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
«__» _____ 2025 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи

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

спеціальність *125 Кібербезпека та захист інформації*
(код і назва спеціальності)

освітній ступень *магістр*

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

на тему: «Метод виявлення вразливостей для SQL-ін’єкцій у веб-застосунках з використанням штучного інтелекту»

Виконавець: студент ІІ курсу, групи КБм-21

_____ **Олексій ПОДУС**
(підпис) (Ім’я, ПРІЗВИЩЕ)

	Ім’я, ПРІЗВИЩЕ	**Підпис**
Науковий керівник	Лариса МИРУТЕНКО	
Нормоконтроль	Юрій БАБЕНКО	

Київ 2025

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
«25» жовтня 2024 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи

спеціальності *125 Кібербезпека та захист інформації*
(код і назва спеціальності)

освітній ступень *магістр*

Здобувача(ки) КБм-21 (група) Подуса Олексія Сергійовича (прізвище ім'я по-батькові)

Тема кваліфікаційної роботи Метод виявлення вразливостей для SQL-ін'єккцій у веб-застосунках з використанням штучного інтелекту

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Рішення засідання кафедри кібербезпеки та захисту інформації факультету інформаційних технологій протокол № 4 від 24.10.2024 р.

2. МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

.

Об'єкт досліджень	Процес виявлення SQL-ін'єкцій у веб-застосунках.
Предмет досліджень	Методи використання моделей великих мов для аналізу та виявлення SQL-ін'єкцій, ефективність та точність таких методів.
Мета	Розробка та аналіз методу виявлення SQL-ін'єкцій на основі моделей великих мов у веб-застосунках
Вихідні дані для проведення роботи	Набори SQL-ін'єкцій (зловмисних та легітимих) та моделі штучого інтелекту

3. ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Наукова новизна запропоновано метод виявлення SQL-ін'єкцій на основі моделей великих мов у веб-додатках, що дозволяє підвищити точність та ефективність аналізу порівняно з традиційними методами.

Практична цінність використання методу в комбінуванні з сучасними методами, підвищення ефективності та точності виявлення SQL вразливостей

4. ЕТАПИ ВИКОНАННЯ РОБОТИ

Найменування етапів робіт	Строки виконання робіт (початок-кінець)
Уточнення постановки задачі	25.10.2024 – 08.12.2024
Аналіз літературних джерел	09.12.2024 – 17.01.2025
Ознайомлення з сучасними методами захисту від SQL-ін'єкцій у веб-застосунках	18.02.2025 – 20.01.2025
Формування набору тестових запитів та вибір AI-моделей для аналізу	21.01.2025 – 23.01.2025
Розробка програмного застосунку з використанням ШІ	24.01.2025 – 05.02.2025
Проведення експериментального дослідження та тестування	06.02.2025 – 15.02.2025
Аналіз результатів експериментів, оцінка ефективності та перспектива розвитку	16.02.2025 – 24.02.2025
Оформлення пояснювальної записки згідно методичних рекомендацій	25.02.2025 – 15.05.2025
Подача пакету документів на розгляд ЕК	15.05.2025 – 19.05.2025

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

Завдання прийняв до виконання _____ Олексій ПОДУС
(підпис) (Ім'я, ПРІЗВИЩЕ)

Дата видачі завдання: 25.10.2024 р.

Термін подання кваліфікаційної роботи до ЕК 19.05.2025 р.

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Метод виявлення вразливостей для SQL-ін'єкцій у веб-застосунках з використанням штучного інтелекту»: 82 сторінки, 27 рисунків, 3 таблиці, 48 літературних джерел.

Мета роботи – розробка та аналіз методу виявлення SQL-ін'єкцій на основі моделей великих мов у веб-додатках.

Об’єкт дослідження – процес виявлення SQL-ін'єкцій у веб-додатках.

Предмет дослідження – методи використання LLM для аналізу та виявлення SQL-ін'єкцій, ефективність та точність таких методів.

Наукова новизна: запропоновано метод виявлення SQL-ін'єкцій на основі LLM у веб-додатках, що дозволяє підвищити точність та ефективність аналізу порівняно з традиційними методами.

У роботі розглянуто основні типи SQL-ін'єкцій, методи захисту та існуючі інструменти виявлення. Проаналізовано можливості використання LLM для виявлення та запобігання SQL-ін'єкціям. Розроблено веб-додаток, що реалізує запропонований метод, та проведено експериментальні дослідження на реальних кейсах SQL-ін'єкцій.

Актуальність роботи: SQL-ін'єкції є одними з найпоширеніших та найнебезпечніших кібератак. Використання штучного інтелекту, зокрема LLM, дозволяє підвищити ефективність виявлення та запобігання таким атакам.

Ключові слова: SQL-ін'єкції, кібербезпека, штучний інтелект, LLM, виявлення атак, аналіз безпеки

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

SQL – Structured Query Language
ШІ – Штучний Інтелект
LLM – Large Language Model
SQLI – Structured Query Language Injection
APT – Advanced Persistent Threat
IT – Інформаційні технології
HTTP – Hypertext Transfer Protocol
URL – Uniform Resource Locator
DNS – Domain Name System
БД – База Даних
HTML – Hypertext Markup Language
API – Application Programming Interface
XSS – Cross-site Scripting
PCI DSS – Payment Card Industry Data Security Standard
SIEM – Security Information and Event Management
DAST – Dynamic Application Security Testing
SAST – Static Application Security Testing
IAST – Interactive Application Security Testing
WAF – Web Application Firewall
ML – Machine Learning
DL – Deep Learning
CSS – Cascading Style Sheets
SSR – Side Server Rendering
SSG – Static Site Generator
UI – User Interface
CVE – Common Vulnerabilities and Exposures
OWASP – Open Worldwide Application Security Project

ЗМІСТ

ВСТУП

В сучасних умовах розвитку інформаційних технологій питання захисту веб-застосунків від кіберзагроз набуває особливої актуальності. Однією з найпоширеніших та найнебезпечніших атак на веб-застосунки є SQL-ін'єкції. Ці атаки дозволяють зловмисникам маніпулювати базами даних, отримуючи несанкціонований доступ до конфіденційної інформації, змінюючи або видаляючи дані.

Ефективне виявлення та запобігання SQL-ін'єкціям є критично важливим для забезпечення безпеки веб-застосунків та захисту даних користувачів. Традиційні методи виявлення вразливостей часто виявляються недостатньо ефективними перед складними та постійно еволюціонуючими атаками. У зв'язку з цим, використання ШІ для виявлення SQL-ін'єкцій стає перспективним напрямом досліджень.

Штучний інтелект, зокрема моделі великих мов, демонструють значний потенціал у сфері кібербезпеки. Їх здатність аналізувати великі обсяги даних, виявляти аномалії та навчатися на прикладах робить їх ефективним інструментом для виявлення складних кібератак, таких як SQL-ін'єкції.

У цій роботі досліджуються можливості застосування моделей великих мов для виявлення SQL-ін'єкцій у веб-застосунках. Розглядаються основні типи SQL-ін'єкцій, методи захисту та існуючі інструменти виявлення. Проаналізовано потенціал використання LLM для підвищення ефективності виявлення та запобігання SQL-ін'єкціям.

У роботі також представлено огляд сучасних досліджень у сфері застосування штучного інтелекту для кібербезпеки, зокрема для виявлення вразливостей веб-застосунків. Розглянуто різні підходи та методи, що використовуються виявлення потенційних загроз. Розроблено веб-додаток, що реалізує запропонований метод, та проведено експериментальні дослідження на реальних кейсах SQL-ін'єкцій.

РОЗДІЛ 1
АНАЛІЗ SQL-ІН’ЄКЦІЙ ТА ІСНУЮЧИХ МЕТОДІВ ЇХ ВИЯВЛЕННЯ

1.1.SQL-ін'єкції: визначення, механізм атаки та наслідки

SQL-ін'єкція, також відома як SQLI, є поширеним вектором атаки, який використовує шкідливий SQL-код для маніпулювання внутрішніми базами даних з метою отримання доступу до інформації, яка не призначена для відображення. Ця інформація може включати будь-яку кількість елементів, в тому числі конфіденційні дані компанії, списки користувачів або приватні дані клієнтів.

Вплив SQL-ін'єкцій на бізнес може мати далекосяжні наслідки. Успішна атака може призвести до несанкціонованого перегляду списків користувачів, видалення цілих таблиць, а в деяких випадках - до отримання зловмисником адміністративних прав на базу даних, що може завдати значної шкоди бізнесу.

Підраховуючи потенційну вартість SQLi, важливо враховувати втрату довіри клієнтів у разі викрадення особистої інформації, такої як номери телефонів, адреси та дані кредитних карток.

Хоча цей вектор може бути використаний для атаки на будь-яку базу даних SQL, веб-сайти є найчастішими цілями [1].

Що таке SQL-запити [2]:

SQL, що розшифровується як Structured Query Language (мова структурованих запитів), – це мова, яка використовується для спілкування з базами даних та маніпулювання ними. SQL-запити – це ті мови, які є стандартними для систем управління реляційними базами даних і використовуються для виконання таких завдань, як оновлення, або отримання даних з бази.

SQL-запити – це команди, що використовуються для спілкування з базою даних. Вони дозволяють виконувати такі завдання, як пошук, оновлення або отримання даних, що зберігаються в базі даних. Наприклад, веб-додаток для

електронної комерції може використовувати SQL-запит для отримання інформації про конкретний товар з бази даних товарів.

Однак, якщо додаток не розроблений безпечно, він може дозволити користувачеві вводити SQL-запити безпосередньо. У такому випадку зловмисник може ввести спеціально створений SQL-запит, який може призвести до несанкціонованого доступу, витоку даних або навіть їх видалення. Це називається SQL-ін'єкція.

Атака SQL Injection передбачає вставку SQL-запиту через вхідні дані від клієнта до програми. Успішна атака дозволяє зловмиснику маніпулювати SQL-запитами, які додаток надсилає до своєї бази даних. Зазвичай вона складається з наступних кроків:

Визначення вразливих вхідних даних: Зловмисники спочатку визначають вхідні дані у веб-додатку, які є вразливими для SQL-ін'єкцій. Це можуть бути текстові поля у формі, параметри URL-адреси або будь-які інші механізми введення.

Створення шкідливого SQL-запиту: Після виявлення вразливого поля зловмисники створюють інструкцію SQL, яку потім вставляють у запит, що виконується додатком. Цей оператор призначений для модифікації початкового SQL-запиту для виконання дій, не передбачених розробниками програми.

Обхід заходів безпеки програми: Зловмисникам часто доводиться обходити заходи безпеки, такі як перевірка введення або екранування спеціальних символів. Вони досягають цього за допомогою таких методів, як конкатенація рядків або використання синтаксису SQL для коментування частин оригінального запиту.

Виконання шкідливого запиту: Коли програма виконує SQL-запит, вона включає зловмисне введення зловмисника. Цей модифікований запит може виконувати такі дії, як несанкціонований перегляд даних, видалення даних або навіть зміна схеми бази даних.

Вилучення або маніпулювання даними: Залежно від типу атаки, результатом може бути вилучення конфіденційної інформації (наприклад, облікових даних користувача), зміна існуючих даних, додавання нових даних або навіть видалення значної частини бази даних.

Використання вразливостей сервера баз даних: Складні SQL-ін'єкції можуть використовувати вразливості в сервері бази даних, поширюючи атаку за межі бази даних на рівень сервера. Це може включати виконання команд в операційній системі або доступ до інших частин файлової системи сервера.

Цей процес використовує динамічне виконання SQL в додатках, де вхідні дані користувача безпосередньо включаються в оператори SQL без належної перевірки або екранування. Він використовує спосіб побудови SQL-запитів, часто таким чином, що розробники не очікували.

Реальні приклади атак SQL-ін'єкцій [2]:

За останні 20 років багато атак SQL-ін'єкцій були спрямовані на великі веб-сайти, бізнес-платформи та соціальні мережі. Деякі з цих атак призвели до серйозних витоків даних. Нижче наведено кілька найвідоміших прикладів.

Порушення, спричинені ін'єкціями SQL:

Хакери GhostShell з групи APT Team GhostShell атакували 53 університети, використовуючи SQL-ін'єкції, викрали та опублікували 36 000 особистих записів студентів, викладачів та співробітників.

Турецький уряд – інша група APT, колектив RedHack, використала SQL-ін'єкцію, щоб зламати веб-сайт турецького уряду та стерти борги перед державними установами.

7-Eleven – команда зловмисників використала SQL-ін'єкцію для проникнення в корпоративні системи кількох компаній, в першу чергу роздрібної мережі 7-Eleven, і викрала 130 мільйонів номерів кредитних карток.

HBGary – хакери, пов'язані з групою активістів Anonymous, використали SQL-ін'єкцію, щоб вивести з ладу веб-сайт компанії, що займається IT-безпекою. Атака стала відповіддю на те, що генеральний директор HBGary оприлюднив інформацію про те, що у нього є імена членів організації Anonymous.

Найвідоміші вразливості SQL Injection:

Уразливість Tesla – у 2014 році дослідники безпеки оприлюднили інформацію про те, що їм вдалося зламати веб-сайт компанії Tesla за допомогою SQL ін'єкції, отримати адміністративні привілеї та викрасти дані користувачів.

Уразливість в Cisco – у 2018 році в Cisco Prime License Manager було виявлено SQL ін'єкцію. Уразливість дозволяла зловмисникам отримати командний доступ до систем, на яких був розгорнутий менеджер ліцензій. Компанія Cisco виправила уразливість.

Уразливість у Fortnite – популярній онлайн-грі з більш ніж 350 мільйонами користувачів. У 2019 році було виявлено SQL-ін'єкційну уразливість, яка могла дозволити зловмисникам отримати доступ до облікових записів користувачів. Вразливість була виправлена.

Типовий SQL-запит до бази даних інтернет-магазину може виглядати наступним чином [1]:

```
SELECT ItemName, ItemDescription
FROM Item
WHERE ItemNumber = ItemNumber
```

Рисунок 1.1 – Приклад SQL-запиту до інтернет магазину

На основі цього веб-додаток формує рядковий запит, який надсилається до бази даних у вигляді одного SQL-запиту:

```
sql_query= "
SELECT ItemName, ItemDescription
FROM Item
WHERE ItemNumber = " & Request.QueryString("ItemID")
```

Рисунок 1.2 – Приклад SQL-запиту до інтернет магазину

На основі даних, наданих користувачем, можна згенерувати наступний SQL-запит (http://www.estore.com/items/items.asp?itemid=999):

```
SELECT ItemName, ItemDescription
FROM Item
WHERE ItemNumber = 999
```

Рисунок 1.3 – Приклад SQL-запиту до інтернет магазину

Як можна зрозуміти з синтаксису, цей запит надає назву та опис товару з номером 999.

