продуктів, наприклад ESB. Саме тут використовуються високотехнологічні системи для маршрутизації повідомлень та застосування бізнес-правил. Можна сказати, що мікросервіси мають розумні кінцеві точки, які обробляють інформацію та застосовують логіку, а також канали, через які проходить інформація.

- Оскільки мікросервіси включають різноманітні технології, старі методи централізованого управління не є оптимальними. Спільнота мікросервісів віддає перевагу децентралізованому управлінню, щоб її розробники могли створювати інструменти, які можуть використовувати інші для вирішення тих самих проблем. Як і децентралізоване управління, архітектура мікросервісів надає перевагу децентралізованому управлінню даними. Монолітні системи використовують єдину логічну базу даних для різних програм. У програмі мікросервісу кожна служба зазвичай керує своєю унікальною базою даних.

- Мікросервіси створені, щоб справлятися з невдачами. Оскільки взаємодіють кілька різноманітних служб, цілком можливо, що служба може вийти з ладу. У цих випадках клієнт повинен дозволити своїм сусіднім службам функціонувати, поки він відхиляється. Однак моніторинг мікросервісів може допомогти запобігти ризику збою. Зі зрозумілих причин ця вимога ускладнює мікросервіси порівняно з монолітною системною архітектурою.

- Це еволюційний дизайн і, знову ж таки, ідеальний для еволюційних систем, де ви не можете повністю передбачити, які майбутні пристрої матимуть доступ до вашої програми. Багато додатків починаються на основі монолітної архітектури, але, оскільки з’явилися кілька непередбачених вимог, їх можна повільно переробити до мікросервісів, які взаємодіють із старішою монолітною архітектурою через API.

Як і в будь-якому іншому випадку, чи підходить вам архітектура мікросервісу, залежить від вимог, оскільки всі вони мають свої плюси та мінуси.

Переваги мікросервісної архітектури:

- Дає розробникам свободу самостійно розробляти та розгортати служби.
- Може бути розроблений досить невеликою командою.
- Коди для різних сервісів можуть бути написані різними мовами (хоча багато практиків не рекомендують цього).
- Проста інтеграція та автоматичне розгортання (з використанням інструментів постійної інтеграції з відкритим кодом).
- Простий для розуміння та модифікації для розробників, тому може допомогти новому члену команди швидко стати продуктивним.
- Розробники можуть використовувати новітні технології Код організовано навколо бізнес-можливостей.
- Запускає веб-контейнер швидше, тому розгортання також відбувається швидше.
- Якщо потрібні зміни в певній частині програми, можна змінити та повторно розгорнути лише відповідну службу – не потрібно змінювати та повторно розгортати всю програму.
- Краща ізоляція помилок: якщо один мікросервіс вийде з ладу, інший продовжить працювати.
- Легко масштабувати та інтегрувати зі сторонніми службами.
- Немає довгострокових зобов’язань щодо стеку технологій.

Недоліки мікросервісної архітектури:

- Через розподілене розгортання тестування може стати складним і виснажливим – і часто перешкоджає деяким перевагам масштабування мікросервісів.
- Збільшення кількості послуг може призвести до виникнення інформаційних бар’єрів.
- Додаткова складність, оскільки розробники зменшують відмовостійкість, мережеву затримку та різні формати повідомлень, а також балансування навантаження.

- Будучи розподіленою системою, це може призвести до дублювання зусиль.
- Коли кількість послуг збільшується, інтеграція та управління цілими продуктами може ускладнитися.
- Розробникам доводиться мати справу з додатковою складністю розподіленої системи.
- Розробники повинні реалізувати механізм зв'язку між сервісами.
- Обробка випадків використання, які охоплюють більше однієї служби без використання розподілених транзакцій, не тільки складна, але й вимагає співпраці між різними командами.

Традиційна SOA є ширшою структурою і може означати різні речі. Деякі прихильники мікросервісів взагалі відкидають тег SOA, а інші вважають мікросервіси ідеальною вдосконаленою формою SOA. У будь-якому випадку є досить чіткі відмінності, щоб виправдати окрему концепцію мікросервісу. Типова модель SOA, зазвичай має більше залежних ESB, а мікросервіси використовують швидші механізми обміну повідомленнями. SOA також зосереджена на імперативному програмуванні, тоді як архітектура мікросервісів зосереджена на стилі програмування адаптивного актора. Крім того, моделі SOA, як правило, мають велику реляційну базу даних, тоді як мікросервіси часто використовують бази даних NoSQL. Але справжня різниця пов’язана з методами архітектури, які використовуються для отримання інтегрованого набору послуг. Оскільки в цифровому світі все змінюється, гнучкі методи розробки, які можуть йти в ногу з вимогами еволюції програмного забезпечення, є неоціненними.

Більшість методів, які використовуються в архітектурі мікросервісів, походять від розробників, які створили програмні додатки для великих корпоративних організацій і які знають, що сучасні кінцеві користувачі очікують динамічного, але узгодженого досвіду на широкому діапазоні пристроїв. Масштабовані, адаптовані, модульні та швидкодоступні хмарні програми користуються великим попитом. І це змусило багатьох розробників змінити свій підхід.

Незалежно від того, чи стане архітектура мікросервісів улюбленим стилем розробників у майбутньому, очевидно, це потужна ідея, яка пропонує серйозні переваги для розробки та впровадження корпоративних програм. Багато розробників і організацій, ніколи не вживаючи назви і навіть не називаючи свою практику SOA, використовували підхід до використання API, які можна класифікувати як мікросервіси.

1.4 Гібридна архітектура

Одним з нових методів організації обчислень є гібридна архітектура кластерних обчислень. Основна ідея гібридної архітектури організації обчислень ґрунтується на твердженні, що оскільки значна частина задач великої розмірності, що розв'язуються методами паралельного програмування, завдяки своїй структурі може бути розв'язана на графічних процесорах зі значно нижчими затратами часу, ніж на центральному процесорі, то можна досягти значного збільшення швидкодії і рівня паралелізму кластерних обчислень [6].

У сфері архітектури програмного забезпечення не існує універсального рішення, яке б підходило всім. Кожна організація має унікальні потреби, застарілі системи та різний рівень ресурсів. Усвідомлюючи це, деякі компанії прийняли гібридну архітектуру, яка поєднує в собі сильні сторони як мікросервісів, так і монолітів.

Гібридна архітектура означає навмисне поєднання мікросервісів і монолітних компонентів в одній програмі або системі. Цей підхід визнає, що не всі частини програми можуть отримати користь від мікросервісів, і спроба повної міграції може бути складною або непрактичною. Стратегічно використовуючи мікросервіси та моноліти, організації можуть досягти балансу між гнучкістю, масштабованістю та збереженням наявної функціональності.

Переваги гібридної архітектури:

- Поступова міграція: застосування гібридної архітектури дозволяє організаціям поступово переходити від монолітної системи до мікросервісів.

Визначивши конкретні області, які виграють від гнучкості мікросервісів, ці компоненти можна розробити та інтегрувати разом із існуючою монолітною програмою, мінімізуючи збої та ризики.

- Залучення наявних інвестицій: у багатьох випадках монолітні програми вдосконалювалися протягом багатьох років і мають цінну бізнес-логіку та функціональність. Інтеграція мікросервісів в існуючий моноліт дає можливість організаціям використовувати свої інвестиції замість повного переписування. Це максимізує віддачу від інвестицій і зберігає інтелектуальну власність, вбудовану в монолітні компоненти.

- Покращена масштабованість: мікросервіси чудово забезпечують масштабованість для певних служб або функцій у програмі. Ізолюючи та відокремлюючи ці області як мікросервіси, організації можуть самостійно масштабувати їх за потреби, зберігаючи ефективну роботу решти монолітної системи. Такий підхід дозволяє ефективніше використовувати ресурси та краще реагувати на мінливі вимоги.

- Гнучкість та інновації: мікросервіси пропонують розробникам свободу вибору різних технологій, мов програмування та фреймворків. Впроваджуючи мікросервіси поряд із монолітом, організації можуть використовувати сучасні практики та технології розробки, зберігаючи при цьому стабільність і надійність існуючої системи. Ця гнучкість сприяє інноваціям і дає змогу командам розробників досліджувати нові рішення.

Недоліки гібридної архітектури:

- Складність: Управління гібридною архітектурою створює складність, оскільки організації повинні забезпечити безперебійну інтеграцію та зв’язок між мікросервісами та монолітами. Чітко визначені інтерфейси, протоколи зв’язку та шаблони інтеграції є важливими для підтримки цілісної системи.

- Узгодженість даних: оскільки дані можуть спільно використовуватися між монолітними компонентами та мікросервісами, забезпечення узгодженості та синхронізації даних стає критичним. Організації повинні розробити ефективні

стратегії керування та інтеграції даних, щоб запобігти невідповідності даних і підтримувати цілісність системи.

- Операційні витрати: Управління двома різними архітектурними парадигмами вимагає додаткових операційних витрат. Організації повинні інвестувати в створення та підтримку необхідної інфраструктури, моніторингу, розгортання та механізмів масштабування як для мікросервісів, так і для монолітних компонентів.

Гібридна архітектура, яка поєднує мікросервіси та моноліти, пропонує організаціям практичний підхід до модернізації їхніх додатків, одночасно використовуючи наявні інвестиції та функціональність. Ретельно визначаючи сфери, які можуть отримати користь від мікросервісів, організації можуть досягти покращення.

Висновки до розділу 1

У цьому розділі ми розглянули різноманітні способи побудови архітектури програмного забезпечення, включаючи клієнт-серверну, компонентну, мікросервісну та гібридну архітектури. Кожен з цих підходів має свої переваги і недоліки, і вибір конкретного типу архітектури залежить від потреб проекту, його масштабу та вимог до функціональності.

Клієнт-серверна архітектура є однією з найпоширеніших та найстаріших моделей, що використовується для побудови систем. Вона розділяє функціональність між клієнтськими та серверними компонентами, що дозволяє досягти високої масштабованості та ефективності.

Компонентна архітектура використовує підхід розробки програмного забезпечення, у якому система розбивається на окремі компоненти або модулі, які можуть бути розроблені та підтримані незалежно один від одного. Це сприяє перевикористанню коду, полегшує тестування та забезпечує більшу гнучкість системи.

Мікросервісна архітектура стає все більш популярною в останні роки, особливо в умовах росту інтернет-орієнтованих додатків. Вона базується на розбитті системи на невеликі, незалежні сервіси, які можуть бути розвинуті, впроваджені та масштабовані окремо один від одного.

Гібридна архітектура, у свою чергу, поєднує різні підходи та моделі для досягнення оптимальних результатів для конкретного проекту. Вона може використовувати елементи клієнт-серверної, компонентної та мікросервісної архітектур, комбінуючи їх переваги з урахуванням потреб системи.

Загалом, вибір певного типу архітектури програмного забезпечення повинен бути обдуманим та обґрунтованим, враховуючи потреби та особливості конкретного проекту, його масштаб та прогнозовану динаміку розвитку.

РОЗДІЛ 2
ДОСЛІДЖЕННЯ ЗАГРОЗ КОМПОНЕНТАМ АРХІТЕКТУРИ

2.1 Види атак на рівні мережі

Сервіси спілкуються один з одним за допомогою повідомлень через мережу. Тому одним із векторів атаки є канали зв’язку для надсилання повідомлень. Ці атаки спрямовані на канали, які використовуються службами для зв’язку одна з одною, наприклад, шляхом перехоплення або маніпулювання мережевим трафіком.

Атака на відмову в обслуговуванні – напад на комп'ютерну систему з наміром зробити комп'ютерні ресурси недоступними користувачам, для яких комп'ютерна система була призначена. Одним із найпоширеніших методів нападу є насичення атакованого комп'ютера або мережевого устаткування великою кількістю зовнішніх запитів таким чином атаковане устаткування не може відповісти користувачам, або відповідає настільки повільно, що стає фактично недоступним [7].

DDoS, або розподілена відмова в обслуговуванні, — це тип кібератаки, який намагається зробити веб-сайт або мережевий ресурс недоступним шляхом заповнення його зловмисним трафіком, щоб він не міг працювати.

У розподіленій атаці типу «відмова в обслуговуванні» (DDoS) зловмисник переповнює свою ціль небажаним інтернет-трафіком, так що звичайний трафік не може досягти свого призначення.

На високому рівні атака DDoS або DoS схожа на неочікувану пробку, спричинену сотнями фіктивних запитів на перевезення. Здається, запити є законними для служб спільного перевезення, і вони відправляють водіїв на пікап, який неминуче забиває вулиці міста. Це перешкоджає регулярному законному трафіку прибути до місця призначення.

Під час DDoS-атаки зловмисники використовують велику кількість зловмисних машин і підключених пристроїв в Інтернеті, включаючи пристрої Інтернету речей, смартфони, персональні комп'ютери та мережеві сервери, щоб відправити потік трафіку до цілей.

DDoS-атака на веб-сайт компанії, веб-програму, API, мережу або інфраструктуру центру обробки даних може призвести до простою та перешкодити законним користувачам купувати продукти, користуватися послугами, отримувати інформацію чи будь-який інший доступ.

DDoS-атаки використовують мережі пристроїв, підключених до Інтернету, щоб відрізати користувачів від сервера або мережевого ресурсу, наприклад веб-сайту чи програми, до якого вони часто мають доступ.

Щоб розпочати DDoS-атаку, зловмисники використовують зловмисне програмне забезпечення або користуються вразливими місцями безпеки, щоб зловмисно заразити та отримати контроль над машинами та пристроями. Кожен комп'ютер або інфікований пристрій, званий «ботом» або «зомбі», стає здатним поширювати зловмисне програмне забезпечення далі та брати участь у DDoS-атаках. Ці боти утворюють армії ботів, які використовують свою силу в кількості та збільшують масштаб атаки. А оскільки зараження пристроїв Інтернету речей часто залишається непоміченим — як той надокучливий зомбі з фільму B, про якого ви навіть не підозрювали, що він заражений — законні власники пристроїв стають вторинними жертвами або невідомими учасниками, а зловмисникам залишається важко ідентифікувати постраждалу організацію.

Після того, як зловмисник створив ботів, він може надсилати віддалені інструкції кожному боту, спрямовуючи DDoS-атаку на цільову систему. Коли бот атакує мережу або сервер, зловмисник дає вказівку окремим ботам надсилати запити на IP-адресу жертви. Подібно до того, як ми, люди, маємо єдині у своєму роді відбитки пальців, наші пристрої мають унікальну адресу, яка ідентифікує їх в Інтернеті чи локальній мережі. Переважний трафік призводить до відмови в обслуговуванні, перешкоджаючи нормальному трафіку отримати доступ до веб-сайту, веб-програми, API або мережі.

Іноді бот-мережі з їхніми мережами скомпрометованих пристроїв орендуються для інших потенційних атак через послуги «атаки за наймом». Це дозволяє людям зі зловмисними намірами, але без підготовки чи досвіду, легко запускати DDoS-атаки самостійно.

Існує багато різних типів DDoS-атак, і зловмисники часто використовують більше ніж один тип, щоб завдати шкоди своїм цілям. Три ключові типи – об'ємні атаки, атаки на протокол і атаки на прикладному рівні. Метою всіх атак є суттєве уповільнення або зупинка законного трафіку в досягненні його призначення. Наприклад, це може означати заборону користувачеві отримати доступ до веб-сайту, купити продукт або послугу, переглянути відео або взаємодіяти в соціальних мережах. Крім того, роблячи ресурси недоступними або знижуючи продуктивність, DDoS може призвести до зупинки бізнесу. Це може призвести до того, що співробітники не зможуть отримати доступ до електронної пошти чи веб-додатків або працюватимуть у звичайному режимі.