Наслідки успішної атаки SQL-ін'єкції можуть бути наступними [2]:

Викрадення облікових даних – зловмисники можуть отримати облікові дані через SQLi, а потім видавати себе за користувачів і використовувати їхні привілеї.

Несанкціонований доступ до баз даних – зловмисники можуть отримати доступ до конфіденційних даних на серверах баз даних.

Зміна даних – зловмисники можуть змінювати або додавати нові дані до бази даних, до якої отримали доступ.

Видалення даних – зловмисники можуть видаляти записи в базі даних або видаляти цілі таблиці.

Латеральне переміщення – зловмисники можуть отримати доступ до серверів баз даних з привілеями операційної системи та використовувати ці дозволи для доступу до інших чутливих систем.

1.2. Класифікація SQL-ін'єкцій

SQL-ін'єкції зазвичай поділяються на три типи: In-band SQLi (класичні), Inferential SQLi (сліпі) та Out-of-band SQLi [2].

In-band SQLi:

Зловмисник використовує один і той самий канал зв'язку для запуску атаки та збору її результатів. Простота та ефективність In-band SQLi робить його одним з найпоширеніших типів SQLi-атак. Існує два різновиди цього методу:

Error-Based SQL Injection:

Error-Based SQL Injection – це техніка In-band SQLi, яка дозволяє зловмисникам використовувати помилки, що виводяться з бази даних, для маніпулювання її даними. Вона працює шляхом створення запиту, що спричиняє помилку, яка надає зловмиснику інформацію про структуру бази даних.

In-band SQLi дозволяє зловмисникам використовувати один канал зв’язку як для проведення атаки, так і для отримання даних. Для вилучення інформації потрібно знайти та використати вразливість у застосунку. Зазвичай така вразливість дозволяє серверу виводити помилку SQL замість очікуваних даних. Аналізуючи цю помилку, зловмисник може отримати відомості про структуру бази даних [3].

Приклад використання Error-Based Injection:

Використання арифметичних операцій для отримання помилок

При Error-Based Injection зловмисник намагається змусити сервер бази даних повернути детальне повідомлення про помилку, яке містить конфіденційну інформацію про її структуру.

Один із способів досягти цього – використання арифметичних операцій над параметром у запиті, що може призвести до помилки.

Сценарій атаки:

Припустимо, що веб-застосунок має URL-адресу, яка приймає числовий ідентифікатор товару:

https://example.com/product.php?id=10

Зловмисник може спробувати використати поділ на нуль, що призведе до помилки в SQL-сервері:

https://example.com/product.php?id=10/0

Якщо сервер бази даних не обробляє винятки коректно, він може повернути таке повідомлення про помилку:

SQL Error: Division by zero in query "SELECT * FROM products WHERE id = 10/0"

Що отримує зловмисник:

Аналізуючи помилку, зловмисник може дізнатися:

Тип бази даних – наприклад, MySQL або PostgreSQL, оскільки повідомлення про помилку залежить від конкретного SQL-сервера.

Конкретний синтаксис запиту – зокрема, що значення id напряму використовується в SQL-запиті без валідації.

Місце виконання запиту – оскільки помилка виникла в WHERE id = 10/0, стає зрозуміло, що параметр id безпосередньо впливає на SQL-запит.

Для досвідченого зловмисника цього достатньо, щоб зрозуміти вразливість і спробувати більш складні варіанти SQL-ін'єкцій для викрадення даних або зміни вмісту бази даних.

Автоматизація атаки:

Зловмисник може використати спеціальні скрипти або інструменти, такі як SQLmap, щоб тестувати різні варіанти арифметичних операцій (наприклад, id=10-99999) і виявляти вразливі місця у веб-додатку.

Union SQL Injection:

Union SQL Injection дозволяє зловмисникам витягувати конфіденційну інформацію з бази даних. Вона дозволяє зловмисникам розширювати результати, отримані за оригінальним запитом.

Union SQL-ін'єкція може призвести до вилучення вмісту бази даних і може бути використана для виконання команд на цільовому сервері. Однак зловмисники можуть використовувати оператор UNION лише в тому випадку, якщо їхній шкідливий запит має таку саму структуру, включаючи кількість і тип даних у стовпцях, що й оригінальний запит [4].

Основні відмінності між Union-Based та Error-Based SQLi [4]:

Error-Based SQLi - зловмисник намагається отримати конфіденційну інформацію про базу даних, її конфігурацію, структуру або вміст за допомогою повідомлень про помилки SQL.

Union-Based SQLi - зловмисник використовує оператор UNION для об'єднання безпечного SQL-запиту зі зловмисним. Шкідливий оператор повинен використовувати ті ж стовпці і типи даних, що і початковий оператор. Вразлива база даних обробляє об'єднаний оператор і виконує шкідливий код.

Приклад використання Union-Based Injection:

Зловмисник використовує оператор UNION для того, щоб комбінувати кілька SELECT-запитів, що дозволяє йому отримувати інформацію з інших таблиць або навіть з інших баз даних.

Сценарій атаки:

Припустимо, веб-застосунок має URL-адресу, яка приймає параметр для відображення інформації про продукт:

https://example.com/product.php?id=5

Зловмисник може спробувати додати оператор UNION до цього параметра, щоб об'єднати результати оригінального запиту з результатами зловмисного запиту. Він може спробувати таку зміну:

https://example.com/product.php?id=5 UNION SELECT null, username, password FROM users

Тут зловмисник додає до запиту UNION SELECT, який спробує отримати дані з таблиці users (наприклад, логіни та паролі). Оскільки запит виконується без перевірки введених даних, база даних об'єднає результати запиту для товару з результатами запиту для користувачів.

Що отримує зловмисник:

Якщо запит виконується успішно, результатом буде комбінація даних з таблиці товарів і таблиці користувачів, що дозволить зловмиснику отримати чутливу інформацію, таку як імена користувачів і паролі.

Таблиця 1.1

Зловмисник може побачити результати у вигляді таблиці:

id	**username**	**password**
5	admin	12345
6	user1	password123

Продовження таблиці 1.1

7	user2	qwerty

Інформація, яка міститься у повідомленні про помилку або результатах запиту, дає зловмиснику важливі відомості:

Місце вразливості – оператор UNION дозволяє з'єднати таблиці, і зловмисник може зрозуміти, які таблиці існують у базі даних.

Структура таблиць – зловмисник може отримати доступ до імен таблиць і полів, таких як username та password, які використовуються в запиті.

Автоматизація атаки:

Зловмисник може автоматизувати процес за допомогою інструментів типу SQLmap, щоб швидко знаходити параметри, у яких можливо застосувати техніку Union-Based Injection. Інструмент автоматично генерує різні варіанти запитів і знаходить відповідні таблиці, що дозволяє витягувати дані з бази без необхідності вручну вводити кожен запит.

Blind SQL Injection:

Blind SQL Injections (сліпі) виникають, коли веб-додаток піддається SQL-ін'єкції, але його HTTP-відповіді не містять результатів SQL-запиту або будь-яких деталей про помилки в базі даних. Це відрізняється від звичайної SQL-ін'єкції, в якій помилка в базі даних або результат шкідливого SQL-запиту відображається у веб-додатку і стає видимим зловмиснику.

При Blind SQL Injections зловмисники ніколи не бачать результат виконання SQL-запитів. Проте вони можуть бачити, чи нормально завантажується додаток або веб-сторінка, і визначити, скільки часу SQL-серверу потрібно для обробки SQL-запиту, який зловмисник передав користувачеві при введенні.

Експлуатація Blind SQL Injections є більш складною і займає більше часу для зловмисника, і зловмисник не може використовувати поширені методи SQLi, такі як UNION, ін'єкції підзапитів або XPATH.

Однак, наслідки для безпеки подібні. Коли зловмисник виконує успішний шкідливий запит, він отримує контроль над сервером бази даних. Це призводить до крадіжки даних (наприклад, номерів кредитних карток) і може призвести до повного захоплення операційної системи веб-сервера за допомогою ескалації привілеїв [5].

Приклад використання Blind SQL Injection (Boolean-Based):

Blind SQL Injection (Boolean-Based) – це тип SQL-ін'єкції, при якому зловмисник не отримує детальної інформації про помилки в базі даних, але може отримати доступ до інформації через зміну логіки запиту. Зловмисник може маніпулювати SQL-запитом таким чином, щоб сервер відповідав різними результатами в залежності від того, чи правильне певне припущення.

Сценарій атаки:

Припустимо, веб-застосунок має URL-адресу, яка приймає параметр для пошуку товару за його ідентифікатором:

https://example.com/product.php?id=5

Зловмисник може спробувати додати умовний вираз для тестування вразливості, наприклад, так:

https://example.com/product.php?id=5 AND 1=1

Запит буде працювати нормально, бо умовний вираз 1=1 завжди істинний. Тепер зловмисник може спробувати змінити вираз на хибний:

https://example.com/product.php?id=5 AND 1=2

Якщо запит не викликає помилки і сторінка не змінюється, це вказує на те, що SQL-запит був оброблений успішно і сервер підтвердив, що є вразливість.

Що отримує зловмисник:

Зловмисник може за допомогою цього методу визначити, чи є SQL-ін'єкція, а також, через комбінацію умов, спробувати витягнути додаткову інформацію про структуру бази даних, таблиці або значення, роблячи зміну в логічних виразах.

Приклад використання Blind SQL Injection (Time-Based):

Blind SQL Injection (Time-Based) – це варіант сліпої SQL-ін'єкції, при якому зловмисник використовує затримку часу, щоб визначити, чи існують певні умови в

базі даних. Зловмисник вносить зміни в SQL-запит, які викликають затримку в відповіді сервера, тим самим підтверджуючи наявність або відсутність певної інформації.

Сценарій атаки:

Припустимо, той самий веб-застосунок має URL, який приймає параметр id для пошуку товару:

https://example.com/product.php?id=5

Зловмисник може додати затримку в SQL-запит, використовуючи конструкцію SLEEP():

https://example.com/product.php?id=5 AND SLEEP(5)

Якщо сервер затримається на 5 секунд, це свідчитиме про те, що SQL-запит був успішно виконаний. Якщо затримки не буде, значить сервер не обробляє запит з такою умовою.

Зловмисник може також використовувати цей метод для більш точного визначення інформації, перевіряючи, наприклад, чи існує певний користувач в базі даних, по черзі змінюючи умови запиту, наприклад:

https://example.com/product.php?id=5 AND (SELECT IF(username='admin', SLEEP(5), 0))

Якщо користувач з іменем admin існує в базі даних, сервер затримається на 5 секунд, і зловмисник буде знати, що ця інформація правильна.

Що отримує зловмисник:

За допомогою Time-Based Blind SQL Injection зловмисник може поступово визначати наявність або відсутність конкретних значень у базі даних. Це дозволяє отримати важливу інформацію, навіть якщо сервер не виводить конкретні помилки, але затримка відповіді є індикатором правильної або хибної умови в запиті.

Out-of-band SQL injection:

Приклад використання Out-of-Band SQL Injection:

Out-of-Band SQL Injection (OOB-SQLi) – це тип SQL-ін'єкцій, коли зловмисник не отримує відповідь від атакованого додатка на тому ж каналі зв'язку, але замість

цього може змусити додаток надсилати дані на віддалену кінцеву точку, яку він контролює.

Out-of-Band SQL Injection можлива лише в тому випадку, якщо на сервері, який ви використовуєте, є команди, що запускають DNS- або HTTP-запити. Втім, це стосується всіх популярних SQL-серверів [6].

Сценарій атаки:

Припустимо, веб-застосунок має URL-адресу, яка приймає параметр для ідентифікації користувача:

https://example.com/user.php?id=5

Зловмисник може спробувати додати в параметр запиту інструкцію для зовнішнього запиту, наприклад, за допомогою функції LOAD_FILE() або SELECT INTO OUTFILE, щоб відправити результат запиту на інший сервер або зберегти його в зовнішньому файлі. Ось приклад атаки, яка використовує функцію LOAD_FILE():

https://example.com/user.php?id=5 UNION SELECT LOAD_FILE ('\\\\attacker.com\\data\\file.txt')

У цьому випадку, якщо сервер має налаштовані дозволи для доступу до файлів або мережевих ресурсів, результат запиту буде надіслано на сервер зловмисника або збережено у файлі, що дозволяє зловмиснику отримати інформацію.

Інший приклад – використання DNS-запитів для відправки результату запиту на сервер зловмисника:

https://example.com/user.php?id=5 UNION SELECT 1, 2, 3, 4, 5, CONCAT('attacker.com', (SELECT username FROM users WHERE id=1)) INTO OUTFILE '\\\\attacker.com\\data\\result.txt'

Що отримує зловмисник:

Передача результату через зовнішній канал: Зловмисник може отримати необхідні дані за допомогою зовнішніх серверів або навіть через DNS-запити.

Можливість виведення конфіденційної інформації: Зловмисник може отримати дані з бази, такі як імена користувачів, паролі, адреси електронної пошти та іншу важливу інформацію, яку неможливо витягнути через стандартні канали.

Обхід обмежень сервера: Оскільки відправка результатів здійснюється через зовнішній канал (наприклад, через DNS-запит чи виведення на інший сервер), цей метод може бути важче виявити або заблокувати.

Автоматизація атаки:

Зловмисник може використовувати інструменти, такі як SQLmap, щоб автоматизувати пошук вразливих точок і відправляти запити через зовнішні канали. Інструменти можуть автоматично спробувати використати методи OOB, щоб отримати дані без необхідності взаємодіяти з основним веб-додатком.

1.3. Методи захисту від SQL-ін'єкцій

Існують кілька ефективних методів захисту від SQL-ін'єкцій, кожен з яких має свої особливості та переваги [8].

1 Використання підготовлених запитів (parameterized queries)

Підготовлені запити є одним з найбільш ефективних способів захисту від SQL-ін'єкцій. Вони дозволяють чітко відокремити код SQL-запиту від переданих користувачем даних. У випадку з параметризованими запитами значення передаються окремо від самого запиту, що забезпечує захист від ін'єкцій. Завдяки цьому, навіть якщо зловмисник спробує вставити небезпечний SQL-код у введення, база даних сприйматиме його як звичайний текст, а не як частину SQL-запиту.