Існують різні шляхи, якими можуть скористатися зловмисники. Модель взаємозв'язку відкритих систем — це багаторівнева основа для різних мережевих стандартів і містить сім різних рівнів. Кожен рівень моделі OSI має унікальне призначення, як поверхи офісної будівлі, де на кожному поверсі виконуються різні бізнес-функції. Зловмисники націлюються на різні рівні залежно від того, який тип веб-ресурсу чи активу, що виходить в Інтернет, вони хочуть порушити.

Намір DDoS-атаки на основі обсягу полягає в тому, щоб перевантажити мережу величезним обсягом трафіку шляхом насичення пропускної здатності передбачуваного ресурсу жертви. Велика кількість трафіку атаки блокує законним користувачам доступ до програми чи служби, запобігаючи надходженню та виходу трафіку. Залежно від мети, зупинка законного трафіку може означати, що клієнт банку може не сплатити рахунок вчасно, покупці електронної комерції не зможуть завершити онлайн-транзакції, пацієнту лікарні може бути виключено з його медичних записів, або громадянин може виявитися нездатним переглядати свої податкові записи в державному органі. Незалежно

від організації, блокування людей у службі, якою вони планують користуватися онлайн, має негативний вплив.

Об’ємні атаки використовують ботів, створені за допомогою армій окремих заражених шкідливим програмним забезпеченням систем і пристроїв. Керовані зловмисником, боти використовуються, щоб спричинити перевантаження між ціллю та Інтернетом загалом із шкідливим трафіком, який насичує всю доступну пропускну здатність.

Непередбачений натиск бот-трафіку може значно сповільнити або запобігти доступ до веб-ресурсу чи інтернет-сервісу. Оскільки боти захоплюють легальні пристрої, щоб посилити DDoS-атаки, які потребують великої пропускної спроможності, часто несвідомо для користувача, організації-жертві важко виявити зловмисний трафік.

Існують різноманітні об’ємні вектори атак DDoS, які використовуються суб’єктами загрози. Багато використовують методи атак відображення та посилення, щоб перевантажити цільову мережу чи службу.

UDP-флуди часто вибирають для DDoS-атак з більшою пропускною здатністю. Зловмисники намагаються перевантажити порти на цільовому хості IP-пакетами, що містять протокол UDP без збереження стану. Хост-жертва потім шукає програми, пов’язані з UDP-пакетами, і, якщо їх не знайдено, надсилає відправнику повідомлення «Destination Unreachable». IP-адреси часто підробляють, щоб анонімізувати зловмисника, і як тільки цільовий хост стає завалений трафіком атаки, система перестає реагувати та стає недоступною для законних користувачів.

Атаки на систему доменних імен або відображення DNS є поширеним типом вектора атак, коли кіберзлочинці або хакери підробляють IP-адресу своєї цілі, щоб надсилати велику кількість запитів до відкритих серверів DNS. У відповідь ці DNS-сервери відповідають на зловмисні запити підробленої IP-адреси, тим самим створюючи атаку на призначену ціль через потоки відповідей DNS. Дуже швидко великий обсяг трафіку, який створюється з відповідей DNS,

переповнює служби організації-жертви, роблячи їх недоступними та не дозволяючи законному трафіку досягти цільового призначення.

Протокол ICMP в основному використовується для обміну повідомленнями про помилки та зазвичай не обмінюється даними між системами. Пакети ICMP можуть супроводжувати пакети протоколу керування передачею (TCP), які дозволяють прикладним програмам і комп'ютерним пристроям обмінюватися повідомленнями через мережу під час підключення до сервера. ICMP-флуд – це метод DDoS-атаки рівня 3, який використовує повідомлення ICMP для перевантаження пропускної здатності цільової мережі.

Протокольні атаки намагаються використати та виснажити обчислювальну потужність різноманітних ресурсів мережевої інфраструктури, таких як сервери чи брандмауери, за допомогою зловмисних запитів на з'єднання, які використовують зв'язок протоколу. Потоки синхронізації і Smurf DDoS є двома поширеними типами DDoS-атак на основі протоколу. Атаки на протоколи можна вимірювати в пакетах за секунду, а також бітах за секунду.

Одним із основних способів підключення людей до Інтернет-додатків є протокол керування передачею. Для цього з'єднання потрібне тристороннє рукостискання від служби TCP, наприклад веб-сервера, і передбачає надсилання пакета SYN (синхронізації), звідки користувач підключається до сервера, який потім повертає пакет SYN-ACK (підтвердження синхронізації), який остаточно відповідає останнім повідомленням ACK (підтвердження) для завершення рукостискання TCP.

Під час атаки SYN flood зловмисний клієнт надсилає велику кількість пакетів SYN, але ніколи не надсилає підтвердження для завершення рукостискання. Через це сервер чекає відповіді на ці напіввідкриті TCP-з'єднання, які зрештою вичерпуються для прийняття нових з'єднань для служб, які відстежують стани з'єднань.

Атака SYN flood схожа на жахливу витівку всього випускного класу справді великої середньої школи, де кожен учень дзвонить в одну піцерію та замовляє пиріг протягом того самого часу. Потім, коли кур'єр йде пакувати речі,

вона розуміє, що піци забагато, щоб помістится в її машину, а в замовленнях немає адрес, тому доставка припиняється.

Назва цієї DDoS-атаки заснована на концепції, згідно з якою численні крихітні зловмисники можуть здолати набагато більшого супротивника величезним обсягом, як і вигадана колонія маленьких блакитних гуманоїдів.

Під час розподіленої атаки типу «відмова в обслуговуванні» Smurf велика кількість пакетів ICMP із підробленим вихідним IP-адресою передбачуваної цілі транслюється в комп'ютерну мережу за допомогою широкомовної IP-адреси. За умовчанням більшість пристроїв у мережі відповідатимуть, надсилаючи відповідь на вихідну IP-адресу. Залежно від кількості комп'ютерів у мережі, комп'ютер жертви може бути сповільненим до повзання від переповненого трафіком.

Атаки на прикладному рівні, що здійснюються шляхом переповнення додатків шкідливими запитами, вимірюються в запитах за секунду. Ці атаки також називаються DDoS-атаками рівня 7. Ці атаки спрямовані на певні веб-програми, а не на цілі мережі, і порушують їх роботу. Хоча їх важко запобігти та пом'якшити, вони є одними з найлегших DDoS-атак.

Однією з найстаріших і найнебезпечніших атак є атака типу Man-In-The-Middle (MITM). Дослівно це перекладається як «людина посередині», тобто коли хакер виступає в ролі посередника при передачі інформації. Цей тип атак є розповсюдженим та руйнівним. Суть атаки man-in-the-middle доволі проста: злочинець таємно перехоплює трафік з одного комп'ютера та відправляє його кінцевому одержувачу, попередньо прочитавши та змінивши на свою користь. Атаки MITM надають можливість робити такі дії як підміна криптовалютного гаманця для викрадення коштів, перенаправлення браузера на шкідливий веб-сайт або ж просто пасивний збір інформації з метою її подальшого злочинного використання. Найбільш очевидний спосіб, коли хтось може це зробити, коли ви користуєтесь незашифрованою, загальнодоступною мережею Wi-Fi, як у аеропортах чи кафе. MITM-атаки трапляються і на мережевому рівні [8].

Атаки людина посередині (MITM) представляють значну загрозу, здатну захопити контроль над сеансами зв'язку. У цих атаках посередник, розташований між двома сторонами, таємно перехоплює сеанс, маніпулюючи процесом спілкування без відома сторін. Це дає зловмиснику змогу викрасти інформацію або присвоїти фальшиві особи, щоб отримати доступ до систем обслуговування. Атаки MITM охоплюють різні конкретні методи, зокрема підробку Wi-Fi, викрадення електронної пошти, підробку DNS і викрадення SSL. Вони часто використовуються для крадіжки особистих даних, таких як облікові дані для входу, електронні листи та фінансові рахунки, сіючи хаос в таких онлайн-системах, як електронний банкінг, ігрові онлайн-платформи та цифрові транзакції.

Атака MITM складається з двох основних етапів:

- Нападник намагається розташуватися між двома сторонами, що спілкуються, щоб перехопити їхній трафік зв’язку. Це дозволяє їм викрадати дані або використовувати підроблені ідентифікаційні дані, щоб отримати доступ до систем обслуговування.
- Після введення в канал зв’язку зловмисник отримує можливість маніпулювати зв’язком між двома сторонами, ініціюючи такі операції, як крадіжка даних або використання підроблених ідентифікаторів для доступу до систем обслуговування.

Поширені типи атак MITM:

- Один із найпростіших і найбільш часто використовуваних методів для виконання атак MITM передбачає створення зловмисної точки доступу Wi-Fi. Зловмисник встановлює точку доступу з іменем, яке виглядає легітимним, наприклад, ім’я сусіднього кафе. Ця точка доступу зазвичай не має шифрування, що робить її вразливою для використання. Коли нічого не підозрюючи користувач підключається до цієї зловмисної точки доступу Wi-Fi, увесь наступний трафік зв’язку перехоплюється зловмисником, що дозволяє викрасти особисту інформацію.

- Підробка ARP, також відома як отруєння ARP, передбачає маніпуляції зловмисником з кешем ARP користувача для перенаправлення трафіку користувача до системи зловмисника. Коли користувачі локальної мережі ініціюють запити на доступ, вони зазвичай пересилаються через шлюз. Спочатку користувач надсилає ARP-запит для отримання MAC-адреси, пов'язаної з IP-адресою шлюзу. Використовуючи це, зловмисник маскується під шлюз і відповідає користувачеві з його власною MAC-адресою. Відповідно, користувач оновлює свій ARP-кеш неправильною MAC-адресою. У результаті весь наступний трафік від користувача ненавмисно спрямовується до системи зловмисника.

- Підробка DNS, також відома як викрадення DNS, передбачає втручання в процес вирішення DNS для перенаправлення користувачів на шахрайські веб-сайти. Спочатку, коли користувач намагається отримати доступ до Інтернету, він надсилає DNS-запит на DNS-сервер, щоб отримати IP-адресу, пов'язану з доменним іменем. Згодом DNS-сервер відповідає зіставленням між ім'ям домену та його відповідною IP-адресою. Однак зловмисники можуть втрутитися в цей процес, змінивши IP-адресу, прив'язану до доменного імені, таким чином перенаправляючи доступ користувача.

- Зловмисники використовують різні методи для викрадення серверів електронної пошти банків або фінансових установ, на яких часто розміщено численні облікові записи електронної пошти користувачів. Отримавши несанкціонований доступ до цих серверів, зловмисники можуть відстежувати електронні листи користувачів і, у деяких випадках, видавати себе за фінансову установу для надсилання шахрайських електронних листів окремим користувачам. Ця тактика дає їм змогу отримувати конфіденційну інформацію користувачів і вводити користувачів в оману для здійснення транзакцій, таких як неавторизовані банківські перекази.

- Зараз доступ до більшості веб-сайтів здійснюється через HTTPS, де між користувачами та серверами веб-сайтів встановлюються з'єднання SSL, що забезпечує цілісність і конфіденційність даних за допомогою сертифікатів SSL.

Прийняття HTTPS допомогло певною мірою пом’якшити атаки MITM. Однак зловмисники наполегливо використовують уразливості, використовуючи такі методи, як викрадення SSL, щоб порушити безпеку HTTPS.

Спуфінг – це кібератака, яка відбувається, коли шахрай маскується під надійне джерело для отримання доступу до важливих даних або інформації. Спуфінг може відбуватися через веб-сайти, електронні листи, телефонні дзвінки, текстові повідомлення, IP-адреси та сервери [9].

Через маніпуляції з IP- та MAC-адресами атака ARP-спуфінгу може змінити спосіб взаємодії пристроїв у вашій мережі між собою. Це дозволяє зловмисникам перехоплювати трафік між вашими пристроями та потенційно викрадати конфіденційну інформацію.

Хоча ARP-спуфінг корисний для тестувальників проникнення для оцінки стану безпеки мережі, він становить значний ризик для синіх команд, оскільки його можна використовувати як сходинку для здійснення різних інших типів атак, таких як людина посередині і атаки на відмову в обслуговуванні.

Знання протоколу ARP та його обмежень має першочергове значення, щоб убезпечитися від цієї атаки та її потенційних загроз. Ця стаття надасть вам додаткові технічні відомості про цю атаку, дослідить деякі інструменти та методи, які часто використовуються, а також відповість на деякі поширені запитання.

Як правило, у локальній мережі пристрої спілкуються один з одним за допомогою двох типів адрес: IP для логічної адресації та MAC для фізичної адресації.

Щоб полегшити зв’язок через фізичну мережу, був розроблений протокол ARP, щоб зіставити IP-адреси третього рівня пристроїв із відповідними MAC-адресами другого рівня.

Таким чином, протокол ARP дозволяє пристроям у мережі знаходити та ідентифікувати один одного за допомогою своїх IP- та MAC-адрес.

Протокол ARP використовує два різні типи пакетів, які називаються відповідно ARP-запитом і ARP-відповіддю, у поєднанні з попередньо показаною таблицею ARP для зберігання відповідності між IP- та MAC-адресами.

ARP працює шляхом трансляції пакета запиту ARP, що містить IP-адресу цільового пристрою, а потім отримання пакета відповіді ARP, що містить MAC-адресу цього пристрою.

Підробка ARP — це форма атаки, під час якої зловмисник надсилає фальсифіковані повідомлення ARP через локальну мережу. Це призводить до пов’язування MAC-адреси зловмисника з IP-адресою законного комп’ютера або сервера в мережі. Таким чином зловмисник опиниться в центрі зв’язку між вузлами мережі, що дозволить йому отримати доступ до переданих даних, змінити трафік або навіть виконати атаки на відмову в обслуговуванні.

Підробка/отруєння ARP змінює ARP-кеш пристроїв, щоб перенаправити трафік на їхні машини. Зазвичай цього можна досягти двома способами: безпосередньо отруїти хости або шлюз мережі.

Протягом багатьох років було розроблено кілька інструментів для дослідницьких цілей і перевірки стану безпеки мережі. Деякі інструменти, які можуть полегшити атаку ARP Spoofing:

- Arpspoof – це інструмент із пакету Dsniff, який можна використовувати для надсилання підроблених повідомлень ARP у локальній мережі. Його можна використовувати для перенаправлення трафіку або для здійснення атак типу "людина посередині".
- Ettercap – це безкоштовний інструмент мережевої безпеки з відкритим вихідним кодом, який спочатку був розроблений для перехоплення мережі, але згодом був розширений, щоб включити інші атаки типу "людина посередині", зокрема ARP-спуфінг.
- Bettercap – це універсальний мережевий хакерський інструмент, написаний на Go. Інструмент містить розширюваний набір функцій для різних типів цілей, включаючи мережі WiFi, пристрої BLE, бездротові пристрої HID і підтримку мереж ipv4 та Ipv6.

Атаки підслуховування можуть здійснюватися зловмисниками, які відстежують мережевий трафік, щоб перехопити або підслухати дані, які надсилаються або отримують користувачі. Цей тип атаки може здійснюватися на бездротові мережі, з'єднання Bluetooth або інші канали зв'язку [10].

Атака підслуховування відбувається, коли хакер перехоплює, видаляє або змінює дані, які передаються між двома пристроями. Прослуховування, також відоме як перехоплення або стеження, покладається на незахищений мережевий зв'язок для доступу до даних, що передаються між пристроями. Це зазвичай відбувається, коли користувач підключається до мережі, у якій трафік не захищений або не зашифрований, і надсилає конфіденційні бізнес-дані колезі. Дані передаються через відкриту мережу, що дає зловмиснику можливість використовувати вразливість і перехоплювати її різними методами. Атаки підслуховування часто буває важко помітити. На відміну від інших форм кібератак , наявність помилки або пристрою для прослуховування не може негативно вплинути на роботу пристроїв і мереж.

За допомогою підслуховування зловмисники можуть використовувати різні методи для здійснення атак, які зазвичай включають використання різних пристроїв для підслуховування для прослуховування розмов і перегляду мережевої активності.

Типовим прикладом електронного підслуховуючого пристрою є прихований жучок, фізично розміщений у будинку чи офісі. Це може статися, якщо залишити жучка під стільцем чи на столі або сховати мікрофон у непомітному предметі, наприклад ручці чи сумці. Це простий підхід, але він може призвести до встановлення складніших пристроїв, які важко виявити, наприклад мікрофонів у лампах чи стельових світильниках, книг на книжковій полиці чи в рамках для картин на стіні.

Сучасні комп'ютеризовані телефонні системи дозволяють перехоплювати телефонні розмови електронним способом без прямого доступу до пристрою. Зловмисники можуть посилати сигнали по телефонній лінії та передавати будь-які розмови, які відбуваються в одній кімнаті, навіть якщо трубка неактивна.

Подібним чином комп’ютери мають складні інструменти зв’язку, які дозволяють зловмисникам перехоплювати комунікаційну діяльність, починаючи з голосових розмов, онлайн-чатів і навіть помилок у клавіатурах, щоб реєструвати текст, який вводять користувачі.

Комп’ютери також випромінюють електромагнітне випромінювання, за допомогою якого досвідчені перехоплювачі можуть реконструювати вміст екрана комп’ютера. Ці сигнали можна передати на відстані до кількох сотень футів і поширити далі через кабелі та телефонні лінії, які можна використовувати як антени.

Зловмисники можуть використовувати пристрої, які вловлюють звук або зображення, такі як мікрофони та відеокамери, і перетворювати їх в електричний формат для прослуховування цілей. В ідеалі це буде електричний пристрій, який використовує джерела живлення в цільовій кімнаті, що позбавляє зловмисника необхідності доступу до кімнати, щоб зарядити пристрій або замінити його батареї.

Деякі пристрої для прослуховування здатні зберігати цифрову інформацію та передавати її на пункт прослуховування. Зловмисники також можуть використовувати міні-підсилювачі, які дозволяють видаляти фоновий шум.

Канал передачі між пристроєм перехоплення та приймачем зловмисника можна прослухати з метою прослуховування. Це можна зробити за допомогою радіочастотної передачі або дроту, який включає активні або невикористовувані телефонні лінії, електричні дроти або незаземлені електричні канали. Деякі передавачі можуть працювати безперервно, але більш складний підхід передбачає дистанційну активацію.

Слабкі паролі полегшують зловмисникам отримання несанкціонованого доступу до облікових записів користувачів, що відкриває їм шлях до корпоративних систем і мереж. Це включає в себе можливість хакерів зламати конфіденційні канали зв’язку, перехоплювати діяльність і розмови між колегами, а також викрадати конфіденційні чи цінні бізнес-дані.

Користувачі, які підключаються до відкритих мереж, які не вимагають паролів і не використовують шифрування для передачі даних, створюють ідеальну ситуацію для зловмисників для підслуховування. Хакери можуть стежити за діяльністю користувачів і стежити за спілкуванням, що відбувається в мережі.

2.2 Види атак на сервіси архітектури

Атаки спрямовані на певні служби, наприклад намагаються подолати елементи керування автентифікацією та авторизацією або запровадження шкідливого коду.

Уразливість порушення контролю доступу — це недолік безпеки, який дозволяє неавторизованим користувачам отримувати доступ, змінювати або видаляти дані, до яких вони не повинні мати доступу. Ця вразливість вважається однією з найбільш критичних загроз безпеці веб-додатків. Це відбувається, коли програма не може належним чином забезпечити контроль доступу, дозволяючи зловмисникам обійти авторизацію та виконувати завдання, як якщо б вони були законним користувачем.

Ця вразливість може існувати в різних формах, таких як неадекватне керування сеансами, неправильне застосування ролевих елементів керування доступом або небезпечні прямі посилання на об’єкти. Розробники та фахівці з безпеки зобов’язані розуміти ризики, пов’язані з порушенням контролю доступу, і вживати необхідних заходів для їх зменшення.

Open Web Application Security Project зазначає несправний контроль доступу як критичний ризик безпеки веб-додатків №1 (згідно зі списком OWASP Top 10, оновленим у 2021 році) [11].

Наслідки порушення контролю доступу можуть бути катастрофічними для організацій. Несанкціонований доступ до конфіденційних даних може призвести до витоку даних, крадіжки особистих даних, фінансових втрат і шкоди репутації

компанії. У найгіршому випадку це може навіть призвести до повного злому системи, коли зловмисники отримають повний контроль над системою.

Ризик, пов’язаний із порушенням контролю доступу, високий, оскільки це безпосередньо впливає на конфіденційність, цілісність і доступність даних. Зловмисник, який використовує цю вразливість, потенційно може отримати доступ, змінити або видалити будь-які дані в системі. Це включає дані користувача, дані системи, дані програм тощо. Чим більша система та чутливіші дані, тим вищий ризик.

Порушений контроль доступу — це вразливість, яку не можна ігнорувати, і організації повинні вживати профілактичних заходів для її виявлення та пом’якшення. Це передбачає регулярне тестування безпеки, належне проектування та впровадження засобів контролю доступу, а також постійний моніторинг і оновлення заходів безпеки.

Існує кілька способів, за допомогою яких зловмисник може використати вразливості зламаного контролю доступу:

- Маніпуляції з URL-адресами — це простий метод, який використовують зловмисники для використання вразливостей у порушеному контролі доступу. Це передбачає зміну URL-адреси з метою обійти контроль доступу та отримати несанкціонований доступ до конфіденційних даних або функцій. Якщо програма не забезпечує належним чином контроль доступу, зловмисник може просто змінити URL-адресу для доступу до обмежених ресурсів.
- Кінцеві точки — це точки взаємодії між програмою та рештою системи. Це можуть бути API, мікросервіси або будь-які інші служби, на які покладається програма. Якщо ці кінцеві точки не захищені належним чином, вони можуть бути використані зловмисниками для обходу контролю доступу.
- Поширеним методом, який використовують зловмисники, є підвищення привілеїв. Це передбачає отримання несанкціонованого доступу до облікового запису нижчого рівня з подальшим підвищенням привілеїв цього облікового запису для отримання доступу до більш конфіденційних даних або функцій.

• Незахищені прямі посилання на об’єкти – це тип уразливості порушеного контролю доступу, коли програма відкриває прямі посилання на внутрішні об’єкти впровадження. Це може включати ключі бази даних, шляхи до файлів або будь-які інші внутрішні посилання. Якщо зловмисник може здогадатися або застосувати ці посилання грубою силою, він зможе обійти елементи керування доступом і отримати прямий доступ до конфіденційних даних.

Впровадження коду — це термін, який використовується для опису атак, які вводять код у програму. Цей введений код потім інтерпретується програмою, змінюючи спосіб виконання програми. Атаки з впровадженням коду зазвичай використовують вразливість додатка, що дозволяє обробляти недійсні дані. Цей тип атак використовує погану обробку ненадійних даних, і ці типи атак зазвичай стають можливими через відсутність належної перевірки вхідних/вихідних даних. Зловмисники можуть ввести код у комп’ютерну програму з таким типом уразливості.

Ін’єкційна атака – це форма кібератаки, під час якої інформація надсилається, щоб змінити інтерпретацію системою команд. Під час ін'єкційної атаки зловмисник надсилає шкідливу інформацію інтерпретатору. Ін’єкційну атаку можна здійснити на дані з багатьох різних місць, як-от змінні середовища, параметри, онлайн-сервіси та типи користувачів, але не тільки вони. Топ-10 OWASP висвітлює ін’єкційні атаки як критичну вразливість у програмах [13].

Ін’єкційні атаки можуть спричинити втрату даних, пошкодження даних, порушення безпеки та, можливо, втрату контролю над цільовим хостом і розкриття конфіденційної інформації, пов’язаної з хостом. Успішна ін’єкція також може дозволити зловмисникам отримати доступ до бази даних без дозволу. Це дозволяє зловмисникам переглядати таблиці, отримувати з них важливу інформацію, змінювати їх і навіть отримувати доступ як адміністратор.

Типів ін’єкційних атак:

• SQL-ін’єкція – це слабка сторона веб-безпеки, яка може дозволити зловмиснику змінити SQL-запити, що виконуються в базі даних. Це можна

використовувати для отримання конфіденційної інформації, наприклад структури бази даних, її таблиць, стовпців і набору даних.

- Міжсайтовий сценарій, також відомий як XSS, дозволяє зловмиснику контролювати, як користувачі взаємодіють із уразливою до нього програмою. Зловмисник може обійти правило «одного походження», яке покликане запобігти спілкуванню між різними веб-сайтами. Міжсайтовий сценарій створює діри в безпеці, які дозволяють зловмиснику зайняти місце користувача-жертви, зробити все, що користувач може зробити, і отримати доступ до будь-яких даних користувача. Якщо користувач, якого атакують, має привілейований доступ до програми, зловмисник може отримати повний контроль над даними та функціями програми. Міжсайтовий сценарій виконується шляхом внесення змін до веб-сайту, який можна атакувати, щоб він надсилав шкідливий JavaScript назад користувачам. Коли шкідливий код запускається у браузері жертви, зловмисник може повністю змінити спосіб використання жертвою програми.

- Якщо програма має недолік під назвою Ін’єкція команд ОС, яка також відома як ін’єкція оболонки, зловмисник може виконувати будь-які команди на сервері активної програми. Інструкції, які запускає зловмисник, виконуються операційною системою за допомогою дозволів веб-сервера. Зловмисники можуть використовувати ескалацію привілеїв та інші вразливості, щоб скористатися цими недоліками впровадження команд.

- Програма має вразливість до впровадження коду, якщо зловмисник може представити код програми як введений користувачем і переконати сервер виконати його. Наприклад, якщо вразливу програму було написано мовою PHP, зловмисники могли вставити код PHP на веб-сервер. Потім інтерпретатор PHP на веб-сервері прочитає код і запустить його.

- Коли зловмисне корисне навантаження впроваджується в шаблон за допомогою рідного синтаксису мови шаблону, а потім шаблон запускається на сервері, це називається ін’єкцією шаблону на стороні сервера. Веб-сторінки можна створювати за допомогою механізмів шаблонів шляхом об’єднання існуючих шаблонів із динамічними даними. Коли введені користувачем дані

додаються до шаблону, а не передаються як дані, може відбутися атака шаблону на стороні сервера. Це дає змогу зловмисникам вставляти довільні директиви шаблонів і сіяти хаос у системі шаблонів, іноді навіть захоплюючи повний контроль над сервером.

- Заголовок хосту веб-сайту або веб-програми визначає, який веб-сайт або веб-програма має відповідати за обробку вхідного запиту HTTP. Вміст цього заголовка оцінюється веб-сервером перед тим, як він пересилає запит на вказаний веб-сайт або онлайн-додаток. Якщо зловмисник доставить довільний хост на справжній віртуальний хост, це може призвести до отруєння веб-кешу, а також до виконання незаконних дій, як-от скидання пароля.

- Протокол, який використовується для доступу та керування службами каталогів на IP-серверах, називається Lightweight Active Directory Protocol або скорочено LDAP. Протокол LDAP — це клієнт-серверний протокол, який використовується для перевірки користувачів, керування ресурсами та встановлення дозволів. Він також надає доступ до бази даних каталогу. Коли зловмисник додає шкідливі заяви до запиту, сервер отримує шкідливі запити LDAP, що може вплинути на безпеку системи. Якщо зловмиснику вдасться впровадити зловмисний код у LDAP, зловмисник не лише матиме доступ до даних, які не повинні бути видимі, але й зможе маніпулювати структурою LDAP.

Атака SQL-ін’єкції здійснюється, коли ворожий хакер вставляє оператор SQL у дані, які розміщуються у веб-формі, полі коментаря, рядку запиту чи будь-якому іншому каналі введення, доступному для людей ззовні. Шкідливий код зазвичай має форму SQL-запиту, який намагається отримати конфіденційну інформацію. Однак він також може приймати форму оператора SQL, призначеного для зміни вмісту бази даних, аж до видалення таблиць бази даних.

Якщо цільова програма вразлива до SQL-ін’єкцій, вона надсилає ці дані до бази даних без попередньої перевірки безпеки. Після цього, замість збереження коментаря чи отримання інформації про обліковий запис, сервер бази даних виконуватиме шкідливі SQL-запити, які зловмисники ввели в систему.

Зловмисники все одно можуть виявити інформацію за допомогою сліпого впровадження SQL, навіть якщо сприйнятлива програма не розкриває дані явно.

Ін’єкції SQL є одним із найстаріших і найбільш смертоносних типів вразливостей, які можуть вплинути на онлайн-програми.

Помилки ідентифікації та автентифікації — це вразливі місця в безпеці, які можуть виникнути, коли системі або програмі не вдається правильно ідентифікувати або аутентифікувати користувача. Це може дозволити зловмисникам отримати несанкціонований доступ до систем і даних [14].

Найпоширеніших помилки ідентифікації та автентифікації [15]:

- Ненадійні або повторно використовувані паролі.
- Атаки грубою силою.
- Підміна облікових даних.
- Відсутня або слабка багатофакторна автентифікація.
- Неперевірені перенаправлення та перенаправлення.

Уразливості з неправильною конфігурацією є поширеною і часто упускати з виду загрозою безпеки, яка може вплинути практично на будь-яку технологічну систему, від невеликих мереж до великих хмарних інфраструктур. Неправильна настройка відноситься до ситуації, коли система або пристрій настроєно неправильно, що може привести до несподіваного і потенційно небезпечного поведінки. Неправильні налаштування можуть виникати з багатьох причин, включаючи людську помилку, погану документацію або неповне тестування, і вони можуть мати серйозні наслідки для безпеки і цілісності системи [16].

Помилки в конфігурації безпеки можуть бути викликані такими проблемами [17]:

- Людська помилка.
- Погане або слабке шифрування.
- Надмірні привілеї.
- Неправильно налаштоване журналювання.
- Неправильна версія.
- Ненадійні послуги.

- Неправильні налаштування, пов’язані з інструментами безпеки.
- Використання стандартних налаштувань.

Неправильна конфігурація безпеки дозволяє зловмисникам отримати несанкціонований доступ до мереж, систем і даних, що, у свою чергу, може завдати значних фінансових і репутаційних збитків вашій організації. Помилки в конфігурації безпеки настільки небезпечні, що вони можуть відбуватися в багатьох місцях у вашому середовищі, включаючи стек програм, хмарні служби, мережевий рівень, веб-сервери та сервери програм, бази даних, віртуальні машини, контейнери, сховище та в коді.

В списку OWASP помилки неправильної конфігурації безпеки поміщено під шосте місце. Це не дивно, оскільки неправильні конфігурації безпеки були незмінними членами списку 10 найпопулярніших протягом багатьох років [18].

Цього року неправильні конфігурації зайняли першу позицію як інцидент номер один, пов’язаний із безпекою, перевершивши розкриті дані користувачами і компрометацію облікового запису минулого року [19].

2.3 Види атаки на контейнери

Під час атаки на віртуалізацію на основі контейнерів намагаються використати контейнери разом із їх ізоляцією, використовуючи вразливості у середовищі виконання контейнера, уразливості зображень, неправильну конфігурацію зображень або використання ненадійних зображень. Ці атаки націлені на середовище контейнера, наприклад, використовуючи недоліки в хост-системі або середовищі виконання контейнера.

Уразливість віддаленого виконання коду (RCE) - це тип вразливості в системі безпеки, яка дозволяє зловмиснику виконувати довільний код або команди в цільовій системі або додатку віддалено, зазвичай по мережі [20].

Уразливості RCE можуть з’являтися в будь-якому типі комп’ютерного програмного забезпечення, майже в кожній мові програмування та на будь-якій платформі. Існують уразливості RCE, наприклад, в автономних програмах

Windows, написаних на C#, веб-програмах та API, написаних на PHP, мобільних програмах, написаних на Java, і навіть у самих операційних системах.