2 Збережені процедури (Stored Procedures)

Збережені процедури є ще одним способом захисту від SQL-ін'єкцій, якщо вони реалізовані правильно. Вони дозволяють виконувати складні запити на сервері, зменшуючи ризики завдяки попередньо визначеним SQL-операціям. Однак варто зазначити, що неправильно реалізовані збережені процедури можуть бути вразливими до SQL-ін'єкцій, тому їх використання має бути обмежене лише необхідними операціями.

3 Валідація вводу за принципом allow-list

Для деяких частин SQL-запитів, таких як імена таблиць, стовпців або параметри сортування, не можна використовувати параметризовані запити. У таких випадках

важливо впроваджувати валідацію вводу за принципом allow-list, де значення параметрів мають бути перевірені на відповідність списку дозволених значень. Це допомагає забезпечити, що лише легітимні значення будуть використовуватись у запитах, уникаючи можливості для SQL-ін'єкцій.

4 Мінімізація прав доступу

Одним із важливих заходів для захисту від SQL-ін'єкцій є обмеження прав доступу до бази даних. Кожен користувач або додаток має отримувати лише ті права, які необхідні для виконання конкретних операцій. Це мінімізує потенційну шкоду від успішної атаки, оскільки навіть якщо зловмисник отримає доступ до бази даних, його можливості будуть обмежені.

5 Валідація вводу

Валідація вводу є важливою складовою безпеки навіть при використанні параметризованих запитів або збережених процедур. Це дозволяє додатково перевірити, чи є дані від користувача безпечними для подальшого використання у SQL-запитах. Всі введені дані повинні бути перевірені на відповідність очікуваному формату перед тим, як їх буде передано в запит.

6 Уникання динамічних SQL-запитів

Використання динамічних SQL-запитів, де SQL-код формується на основі введених даних, є одним із найбільш вразливих підходів до побудови запитів. Такий підхід слід уникати, оскільки він відкриває двері для ін'єкцій. Якщо динамічні запити неминучі, необхідно застосовувати валідацію вводу або екранування, щоб запобігти вставці небезпечного SQL-коду.

7 Операційна система та мінімізація прав доступу на рівні БД

Крім мінімізації прав доступу до самої бази даних, також важливо обмежити привілеї операційної системи для користувача, під яким працює база даних. Наприклад, база даних не повинна працювати з правами адміністратора або системного користувача, оскільки це може призвести до серйозних наслідків у разі компрометації сервера.

1.4. Існуючі інструменти для виявлення SQL-ін'єкцій

Виявлення та зупинка атак SQL-ін'єкцій має важливе значення для збереження безпеки та цілісності онлайн-систем через руйнівні наслідки, які вони можуть спричинити. Існує безліч технологій виявлення SQLi, які допоможуть знайти і вирішити ці проблеми. Нижче розглянуто деякі з найкращих технологій виявлення SQLi [9].

1. Sqlmap

На GitHub можна знайти sqlmap, автоматизований інструмент для тестування SQLi та захоплення баз даних. Ця програма для тестування на проникнення з відкритим вихідним кодом автоматизує процес виявлення та використання вразливостей SQLi або інших атак, які захоплюють сервери баз даних.

Основні методи SQLi, в тому числі булеві сліпі, запити на основі помилок, сліпі запити на основі часу, стекові та UNION запити, підтримуються sqlmap. Крім того, ця утиліта взаємодіє з широким спектром систем управління базами даних, таких як IBM DB2, MySQL, PostgreSQL, Microsoft SQL Server, Oracle, Microsoft Access тощо.

Крім того, вона має механізм виявлення, багато методів тестування на проникнення, інструменти для створення відбитків баз даних, вилучення даних, доступу до базової файлової системи та позасмугового виконання команд в операційній системі (ОС).

Основні можливості sqlmap полягають в наступному:

- Повна сумісність з багатьма системами управління базами даних.

- Використовуючи локальну базу даних SQLite 3, програма може створювати прототипи таблиць, записів та структури внутрішньої бази даних.

- Булеві сліпі, часові сліпі, запити на основі помилок, запити на основі об'єднання та стекові запити - це п'ять основних методів SQLi, які підтримуються програмою.

- Інструмент автоматично обробляє заголовок HTTP Set-Cookie програми.

Основні переваги sqlmap полягають у наступному:

- Це безкоштовна програма з відкритим вихідним кодом.

- Може бути змінений для пристосування до різних тестових сценаріїв.

- Дозволяє інтегруватися з w3af та Metasploit, двома основними інструментами IT-безпеки з відкритим кодом.

Основні недоліки sqlmap полягають в наступному:

- Це всього лише утиліта командного рядка.

- Це може призвести до помилкових позитивних результатів.

2. Invicti

За допомогою Invicti, системи управління веб-безпекою, можна автоматизувати обов'язки з безпеки в життєвому циклі розробки програмного забезпечення, знаходячи вразливості в онлайн-додатках і призначаючи їх для виправлення. Платформа, яка включає SQLi як один з основних компонентів, використовує технологію Proof-based Scanning для пошуку та перевірки вразливостей і показує результати, які не є хибнопозитивними.

Вона може виявляти вразливості SQL-ін'єкцій, а також міжсайтовий скриптинг (XSS) та інші недоліки в онлайн-сервісах, веб-додатках і веб-інтерфейсах. Платформа може бути інтегрована в налаштування DevOps і включає в себе інструменти для тестування безпеки, а також генератор звітів. Вона підтримує додатки JavaScript та AJAX і перевіряє веб-сервери, включаючи Apache, Nginx.

Основні можливості Invicti наступні:

- Він може автоматично підтримувати актуальну, повну інвентаризацію ваших API і веб-додатків.

- Він надає складну аналітику та звіти.

- За допомогою програми можна визначити точне місце розташування вразливостей.

- Пропонуючи відстеження версій технологій, Invicta допомагає мінімізувати ризики, спричинені застарілими фреймворками та бібліотеками.

Основні переваги Invicti полягають у наступному:

- Окрім виявлення SQLi, програма забезпечує безпеку веб-додатків.

- Для тих, хто зацікавлений спробувати інструмент, вона надає безкоштовну пробну версію.

- Для допомоги новим користувачам доступна велика документація.

Основними недоліками Invicti є наступні:

- Відсутність прозорості в ціноутворенні.

- Більшість функцій є ексклюзивними для корпоративного плану.

3.Burp Scanner

Сканер веб-вразливостей Burp Suite використовує дослідження PortSwigger, щоб допомогти клієнтам в автоматичному виявленні різноманітних вразливостей, присутніх у веб-додатках. Для перевірки вразливостей, які не бачать традиційні сканери, включаючи асинхронні SQL-ін'єкції та сліпу підробку запитів на стороні сервера (SSRF), Burp Collaborator, наприклад, виявляє взаємодію між цільовим об'єктом і зовнішнім сервером.

Механізм сканування Burp Scanner, який лежить в основі таких пакетів, як Burp Suite Enterprise Edition і Burp Suite Professional, здатний долати такі перешкоди, як функціональні можливості, що залежать від стану, нестабільні або перевантажені URL-адреси і токени міжсайтової підробки запитів (CSRF). JavaScript відображається і сканується вбудованим браузером Chromium, а алгоритм сканування створює цільовий профіль так само, як це зробив би тестувальник.

Основні можливості Burp Scanner наступні:

- Ручна перевірка на наявність вразливостей, включаючи міжсайтовий скриптинг, SQL ін'єкції та інші 10 найпоширеніших помилок OWASP.

- Автоматизоване сканування вразливостей.

- Забезпечує реалістичні сценарії та автоматичне сканування обсягу.

- Складні функції сканування, включаючи користувацькі скрипти та обробку сеансів.

Основні переваги Burp Scanner наступні:

- Забезпечує автоматизоване і ручне тестування додатків.

- Доступна безкоштовна пробна версія.

Основні недоліки Burp Scanner наступні:

- Складний користувальницький інтерфейс, особливо для недосвідчених користувачів.

IT-команди можуть отримувати інформацію з віддалених серверів за допомогою jSQL Injection, рішення на основі Java. Це одне з численних безкоштовних рішень для SQLi з відкритим вихідним кодом. Він працює з Java версій 11-17 і підтримує Windows, Linux і Mac OS X.

4. jSQL Injection

Багато додаткових інструментів і дистрибутивів для сканування вразливостей і тестування на проникнення, таких як Kali Linux, Pentest Box, Parrot Security OS,

ArchStrike і BlackArch Linux, мають jSQL Injection, оскільки він є чудовим стримуючим фактором SQLi. Крім того, він забезпечує автоматизовану ін'єкцію 33 різних механізмів баз даних, таких як Hana, Ingres, MySQL, Oracle, PostgreSQL, SQL Server, Teradata, Access і DB2. Він надає скриптові пісочниці для SQL і фальсифікацій, а також дозволяє користувачеві керувати різними схемами і процесами ін'єкцій.

Основні можливості jSQL Injection наступні:

- Забезпечує автоматичну ін'єкцію механізму бази даних.

- В інших пакетах для тестування вразливостей часто використовується утиліта SQLi.

- Інструмент підтримує багато тактик ін'єкцій, включаючи звичайну, помилкову, сліпу та тимчасову.

- Для спрощення підключення до бази даних та отримання даних передбачено функцію дактилоскопіювання бази даних.

Основні переваги jSQL Injection наступні:

- Відкритий вихідний код і безкоштовність.

- Значна бібліотека документації.

Основні недоліки jSQL Injection наступні:

- Мало можливостей для створення звітів.

- Не підходить для великих підприємств і є портативним.

5. AppSpider

Rapid7 створила AppSpider, сканер безпеки веб-додатків. Інструмент імітує реальні атаки і безперервно моніторить додатки, щоб забезпечити їхню захищеність від SQLi. Програма призначена для оцінки як складних, так і портативних додатків, досліджуючи їхні найпотаємніші куточки, щоб знайти можливі недоліки безпеки. Крім того, інструмент може запропонувати всебічну інформацію, що дозволить інженерам швидко та ефективно виправити проблеми.

AppSpider можна інтегрувати з низкою інших технологій, щоб задовольнити потреби користувачів у безпеці додатків. Він може інтегруватися з платформами відстеження проблем, такими як Jira, інструментами автоматизованого тестування, такими як Selenium, фреймворками документації API, такими як Swagger, і технологіями безперервної інтеграції (CI), такими як Jenkins і Bamboo.

Основні функції AppSpider полягають у наступному:

- AppSpider полегшує створення звітів про усунення вразливостей завдяки функції перевірки вразливостей в один клік.

- Завдяки можливостям глобального перекладу, програма здатна ретельніше сканувати сучасні онлайн-додатки.

- AppSpider надає звіти у форматі HTML разом із плагіном для Chrome.

- За допомогою цього інструменту можна перевіряти вразливості, що не входять до топ-10 OWASP.

Основні переваги AppSpider наступні:

- Забезпечується розширене тестування уразливостей за межами SQLi.

- Тестування прискорюється завдяки функції перевірки вразливостей в один клік.

Основними недоліками AppSpider є наступні:

- Недостатня кількість інструкцій про те, як користуватися продуктом.

- Не кожна цінова стратегія зрозуміла.

- Єдина операційна система, яку він підтримує - це Windows.

6. Acunetix

Тестування SQLi є компонентом загальної функціональності сканування веб-додатків Acunetix by Invicti. Для Linux і Windows його багатопотоковий сканер може

швидко сканувати сотні тисяч сторінок. Він знаходить типові проблеми з налаштуваннями веб-сервера і відмінно справляється з перевіркою WordPress.

Acunetix підтримує актуальний список усіх веб-сайтів, додатків та API, створюючи його автоматично. Ця програма пропонує макроси для автоматизації сканування в захищених паролем і важкодоступних регіонах. Вона також сканує SPA-сайти, веб-сайти з великою кількістю скриптів, а також програми HTML5 і JavaScript.

Основні можливості Acunetix такі:

- Програма включає в себе тестування на більш ніж 7000 вразливостей на додаток до SQLi.

- Acunetix надає складну функцію запису макросів, яка дозволяє шукати захищені паролем частини вашого веб-сайту.

- Інструмент може визначити, наскільки серйозною є вразливість, і запропонувати корисну інформацію.

- Завдяки інтегрованим функціям управління вразливостями Acunetix розробники можуть швидше вирішувати проблеми з додатками.

Основні переваги Acunetix наступні:

- Сприяє комплексному скануванню веб-сторінок, складних онлайн-додатків і веб-додатків.

- Забезпечуючи перевірку вразливостей, він зменшує кількість хибних спрацьовувань, визначаючи, які з них є справжніми.

- У Jenkins його можна встановити як плагін.

Основні недоліки Acunetix наступні:

- Відсутня прозорість у ціноутворенні.

- Не доступний безкоштовно.

7. Qualys WAS

Qualys WAS аналізує веб-додатки і створює вичерпні звіти про будь-які виявлені вразливості, використовуючи комбінацію автоматизованих і людських методологій тестування. Qualys WAS може виявити численні вразливості, такі як SQL-ін'єкції, міжсайтовий скриптинг (XSS) та інші поширені вразливості веб-додатків.

Окрім сканування вразливостей, Qualys WAS пропонує ряд інших функцій, включаючи комплексні звіти, безперешкодну взаємодію з іншими рішеннями для забезпечення безпеки та підтримку вимог стандартів, таких як PCI DSS.

Основні можливості Qualys наступні:

- Інтеграція з іншими продуктами SIEM для управління подіями та інформацією про безпеку.

- Мільйонами активів можна керувати за допомогою масштабованого програмного забезпечення.

- Підтримує ретельну звітність, яка включає запропоновані ремонтні дії, технічну інформацію та рівні серйозності.

- Qualys WAS полегшує автономне сканування і сканування.

Основні переваги Qualys полягають в наступному:

- Неймовірна адаптивність.

- Завдяки цій технології користувачі отримують повний доступ до свого прикладного середовища.

- Функції звітності та перевірки відповідності інтегровані прямо в нього.

Основні недоліки Qualys наступні:

- Не підходить для компаній з локальним середовищем.

- Для її налаштування та управління може знадобитися певне технічне ноу-хау.

8. HCLAppScan

HCL Technologies придбала AppScan, продукт для тестування безпеки веб-додатків, створений IBM. Існують хмарні та локальні версії утиліти. Він може бути використаний для перевірки веб-додатків на наявність різних вразливостей, таких як SQLi, з використанням різних фреймворків і технологій, таких як PHP, Java і .NET.

AppScan є гнучким рішенням для тестування безпеки веб-додатків, оскільки надає різноманітні можливості сканування, такі як динамічне сканування (DAST) і статичне сканування (SAST), на додаток до альтернативних варіантів ручного тестування.