Деякі атаки RCE можуть відбуватися із затримкою. Наприклад, програма може спочатку зберегти корисне навантаження RCE у файлі конфігурації, а потім виконати його лише пізніше, можливо, навіть кілька разів. Цей тип уразливості RCE називається збереженою RCE.

Кожна поширена мова програмування, яка використовується у веб-розробці, має функції для оцінки коду цією мовою під час виконання. Щоразу, коли розробники використовують такі функції у веб-додатках, вони вводять можливість віддаленого виконання коду на стороні веб-сервера. Якщо розробник дозволяє такій функції, обробляти несанкціонований ввід користувача, зловмисник може впровадити код, включивши його до введення користувача. Приклади введених даних, якими керує користувач, включають текст із веб-форм, вміст заголовків HTTP, файли, завантажені користувачем, або навіть модифіковані файли cookie.

Віддалене виконання коду є однією з найсерйозніших уразливостей, оскільки наслідки кібератак RCE практично необмежені, особливо у випадку веб-додатків. Найпоширенішим способом подальших дій зловмисників у разі використання вразливостей веб-RCE є встановлення веб-оболонок. Таке корисне навантаження експлойту RCE включає код, який дозволяє зловмиснику отримати доступ до оболонки на цільовій машині для виконання системних команд. Оболонка може бути зворотною оболонкою , яка дозволяє зловмиснику уникнути більшості брандмауерів.

Веб-оболонка має ті самі привілеї, що й веб-сервер, але зазвичай вони обмежені. Однак, як тільки зловмисник отримає доступ до оболонки на віддаленій машині, він може спробувати знайти інші вразливості та використати підвищення привілеїв, щоб отримати root-доступ.

Несанкціонований доступ — це процес отримання входу або доступу до системи, фізичної чи електронної, без дозволу власника чи адміністратора. Такий доступ можна отримати, обходячи заходи безпеки, використовуючи вразливі

місця системи або використовуючи вкрадені облікові дані. Несанкціонований доступ є серйозним порушенням законів про конфіденційність і може призвести до тяжких наслідків, у тому числі до судових позовів [21].

У сфері кібербезпеки несанкціонований доступ означає порушення комп'ютерних систем, мереж або баз даних. Ці порушення зазвичай пов'язані з проникненням хакерів у систему з метою викрадення, зміни або знищення інформації. Однак важливо зазначити, що неавторизований доступ не обмежується атаками зовнішніх хакерів. Це також може включати доступ співробітника до файлів або інформації за межами його рівня авторизації.

Дедалі більш поширена загроза несанкціонованого доступу викликає серйозні занепокоєння щодо безпеки даних, конфіденційності та цілісності цифрових систем. Це становить значний ризик для окремих осіб, корпорацій і урядів.

Коли неавторизовані особи отримують доступ до системи, вони часто націлюються на конфіденційні дані, такі як фінансові записи, особиста ідентифікаційна інформація, комерційна таємниця або інтелектуальна власність. Це вторгнення може призвести до значних фінансових втрат, шкоди репутації компанії та потенційних юридичних наслідків.

Ще один серйозний ризик – це можливість шахрайства. Маючи доступ до конфіденційних даних, кіберзлочинці можуть здійснювати різноманітні шахрайські дії. Це може бути шахрайство з кредитними картками, маніпуляції з банківськими рахунками або навіть створення фальшивого бізнесу.

У деяких випадках несанкціонований доступ може бути використаний для саботажу організаційних систем або пошкодження веб-сайтів. Це може включати порушення функціонування мережі, впровадження зловмисного коду на веб-сайт або навіть взяття під контроль системи, викликаючи повсюдний хаос і збої. Ці дії можуть завдати значної шкоди бізнесу. Вони не тільки можуть призвести до фінансових втрат, але й можуть зіпсувати репутацію компанії, що призведе до втрати довіри серед клієнтів і клієнтів.

Хоча це може бути менш поширеним, несанкціонований доступ також може призвести до фізичних пошкоджень. Наприклад, якщо хакер отримує контроль над промисловою системою керування, він може спричинити збій у роботі обладнання, що призведе до потенційних аварій або пошкодження обладнання. Цей ризик особливо гострий у таких галузях, як виробництво, енергетика чи транспорт, де несправність обладнання може призвести до значної загрози безпеці. Це підкреслює важливість надійних заходів безпеки не лише для захисту даних, але й для забезпечення фізичної безпеки працівників та інфраструктури.

Одним із найпоширеніших способів несанкціонованого доступу є неправильно реалізовані процеси автентифікації. Автентифікація — це захід безпеки, який використовується для перевірки особи або пристрою, які намагаються отримати доступ до системи. Якщо процес автентифікації погано розроблений, реалізований або неправильно налаштований, неавторизованим особам стає легко обійти його та отримати доступ до системи.

Коли система не блокує користувача після певної кількості невдалих спроб входу, вона залишає двері відкритими для атаки грубою силою, коли зловмисник намагається використовувати різні комбінації паролів, доки не знайде правильний.

Погано реалізованої автентифікації є те, що система не вимагає регулярної зміни пароля. У такій ситуації неавторизована особа, якій вдається отримати дійсний пароль, може продовжувати використовувати його протягом тривалого періоду, не будучи виявленим.

Одним із способів несанкціонованого доступу є фішингові атаки. Це передбачає надсилання оманливих електронних листів або повідомлень, які обманом змушують одержувачів розкрити свої облікові дані або натиснути зловмисні посилання. Як тільки одержувач наживеться, зловмисник може отримати доступ до його облікових записів або заразити його системи шкідливим програмним забезпеченням.

Іншим методом отримання несанкціонованого доступу є атаки паролем. Це передбачає спробу вгадати або зламати пароль користувача за допомогою різних методів. Вони можуть включати атаки грубої сили, коли пробуються всі можливі комбінації паролів, або атаки за словником, коли використовуються загальні слова чи фрази.

Неавторизований доступ також може статися через використання вразливостей програмного забезпечення. Це недоліки або слабкі сторони програмного забезпечення, які можна використати для отримання несанкціонованого доступу або виконання інших зловмисних дій.

Також джерелом несанкціонованого доступу є внутрішні загрози. Інсайдерські загрози стосуються загроз безпеці, які походять зсередини організації, часто від співробітників, колишніх співробітників, підрядників або ділових партнерів, які мають законний доступ до мереж, систем або даних організації.

Висновок до розділу 2

У цьому розділі було докладно розглянуто різноманітні види загроз, які можуть виникнути на різних рівнях компонентів архітектури програмного забезпечення. Аналізуючи атаки на мережевому рівні, ми виявили потенційні загрози, які можуть виникнути через недоліки мережевих протоколів та інфраструктури. Атаки на рівні сервісу надають уявлення про потенційні вразливості та атаки, які спрямовані на самі сервіси програмного забезпечення. Нарешті, розглядаючи атаки на рівні контейнера, ми дослідили загрози, які виникають внаслідок використання контейнеризації для розгортання та управління додатками.

Цей розділ підкреслює необхідність постійного моніторингу та аналізу заходів безпеки на різних рівнях архітектури програмного забезпечення. На основі проведеного дослідження можна зробити висновок, що жоден рівень компонентів архітектури не є імунним до потенційних загроз, і необхідно

приділяти увагу всім аспектам безпеки від мережевих протоколів до контейнеризації. Лише за допомогою цілеспрямованих заходів захисту можна забезпечити високий рівень безпеки та стійкості програмного забезпечення у сучасному цифровому середовищі.

РОЗДІЛ 3
ДОСЛІДЖЕННЯ ЗАСОБІВ ТА МЕХАНІЗМІВ ЗАХИСТУ

Дослідження засобів та механізмів захисту архітектури програмного забезпечення є важливим етапом у забезпеченні безпеки та стійкості системи. Цей процес включає в себе аналіз та вивчення різноманітних засобів та методів, які можуть бути використані для захисту різних компонентів та елементів архітектури від потенційних загроз та атак. Дослідження цього аспекту дозволяє виявити найбільш ефективні та оптимальні засоби захисту, які відповідають конкретним потребам та вимогам системи, та сприяє створенню комплексної стратегії безпеки.

3.1 Захист на рівні мережі

За даними компанії Cloudwards, кожні 14 секунд нова організація зазнає атаки програм-вимагачів. І це незважаючи на те, що світові витрати на кібербезпеку зростають і становлять близько 150 млрд. доларів на рік. Саме тому зміцнення мережевої безпеки організації є нагальною потребою [22].

Мережі з’єднують пристрої одна з одною, щоб користувачі могли отримувати доступ до таких ресурсів, як програми, дані або навіть інші мережі, такі як Інтернет. Безпека мережі захищає та контролює зв’язки та комунікації в мережі за допомогою комбінації апаратних засобів, програмного забезпечення та примусових політик.

Мережі та мережева безпека мають широкий діапазон складності, щоб відповідати широкому спектру потреб. Людям, які хочуть захистити середовище малого або домашнього офісу, не потрібні ті самі інструменти та методи, що й малому чи середньому бізнесу, не кажучи вже про підприємство з тисячами пристроїв і десятками, якщо не сотнями мереж.

Основні типи загроз для мережі включають:

- Погані користувачі: неавторизовані користувачі (хакери, неправильно налаштовані програми тощо), які навмисно чи ненавмисно підключаються до мережі.
- Погані пристрої: неавторизовані пристрої або пристрої в скомпрометованому стані, які навмисно чи ненавмисно загрожують цілісності чи продуктивності мережі.
- Шкідливий трафік: мережевий трафік з метою порушити роботу або викрасти дані.
- Зловмисні наміри: авторизовані користувачі або пристрої, підключені до мережі з наміром порушити роботу або викрасти дані, зазвичай характеризуються як внутрішня загроза.
- Погане технічне обслуговування: мережеві компоненти навмисно чи ненавмисно залишаються вразливими через відсутність оновлень або застарілу технологію.
- Порушення роботи: навмисні атаки, ненавмисні неправильні налаштування та інші потенційні ситуації, які загрожують безвідмовній роботі або пропускній здатності мережі.

Кожну з цих категорій можна розглянути за допомогою технічних засобів контролю, які використовують апаратне забезпечення, програмне забезпечення або конфігурації для авторизації, автентифікації, полегшення, захисту та моніторингу мережевого трафіку.

Основні технічні засоби контролю:

- Контроль доступу користувачів;
- Елементи керування виявленням активів;
- Контроль моніторингу руху;
- Контроль стійкості, технічного обслуговування та тестування.

Ці інструменти значною мірою залежать від ефективного визначення адміністративних засобів контролю, які визначають і визначають політику, яка буде реалізована за допомогою технічних засобів контролю. Політики, як правило, є письмовими документами, які детально описують вимоги, які будуть

дотримуватись, наприклад, складність пароля. Ці політики також надають контрольні показники, за якими продуктивність технічного інструменту буде відстежуватися, вимірюватися та звітуватися щодо ключових показників ефективності і відповідності.

Інструменти також залежать від фізичних засобів контролю , які також повинні бути реалізовані проти зловмисного фізичного доступу для знищення або компрометації мережевого обладнання, такого як маршрутизатори, кабелі, комутатори, брандмауери та інші мережеві пристрої. Ці засоби фізичного контролю не покладаються на IT-технології, і передбачається їх наявність.

Усі аспекти безпеки мережі прагнуть виключити несанкціонований доступ до активів або зв’язку. Покращена мережева безпека відстежує авторизовані, але невідповідні дії або незвичайну поведінку, які можуть свідчити про компрометацію, діяльність зловмисного програмного забезпечення або внутрішню загрозу.

Аудит мережевих журналів і журналів активності користувачів слід використовувати для перевірки успішного впровадження політик безпеки мережі та засобів контролю. Для перевірки належної реалізації та конфігурації слід використовувати тестування на проникнення та сканування вразливостей.

Для захисту від неавторизованих користувачів мережева безпека повинна запровадити ефективний контроль доступу. За своєю суттю контроль доступу вимагає, щоб мережі регулювали користувачів, які можуть підключатися до мережі, і визначали, до яких мережевих ресурсів може отримати доступ конкретний користувач.

Active Directory: найменші організації можуть хвилюватися лише про доступ до пристрою, інакше відомий як облікові дані для входу (ім’я користувача/пароль). У міру зростання організації формалізований і централізований контроль за допомогою Active Directory або еквівалентного полегшеного інструменту протоколу доступу до каталогу заощадить час і дозволить швидше відповідати на запити змін.

Віртуальна приватна мережа: для віддаленого доступу протокол віддаленого робочого столу більше не можна вважати безпечним. Замість цього організації повинні використовувати рішення віртуальної приватної мережі. Обладнання на місці може бути дорогим і важким у розгортанні та обслуговуванні для найменших організацій. Натомість багато хто звертається до постачальників послуг VPN як послуги, які іноді називають корпоративними рішеннями VPN, які надають рішення VPN щомісяця та для кожного користувача.

У міру зростання організацій керування користувачами займатиме багато часу, а потенційні втрати через порушення зростуть. Кращий контроль доступу до безпеки мережі може покращити безпеку та зменшити витрати та ризики.

Інструменти керування доступом: покращене керування користувачами з більш детальним і автоматизованим контролем доступу можна досягти за допомогою керування ідентифікацією та доступом або керування привілейованим доступом. Деякі інструменти навіть інтегруються з програмним забезпеченням відділу кадрів, щоб увімкнути одночасне й автоматизоване надання IT-доступу до IT-підключення та відключення IT-доступу. Ці складніші інструменти заощаджують час на керування та можуть полегшити створення детального розмежування між користувачами та активами, до яких їм може знадобитися доступ.

Керування доступом до хмари: навіть менші організації тепер використовують хмарні ресурси, але більшість внутрішніх мережевих елементів керування не поширюються на ресурси, розміщені за межами мережі, наприклад Office 365, документи Google, інші рішення програмного забезпечення як послуги та навіть окремі мережі відділень. Хмарні брокери безпеки доступу до хмари і захищені браузерні програми можуть надавати консолідовані рішення для захисту користувачів як у хмарі, так і в локальних мережах.

Найбільші організації використовують різноманітні хмарні ресурси та потребують централізованого керування декількома офісами, багато з яких є міжнародними відповідно до різних нормативних актів. Крім того, підприємства

повинні контролювати широкий спектр користувачів, таких як співробітники філій, підрядники, корпоративні клієнти та навіть програми.

Доступ до програми: роздрібний веб-сайт може робити більше запитів, ніж користувачі будь-якого ресурсу. Викликами API пов’язаних додатків або прямі з’єднання з активами необхідно керувати як на рівні додатків, так і на рівні мережі.

Покращене керування доступом до хмари. Хмарні ресурси, такі як захищені веб-шлюзи , робочий стіл як послуга, Azure Active Directory і подібні рішення, контролюють доступ користувачів і керування ними в масштабі. Ці рішення також дозволяють централізовано контролювати територіально розподілену робочу силу та збір активів.

Безпека мережі стосується лише користувачів, які стосуються доступу до мережі. Хоча ефективна мережева безпека виходить за межі мережі, ефективна безпека мережі залежить від ефективної автентифікації користувача в іншому місці стеку безпеки.

Безпека AD: надійний і безпечний Active Directory відіграє вирішальну роль для більшості організацій у контролі доступу користувачів. Хоча засоби безпеки Active Directory безпосередньо не пов’язані з мережевою безпекою, можна встановити на серверах для захисту AD від зловмисних і недбалих дій.

Покращені паролі: організації, яким потрібна покращена безпека, зазвичай збільшують вимоги до надійності паролів, щоб додати складність або частіше змінювати паролі. Менеджери паролів допомагають користувачам відповідати суворішим вимогам, а також можуть забезпечити централізований контроль. Підприємства також можуть застосовувати технології єдиного входу, щоб оптимізувати доступ до хмарних ресурсів.

Двофакторна автентифікація: у сучасному середовищі, наповненому програмами-вимагачами, двофакторну автентифікацію також слід вважати мінімальною вимогою для всіх форм віддаленого доступу. SMS не вважається оптимальним рішенням, але часто може бути найзручнішим і найпростішим у

розгортанні. Для покращення безпеки використання мобільних телефонів доступні безкоштовні програми автентифікації від Google, Microsoft та інших.

Багатофакторна автентифікація: організації, що розвиваються, стикаються з підвищеним ризиком злому, оскільки потенційні збитки від викрадених облікових даних зростають із розміром компанії та репутацією. Щоб зменшити цей ризик, багато хто використовує багатофакторну автентифікацію для забезпечення покращеної безпеки порівняно з 2FA, особливо коли програми або маркери замінюють вразливий текст SMS як фактор. Біометричні та безпарольні рішення можуть бути дорожчими, але їх важко підробити.

Неавторизовані пристрої можуть перехоплювати або перенаправляти мережевий трафік за допомогою таких атак, як підключення неавторизованих комп’ютерів до мережі, розгортання сніферів пакетів для перехоплення мережевого трафіку або надання фішингового посилання для атаки типу людина посередині для викрадення облікових даних і даних. Подібним чином підроблена система доменних імен та IP-адреси можуть перенаправляти користувачів із законних з’єднань на небезпечні та шкідливі веб-сайти.

Окрім шкідливих пристроїв, мережі повинні виявляти потенційно небезпечні пристрої, які можуть бути запроваджені через непорозуміння чи недбалість. Наприклад, ноутбук без важливих оновлень безпеки, зламаний планшет або холодильник із підтримкою Wi-Fi, підключений у кімнаті відпочинку без дозволу.

Корпоративні середовища виходять далеко за межі одного офісу та включають широкий спектр фізичних і віртуальних активів. Віртуальні середовища, розміщені в хмарі або в локальних центрах обробки даних, не підлягають візуальній перевірці та повинні контролюватися за допомогою програмного забезпечення. Пристрої IoT, такі як камери безпеки, датчики температури або монітори тепла, будуть додані до мереж і часто мають недоліки безпеки. Операційна технологія, також відома як промисловий Інтернет речей, використовує інтелектуальні насоси, конвеєрні стрічки, двигуни та виробниче

обладнання і операційні групи, які встановлюють пристрої, не завжди можуть інформувати про них команду безпеки мережі.

3.2 Безпека даних

Безпека даних це – набір стандартів і технологій, які захищають дані від навмисного або випадкового знищення, змін або розкриття. Технологія захисту даних буває різних форм і захищає дані від зростаючого числа загроз. Багато з цих загроз виходять із зовнішніх джерел, але організації також повинні зосередити свої зусилля на захисті своїх даних зсередини [23].

Ефективні рішення для захисту інформації мають враховувати рівень конфіденційності наборів даних і нормативно-правові вимоги до вашої організації. Нижче наведено типи захисту даних, що дають змогу запобігати порушенням їх безпеки, ефективно дотримуватися нормативних вимог і уникати шкоди для репутації [24].

Безпека даних захищає дані компанії від внутрішніх і зовнішніх загроз і є критично важливим елементом бізнес-операцій. Це охоплює захист цифрової інформації від численних загроз несанкціонованого доступу, пошкодження даних або крадіжки протягом її життєвого циклу. Це всеохоплюючий термін, який охоплює апаратне забезпечення та його фізичну безпеку, а також захист пристроїв зберігання даних, адміністративний контроль, доступність і безпеку програм. Він також охоплює політику та протоколи компанії. Якщо захист даних сплановано та реалізовано правильно, активи даних компанії захищені від різноманітних форм атак кіберзлочинців. Це також гарантує, що дані захищені від внутрішніх загроз і можливої людської помилки – остання продовжує бути поширеною причиною витоку даних. Переважна більшість порушень безпеки даних викликана помилками людини.

У рамках своїх процедур захисту даних компанії розгортають ряд інструментів і технологій, щоб підвищити видимість своїх критично важливих даних і контролювати їх використання. Ці інструменти можуть захистити

шифрування та маскування даних . Конфіденційні файли видаляються, а звітування автоматизується для спрощення процедур аудиту. Усе це сприяє дотриманню нормативних вимог.

Найзначнішим впливом порушення безпеки на корпорацію є фінансова шкода. Однак капіталу бренду та вартості бренду організації також завдано шкоди. Опитування споживачів показують, що більшість споживачів припинили б свої стосунки з брендом, якби він порушив безпеку даних.

Компанія часто збирає широкий спектр даних, які не призначені для передачі. Це можуть бути особисті дані клієнтів, постачальників, замовників тощо. Заходи захисту даних зберігають усі види інформації в безпеці та в межах, де вона повинна бути. Якби особисті дані клієнтів були оприлюднені. Наслідки, як індивідуально, так і для організації, будуть величезними.

Сьогодні люди цінують свою конфіденційність як ніколи, і наявність надійного плану захисту даних допомагає зміцнити довіру в організації та з усіма клієнтами.

Коли організація захищає конфіденційні дані від цікавих очей хакерів, вона може залишатися попереду конкурентів. Витік даних, пов'язаних з бізнес-планами, може уповільнити прогрес і розвиток бізнесу.

Коли організація стикається з витоком даних, клієнти обирають юридичні заходи для захисту своїх інтересів. Це означає, що вони можуть порушувати судові справи проти організації, і якщо буде виявлено будь-яку невідповідність, на неї будуть накладені штрафи. Крім того, організації, можливо, доведеться виплатити компенсацію своїм клієнтам, не кажучи вже про величезні витрати на судовий розгляд. Правильні протоколи безпеки даних можуть запобігти цьому.

Якщо кіберзлочинці атакують організацію, це не завжди для того, щоб викрасти дані, але також можуть підробити їх. Хакери можуть видаляти, змінювати та пошкоджувати дані. Вони можуть захоплювати процеси за допомогою смертоносних троянів або навіть впроваджувати програми-вимагачі в системи інформаційних технологій. Результати можуть виявитися катастрофічними. Протоколи безпеки даних значною мірою захищають бізнес.

Види заходів для впрвадження захисту даних:

- Шифрування даних. Алгоритм кодує текстові символи до нечитабельного формату, щоб авторизовані користувачі могли лише прочитати його. Якщо є багато конфіденційної інформації, такі рішення, як шифрування файлів і наборів даних, захищають дані за допомогою шифрування та/або токенізації. Більшість із цих рішень також мають функції керування ключами безпеки.

- Стирання даних. Хоча видалення даних є стандартною процедурою, вона може бути не ретельною. Програмне забезпечення для перезапису даних, що зберігаються на будь-якому пристрої. Він перевіряє, що дані не можна відновити. Це сучасний еквівалент листа, який знищується після прочитання. Однією з переваг рішення для віртуалізації даних є те, що воно не зберігає дані, тому видалення потрібне лише на вихідних системах. Це забезпечує додаткове керування та усуває потенційну неузгодженість даних.

- Маскування даних. Ідентифікаційна інформація маскується, щоб різні команди могли продовжувати розробляти програми та навчати новобранців точними даними. Усі розробки відбуваються з фактичними даними в сумісних середовищах. Рівень віртуалізації даних також може реалізувати захист на основі рядків і стовпців на основі користувачів і ролей під час виконання.

- Стійкість даних. Наскільки швидко організація може відновити роботу після збою, будь то апаратне забезпечення, недостатність живлення чи будь-який інший фактор, що впливає на доступність даних команди. Швидкість відновлення має вирішальне значення для зменшення впливу на організацію.

Створення плану безпеки даних полягає в об’єднанні кількох змінних і забезпеченні того, щоб усі вони працювали разом у режимі реального часу для забезпечення безпеки даних. Реалізація плану безпеки даних повністю залежить від розміру та архітектури обчислювальної структури компанії. Хоча неможливо забезпечити 100% надійний підхід до створення плану безпеки даних, є певні ключові елементи, про які організація повинна пам’ятати.

Важливим першим кроком до безпеки даних є забезпечення безпеки незалежно від того, де вони зберігаються. Найкращі методи забезпечення цифрової та фізичної безпеки мають бути пріоритетними:

- Доступ на основі ідентифікатора користувача: в ідеалі, коли працюєте з конфіденційною інформацією, найкраще обмежити доступ до даних лише тим, хто працюватиме з нею. Створення протоколів на основі ідентифікатора користувача є простим, але ефективним способом гарантувати, що доступ до даних отримають лише ті, кому потрібен доступ. Таким чином дані залишаються безпечнішими, навіть якщо імена користувачів і логіни викрадено.
- Використовуйте шифрування скрізь: шифрування – це чудовий спосіб гарантувати, що хакери не зможуть використати будь-яку інформацію для створення проблем. Для додаткового захисного рівня розгляньте можливість шифрування всіх інформаційних передач.
- Створення методів автентифікації: захист даних користувача може починатися прямо з джерела. Незалежно від того, чи відбувається вхід для першого чи кількох типів, включаючи етап перевірки за допомогою протоколів автентифікації, таких як вхід із соціальної мережі, може мати велике значення.

Кіберзлочинці постійно вдосконалюють способи здійснення атак. З кожним новим рішенням атаки стають все більш витонченими, і компаніям потрібно переконатися, що їхні протоколи безпеки даних можуть не відставати. Найкращі практик, щоб гарантувати, що організація добре підготовлена до атак і має швидкі та надійні рішення у разі порушення:

- Регулярно проводите стрес-тестування систем: працюйте в наступі. Переконайтесь, що дані ніколи не будуть втрачені за допомогою безпечного процесу відновлення даних. Автоматизація — це режим моніторингу, який часто віддають перевагу, але він не встигає за людською креативністю у пошуку нових способів злому. Створіть внутрішню команду для регулярного стрес-тестування систем або залучіть для цього послуги зовнішньої організації.
- Впроваджуйте найкращі методи для співробітників: найпоширеніші форми атак на безпеку даних відбуваються через USB-пастки або фішингові

електронні листи, жертвами яких стають співробітники. Не кожен працівник усвідомлює можливі загрози безпеці та різні способи їх виникнення, і це може завдати значної шкоди даним компанії. Регулярне навчання та інформування співробітників про різні форми атак може захистити дані та зменшити ризики.

- Створення детального план реагування: завжди будьте напоготові. У разі нападу завжди майте план реагування. Це має бути якомога всеохоплюючим і мати завдання, окреслені для кожного. Швидке введення в дію плану відновлення може значною мірою пом’якшити негативний вплив атаки, заощадивши організацію з багатьох причин. Кожен відділ має знати про план: від IT до керівництва та керівників тощо.
- Резервне копіювання всіх даних. Значною мірою відновлення даних залежить від надійних резервних копій даних. Резервну копію слід створювати незалежно від систем даних, які організація регулярно використовує.

Дані можуть застаріти, але все одно можуть стати джерелом небезпеки, якщо їх витік. Щоб уникнути цього, компанія в ідеалі повинна стерти всі невикористовувані дані. Старі паролі користувачів, чудовий приклад для розгляду. Незважаючи на попередження, люди, як правило, використовують той самий пароль на кількох платформах, навіть якщо його змінюють в одному місці.

Призначте всім даним термін дії. Ці параметри можна створити внутрішньо на основі типу даних, з якими працюється. Коли прийде час утилізувати, переконайтеся, що все стерто, включно з резервними копіями та проектами, які могли отримати доступ до цієї інформації з будь-якою метою.

Ніколи не забувайте про фізичне зберігання: дані часто зберігаються на фізичних пристроях. Це можуть бути зовнішні накопичувачі, USB-накопичувачі, друковані файли та документація та інші формати. Про них зазвичай забувають, але вони можуть бути значним джерелом витоку даних.

Кілька стандартів керування даними можуть допомогти зменшити ймовірність витоку даних. Правила відповідальності в основному ґрунтуються на тому, де територіально розташована організація та як бізнес працює в цьому регіоні. Без якісних даних організація не зможе забезпечити перевагу у

висококонкурентному світі. Компанії постійно працюють над безпекою даних, щоб запобігти зламам і наслідкам атаки на компанію. Інвестиції в сектор збільшуватимуться з кожним роком і з підвищенням рівня складності атак. Однак чим сильніші протоколи безпеки даних компанії, тим вищі шанси гарантувати, що дані завжди будуть у безпеці.

3.3 Моніторинг та керування подіями

Керування захистом інформації (SIEM) – це рішення, що допомагає організаціям виявляти, аналізувати й усувати загрози безпеці, перш ніж вони зможуть зашкодити бізнес-операціям [25].

Система моніторингу та управління – набір систем або комплексний інструмент, які дозволяють спостерігати та проводити аналіз поточного функціонування інфраструктури в режимі реального часу згідно з раніше отриманими даними [26].

Технологія централізує інформацію про безпеку з кількох кінцевих точок, серверів і додатків, щоб допомогти контролювати IT-інфраструктуру, перевіряти аномалії в режимі реального часу, сповіщати команди безпеки щоразу про аномальну подію та вести детальні журнали даних усіх подій. Як централізована платформа дозволяє швидко й ефективно виявляти потенційні інциденти безпеки та реагувати на них.

Незалежно від розміру організації, важливо вживати активних заходів для моніторингу та пом’якшення ризиків IT-безпеки. Рішення SIEM приносять різну користь підприємствам і стали важливим компонентом оптимізації робочих процесів безпеки.

Якщо у вашій мережі багато різних пристроїв і систем, може бути важко відстежувати всі події безпеки, які відбуваються. SIEM може допомогти вам централізувати моніторинг безпеки та спростити виявлення потенційних загроз.

Центральна інформаційна панель забезпечує уніфікований перегляд системних даних, попереджень і сповіщень, що дозволяє командам ефективно спілкуватися та співпрацювати під час реагування на загрози.

Агрегуючи та співвідносячи дані про події безпеки з багатьох джерел, рішення SIEM зменшують помилкові спрацьовування, пов’язані з окремими інструментами безпеки. Це дозволяє командам безпеки зосередити свої зусилля на розслідуванні справжніх інцидентів безпеки та реагуванні на них, підвищуючи загальну ефективність роботи.

Рішення SIEM є популярним вибором для організацій, які підпадають під різні форми дотримання нормативних вимог. Завдяки автоматичному збору та аналізу даних, які він забезпечує, SIEM є цінним інструментом для збору та перевірки даних відповідності в усій бізнес-інфраструктурі. Він забезпечує централізоване ведення журналів, журнали аудиту та можливості звітування, які демонструють дотримання політик і стандартів безпеки.

Рішення SIEM можуть генерувати звіти про відповідність у режимі реального часу для PCI-DSS, GDPR, PECR, NIS і Закону про свободу інформації, основних стандартів відповідності Великобританії, зменшуючи навантаження на управління безпекою та виявляючи потенційні порушення, щоб їх можна було усунути на ранній стадії.

Багато рішень SIEM постачаються з готовими додатками, які можуть створювати автоматичні звіти, розроблені відповідно до вимог відповідності.

Коли трапляється інцидент із безпекою, час має вирішальне значення. Рішення SIEM надають попередження та сповіщення в режимі реального часу, що дозволяє швидко реагувати на загрози та зменшувати потенційну шкоду.

Інструменти SIEM постійно відстежують мережевий трафік, системні журнали та інші джерела на наявність підозрілих дій або аномалій, які можуть свідчити про порушення безпеки або спробу вторгнення. Зіставляючи події з різних джерел, SIEM може виявляти складні моделі атак, які інакше могли б залишитися непоміченими. SIEM також може автоматизувати певні дії реагування, щоб швидко локалізувати або вирішити інциденти безпеки.

Більшість інформаційних панелей SIEM також включають візуалізацію даних у реальному часі, що допомагає IT-командам помічати сплески або тенденції підозрілої активності. Використовуючи налаштовані попередньо визначені правила кореляції, адміністратори можуть негайно отримувати сповіщення та вживати відповідних заходів для пом’якшення загроз, перш ніж вони матеріалізуються у більш значні проблеми безпеки.