AppScan пропонує комплексні функції звітування, такі як запропоновані коригувальні дії та рейтинги серйозності вразливостей. Крім того, він здатний

інтегруватися з іншими рішеннями для забезпечення безпеки, включаючи SIEM-системи та платформи управління вразливостями.

Основні можливості AppScan наступні:

- Програмне забезпечення підтримує декілька функцій тестування додатків, включаючи DAST, SAST та інтерактивне сканування (IAST).

- Параметри ручного тестування можна змінювати відповідно до конкретних тестових ситуацій.

- Є можливість генерувати вичерпні звіти.

- Програма сумісна з багатьма різними фреймворками і технологіями веб-додатків.

Основні переваги AppScan наступні:

- Можливість ретельного сканування.

- Доступна безкоштовна пробна версія.

- Підтримується велика кількість технологій веб-додатків.

9. Imperva

Виявлення SQLi забезпечується платформою кібербезпеки Imperva як частина її рішень для захисту веб-додатків. Атаки з використанням SQL-ін'єкцій є однією з багатьох форм атак, від яких покликаний захищати брандмауер веб-додатків Imperva SecureSphere Web Application Firewall (WAF).

Платформа використовує передові методи, такі як алгоритми машинного навчання, поведінковий аналіз і виявлення на основі сигнатур, щоб миттєво виявляти і зупиняти атаки SQLi.

Основні особливості Imperva полягають у наступному:

- Складні методи виявлення, такі як алгоритми машинного навчання, поведінковий аналіз і виявлення на основі сигнатур.

- Запобігання атакам SQLi та моніторинг в режимі реального часу.

- Підтримка великої кількості веб-додатків і баз даних.

- Ретельне звітування та дослідження вразливостей і атак SQLi.

- Інтеграція з брандмауерами баз даних і WAF, а також іншими рішеннями для забезпечення безпеки.

- Підтримка стандартів NIST та інших нормативних вимог.

Основні переваги Imperva полягають в наступному:

- Вони надають безкоштовну пробну версію.
- Виявлення загроз та звітування в режимі реального часу.

Основним недоліком Imperva є непрозоре ціноутворення.

1.5. Використання штучного інтелекту для виявлення та запобігання SQL-ін'єкціям у кібербезпеці: сучасний стан та перспективи

Штучний інтелект у кібербезпеці використовується для автоматичного виявлення та блокування загроз у реальному часі, аналізуючи великі обсяги даних та шукаючи аномальні патерни, які можуть вказувати на потенційну атаку. Моделі ШІ, зокрема машинне навчання (ML) та глибинне навчання (DL), здатні виявляти навіть нові або раніше не зафіксовані загрози, навчатися на історичних даних та адаптуватися до змінюваних умов безпеки.

Машинне навчання (ML): За допомогою алгоритмів ML можна навчити систему на прикладах нормальних та аномальних запитів до бази даних, щоб вона могла автоматично виявляти підозрілі активності.

Глибинне навчання (DL): Використовує багатошарові нейронні мережі для аналізу складних патернів та детекції більш складних форм SQL-ін'єкцій.

Використання ШІ для виявлення SQL-ін'єкцій передбачає аналіз вхідних запитів до бази даних на наявність аномальних або підозрілих патернів. Замість використання жорстких правил та фільтрів, ШІ підходить до виявлення ін'єкцій більш гнучко, орієнтуючись на навчання з великої кількості даних.

Аналіз поведінки: Моделі ШІ можуть вивчати поведінку користувачів і визначати, які запити є підозрілими. Це може бути, наприклад, спроба виконати операції, які зазвичай не здійснюються користувачами або незвичні патерни доступу до бази даних.

Аномалії в запитах: ШІ може автоматично порівнювати вхідні SQL-запити з відомими патернами ін'єкцій, такими як використання ключових слів, які зазвичай асоціюються з SQL-ін'єкціями (наприклад, OR 1=1, --, ' OR 1=1 --).

Використання моделей машинного навчання для класифікації: Алгоритми машинного навчання (наприклад, класифікаційні моделі) можуть бути використані для аналізу структури SQL-запитів та визначення, чи є вони потенційно шкідливими. Моделі можуть бути навчені на даних з реальних атак для підвищення точності виявлення.

Одним з великих переваг використання ШІ в кібербезпеці є можливість автоматичного реагування на загрози. У разі виявлення SQL-ін'єкції система на основі ШІ може:

- Автоматично блокувати або відхиляти підозрілі запити.

- Інформувати адміністратора про потенційну атаку, з можливістю зупинки її у реальному часі.

- Адаптуватися до нових методів атак, оскільки ШІ-моделі можуть бути перепідготовлені або адаптовані до нових типів SQL-ін'єкцій.

- Автоматизація цих процесів дозволяє знизити час реакції на атаки та підвищити рівень захисту, не вимагаючи постійної участі адміністратора.

ШІ має великий потенціал у покращенні безпеки від SQL-ін'єкцій та інших атак. Зокрема:

Вдосконалення точності виявлення: Використання нових архітектур нейронних мереж (наприклад, трансформери, глибокі рекурентні нейронні мережі) може підвищити точність виявлення нових типів SQL-ін'єкцій.

Адаптивні системи захисту: Завдяки здатності моделей ШІ до навчання на нових даних, вони можуть автоматично адаптуватися до нових технік атак, таких як використання нестандартних або невідомих до цього моменту методів ін'єкцій.

Інтеграція з іншими методами безпеки: ШІ може працювати в поєднанні з іншими технологіями безпеки, такими як фаєрволи для веб-додатків (WAF), щоб підвищити загальний рівень захисту.

На ринку вже існують деякі рішення, які використовують ШІ для захисту від SQL-ін'єкцій:

Web Application Firewalls (WAFs): Веб-брандмауери, такі як ModSecurity та AWS WAF, використовують техніки машинного навчання для виявлення підозрілих запитів до веб-додатків і їх блокування.

Бази даних з інтеграцією ШІ: Деякі сучасні бази даних також інтегрують механізми машинного навчання для виявлення SQL-ін'єкцій та запобігання їм.

Висновок до першого розділу

У результаті аналізу SQL-ін'єкцій було визначено, що вони є одним із найнебезпечніших методів атак на веб-застосунки, що може призвести до несанкціонованого доступу до баз даних, компрометації конфіденційної інформації та порушення цілісності даних. Було розглянуто різні типи SQL-ін'єкцій, включаючи In-band, Blind та Out-of-band атаки, кожна з яких має свої особливості та методи реалізації.

Для захисту від SQL-ін'єкцій використовують різні підходи, зокрема параметризовані запити, валідацію введених даних, мінімізацію прав доступу та застосування веб-брандмауерів (WAF). Крім того, розглянуто сучасні інструменти для виявлення SQL-атак, такі як sqlmap, Invicti та Burp Suite, що допомагають у тестуванні безпеки веб-додатків.

Окрему увагу приділено використанню штучного інтелекту для детекції та запобігання SQL-ін'єкціям. Методи машинного навчання дозволяють аналізувати запити та виявляти аномальні дії, що підвищує ефективність виявлення атак. Таким чином, застосування ШІ може стати перспективним напрямом розвитку систем кібербезпеки.

РОЗДІЛ 2
РОЗРОБКА МЕТОДУ ВИЯВЛЕННЯ SQL-ІН'ЄКЦІЙ НА ОСНОВІ ШТУЧНОГО ІНТЕЛЕКТУ

2.1. Постановка завдання та вимоги до системи

SQL-ін'єкції є однією з найнебезпечніших загроз для веб-додатків. Вони можуть призвести до витоку конфіденційних даних, несанкціонованого доступу до баз даних або навіть повного компрометації системи. Попри розвиток засобів кіберзахисту, атаки цього типу залишаються актуальними, оскільки зловмисники постійно вдосконалюють свої методи.

Стандартні механізми захисту, такі як ручне тестування, використання WAF або застосування регулярних виразів, мають низку недоліків [15]. Вони можуть блокувати легітимні запити, не розпізнавати складні методи обходу фільтрів або вимагати постійного оновлення правил виявлення загроз. Це створює потребу у гнучких і адаптивних підходах до аналізу вхідних даних.

Розроблений веб-додаток дозволяє аналізувати текстові запити та виявляти потенційні SQL-ін'єкції на основі штучного інтелекту. В якості інструменту аналізу - LLM, які мають можливість гнучкої обробки тексту та визначення характерних ознак атак. У веб-додатку реалізовано можливість вибору різних моделей, серед яких GPT-4, GPT-4.o, GPT-4 Turbo, Claude-3 Sonnet, Opus, Haiku.

Система працює наступним чином:

Користувач вводить текст запиту, який потенційно може містити SQL-ін'єкцію.

Запит передається вибраній моделі ШІ разом із спеціально сформованим промптом, що містить контекст завдання.

Модель аналізує текст і повертає відповідь із поясненням, чи містить запит ознаки SQL-ін'єкції.

Результат виводиться у зручному форматі для користувача.

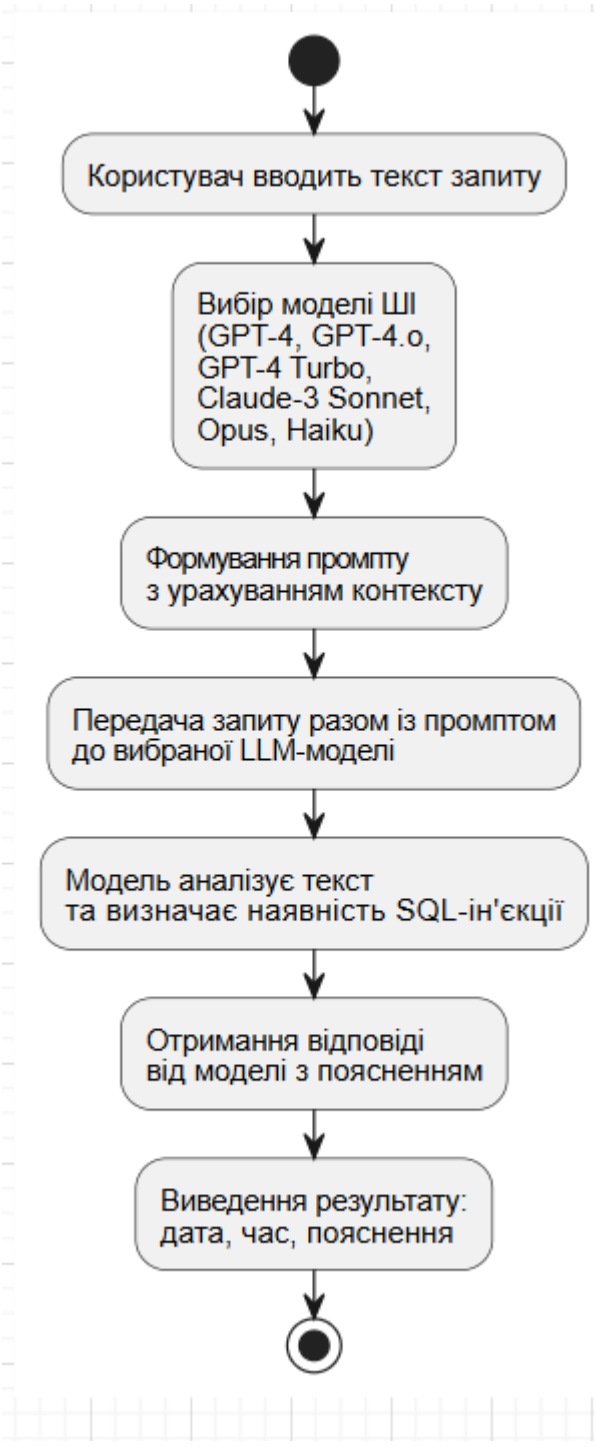

Рисунок 2.1 – Алгоритм роботи

Функціональні вимоги:

- Аналіз вхідних запитів на наявність SQL-ін'єкцій.
- Використання моделей ШІ
- Формування та адаптація промптів для підвищення точності аналізу.
- Виведення пояснення щодо результату аналізу.

- Гнучкість у виборі моделей для порівняння їх ефективності.

Нефункціональні вимоги

- Швидкість обробки одного запиту
- Можливість додавати нові моделі ШІ без значних змін у структурі веб-додатку.
- Гнучкість у налаштуванні промптів для різних сценаріїв аналізу.
- Підтримка роботи у браузері без необхідності встановлення додаткового програмного забезпечення.
- Використання HTTPS для захисту переданих даних.

2.2.Огляд моделей LLM, використаних для детекції атак

Зростання поширеності та складності кібератак, особливо SQL-ін’єкцій, становить значну загрозу для веб-додатків та безпеки даних. Традиційні методи виявлення, хоча й цінні, часто зазнають труднощів з новими та заплутаними векторами атак. LLM відкрила нові можливості для аналізу текстових даних та виявлення складних патернів, що робить їх потенційними інструментами для покращеного виявлення загроз [25].

Загальні можливості LLM:

LLM володіють значною здатністю розуміти та генерувати природну мову. Вони можуть аналізувати текстові патерни та розуміти контекст. Їхні можливості поширюються на аналіз та генерацію коду. LLM можуть навчатися на великих обсягах даних, що дозволяє їм виявляти тонкі аномалії та патерни, які свідчать про зловмисну діяльність.

Застосування в кібербезпеці:

LLM досліджуються для різних завдань кібербезпеки, включаючи аудит коду, виявлення вразливостей та аналіз шкідливого програмного забезпечення. Вони можуть допомогти у генеруванні тестових випадків для вразливостей безпеки. LLM

можуть використовуватися для аналізу розвідки загроз, виявлення фішингових спроб та інших веб-атак.

Конкретна актуальність для виявлення SQL-ін’єкцій:

Атаки SQL-ін’єкцій передбачають маніпулювання SQL-запитами через зловмисні вхідні дані, часто замасковані у вигляді звичайного тексту. Розуміння природної мови LLM може допомогти їм виявити відхилення від очікуваних структур запитів та розпізнати патерни, пов’язані зі спробами SQL-ін’єкцій. Деякі дослідження вивчають використання LLM для генерування корисних навантажень SQL-ін’єкцій з метою тестування.