SIEM отримує дані про події з широкого спектру джерел у всій IT-інфраструктурі організації, включаючи локальні та хмарні середовища.

Дані журналу подій від користувачів, кінцевих точок, додатків, джерел даних, хмарних робочих навантажень і мереж, а також дані апаратного та програмного забезпечення безпеки, наприклад брандмауерів або антивірусного програмного забезпечення, збираються, співвідносяться та аналізуються в режимі реального часу.

Деякі рішення SIEM також інтегруються зі сторонніми каналами аналізу загроз, щоб порівняти дані внутрішньої безпеки з попередньо розпізнаними сигнатурами та профілями загроз. Інтеграція з новинами про загрози в реальному часі дозволяє командам блокувати або виявляти нові типи сигнатур атак.

Рішення SIEM значно покращують середній час виявлення і середній час відповіді для груп IT-безпеки, розвантажуючи ручні робочі процеси, пов’язані з поглибленим аналізом подій безпеки. Ця можливість централізованого керування журналами не тільки полегшує виявлення загроз у режимі реального часу, але й підтримує криміналістичний аналіз, аудит відповідності та усунення неполадок IT.

Із зростанням популярності віддаленої роботи, додатків організаціям потрібен рівень видимості, необхідний для пом’якшення мережевих ризиків за межами традиційного периметра мережі. Рішення SIEM відстежують всю мережеву активність усіх користувачів, пристроїв і програм, значно підвищуючи прозорість усієї інфраструктури та виявляючи загрози незалежно від того, де здійснюється доступ до цифрових активів і послуг.

Інсайдерські загрози, навмисні чи ненавмисні, становлять значний ризик для організацій. SIEM може допомогти виявити аномальну поведінку інсайдерів, як-от несанкціонований доступ до конфіденційних даних або незвичайні моделі активності, допомагаючи запобігти витоку даних і інсайдерським атакам.

Рішення SIEM ідеально підходять для проведення комп'ютерних криміналістичних розслідувань після інциденту безпеки. Рішення SIEM дозволяють організаціям ефективно збирати та аналізувати дані журналу з усіх своїх цифрових активів в одному місці. SIEM зберігає довгострокові історичні дані для полегшення аналізу відповідності, відстеження та звітування. Це дає їм можливість відтворити минулі інциденти або проаналізувати нові, щоб розслідувати підозрілу діяльність і запровадити більш ефективні процеси безпеки.

Поєднання SIEM зі штучним інтелектом стає все більш важливим і приносить значні успіхи в моніторингу безпеки. Використовуючи глибоке машинне навчання, яке автоматично вивчає поведінку мережі, ці рішення можуть обробляти складні протоколи ідентифікації загроз і реагування на інциденти за менший час, ніж фізичні групи.

Алгоритми штучного інтелекту можуть аналізувати величезні обсяги даних із різних джерел, виявляючи тонкі аномалії та підозрілі шаблони, які традиційні правила SIEM можуть пропустити. Це забезпечує швидше та точніше виявлення загроз.

Штучний інтелект також може вивчати попередні сповіщення та контекст, щоб розпізнавати справжні загрози, значно зменшуючи навантаження на команди безпеки щодо розслідування помилкових спрацьовувань.

Висновки до розділу 3

В цьому розділі було досліджено різноманітні аспекти забезпечення безпеки та стійкості архітектури програмного забезпечення через застосування різних засобів та механізмів, важливі аспекти безпеки, починаючи з мережевих

аспектів, які визначають ефективне функціонування системи в мережевому середовищі, та закінчуючи моніторингом подій для виявлення аномалій та небезпек.

Дослідження дозволило визначити ключові аспекти безпеки, які потребують особливої уваги та заходів захисту. Засоби захисту на рівні мережі виявилися невід'ємною складовою для забезпечення безпеки передачі даних, тоді як безпека даних стала критично важливою у зв'язку зі збільшенням обсягу та значенням інформації. Моніторинг та керування подіями надають можливість вчасно виявляти та реагувати на потенційні загрози та інциденти безпеки.

Ретельне вивчення та ефективне впровадження засобів та механізмів захисту є ключовими для забезпечення безпеки та надійності архітектури програмного забезпечення в умовах постійно зростаючих кіберзагроз.

РОЗДІЛ 4
ВДОСКОНАЛЕННЯ МЕТОДУ ЗАХИСТУ АРХІТЕКТУРИ

4.1 Аналіз існуючих методів захисту від ін'єкцій коду

Впровадження коду – це тип атаки, коли зловмисник може виконати шкідливий код у комп’ютерній програмі. Це може призвести до маніпулювання поведінкою програми та потенційно поставити під загрозу безпеку системи [27].

Атаки з впровадженням коду є значною загрозою для безпеки веб-додатків. Ці атаки відбуваються, коли зловмисник може впровадити шкідливий код у веб-програму, який потім виконується інтерпретатором програми. Наслідки успішної атаки з впровадженням коду можуть бути серйозними: від несанкціонованого доступу до конфіденційних даних до повної компрометації основної системи. Щоб запобігти атакам із впровадженням коду, вкрай важливо дотримуватися передових практик у розробці веб-додатків і безпеки.

Перевірка введених даних: одним із найефективніших способів запобігання атакам із впровадженням коду є перевірка та дезінфекція всіх введених користувачами даних. Це передбачає ретельну перевірку даних, які надає користувач, на наявність будь-яких потенційно зловмисних символів або шаблонів. Перевірку введених даних слід виконувати як на стороні клієнта, так і на стороні сервера. Перевірка на стороні сервера особливо важлива, оскільки зловмисники можуть обійти перевірку на стороні клієнта.

Наприклад, якщо веб-додаток дозволяє користувачам вводити свої імена, процес перевірки введення має гарантувати, що приймаються лише дійсні символи. Будь-який вхід, який не відповідає очікуваному формату, має бути відхилений.

Параметризовані запити: під час взаємодії з базою даних вкрай важливо використовувати параметризовані запити або підготовлені оператори замість динамічного створення SQL-запитів за допомогою введення користувача.

Параметризовані запити відокремлюють код SQL від наданих користувачем даних, не дозволяючи інтерпретатору розглядати введені користувачем дані як виконуваний код.

Наприклад, розглянемо форму входу, де користувач вводить своє ім’я користувача та пароль. Замість того, щоб безпосередньо об’єднувати вхідні значення в SQL-запит, слід використовувати параметризований запит. Це гарантує, що введені користувачем дані розглядаються як дані, а не як частина коду SQL.

Екранування та кодування: ще одна важлива практика полягає в тому, щоб правильно екранувати та кодувати введені користувачем дані, коли вони відображаються у виводі програми. Це запобігає інтерпретатору від неправильної інтерпретації введення користувача як виконуваного коду.

Наприклад, якщо веб-додаток дозволяє користувачам публікувати коментарі, будь-які спеціальні символи або HTML-теги в коментарі повинні бути належним чином екрановані або закодовані, перш ніж відображати їх на сторінці. Це перешкоджає браузеру інтерпретувати введені дані як HTML-код, таким чином зменшуючи ризик атак із впровадженням коду.

Принцип найменших привілеїв: важливо дотримуватися принципу найменших привілеїв під час розробки та налаштування середовища програми. Це означає, що кожен компонент програми повинен мати лише мінімальні привілеї, необхідні для виконання призначеної функції. Обмеживши привілеї кожного компонента, вплив успішної атаки впровадження коду можна мінімізувати.

Наприклад, веб-сервер, на якому працює програма, не повинен мати зайвих прав адміністратора. Крім того, обліковий запис користувача бази даних, який використовується програмою, повинен мати обмежені права доступу, що дозволяє виконувати лише необхідні операції з базою даних.

Регулярні оновлення та виправлення: Оновлення веб-програми та програмного забезпечення, що лежить в її основі, має вирішальне значення для запобігання атакам із впровадженням коду. Розробники повинні регулярно

застосовувати виправлення безпеки та оновлення, надані постачальниками програмного забезпечення. Ці оновлення часто включають виправлення відомих вразливостей, якими можна скористатися під час атак із впровадженням коду.

4.2 Розробка вдосконаленого методу захисту

Під час аналізу існуючих загроз і варіантів захисту, було виявлено, що однією з найменш захищених вразливостей є ін'єкція коду. Це пояснюється тим, що для захисту від даної загрози часто використовується перевірка даних за типом. Проте, у багатьох сучасних веб-додатках поля, в які можуть вводитися дані, такі як нотатки, повідомлення або коментарі, можуть містити велику кількість інформації. У таких випадках, де типом даних є рядок, перевірка даних за типом може бути неефективною через можливість великого обсягу введених даних. Це створює потенційну уразливість для ін'єкції коду, оскільки шкідливий код може легко пройти перевірку типу даних.

Наприклад, у разі веб-додатків, де користувачі можуть залишати коментарі чи відгуки, велика кількість тексту, яку можна ввести, може використовуватися зловмисниками для впровадження шкідливого коду. Такі ін'єкції можуть бути в основному засновані на мові SQL, HTML або JavaScript, в залежності від контексту вразливого поля.

Ін'єкція коду залишається серйозною загрозою для безпеки веб-додатків, і вимагається ретельна перевірка та фільтрація вхідних даних.

Проблема ін’єкції коду, яка існує довгий час, заснована на традиційних методах захисту, які, на жаль, демонструють свої обмеження. У сучасному контексті, з розвитком штучного інтелекту та машинного навчання, можливе вдосконалення заходів безпеки, використовуючи моделі, які здатні до комплексної перевірки вхідних даних, не обмежуючись лише типом та довжиною.

Окрім перевірки типу та довжини вхідних даних, ефективні заходи проти ін’єкції коду можуть базуватися на ряді додаткових критеріїв. Зокрема,

врахування контексту введених даних, аналіз потенційно небезпечних символів або шаблонів, а також визначення аномальних змін у введених даних можуть бути ефективними стратегіями для виявлення та запобігання ін’єкціям.

Спочатку було вирішено спробувати обійти існуючі засоби захисту. Для цього було створено веб-додаток з полем для введення нотаток, який зображений на рисунку 4.1.

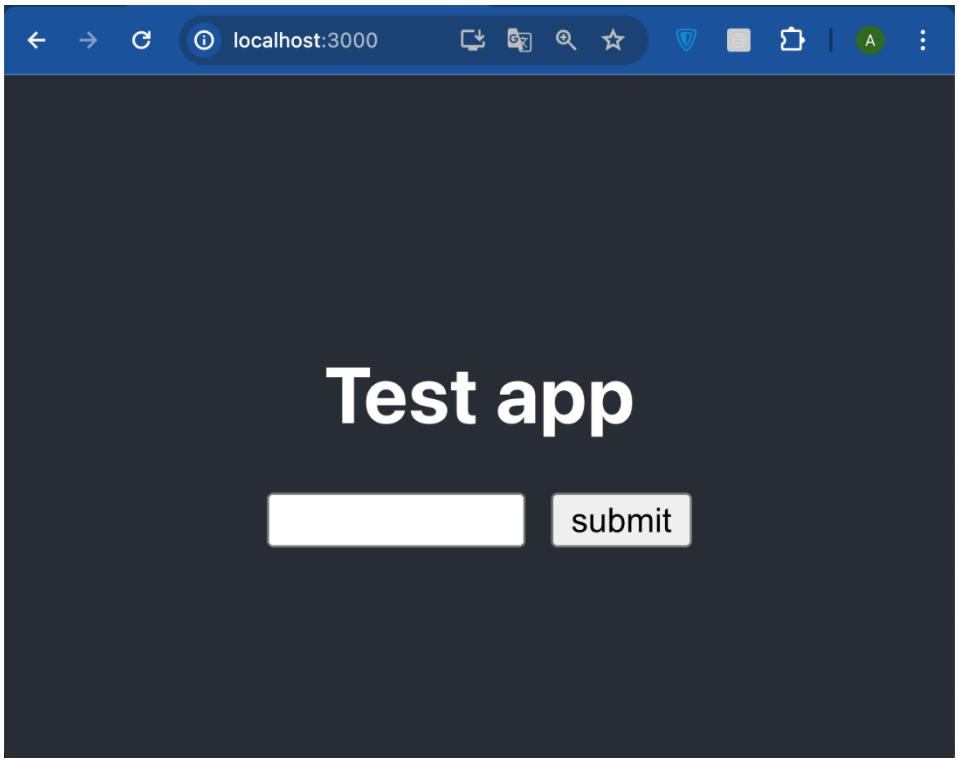

Рисунок 4.1 – Створений веб-додаток з полем для вводу даних

На рівні серверу був реалізований традиційний метод захисту, який полягав у встановленні перевірки на тип вхідних даних. Конкретно, здійснювалася перевірка на те, що введені дані є типом рядок та не перевищують обмеження у 256 символів.

У подальшому етапі, було випробувано введення зловмисного коду у поле тексту, зображено на рисунку 4.2. Незважаючи на те, що зазначений код мав недоброякісні наміри, він успішно пройшов перевірку на сервері та був виконаний, результат на рисунку 4.3.

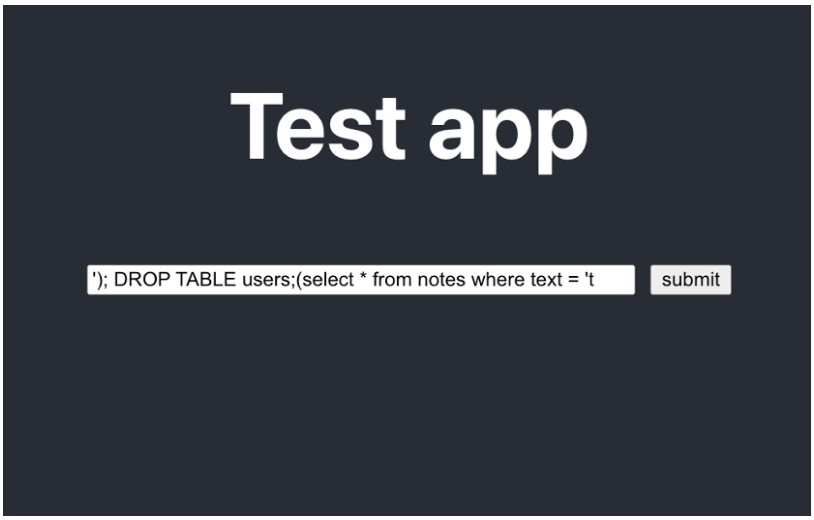

Рисунок 4.2 – Введення зловмисного коду в веб-додатку

```
sql: "INSERT INTO notes (id,text) VALUES (265052, ''); DROP TABLE users;(select * from notes where text = 't')",
parameters: {}
```

Рисунок 4.3 – Виконання зловмисного коду на сервері

Внаслідок експериментальних досліджень, що продемонстрували обмежену ефективність існуючих методів захисту від ін'єкцій коду, було прийнято рішення про розробку модуля штучного інтелекту з метою виявлення потенційних загроз.

На початковому етапі було здійснено пошук та аналіз набору даних, що містив приблизно 34 000 прикладів запитів, включаючи ін'єкції коду, частину цього набору можна оглянути на рисунку 4.4.

	Sentence	Label
1	" or pg_sleep (__TIME__) --	1
2	create user name identified by pass123 tempora...	1
3	%29	1
4	' AND 1 = utl_inaddr.get_host_address ((S...	1
5	select * from users where id = '1' or @ @1 = ...	1
...	...	...
33756	syrett	0
33757	arrechea bellveh	0
33758	1664	0
33759	almaluez	0
33760	f6lo40r06	0

33725 rows × 2 columns

Рисунок 4.4 – Частина набору даних

Далі було видалено дані які дублювалися чи були порожні.

Для трансформації рядкових значень в числові дані у контексті створення моделі штучного інтелекту, використовувалась бібліотека NLTK, як показано на рисунку 4.5.