Використання LLM для генерування тестових випадків свідчить про проактивний підхід до безпеки, коли LLM можуть допомогти виявити потенційні вразливості, створюючи різноманітні сценарії атак, включаючи варіації SQL-ін’єкцій. LLM можуть виступати потужними інструментами для тестування безпеки, автономно генеруючи широкий спектр корисних навантажень SQL-ін’єкцій, потенційно виявляючи вразливості, які можуть бути пропущені традиційними сканерами на основі правил. Оскільки LLM можуть розуміти базову структуру та синтаксис SQL, вони можуть створювати складні та різноманітні рядки атак, включаючи заплутані або нові форми SQL-ін’єкцій. Це дозволяє проводити більш комплексне тестування захисту веб-додатків.

GPT-4:

- Розроблено OpenAI
- Випущено у 2023 році
- Тип: Велика мовна модель
- Архітектура: Покращена версія трансформерної нейромережі
- Контекстне вікно: До 32 тис. токенів
- Ключові особливості: Значно покращена точність, краще розуміння складної логіки та текстових патернів, стійкість до помилок та маніпуляцій, кращий аналіз довгих документів та підтримка довготривалого контексту, використання для НЛП, аналізу даних, генерації коду та логічних задач.

- Обмеження: Потребує більше обчислювальних ресурсів (повільніша), дорожча у використанні, все ще можливі помилки у складних доменних задачах.
- Актуальність для виявлення SQL-ін’єкцій: Покращена точність та розуміння складних структур роблять її більш здатною виявляти складні атаки SQL-ін’єкцій. Здатність аналізувати довші контексти корисна для виявлення багатокрокових атак або ін’єкцій у складних запитах.
- Аналіз та наслідки: Вища точність GPT-4 свідчить про нижчий рівень хибнопозитивних та хибнонегативних результатів, що робить її надійнішою для критичних застосувань безпеки. Однак збільшення обчислювальних витрат може обмежити її використання в сценаріях реального часу з високою пропускною здатністю. Покращена стійкість до маніпуляцій у GPT-4 є вирішальною для застосувань безпеки, оскільки зловмисники можуть намагатися створювати запити, які обманюють модель виявлення, змушуючи її ігнорувати шкідливий код. Ця підвищена надійність робить GPT-4 більш надійним варіантом для виявлення спроб SQL-ін’єкцій. Якщо зловмисник розуміє, як працює система виявлення на основі LLM, він може спробувати створити корисні навантаження SQL-ін’єкцій, які дещо відрізняються від відомих патернів, щоб уникнути виявлення. Покращена надійність GPT-4 означає, що її менш імовірно вдасться обдурити такими зловмисними прикладами, забезпечуючи сильніший захист.

GPT-4-o:

- Розробник: OpenAI .
- Дата випуску: Травень 2024 року .
- Тип моделі: Мультимодальна велика мовна модель.
- Архітектура: Generative Pre-trained Transformer .
- Контекстне вікно: 128 тис. токенів .
- Ключові особливості: Багатомовність, мультимодальність (текст, зображення, аудіо), швидший та дешевший API порівняно з GPT-4 Turbo , покращена продуктивність для неанглійських мов та задач комп’ютерного зору.

- Актуальність для виявлення SQL-ін’єкцій: Збільшене контекстне вікно є дуже корисним для аналізу великих та складних SQL-запитів, потенційно покращуючи виявлення складних ін’єкцій. Її багатомовні можливості можуть бути корисними для додатків з неанглійськими параметрами або коментарями запитів. Дослідження вказують на її ефективність у генеруванні та валідації корисних навантажень SQLi .
- Аналіз та наслідки: "Omni"-природа GPT-4-o передбачає майбутнє, де аналіз безпеки може включати не лише текст запитів, але й пов’язані візуальні чи слухові сигнали, хоча пряме застосування до SQL-ін’єкцій потребує подальшого вивчення. Покращена швидкість та економічність порівняно з GPT-4 Turbo можуть зробити її більш життєздатним варіантом для ширшого розгортання в системах безпеки. Здатність GPT-4-o ефективніше обробляти довші контексти може призвести до покращеного виявлення багатокрокових атак SQL-ін’єкцій, де шкідливе навантаження будується через кілька запитів або взаємодій. Деякі складні методи SQL-ін’єкцій передбачають введення невеликих фрагментів шкідливого коду через кілька, на перший погляд, безпечних запитів, які потім об’єднуються базою даних для виконання повної атаки. Модель з великим та ефективним контекстним вікном може краще відстежувати ці фрагментовані атаки та ідентифікувати загальний зловмисний намір.

GPT-4 Turbo:

- Розробник: OpenAI
- Дата випуску: 2023
- Тип моделі: Оптимізована версія GPT-4
- Архітектура: Немає інформації.
- Контекстне вікно: До 128 тис. токенів
- Ключові особливості: Ті ж можливості, що й GPT-4, але швидше та дешевше, збільшене контекстне вікно для обробки великих обсягів інформації, краще підходить для інтерактивних застосунків.

- Обмеження: Якість відповіді може бути дещо нижчою порівняно з GPT-4 у складних логічних запитах, обмежена гнучкість у спеціалізованих завданнях.
- Актуальність для виявлення SQL-ін’єкцій: Дуже велике контекстне вікно дозволяє аналізувати надзвичайно довгі або складні SQL-запити, які можуть містити складні або заплутані спроби ін’єкцій . Дослідження вивчають її використання для виявлення ін’єкцій запитів, пов’язаної вразливості безпеки .
- Аналіз та наслідки: Економічність та швидкість GPT-4 Turbo порівняно зі стандартним GPT-4 роблять її більш практичним варіантом для аналізу великих обсягів даних запитів у реальному часі або майже в реальному часі. Однак потенційний компроміс у точності для дуже складної логіки необхідно враховувати в критичних застосуваннях безпеки. Здатність GPT-4 Turbo обробляти 128 тис. токенів може дозволити аналізувати цілі журнали додатків або навіть вихідний код разом з аналізом запитів для виявлення потенційних вразливостей, що призводять до SQL-ін’єкцій. Іноді вразливість, що дозволяє SQL-ін’єкцію, полягає не безпосередньо в самому запиті, а в тому, як вхідні дані користувача обробляються в коді програми перед використанням у запиті. LLM з великим контекстним вікном потенційно може аналізувати як код, так і згенеровані запити для виявлення цих базових вразливостей.

Claude-3 Sonnet:

- Розробник: Anthropic.
- Дата випуску: 2024.
- Тип моделі: Велика мовна модель.
- Архітектура: Немає інформації.
- Контекстне вікно: 200 тис. токенів .
- Ключові особливості: Велике контекстне вікно, покращена продуктивність, підтримка зображень
- Обмеження: Хоча Claude 3 Sonnet є потужною моделлю, вона може бути менш ефективною для завдань, які вимагають максимальної точності та глибокого розуміння, порівняно з моделлю Claude 3 Opus. Крім того, через великий розмір

моделі, можуть виникати затримки у відповідях під час обробки дуже великих обсягів даних.

- Актуальність для виявлення SQL-ін’єкцій: Велике контекстне вікно корисне для аналізу великих журналів запитів або дуже складних SQL-виразів на предмет потенційних спроб ін’єкцій . Дослідження показують її високу продуктивність у задачах кодування , що може призвести до кращого розуміння синтаксису SQL та виявлення зловмисних патернів.
- Аналіз та наслідки: Баланс між навичками та швидкістю, який пропонує Sonnet, може зробити її вдалим компромісом між швидкістю та точністю GPT-4 для певних завдань аналізу безпеки. Її висока продуктивність у кодуванні свідчить про те, що вона може бути особливо вправною у розумінні тонкощів SQL та виявленні тонких методів ін’єкцій. Покращена здатність Claude 3 Sonnet розуміти нюанси та складні інструкції може бути цінною для виявлення складних атак SQL-ін’єкцій, які покладаються на тонкі маніпуляції параметрами або синтаксисом запиту. Зловмисники часто намагаються обійти фільтри безпеки, незначно змінюючи свої корисні навантаження. LLM, яка добре розуміє нюансовану мову та код, з більшою ймовірністю розпізнає ці тонкі варіації як зловмисні.

Claude-3 Opus:

- Розробник: Anthropic
- Дата випуску: Березень 2024 .
- Тип моделі: Велика мультимодальна модель .
- Архітектура: Немає інформації.
- Контекстне вікно: 200 тис. токенів, розширюється до 1 млн для певних випадків використання .
- Ключові особливості: Найрозумніша модель у сімействі Claude 3, відмінно справляється зі складними міркуваннями, математикою та кодуванням, мультимодальні можливості (текст та зображення), покращена вільність у неанглійських мовах .

- Обмеження: Claude 3 Opus не може розуміти код, створювати зображення, взаємодіяти з користувачем за допомогою голосу або підтримувати додаткові плагіни, що може бути вирішальним для користувачів, які сильно покладаються на ці функції. Модель дотримується строгих етичних принципів, які обмежують її здатність виконувати певні завдання, наприклад, участь у рольових іграх або моделюванні особистості. Через свою передову архітектуру Claude 3 Opus вимагає значних обчислювальних ресурсів, що може бути обтяжливим для невеликих та середніх команд.
- Актуальність для виявлення SQL-ін’єкцій: Її вища інтелектуальність та здатність до міркувань у поєднанні з дуже великим контекстним вікном роблять її дуже перспективною для виявлення навіть найскладніших та заплутаних атак SQL-ін’єкцій . Дослідження вказують на її високу продуктивність у виявленні вразливостей програмного забезпечення .
- Аналіз та наслідки: Майже людський рівень розуміння та вільного володіння мовою свідчать про те, що Opus може бути дуже ефективною у розумінні наміру, що стоїть за SQL-запитами, навіть якщо вони створені так, щоб виглядати безпечними. Здатність обробляти контекст до 1 млн токенів для конкретних випадків використання може бути безцінною для аналізу надзвичайно великих наборів даних запитів або навіть усього коду програми. Здатність Claude 3 Opus виявляти обмеження оцінок та розпізнавати штучно вставлену інформацію натякає на потенціал розпізнавання складних методів SQL-ін’єкцій, розроблених для обходу традиційних методів виявлення. Якщо модель може виявляти аномалії в сценарії тестування, вона також може виявляти тонкі аномалії в SQL-запитах, характерні для складних атак ін’єкцій. Це може включати розпізнавання незвичайних структур коду, неочікуваних перетворень типів даних або інших тонких маніпуляцій.

Claude-3 Haiku:

- Розробник: Anthropic
- Дата випуску: Березень 2024 .
- Тип моделі: Велика мультимодальна модель .

- Архітектура: Немає інформації
- Контекстне вікно: 200 тис. токенів .
- Ключові особливості: Найшвидша та найпростіша модель у сімействі Claude 3, майже миттєва реакція, економічність, підтримка базових мультимодальних завдань.
- Обмеження: Знижена здатність обробки складності порівняно з іншими моделями Claude 3 .
- Актуальність для виявлення SQL-ін’єкцій: Хоча її знижена здатність обробки складності може обмежити її можливість виявлення дуже складних атак, її швидкість та економічність можуть зробити її придатною для моніторингу потоків запитів у реальному часі, потенційно виявляючи поширені або простіші патерни ін’єкцій .
- Аналіз та наслідки: Швидкість Haiku робить її кандидатом для розгортання у високопродуктивних системах, де пріоритетом є негайне виявлення поширених загроз, а не глибокий аналіз кожного окремого запиту. Її економічність також може бути привабливою для масштабних розгортань. Здатність Haiku швидко та точно перекладати може бути корисною у глобальних програмах, де SQL-запити або параметри можуть бути різними мовами. У програмах, що обслуговують глобальну базу користувачів, SQL-запити можуть містити коментарі або параметри різними мовами. Можливості перекладу Haiku можуть допомогти в нормалізації цих запитів до однієї мови для полегшення аналізу системою виявлення.

Model Comparison Overview

Comparison Overview: Side-by-Side Performance Analysis of GPT-4o vs GPT-4 vs GPT-4 Turbo LLM Models Across Key Metrics and Benchmarks.

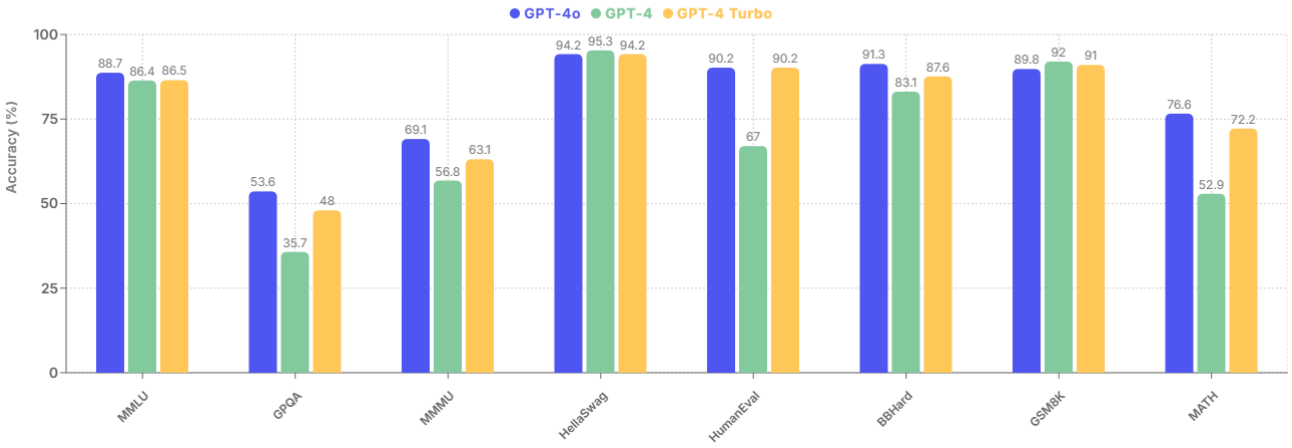

Рисунок 2.2 – Огляд порівняння моделей GPT

LLM Model Performance Overview

Performance Overview : Visualizing and Analyzing Key Metrics of Two Leading LLM Models for Performance Comparison.

MODEL	GPT-4O	GPT-4	GPT-4 TURBO
Context size	128K	32K	128K
Cutoff date	Oct 2023	November 2023	November 2023
Input/output cost	$0.005 / $0.015	$0.06 / $0.12	$0.01 / $0.03
Latency (TTFT)	0.48s	0.60s	0.60s
Throughput	80t/s	28.5t/s	28.5t/s

Рисунок 2.3 – Огляд ефективності моделей GPT

Comparing GPT-4o vs GPT-4 vs GPT-4 Turbo

A detailed comparison of GPT-4o vs GPT-4 vs GPT-4 Turbo performance and features.

BENCHMARK	GPT-4O	GPT-4	GPT-4 TURBO
MMLU	88.7%	86.4%	86.5%
GPQA	53.6%	35.7%	48%
MMMU	69.1%	56.8%	63.1%
HellaSwag	94.2%	95.3%	94.2%
HumanEval	90.2%	67%	90.2%
BBHard	91.3%	83.1%	87.6%
GSM8K	89.8%	92%	91%
MATH	76.6%	52.9%	72.2%

Рисунок 2.4 – Деталізоване порівння у бенчмарках GPT моделей

Model Comparison Overview

Comparison Overview: Side-by-Side Performance Analysis of Claude 3 Haiku vs Claude 3 Opus vs Claude 3 Sonnet LLM Models Across Key Metrics and Benchmarks.

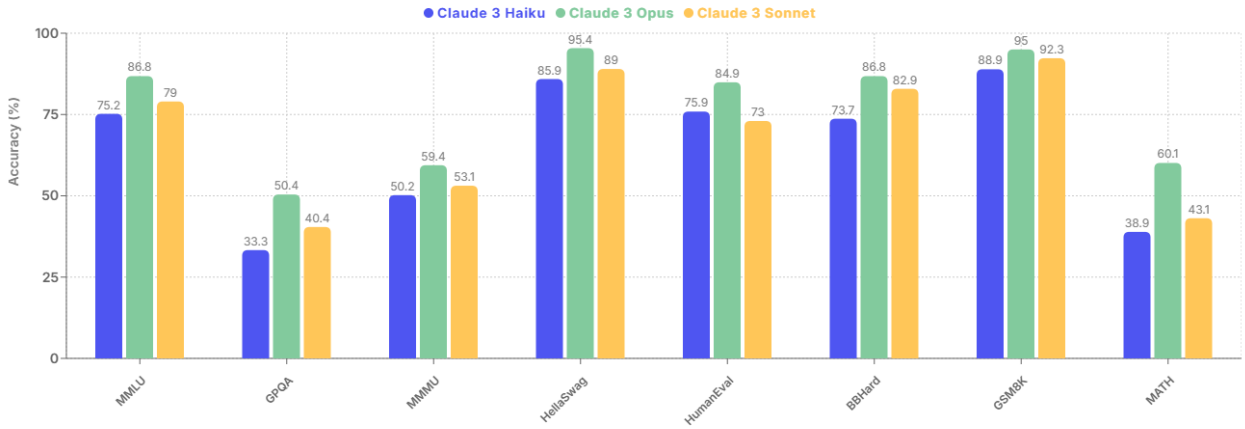

Рисунок 2.5 – Огляд порівняння моделей Claude

LLM Model Performance Overview

Performance Overview : Visualizing and Analyzing Key Metrics of Two Leading LLM Models for Performance Comparison.

MODEL	CLAUDE 3 HAIKU	CLAUDE 3 OPUS	CLAUDE 3 SONNET
Context size	200K	200K	256K
Cutoff date	March 2024	May 2024	May 2024
Input/output cost	$0.00025 / $0.00125	$0.015 / $0.075	$0.0002 / $0.0011
Latency (TTFT)	0.55s	1.66s	0.65s
Throughput	134.3t/s	23.0t/s	170.4t/s

Рисунок 2.6 – Огляд ефективності моделей Claude

Comparing Claude 3 Haiku vs Claude 3 Opus vs Claude 3 Sonnet

A detailed comparison of Claude 3 Haiku vs Claude 3 Opus vs Claude 3 Sonnet performance and features.

BENCHMARK	CLAUDE 3 HAIKU	CLAUDE 3 OPUS	CLAUDE 3 SONNET
MMLU	75.2%	86.8%	79%
GPQA	33.3%	50.4%	40.4%
MMMU	50.2%	59.4%	53.1%
HellaSwag	85.9%	95.4%	89%
HumanEval	75.9%	84.9%	73%
BBHard	73.7%	86.8%	82.9%
GSM8K	88.9%	95%	92.3%
MATH	38.9%	60.1%	43.1%

Рисунок 2.7 – Деталізоване порівння у бенчмарках Claude моделей

2.3. Архітектура та технології веб-додатку

Веб-додаток побудований із використанням технологій, які дозволяють забезпечити швидкість розробки, високу продуктивність та зручність для користувачів. Для побудови інтерфейсу використовується React, що дозволяє створювати динамічні інтерфейси, TypeScript для типізації та безпечної роботи з даними, а також Tailwind CSS для швидкого й ефективного стилізування.

React – це одна з найпопулярніших бібліотек для створення користувацьких інтерфейсів, яка дозволяє ефективно оновлювати UI на основі змін у стані додатка. Завдяки компонентному підходу React дозволяє модульно розробляти інтерфейси, що зручно для масштабування та підтримки. Для побудови веб-додатку було використано функціональні компоненти, які дають можливість описувати логіку та UI в єдиному місці.

TypeScript є важливою частиною стеку, оскільки додає типізацію до JavaScript, що дозволяє мінімізувати помилки під час розробки. Використання TypeScript дає змогу працювати з більш безпечним і передбачуваним кодом, що знижує ймовірність помилок і покращує підтримку великого коду. У проекті TypeScript використовується для типізації даних, компонентів, а також для взаємодії з API і базою даних, що робить розробку більш стабільною та зрозумілою.

Tailwind CSS використовується для швидкої та ефективної стилізації інтерфейсу. Це CSS-фреймворк, що дозволяє за допомогою утилітних класів створювати адаптивні та естетичні дизайни без необхідності написання додаткового CSS. Tailwind CSS забезпечує високу гнучкість, дозволяючи розробнику легко контролювати відступи, розміри, кольори та інші стилістичні аспекти елементів інтерфейсу.

Архітектура додатка

Next.js забезпечує основу для побудови веб-додатка, інтегруючи можливості серверного рендерингу (SSR) і статичної генерації контенту (SSG), що дозволяє значно покращити продуктивність додатка. Веб-додаток розділений на фронтенд і бекенд, що взаємодіють через API-запити для обробки SQL-запитів і їх аналізу.

Frontend: Фронтенд додатка побудований на Next.js з використанням React і TypeScript. Компоненти користувацького інтерфейсу реалізовано з використанням React-компонентів, що надають гнучкість у створенні сторінок і функціоналу. Стилізація реалізована через Tailwind CSS, що дозволяє швидко і зручно налаштовувати зовнішній вигляд елементів інтерфейсу.

Backend: Серверна частина додатка обробляє запити, що надходять з фронтенду, і використовує API моделей ШІ для аналізу SQL-запитів. Всі результати зберігаються у SQLlite базі даних для подальшого доступу і аналізу [27].

Загальна структура проекту

Файлова структура проєкту організована таким чином, щоб забезпечити зручність у розробці, підтримці та масштабуванні додатка. Основні каталоги та файли виконують різні функції та містять відповідні ресурси.

Основні каталоги:

- app/ - основний каталог додатка, що містить frontend-компоненти та API-маршрути.
- api/ - тут розміщені обробники API-запитів, example-route.
- Файли інтерфейсу: layout.tsx, page.tsx, QueryProvider.tsx, які відповідають за відображення та управління UI.
- fonts/ - шрифти потрібні для нашого додатку
- components/ - містить UI-компоненти, що використовуються в додатку.
- db/ - містить конфігурації та файли, пов'язані з базою даних.
- lib/ - додаткові утиліти та допоміжні функції для роботи додатка.
- migrations/ - файли міграцій для управління схемою бази даних.
- node_modules/ - директорія залежностей проєкту, які встановлюються через npm або pnpm.

Основні конфігураційні файли:

- .env - змінні середовища, що зберігають ключі доступу, URL-адреси та інші налаштування.
- drizzle.config.ts - конфігурація ORM Drizzle для роботи з базою даних.
- next.config.mjs - конфігураційний файл Next.js.
- package.json та package-lock.json - файли для керування залежностями проєкту.
- tailwind.config.ts - налаштування Tailwind CSS.
- tsconfig.json - конфігурація TypeScript.

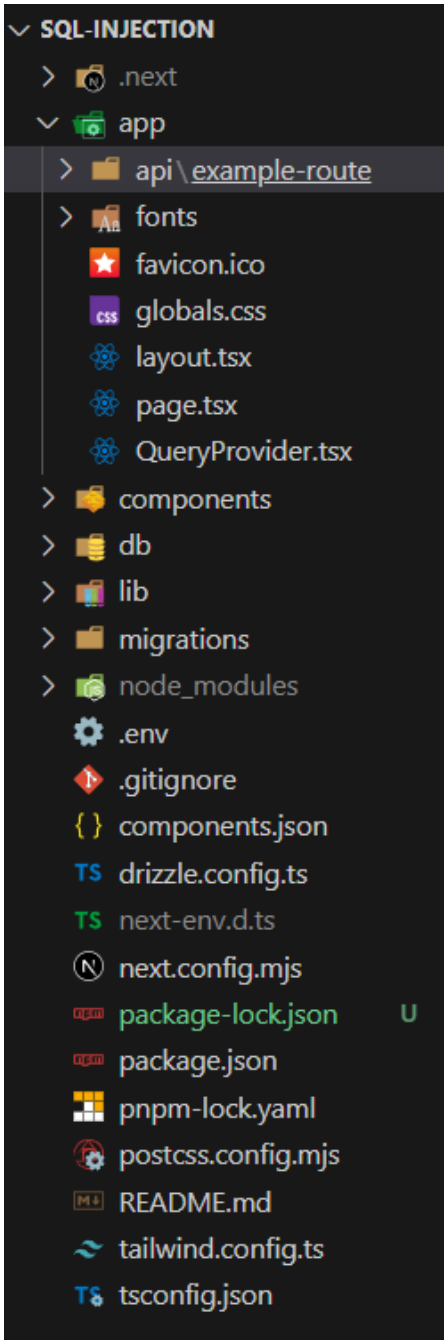

Рисунок 2.8 – Загальна структура проекту

Загалом додаток складається з однієї сторінки page.tsx, яка в собі має лише два основних компоненти: FormCard та AttacksHistory а вони в свою чергу містять в собі всі інші компоненти, які формують UI додатку.

```tsx
import AttacksHistory from "@/components/attacks-history";
import FormCard from "@/components/formCard";
import { revalidatePath } from "next/cache";  68.8k (gzipped: 19.1k)

export default function Home() {
  const revalidate = async () => {
    "use server";
    revalidatePath("/");
  };
  return (
    <div className="p-4 md:grid md:grid-cols-5 md:justify-between flex flex-col space-y-4 md:space-y-0">
      <FormCard revalidate={revalidate} />
      <AttacksHistory />
    </div>
  );
}
```

Рисунок 2.9 – Головна сторінка, яка відображається користувачу

Сторінка має свій layout, тобто файл, який огортає сторінку і дає їй такі речі, як потрібні шрифти, провайдери для роботи з запитами на сервер, метадані для кращого SEO.

```tsx
import type { Metadata } from "next";
import localFont from "next/font/local";  640 (gzipped: 376)
import "./globals.css";
import ReactQueryProvider from "./QueryProvider";

const geistSans = localFont({
  src: "./fonts/GeistVF.woff",
  variable: "--font-geist-sans",
  weight: "100 900",
});
const geistMono = localFont({
  src: "./fonts/GeistMonoVF.woff",
  variable: "--font-geist-mono",
  weight: "100 900",
});

export const metadata: Metadata = {
  title: "SQL injection test",
  description: "Generated by create next app",
};

export default function RootLayout({
  children,
}: Readonly<{
  children: React.ReactNode;
}>) {
  return (
    <html lang="en">
      <body
        className={`${geistSans.variable} ${geistMono.variable} antialiased`}
      >
        <ReactQueryProvider>
          <main>{children}</main>
        </ReactQueryProvider>
      </body>
    </html>
  );
}
```

Рисунок 2.10 – Файл обгортка page.tsx

На сторінці зображена форма для атаки (компонет FormCard). На ній можна вибрати модель LLM, яка буде приймати текст атаки у Input поле та відправляти запит на сервер для перевірки чи є даний текст SQL атакою.

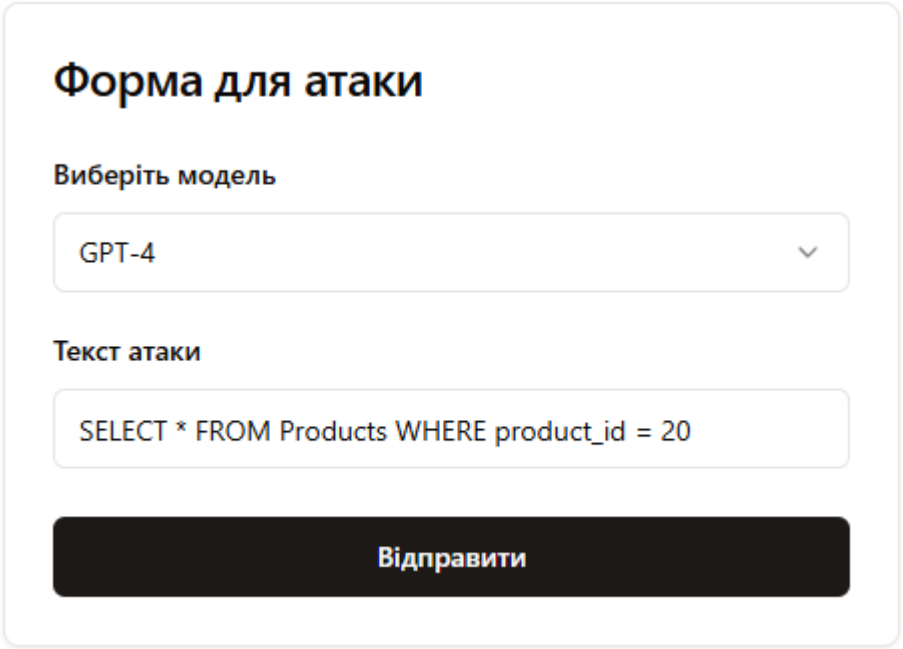

Рисунок 2.11 – Форма для атаки

Також на сторінці є компонент (AttacksHistory) результатів запиту на сервер, де міститься така інформація:

- Чи знайдена атака, чи ні на думку LLM
- Назва моделі, яка використовувалася

Дата атаки

- Час витрачений на обробку запиту
- Пояснення від LLM моделі
- Промпт, який був використаний для LLM моделі
- Сам текст атаки, який аналізувала LLM модель

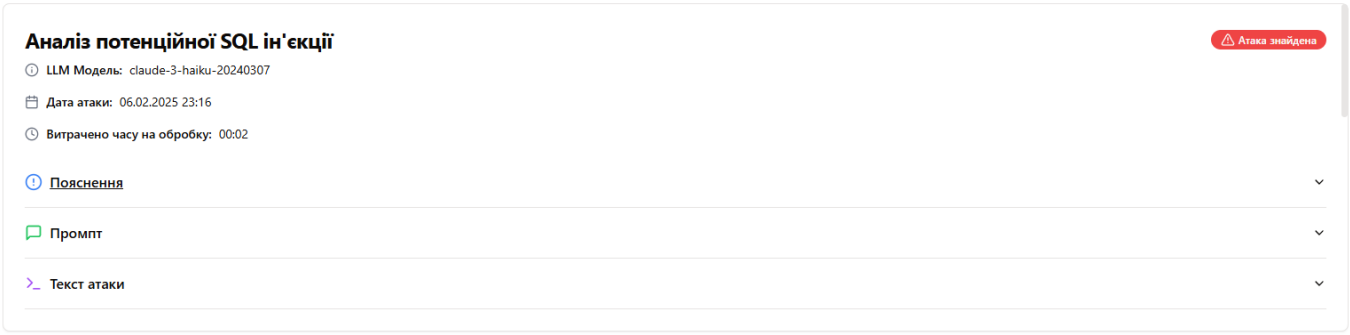

Рисунок 2.12 – Результат роботи LLM моделі

Всі атаки зберігаються у базі даних та підтягуються з неї під час запуску додатку, тому всі варіанти можна подивитися в історії прогортавши вниз використовуючи ScrollBar.