NLTK є провідною платформою для створення програм Python для роботи з даними людської мови. Він надає прості у використанні інтерфейси для більш ніж 50 корпусів і лексичних ресурсів , таких як WordNet, а також набір бібліотек для обробки тексту для класифікації, токенізації, сформування основи, тегування, синтаксичного аналізу та семантичного міркування [28].

```
import nltk
import os
nltk.download('stopwords', download_dir=os.path.abspath(""))

stop_words = stopwords.words('english')


vectorizer = CountVectorizer( )
X = vectorizer.fit_transform(X).toarray()

X
```

Рисунок 4.5 – Трансформація рядкових значень в числа

Після цього етапу дані було розділено на дві незалежні частини: одну для тренування моделі та іншу для оцінки результатів, про що свідчить рисунок 4.6. Цей процес виконувався за допомогою одного з модулів бібліотеки sklearn.

Scikit-learn — це безкоштовна програмна бібліотека машинного навчання для мови програмування Python, яка надає функціональність для створення та тренування різноманітних алгоритмів класифікації, регресії та кластеризації і працює у зв'язці з бібліотеками NumPy та SciPy. Scikit-learn є однією з найбільш популярних бібліотек машинного навчання[29].

```
from sklearn.model_selection import train_test_split


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(26980, 40739)
(26980,)
(6745, 40739)
(6745,)
```

Рисунок 4.6 – Розділення даних за допомогою sklearn

Після виконання попередніх кроків, було прийнято рішення використати бібліотеку TensorFlow для розробки та навчання моделі.

TensorFlow – це безкоштовна бібліотека програмного забезпечення з відкритим кодом для машинного навчання та штучного інтелекту [30].

Після послідовної реалізації дослідницьких експериментів, було визначено необхідність створення моделі, що складається з чотирьох рівнів: два з них є рівнями Conv1D, один рівень - Flatten, та один - Dense. Докладний опис архітектури моделі представлено на рисунку 4.7.

Layer (type)	Output Shape	Param #
conv1d_2 (Conv1D)	(None, 1, 32)	1,303,680
conv1d_3 (Conv1D)	(None, 1, 32)	1,056
flatten_1 (Flatten)	(None, 32)	0
dense_1 (Dense)	(None, 1)	33

Рисунок 4.7 – Архітектура створеної моделі

Conv1D – це згортковий шар для обробки одновимірних послідовностей. Він використовується для виявлення локальних шаблонів у вхідних даних. Під час згортки вікно пройде по вхідному сигналу, виконуючи операцію згортки з фільтрами. Це дозволяє моделі відділювати важливі ознаки від вхідних даних [31].

Flatten – це шар, який перетворює вхідні дані з багатовимірного формату у вектори, що містять всі значення. Використовується для перетворення вихідних даних з попередніх шарів у формат, придатний для подальшої обробки шарами Dense [32].

Dense – це стандартний шар нейронної мережі, в якому кожен нейрон пов'язаний з кожним нейроном попереднього та наступного шарів. У шарі Dense кожен нейрон оброблює вхідні дані, перемножаючи їх на ваги, додаючи

зміщення та застосовуючи активаційну функцію. Він використовується для створення повнозв'язаних шарів у нейронних мережах [33].

Для оптимізації процесу навчання моделі було вибрано оптимізатор Adam та функцію втрат BinaryCrossentropy. Крім того, кількість епох навчання було встановлено на 10, щоб уникнути перенавчання моделі. Повне налаштування моделі відображено на рисунку 4.8.

Adam – це адаптивний метод оптимізації, який використовується для навчання нейронних мереж. Він поєднує в собі переваги двох інших методів оптимізації: методу адаптивного кроку навчання та методу експоненційно зваженого згортання по моменту. Adam використовує експоненційні середні моментів градієнтів і квадратів градієнтів для адаптивного налаштування кроку навчання для кожного параметра. Це дозволяє ефективно оптимізувати функцію втрат, зокрема в умовах великої кількості параметрів та даних [34].

BinaryCrossentropy – це функція втрат, яка використовується для бінарної класифікації, коли модель виробляє вихід у вигляді ймовірності належності прикладів до одного з двох класів. Вона обчислює втрату між справжніми та передбаченими значеннями. Для кожного прикладу вона обчислює втрату на основі розподілу ймовірностей для двох класів, та потім обчислює середню втрату по всіх прикладах. Ця функція відображає, наскільки віддалено передбачені ймовірності від справжніх міток класів [35].

```
model = models.Sequential()
model.add(layers.Conv1D(32, 1, activation = 'relu',
                        input_shape = (1,40739)))
model.add(layers.Conv1D(32, 1, activation = 'relu'))
model.add(layers.Flatten())
model.add(layers.Dense(1, activation = 'sigmoid'))
model.summary()
model.compile(optimizer = tf.keras.optimizers.Adam(learning_rate=0.00005),
              loss = tf.keras.losses.BinaryCrossentropy(),
              metrics = ['accuracy'])

X_train1 = X_train.reshape(-1, 1, 40739)
X_test1 = X_test.reshape(-1, 1, 40739)

history = model.fit(X_train1, y_train, epochs = 10,
                    validation_data = (X_test1, y_test))
```

Рисунок 4.8 – Повне налаштування моделі

Після цього етапу був проведений процес навчання моделі, після чого отриману модель було збережено для подальшого використання. Результати навчання відображені на рисунку 4.9.

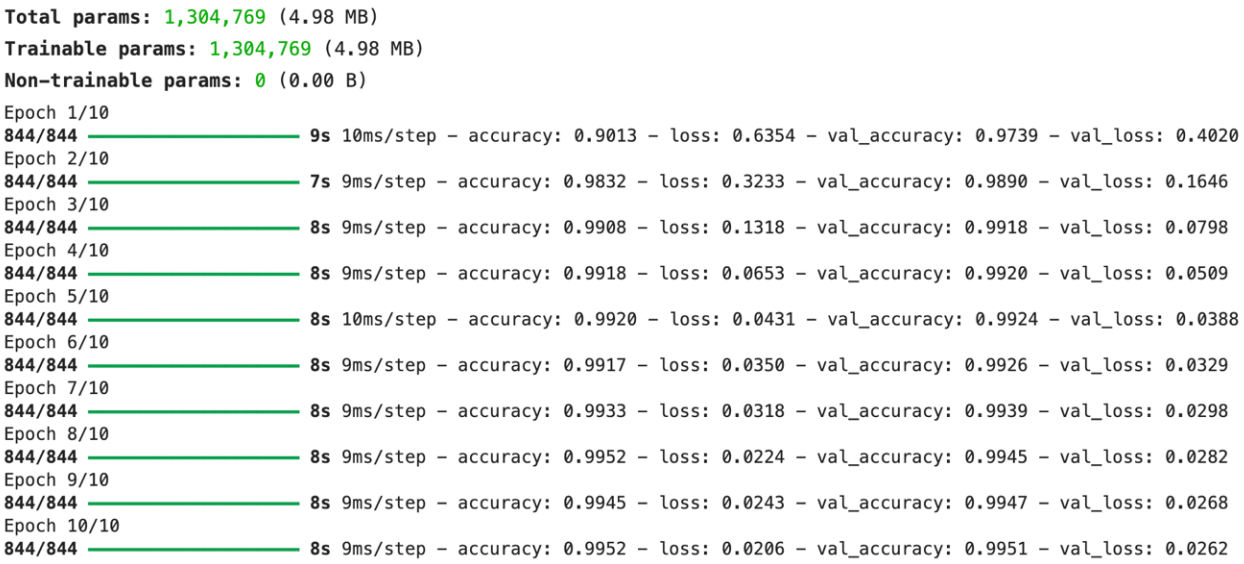

Рисунок 4.9 – Процес навчання моделі

Після цього був створений сервіс, який отримував на вхід рядок із даними та проводив їх перевірку за допомогою раніше створеної моделі штучного інтелекту, повертаючи ймовірність того, що введені дані є небезпечними. Реалізація цього сервісу може бути переглянута на рисунку 4.10.

```python
from flask import Flask, request
import joblib
import numpy as np
import tensorflow as tf


app = Flask(__name__)

new_model = tf.keras.models.load_model('my_model.keras')
loaded_cvec = joblib.load("finalized_countvectorizer.sav")


@app.route("/", methods=['POST'])
def your_route():
    data = request.data.decode('utf-8')

    value = np.array([data])

    new_value = loaded_cvec.transform(value).toarray()
    new_value = new_value.reshape(-1, 1, 40739)
    res = new_model.predict(new_value)

    return str(res[0][0])
```

Рисунок 4.10 – Реалізація сервісу перевірки даних

Після інтеграції сервісу з існуючою серверною частиною, була проведена повторна спроба ін'єкції коду. Модуль штучного інтелекту визначав небезпечні дані, а дані, які не є небезпечними, пропускав. Результати цих випробувань можна побачити на рисунках 4.11 та 4.12.

```
Input data: select I'm happy to say. Sure, I do some validation along the
way, but now it's just 5% of my job.
Code injection probability: 0.00002
Executing (default): INSERT INTO notes (id,text) VALUES (48076, 'select I'
m happy to say. Sure, I do some validation along the way, but now it's jus
t 5% of my job. ')
```

Рисунок 4.11 – Результат запиту до сервісу з безпечними даними

Рисунок 4.12 – Результат запиту до сервісу з ін'єкцією коду

Створений сервіс з використанням моделі штучного інтелекту успішно виконує завдання перевірки вхідних даних і виявлення потенційно небезпечних ін'єкційних вразливостей. Це може свідчити про те, що даний метод, є ефективнішим у виявленні таких проблем порівняно з традиційними методами захисту.

4.3 Оцінка результатів створеного методу

Перевірка створеної моделі штучного інтелекту є надзвичайно важливою для забезпечення її ефективності, надійності та безпеки в реальних умовах експлуатації. Цей процес включає в себе не лише оцінку точності та ефективності моделі на тестових даних, а й перевірку її вразливостей, стійкості до атак, а також виявлення можливих викривлень чи перекосів у вирішенні завдань. Перевірка моделі допомагає підтвердити її відповідність задачам, для яких вона призначена, та визначити можливі обмеження або потенційні проблеми, які потребують виправлення. Тільки завдяки систематичному та комплексному підходу до перевірки можна забезпечити надійність та довіру до використання штучного інтелекту в різних задачах.

Графік навчальної та валідаційної втрат є ключовим інструментом для оцінки продуктивності моделі під час навчання. Навчальна втрата відображає, як ефективно модель вивчає навчальні дані, тоді як валідаційна втрата вказує на її здатність до узагальнення на невідомі дані. Зазвичай на початку навчання обидва типи втрат зменшуються, але після певної кількості епох навчальна втрата може продовжувати зменшуватися, тоді як валідаційна втрата може почати зростати, що може свідчити про перенавчання. Аналіз графіків training та validation loss

допомагає визначити оптимальну кількість епох навчання і виявити потенційні проблеми, такі як недостатня або надмірна інтенсивність навчання. З даного графіку, який представлений на рисунку 4.13, ми можемо зрозуміти, що не відбулося перенавчання моделі.

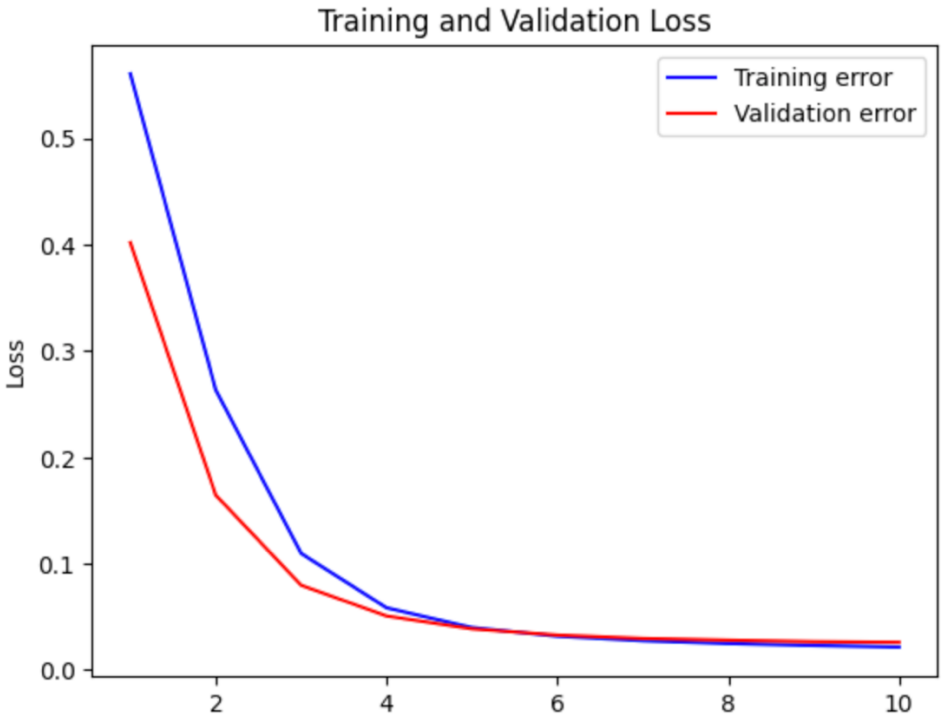

Рисунок 4.13 – Графік втрат для створеної моделі

Графік точності навчання та валідації (training and validation accuracy) є ще одним важливим інструментом для аналізу ефективності моделі під час навчання. Точність навчання відображає, наскільки добре модель навчається на навчальних даних, тоді як точність валідації показує, наскільки добре модель узагальнює свої знання на невідомих даних. Графік точності моделі можна побачити на рисунку 4.14.

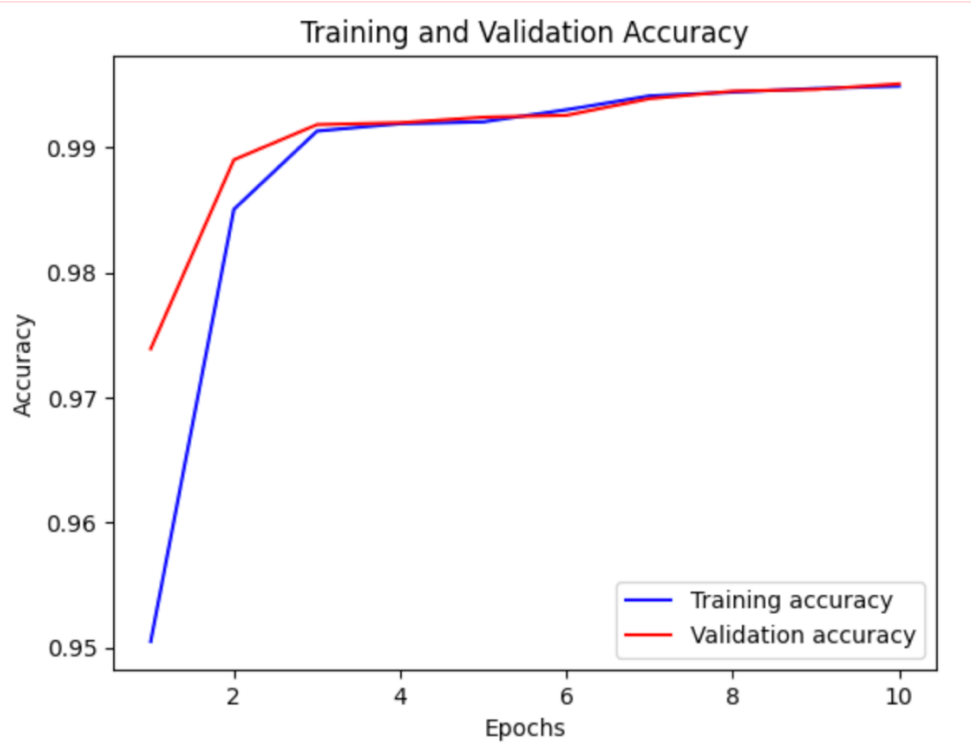

Рисунок 4.14 – Графік точності для створеної моделі

Був проведений тест на точність згорткової нейронної мережі (CNN) та оцінка F1 для CNN на тестових даних. Результати цих тестів можна побачити на рисунку 4.15.