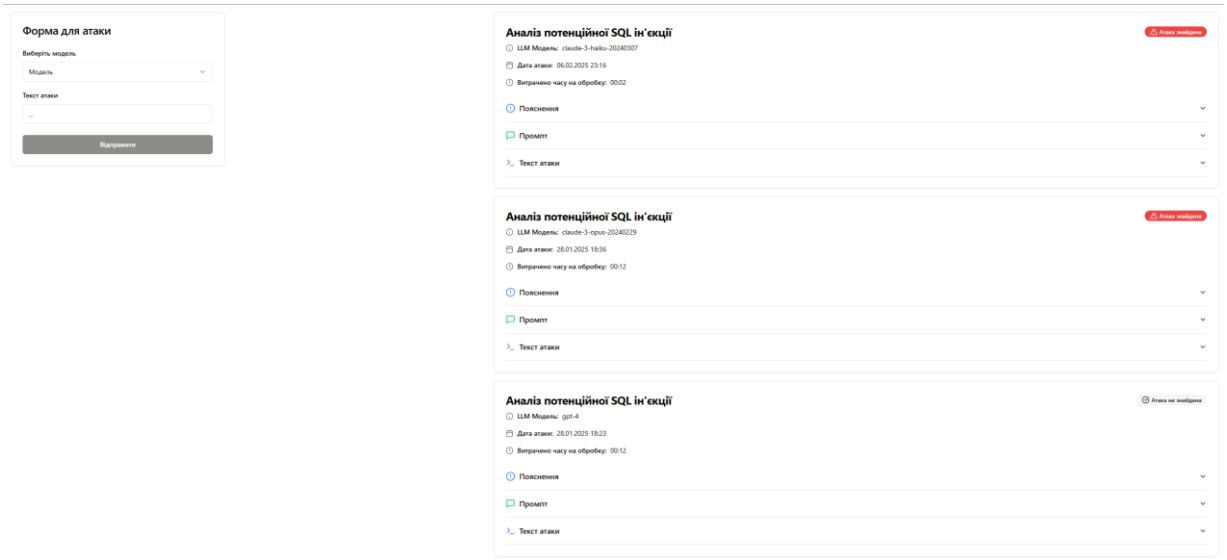

Рисунок 2.13 – Загальний вигляд додатку

У проєкті використовуються готові UI-компоненти з бібліотеки shadcn/ui, які забезпечують високу гнучкість та повторне використання елементів інтерфейсу. Використання таких компонентів значно спрощує процес розробки, дозволяючи розробникам швидко створювати якісний UI без необхідності писати кастомні стилі або логіку з нуля.

Каталог ui/ містить наступні реюзабельні компоненти:

- accordion.tsx – компонент акордеону, який дозволяє згортати та розгортати вміст за допомогою кліка. Використовується для організації великої кількості інформації у стиснутому вигляді.
- badge.tsx – невеликий маркер або бейдж, що використовується для відображення статусів, міток або додаткової інформації поруч із основним контентом.
- button.tsx – стандартний кнопковий компонент із підтримкою стилізації та різних станів (натискання, ховер, неактивний стан тощо).
- card.tsx – компонент картки, що використовується для відображення інформаційних блоків з контентом. Часто містить заголовок, зображення та опис.
- input.tsx – текстове поле для введення даних, що підтримує валідацію та різні режими введення.
- label.tsx – компонент мітки, що використовується для прив’язки до полів введення з метою покращення доступності та UX.
- scroll-area.tsx – зона прокрутки, яка дозволяє кастомізувати поведінку прокручуваного контенту, наприклад, у модальних вікнах або списках.
- select.tsx – випадаючий список для вибору одного з варіантів, із підтримкою клавіатурної навігації та стилізації.

Ці компоненти забезпечують уніфікований дизайн і взаємодію користувача, що робить їх зручними для використання у всьому додатку. Завдяки їхній модульності та налаштовуваності, вони легко адаптуються під конкретні потреби проєкту.

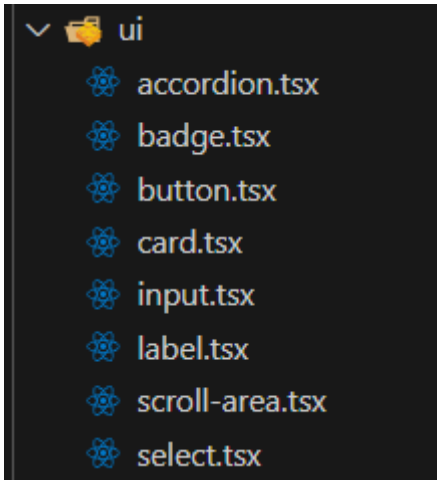

Рисунок 2.14 – Реюзабельні компоненти

Використання цих реюзабельних компонентів дає можливість створити компоненти, які формують наш додаток, а саме:

attacks-history.tsx, formCard – головні компоненти, які відображаються на сторінці (про них було згадано раніше).

loading-animation.tsx – компонент відповідающий за анімацію трьох крапок поки модель думає

selectBasic.tsx – покращена версія звичайного select

sql-injection-card.tsx – карточка одної атаки, яка потім прокидується у attack-history.tsx

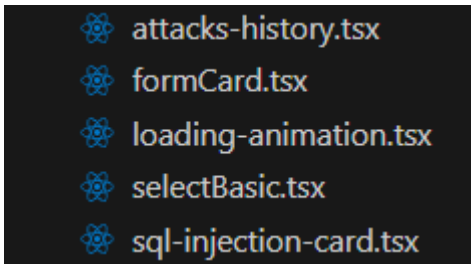

Рисунок 2.15 – Компоненти створені на основі реюзабельних

Файл route.ts є основним серверним маршрутом, що відповідає за аналіз введених користувачем текстів та визначення, чи є вони SQL-атаками.

Файл підключає необхідні бібліотеки для роботи з LLM, базою даних та обробки запитів.

```
import { NextRequest } from "next/server";  39.3k (gzipped: 14.1k)
import { openai } from "@ai-sdk/openai";  35.9k (gzipped: 9.6k)
import { generateObject } from "ai";  78.3k (gzipped: 20.7k)
import * as z from "zod";  63.2k (gzipped: 15.2k)
import { db } from "@/db";
import { attackTable } from "@/db/schema";
import { anthropic } from "@ai-sdk/anthropic";  24k (gzipped: 7.2k)
import { format } from "date-fns";  19.5k (gzipped: 5.6k)
```

Рисунок 2.16 – Імпорт бібліотек

Окрема функція getModel визначає, яку саме LLM-модель використовувати для обробки SQL-запиту.

```
const getModel = (model: string) => {
  switch (model) {
    case "gpt-4":
      return openai("gpt-4");
    case "gpt-4-0":
      return openai("gpt-4o");
    case "gpt-4-turbo":
      return openai("gpt-4-turbo");
    case "claude-3-sonnet-20240229":
      return anthropic("claude-3-sonnet-20240229");
    case "claude-3-opus-20240229":
      return anthropic("claude-3-opus-20240229");
    case "claude-3-haiku-20240307":
      return anthropic("claude-3-haiku-20240307");
    default:
      return openai("gpt-4");
  }
};
```

Рисунок 2.17 – Функція вибору моделі

Next.js дозволяє налаштовувати обмеження часу виконання та динамічність запитів.

```
export const maxDuration = 60;
export const dynamic = "force-dynamic";
```

Рисунок 2.18 – Налаштування API

функція POST приймає вхідні дані та визначає, яка модель буде використана для аналізу.

```
export const maxDuration = 60;
export const dynamic = "force-dynamic";

export async function POST(request: NextRequest) {
  const startTime = Date.now();

  const data = await request.json();

  const searchParams = request.nextUrl.searchParams;

  const searchParamsModel = searchParams.get("model");

  if (!searchParamsModel)
    return new Response(JSON.stringify({ error: "model not found" }), {
      status: 400,
    });

  const model = getModel(searchParamsModel);
```

Рисунок 2.19 – Отримання параметрів запиту

```
const prompt =
  "Ти експерт в кібербезпеці, як ти думаєш це текст є SQL ін'єкцією, яка може нести загрозу? Поясни свою відповідь. Текст атаки: " +
  data.input;
```

Рисунок 2.20 – Формування промпту для LLM

Функція generateObject передає запит у вибрану LLM-модель.

```
const result = await generateObject({
  model: model,
  schema: z.object({
    isSqlInjection: z.boolean(),
    explanation: z
      .string()
      .describe(
        "Чому ти думаєш що це потенційна SQL ін'єкція яка може бути небезпечною ?"
      ),
  }),
  prompt,
});
```

Рисунок 2.21 – Виклик LLM та отримання відповіді

Підрахунок часу, який витратила модель на аналіз запиту

```
const endTime = Date.now();
const processingTime = format(new Date(endTime - startTime), "mm:ss");
```

Рисунок 2.22 – Обчислення часу обробки

Збереження отриманої відповіді у таблиці attackTable у базу даних SQLite

```
await db.insert(attackTable).values({
  isAttack: result.object.isSqlInjection,
  model: searchParamsModel,
  explanation: result.object.explanation,
  input: data.input,
  prompt,
  processingTime,
  createdAt: new Date().toISOString(),
});
```

Рисунок 2.23 – Запис у базу даних

Повертається відповідь клієнту про успішність обробки.

```
  return new Response(JSON.stringify(true), {
    status: 200,
  });
}
```

Рисунок 2.24 – Повернення відповіді API

2.4.Формування промптів та методи підвищення точності аналізу

Промпт – це текстовий запит, який передається мовній моделі для отримання необхідної відповіді. Якість промпту безпосередньо впливає на точність та релевантність результату, тому його формування є важливим етапом у розробці системи аналізу SQL-атак.

Формування промптів для аналізу SQL-атак

Для ефективного визначення потенційних SQL-ін'єкцій промпт повинен містити чіткі інструкції, які спрямовують мовну модель на коректний аналіз. Основні компоненти ефективного промпту:

Чіткість формулювання – модель повинна отримати конкретне завдання без двозначностей.

Контекст – надання додаткової інформації для кращого розуміння завдання.

Приклади – демонстрація прикладів правильних і неправильних запитів допомагає покращити точність відповіді.

Приклад базового промпту:

Ти експерт з кібербезпеки. Оціни, чи містить цей текст SQL-ін'єкцію. Якщо так, поясни, чому він є потенційною загрозою.

Текст: {вхідний_запит}

Покращення точності аналізу

Щоб підвищити точність аналізу SQL-атак, використовуються такі методи:

1. Використання уточнювальних промптів

Модель може отримувати додаткові інструкції, які підсилюють її здатність розпізнавати шкідливі запити. Наприклад:

Ти спеціалізуєшся на аналізі SQL-ін'єкцій. Проаналізуй наступний запит та відповідай у форматі JSON:

{

"isSqlInjection": <true/false>,

"explanation": "Обґрунтування"

}

Запит: {вхідний_запит}

2. Використання різних моделей

Різні мовні моделі мають різний рівень точності та здатність аналізу SQL-атак. Використання GPT-4, Claude-3 та інших моделей дозволяє порівнювати результати та вибирати найбільш точний аналіз.

3. Фільтрація та нормалізація введених даних

Перед передачею тексту у мовну модель слід нормалізувати вхідні дані, наприклад:

Видалення зайвих пробілів та спеціальних символів.

Перетворення тексту у нижній регістр для уніфікації аналізу.

Видалення SQL-коментарів (--, #).

4. Використання ансамблевого підходу

Ансамблевий метод передбачає використання кількох моделей для оцінки одного й того ж запиту. Якщо більшість моделей класифікують запит як SQL-атаку, він вважається небезпечним.

5. Використання постпроцесингу

Після отримання відповіді від мовної моделі можна додатково аналізувати її за допомогою регулярних виразів або фільтрів, щоб уникнути хибнопозитивних або хибнонегативних результатів.

Висновки за другим розділом

У другому розділі дипломної роботи було проведено детальний аналіз та огляд LLM, які використовуються для виявлення SQL-ін'єкцій. Було розглянуто такі моделі, як GPT-4, GPT-4-o, GPT-4 Turbo, Claude-3 Sonnet, Opus та Haiku, а також їхні ключові особливості, переваги та обмеження.

Аналіз показав, що використання LLM для виявлення SQL-ін'єкцій є перспективним напрямком, оскільки ці моделі здатні аналізувати складні текстові патерни та виявляти аномалії, які можуть свідчити про зловмисну діяльність. Зокрема, моделі з великим контекстним вікном (наприклад, GPT-4 Turbo, Claude-3 Opus) дозволяють аналізувати довгі та складні SQL-запити, що підвищує точність виявлення складних ін'єкцій.

Розроблений веб-додаток використовує ці LLM для аналізу текстових запитів та виявлення потенційних SQL-ін'єкцій. Архітектура додатку побудована з використанням сучасних технологій, таких як React, TypeScript та Next.js, що забезпечує високу продуктивність та зручність для користувачів.

Важливим аспектом роботи є формування промптів для LLM. Ефективний промпт повинен містити чіткі інструкції, контекст та приклади, що дозволяє підвищити точність аналізу. Для покращення результатів використовуються методи уточнювальних промптів, фільтрації даних, ансамблевого підходу та постпроцесингу.