Точність згорткової нейронної мережі (Accuracy of CNN) – це показник точності моделі згорткових нейронних мереж на наборі даних. Це вимірюється як відношення кількості правильно класифікованих прикладів до загальної кількості прикладів у тестовому наборі. Точність дає загальне уявлення про те, наскільки ефективно модель класифікує дані.

Оцінка F1 для CNN (F1 Score of CNN) – це показник F1-оцінки моделі згорткових нейронних мереж на наборі даних. F1-оцінка використовується для оцінки точності та повноти класифікації. Вона обчислюється як гармонічне середнє між точністю (відношенням правильно класифікованих екземплярів певного класу до всіх екземплярів, визначених як цей клас) та повнотою (відношенням правильно класифікованих екземплярів певного класу до всіх екземплярів, які дійсно належать до цього класу). F1-оцінка дає більш

збалансовану міру ефективності моделі, особливо в умовах незбалансованих класів даних.

```
Accuracy of CNN on test set : 0.9951074870274277
F1 Score of CNN on test set : 0.9927007299270073
```

Рисунок 4.15 – Результати проведених тестів

У кінці була створена матриця помилок на тестовому наборі даних, її результат можна побачити на рисунку 4.16.

Матриця помилок (confusion matrix) є важливим інструментом для оцінки ефективності класифікаційних моделей, таких як згорткові нейронні мережі. Вона надає детальну інформацію про те, як модель класифікує дані для кожного класу. У матриці помилок кожен рядок представляє фактичні класи, а кожний стовпець прогнозовані класи. Значення у кожній клітинці матриці відображають кількість прикладів, що належать до конкретного класу, які були правильно або неправильно класифіковані моделлю.

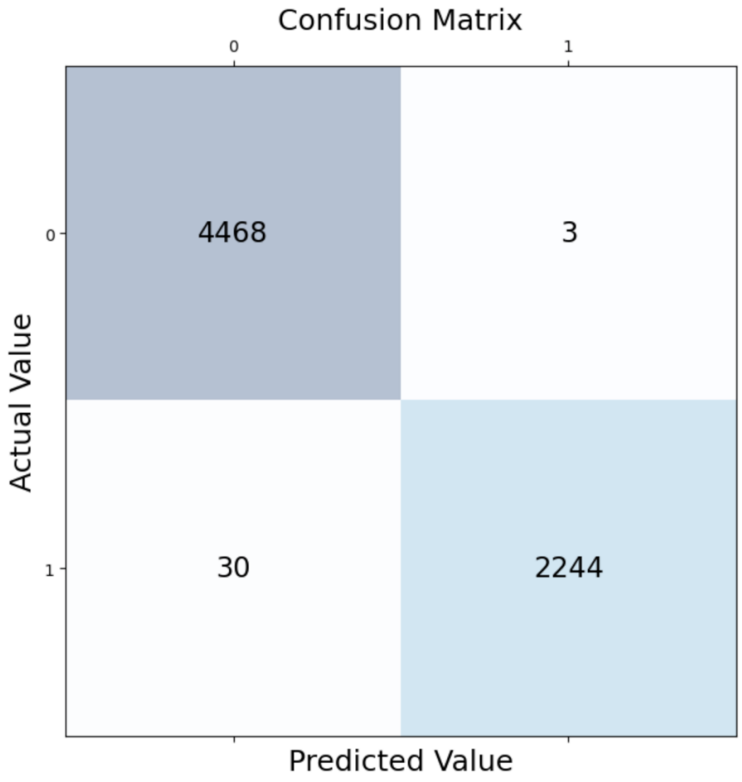

Рисунок 4.16 – Матриця помилок

Після проведення тестів і аналізу результатів матриці помилок, точності, оцінки F1 та інших метрик ефективності можна зробити висновок, що модель штучного інтелекту працює добре. Це підтверджується тим, що модель здатна ефективно класифікувати дані на тестовому наборі з високою точністю та низьким рівнем помилок. Такий результат свідчить про те, що модель може надійно застосовуватися для вирішення конкретних завдань класифікації даних і може бути успішно впроваджена в практичні застосування.

Висновок до розділу 4

Починаючи з аналізу існуючих методів захисту, було докладно розглянуто їх переваги та недоліки з метою визначення областей для подальшого вдосконалення. Важливим етапом цього аналізу є виявлення потенційних вразливостей і недоліків, які можуть бути використані зловмисниками для здійснення ін'єкцій коду та інших атак на систему.

Після аналізу існуючих методів захисту було перейдено до розробки вдосконаленого методу захисту. Було описуємо процес розробки та імплементації нового методу, який має на меті підвищення ефективності та надійності захисту від ін'єкцій коду. Важливою складовою цього процесу є урахування вивчених під час аналізу існуючих методів проблем та недоліків з метою їх виправлення в новій розробці.

Також був проведений аналіз моделі штучного інтелекту, яка використовується для виявлення потенційних загроз ін'єкцій коду. Під час аналізу моделі було досліджено її точність, чутливість до атак, реакцію на навчальні та тестові дані, а також інші ключові метрики ефективності. Цей аналіз допомагає зрозуміти, наскільки надійно модель може виявляти та запобігати

ін'єкціям коду, а також визначити можливі шляхи для покращення її ефективності та надійності.

ВИСНОВКИ

Досліджено різноманітні способи побудови архітектури програмного забезпечення, зокрема клієнт-серверну, компонентну, мікросервісну та гібридну. Кожен з цих підходів має свої переваги та обмеження, але вони надають широкі можливості для розробки різноманітних програмних рішень з урахуванням специфіки проекту та потреб користувачів. Виявлено, що компонентна архітектура сприяє модульності та перевикористанню коду, тоді як мікросервісна архітектура дозволяє побудувати гнучкі та масштабовані додатки.

У контексті дослідження загроз компонентам архітектури були виявлені потенційні атаки на рівні мережі, сервісу та контейнера, які становлять серйозні загрози для безпеки системи. Під час аналізу було з'ясовано, що для забезпечення безпеки даних в хмарних середовищах необхідно використовувати комплексний підхід, що включає в себе захист на рівні мережі, моніторинг подій та застосування сучасних засобів шифрування та аутентифікації.

Розділ, присвячений вдосконаленню методів захисту архітектури, показав, що створення ефективних заходів безпеки від ін'єкцій коду вимагає поєднання аналізу існуючих методів з новими підходами. Результати аналізу допомогли розробити методи, які мають потенціал для підвищення рівня безпеки системи та захисту її від зловмисних атак.

Дослідження виявило, що забезпечення безпеки архітектури програмного забезпечення є складною, але надзвичайно важливою задачею. Підвищення рівня захисту системи передбачає комплексний підхід, що включає в себе аналіз і вдосконалення методів захисту, використання сучасних технологій та постійний моніторинг стану безпеки. Тільки такий підхід може забезпечити надійний захист від різноманітних загроз та забезпечити безпечну експлуатацію програмного забезпечення в сучасному цифровому середовищі.

Створений метод захисту від ін'єкцій коду може бути використаний для покращення безпеки різних компонентів архітектури програмного забезпечення.

Використання цього методу при спілкуванні з базою даних дозволить запобігти потенційним атакам на дані через SQL ін'єкції. Також цей метод може бути ефективним при обробці клієнтських запитів та взаємодії з різними сервісами, забезпечуючи надійний захист від вразливостей і забезпечуючи цілісність та безпеку обробки даних. Використання розробленого методу підвищить рівень безпеки та стійкості архітектури програмного забезпечення в умовах постійно зростаючих загроз та викликів в цифровому середовищі.

Результати кваліфікаційної роботи доповідалися на міжнародній науково-практичній конференції «Проблеми кібербезпеки інформаційно-телекомунікаційних систем» 26 квітня 2024 р. на тему "Improving method of protecting software architecture components" [53].

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Software architecture [Електронний ресурс]: https://en.wikipedia.org/wiki/Software_architecture

2. Client-server Architecture [Електронний ресурс]: https://cs.uwaterloo.ca/~m2nagapp/courses/CS446/1195/Arch_Design_Activity/ClientServer.pdf

3. 2014 Лекції ТСПП [Електронний ресурс]: https://studfile.net/preview/5200675/page:22/

4. Мікросервісна архітектура [Електронний ресурс]: https://medium.com/@IvanZmerzlyi/microservices-architecture-461687045b3d

5. Мікросервісна архітектура ПЗ [Електронний ресурс]: https://qalight.ua/baza-znaniy/shho-take-mikroservisna-arhitektura-pz/

6. РЕАЛІЗАЦІЯ ГІБРИДНОЇ АРХІТЕКТУРИ [Електронний ресурс]: https://science.lpnu.ua/sites/default/files/journal-paper/2018/jun/12973/12108-118.pdf

7. Denial-of-service attack [Електронний ресурс]: https://en.wikipedia.org/wiki/Denial-of-service_attack

8. Атаки MITM і методи захисту від них [Електронний ресурс]: https://hackyourmom.com/kibervijna/shpargalka-mitm/

9. Що таке спуфінг і як запобігти атаці? [Електронний ресурс]: https://nncit.wunu.edu.ua/shho-take-spufing-i-yak-zapobigty-ataczi/

10. Як уникнути атаки підслуховування? [Електронний ресурс]: https://www.teknikservis.com/uk/blog/it-hizmetleri/dinleme-saldirisindan-nasil-korunulur/

11. OWASP Top 10 - 2021 [Електронний ресурс]: https://owasp.org/Top10/

12. WHAT IS CODE INJECTION? [Електронний ресурс]: https://www.contrastsecurity.com/glossary/code-injection

13. A03:2021 – Injection [Електронний ресурс]: https://owasp.org/Top10/A03_2021-Injection/

14. Дослідження методів багатофакторної автентифікації та їх практичного застосування [Електронний ресурс]: https://openarchive.nure.ua/server/api/core/bitstreams/087ebf7e-8e83-42d0-b50d-5f69b199a764/content

15. OWASP Top 10 Identification and Authentication Failures [Електронний ресурс]: https://www.securityjourney.com/post/owasp-top-10-identification-and-authentication-failures

16. НЕПРАВИЛЬНЕ НАЛАШТУВАННЯ БЕЗПЕКИ [Електронний ресурс]: https://cqr.company/ua/web-vulnerabilities/security-misconfiguration/

17. Security Misconfiguration: Impact, Examples and Prevention [Електронний ресурс]: https://www.balbix.com/insights/security-misconfiguration-impact-examples-and-prevention/

18. A05:2021 – Security Misconfiguration [Електронний ресурс]: https://owasp.org/Top10/A05_2021-Security_Misconfiguration/

19. 2022 CLOUD SECURITY REPORT [Електронний ресурс]: https://www.cybersecurity-insiders.com/portfolio/2022-cloud-security-report-isc2

20. ВІДДАЛЕНЕ ВИКОНАННЯ КОДУ [Електронний ресурс]: https://cqr.company/ua/web-vulnerabilities/rce/

21. Unauthorized Access [Електронний ресурс]: https://www.cynet.com/network-attacks/unauthorized-access-5-best-practices-to-avoid-the-next-data-breach/

22. Як підвищити рівень мережевої безпеки [Електронний ресурс]: https://channel4it.com/publications/yak-pdvishchiti-rven-merezhevo-bezpeki-7-osnovnih-sposobv-.html

23. Захист даних [Електронний ресурс]: https://www.intrasystems.ua/solutions/informacyna-bezpeka/zahyst-danyh/

24. Що таке захист даних? [Електронний ресурс]: https://microsoft.com/uk-ua/security/business/security-101/what-is-data-security

25. ЩО таке SIEM? [Електронний ресурс]: https://www.microsoft.com/uk-ua/security/business/security-101/what-is-siem

26. Системи моніторингу та керування [Електронний ресурс]: https://it-solutions.ua/servisi/sistemi-monitoringu-ta-keruvannya

27. Understanding Code Injection [Електронний ресурс]: https://woobewoo.com/glossary/understanding-code-injection-types-prevention-and-fixes/

28. NLTK Documentation [Електронний ресурс]: https://www.nltk.org/

29. Scikit-learn [Електронний ресурс]: https://uk.wikipedia.org/wiki/Scikit-learn

30. TensorFlow [Електронний ресурс]: https://en.wikipedia.org/wiki/TensorFlow

31. 1D convolution layer [Електронний ресурс]: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv1D

32. Flattens the input [Електронний ресурс]: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Flatten

33. Just your regular densely-connected NN layer [Електронний ресурс]: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense

34. Optimizer that implements the Adam algorithm [Електронний ресурс]: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam

35. Computes the cross-entropy loss between true labels and predicted labels [Електронний ресурс]: https://www.tensorflow.org/api_docs/python/tf/keras/losses/BinaryCrossentropy

36. ISO/IEC 27001:2013 Information technology – Security techniques – Information security management systems – Requirements. – Implemented on October 15, 2013. – Geneva: International Organization for Standardization, 2013. – 46 p.

37. ISO/IEC 27002:2013 Information technology – Security techniques – Code of practice for information security controls. – Implemented on October 15, 2013. – Geneva: International Organization for Standardization, 2013. – 114 p.

38. ISO/IEC 27005:2018 Information technology – Security techniques – Information security risk management. – Implemented on June 12, 2018. – Geneva: International Organization for Standardization, 2018. – 54 p.

39. ISO/IEC 27017:2015 Information technology – Security techniques – Code of practice for information security controls based on ISO/IEC 27002 for cloud services. – Implemented on August 15, 2015. – Geneva: International Organization for Standardization, 2015. – 32 p.

40. ISO/IEC 27018:2019 Information technology – Security techniques – Code of practice for protection of personally identifiable information (PII) in public clouds acting as PII processors. – Implemented on November 25, 2019. – Geneva: International Organization for Standardization, 2019. – 38 p.

41. Data Security [Електронний ресурс]: https://www.imperva.com/learn/data-security/data-security

42. What is Data Security? [Електронний ресурс]: https://www.opentext.com/what-is/data-security

43. Secure your applications with strategic architecture [Електронний ресурс]: https://dig8ital.com/post/appsec-architecture

44. Application Security Architecture [Електронний ресурс]: https://avinetworks.com/glossary/application-security-architecture/

45. Attributes of secure web application architecture [Електронний ресурс]: https://www.synopsys.com/blogs/software-security/attributes-of-secure-web-application-architecture.html

46. Application security architecture review [Електронний ресурс]: https://www.guidepointsecurity.com/application-security-architecture-review/

47. Designing and Implementing a Security Architecture [Електронний ресурс]: https://www.xcubelabs.com/blog/designing-and-implementing-a-security-architecture/

48. Cybersecurity Best Practices [Електронний ресурс]: https://www.cisa.gov/topics/cybersecurity-best-practices

49. What is Cybersecurity and Why It is Important? [Електронний ресурс]: https://www.simplilearn.com/tutorials/cyber-security-tutorial/what-is-cyber-security

50. 9 Cybersecurity Best Practices for Businesses in 2024 [Електронний ресурс]: https://www.coursera.org/articles/cybersecurity-best-practices

51. Cybersecurity Tips and Best Practices for Your Business [Електронний ресурс]: https://www.titanfile.com/blog/cyber-security-tips-best-practices/

52. Cybersecurity Best Practices for 2024 [Електронний ресурс]: https://networkats.com/cybersecurity-best-practices-2024/

53. Larysa Myrutenko, Andrii Khomyn, "Improving method of protecting software architecture components", Міжнародна науково-практична конференція «Проблеми кібербезпеки інформаційно-телекомунікаційних систем» № 7, 26 квітня 2024 р. С. 73–75. — (28)