Отже, використання LLM для виявлення SQL-ін'єкцій є ефективним методом, який дозволяє покращити безпеку веб-додатків. Розроблений веб-додаток демонструє можливість практичного застосування цих моделей для аналізу SQL-запитів та виявлення потенційних загроз.

РОЗДІЛ 3
ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ РОЗРОБЛЕНОГО МЕТОДУ

3.1. Опис експериментальної платформи

Для проведення експериментальних досліджень розробленого методу виявлення SQL-ін'єкцій на основі штучного інтелекту було підготовлено відповідне середовище тестування, яке дозволяє об'єктивно оцінити ефективність запропонованого рішення.

Веб-застосунок для детекції SQL-ін'єкцій був розгорнутий на платформі хмарного хостингу Vercel, що забезпечує автоматичне масштабування та мінімальний час відповіді серверу. Додаток розроблено із використанням технологій React.js, Next.js та Tailwind CSS для клієнтської частини, а також TypeScript для забезпечення типобезпеки коду. Для взаємодії із моделями штучного інтелекту використовувався серверний функціонал Next.js API Routes, що дозволяє виконувати запити до сторонніх LLM-сервісів безпосередньо із бекенду додатку.

Тестування проводилося на персональному комп’ютері із такими характеристиками: процесор Intel 12-400, оперативна пам'ять 16 ГБ, SSD-накопичувач об'ємом 512 ГБ. Операційна система – Windows 11 Pro. Веб-браузер для тестування – Google Chrome версії 123.0.6312.106. Використання потужного апаратного забезпечення дозволяло уникнути затримок, пов'язаних із локальними обчисленнями, і зосередитися виключно на оцінці ефективності роботи моделей штучного інтелекту.

У якості інструментів штучного інтелекту було використано декілька мовних моделей великого масштабу, зокрема:

- GPT-4 Turbo;
- GPT-4;
- GPT-4-o;

- Claude 3 Haiku;
- Claude 3 Opus;
- Claude 3 Sonnet;

Доступ до зазначених моделей було отримано через офіційні API, із попередньо налаштованими ключами доступу. Для кожної моделі були визначені однакові параметри запитів, такі як температура генерації відповідей та максимальна кількість токенів, що дозволило забезпечити справедливе порівняння між результатами роботи різних моделей.

Важливим етапом підготовки експериментальної платформи було формування тестової бази SQL-ін'єкцій. Для забезпечення різноманітності тестових кейсів були використані наступні джерела:

Публічно доступні бази даних відомих вразливостей (CVE);

Набір типових запитів SQL-ін'єкцій OWASP [13];

Ручно створені приклади складних та комбінованих атак, що включали кодовані ін'єкції, багаторівневі маніпуляції, використання UNION-запитів та обхід стандартних захистів.

Тестова база складалася зі 20 запитів.

Кожен тестовий запит передавався до API веб-застосунку, після чого модель здійснювала аналіз і повертала відповідь щодо наявності чи відсутності ознак SQL-ін'єкції. Для забезпечення об'єктивності тестування було передбачено автоматичний лог результатів, що містив:

- Текст запиту;
- Модель, яка проводила аналіз;
- Отриману відповідь (SQL-ін'єкція виявлена / не виявлена);
- Час обробки запиту.

Таким чином, експериментальна платформа дозволила створити умови, максимально наближені до реальних, для всебічної перевірки розробленого методу. Комплексний підхід до побудови середовища, стандартизовані параметри тестування, використання кількох LLM-моделей і різноманітних сценаріїв атак

забезпечили достовірність отриманих результатів та надали можливість провести детальну оцінку якості запропонованого рішення.

3.2. Проведення тестувань на реальних кейсах SQL-ін'єкцій та їх аналіз

Для оцінки ефективності моделей машинного навчання у виявленні SQL-ін'єкцій було проведено тестування на двох наборах даних: перший складався з 10 зловмисних запитів, другий – з 10 безпечних запитів. Тестування проводилося з використанням шести моделей: GPT-4, GPT-4-O, GPT-4 Turbo, Claude 3 Haiku, Claude 3 Opus та Claude 3 Sonnet. Метою було визначити здатність моделей коректно ідентифікувати зловмисні запити та уникати хибнопозитивних спрацьовувань на безпечних запитах. Результати тестування представлені в таблицях нижче, де символ "✔" означає, що модель класифікувала запит як зловмисний, а символ "✘" – що запит визнано безпечним.

Перший етап тестування включав аналіз виключно зловмисних запитів, сформованих на основі даних із CVE, OWASP та вручну створених складних атак, таких як кодовані ін’єкції, використання UNION-запитів та обхід стандартних захистів. Оскільки всі запити в цій тестовій базі містили реальні загрози, очікувалося, що моделі мають їх коректно ідентифікувати. Результати цього етапу наведені в таблиці 3.1. Усі моделі продемонстрували 100% точність у виявленні SQL-ін'єкцій. Наприклад, базовий запит SELECT * FROM users WHERE username = 'admin' OR '1'='1' --' AND password = ''; який використовує тавтологію, був коректно класифікований усіма моделями. Складніші запити, такі як SELECT * FROM users WHERE username = 'admin' AND password = 0x414e4420313d31 --'; з hex-кодуванням або SELECT * FROM users WHERE username = 'admin' AND password = %2527%2527%2520%254F%2552%2520%25271%2527%253D%25271 --'; з подвійним URL-кодуванням, також були успішно ідентифіковані. Запити з часовою затримкою, наприклад, SELECT * FROM users WHERE username = 'admin' AND IF(1=1, SLEEP(5), 0) --';, та вкладені запити, як-от SELECT * FROM users WHERE username = 'admin'

AND password = (SELECT password FROM users WHERE username = 'admin') --';, не залишилися поза увагою. Час обробки запитів варіювався: GPT-4-O виявилася найшвидшою з середнім часом 1,1 секунди, тоді як Claude 3 Opus була найповільнішою з середнім часом 13,6 секунд. Загалом, усі моделі показали високу ефективність у розпізнаванні зловмисних SQL-запитів у контрольованому наборі даних.

Таблиця 3.1

ТЕКСТ ЗАПИТУ	**GPT-4**	**GPT-4-O**	**GPT-4 TURBO**	**CLAUDE 3 HAIKU**	**CLAUDE 3 OPUS**	**CLAUDE 3 SONNET**
SELECT * FROM users WHERE username = 'admin' OR '1'='1' --' AND password = '';	✔ /00.08 с	✔ /00.01 с	✔ /00.05 с	✔ /00.02 с	✔ /00.13 с	✔ /00.08 с
SELECT username, password FROM users WHERE username = 'user' UNION SELECT credit_card, pin FROM billing --;	✔ /00.09 с	✔ /00.01 с	✔ /00.06 с	✔ /00.02 с	✔ /00.15 с	✔ /00.04 с
SELECT * FROM users WHERE username = 'admin' AND IF(1=1, SLEEP(5),0) --;	✔ /00.07 с	✔ /00.01 с	✔ /00.07 с	✔ /00.03 с	✔ /00.13 с	✔ /00.04 с

Продовження Таблиці 3.1

SELECT * FROM users WHERE username = 'admin' AND password = 0x4f4e4420313d31 --; -- Hex-encoded 'AND 1=1'	✔ /00.08 с	✔ /00.01 с	✔ /00.05 с	✔ /00.02 с	✔ /00.12 с	✔ /00.05 с
SELECT * FROM users WHERE username = 'admin' --' AND password = 'password';	✔ /00.05 с	✔ /00.02 с	✔ /00.04 с	✔ /00.01 с	✔ /00.13 с	✔ /00.04 с
SELECT * FROM users WHERE username = 'admin' AND 1=(SELECT LOAD_FILE('http://attacker.com')) --;	✔ /00.06 с	✔ /00.01 с	✔ /00.05 с	✔ /00.02 с	✔ /00.14 с	✔ /00.06 с
SELECT * FROM users WHERE username = 'admin'/**/OR/**/1=1--;	✔ /00.06 с	✔ /00.01 с	✔ /00.06 с	✔ /00.02 с	✔ /00.13 с	✔ /00.11 с

Продовження Таблиці 3.1

EXEC sp_executesql 'SELECT * FROM users WHERE username = "admin" OR 1=1 --';	✔ /00.07 с	✔ /00.01 с	✔ /00.05 с	✔ /00.02 с	✔ /00.15 с	✔ /00.04 с
SELECT * FROM users WHERE username = 'admin' AND password = '%2527%252527%2520%2545%252525%20%25271%2527%253D%2527' -- Double URL-encoded " OR '1'='1"	✔ /00.06 с	✔ /00.02 с	✔ /00.07 с	✔ /00.02 с	✔ /00.15 с	✔ /00.05 с
SELECT * FROM users WHERE username = 'admin' AND password =(SELECT password FROM users WHERE username = 'admin') --;	✔ /00.08 с	✔ /00.01 с	✔ /00.06 с	✔ /00.02 с	✔ /00.13 с	✔ /00.07 с

Другий етап тестування був спрямований на оцінку моделей у виявленні безпечних запитів та текстових входів, які не містили жодних ознак SQL-ін'єкцій. Ці запити включали типові операції з базою даних, такі як вибірка даних, оновлення записів, вставка нових даних та видалення записів, а також текстові рядки, що

імітували SQL-запити або містили SQL-подібні слова. Очікувалося, що моделі коректно класифікують ці вхідні дані як безпечні, позначаючи їх символом "❌".

Проте результати, наведені в таблиці 3.2, виявили значну кількість хибнопозитивних спрацьовувань, демонструючи схильність моделей до надмірного виявлення загроз. Зокрема, навіть прості та безпечні конструкції часто помилково ідентифікувалися як SQL-ін'єкції.

Наприклад, запит select username, pass from accounts where id = "1"; -- testing only, який є стандартним запитом вибірки з доданим коментарем, був хибно класифікований як зловмисний усіма моделями. Аналогічно, моделі Claude 3 Haiku та Claude 3 Sonnet помилково позначили запит SELECT FROM users WHERE name = 'test'; як зловмисний, хоча GPT-4, GPT-4-O та GPT-4 Turbo, Claude 3 Opus правильно класифікували його як безпечний.

Текстові вхідні дані також викликали проблеми. Фрази, що лише описували SQL-операції, наприклад, This is an SQL injection test або Normal text '; DROP DATABASE; -- but just kidding!, часто помилково позначалися як атаки. Навіть вирази, що використовувалися для валідації даних, такі як Payload: ' OR (SELECT COUNT(*) FROM users) > 0 -- validation test, були хибно інтерпретовані як зловмисні.

Найменшу кількість помилок серед текстових запитів продемонструвала модель Claude 3 Opus, коректно класифікувавши як безпечні This common text without nothing та Payload: "OR (SELECT COUNT(*) FROM users) > 0 – validation test. Однак, навіть ця модель не уникнула хибнопозитивних спрацювань для інших безпечних запитів.

Загалом, усі моделі продемонстрували тенденцію до надмірного виявлення загроз, що може призвести до блокування легітимних операцій у реальних системах. Час обробки запитів коливався в межах 3 – 16 секунд, що не виявило чіткої залежності від типу запиту, але підкреслює необхідність значного покращення точності класифікації безпечних сценаріїв. GPT-4-O знову показала найшвидший середній час обробки (1,6 секунди), тоді як Claude 3 Opus залишалася найповільнішою (12,3 секунди на запит).

Таблиця 3.2

ТЕКСТ ЗАПИТУ	**GPT-4**	**GPT-4-O**	**GPT-4 TURBO**	**CLAUDE 3 HAIKU**	**CLAUDE 3 OPUS**	**CLAUDE 3 SONNET**
select username, pass from accounts where id = "1"; -- testing only	✓ /00.09 c	✓ /00.07 c	✓ /00.05 c	✓ /00.02 c	✓ /00.14 c	✓ /00.04 c
This is an SQL injection test	✓ /00.08 c	✓ /00.01 c	✓ /00.05 c	✓ /00.02 c	✗ /00.13 c	✓ /00.05 c
Trying to inject: DROP TABLE users	✓ /00.05 c	✓ /00.01 c	✓ /00.03 c	✓ /00.02 c	✓ /00.13 c	✓ /00.04 c
Normal text ; DROP DATABASE; -- but just kidding!	✓ /00.07 c	✓ /00.01 c	✓ /00.05 c	✓ /00.02 c	✓ /00.13 c	✓ /00.04 c
INSERT INTO logs (comment) VALUES ('Attempt to inject here: ' OR 1=1')	✓ /00.07 c	✓ /00.01 c	✓ /00.05 c	✓ /00.01 c	✓ /00.16 c	✓ /00.04 c
SELECT * FROM users where name = 'test'	✗ /00.08 c	✓ /00.01 c	✗ /00.06 c	✓ /00.02 c	✗ /00.11 c	✓ /00.05 c
Fake injection: admin' -- just a joke.	✓ /00.07 c	✓ /00.01 c	✓ /00.04 c	✓ /00.01 c	✗ /00.10 c	✓ /00.04 c
This common text without nothing	✗ /00.08 c	✓ /<00.01 c	✓ /00.05 c	✗ /00.01 c	✗ /00.05 c	✓ /00.03 c

Продовження Таблиці 3.2

UNION SELECT password FROM users (for testing only)	✓ /00.10 с	✓ /00.02 с	✓ /00.11 с	✓ /00.02 с	✓ /00.16 с	✓ /00.04 с
Payload: ' OR (SELECT COUNT(*) FROM users) > 0 -- validation test	✓ /00.12 с	✓ /00.08 с	✓ /00.02 с	✓ /00.02 с	✓ /00.12 с	✓ /00.04 с

Висновки за третім розділом

У результаті проведених експериментальних досліджень було підтверджено ефективність запропонованого методу виявлення SQL-ін'єкцій на основі використання моделей штучного інтелекту великого масштабу. Побудована експериментальна платформа дозволила забезпечити об'єктивне тестування в умовах, наближених до реальних сценаріїв роботи веб-застосунків.

Аналіз результатів тестування на зловмисних запитах продемонстрував високу ефективність усіх розглянутих моделей. Усі шість моделей – GPT-4, GPT-4-O, GPT-4 Turbo, Claude 3 Haiku, Claude 3 Opus та Claude 3 Sonnet – змогли безпомилково виявити всі запропоновані SQL-ін'єкційні запити, незалежно від їх складності чи способу маскування. Це свідчить про здатність сучасних LLM-технологій успішно ідентифікувати загрози у запитах до баз даних.

Водночас, результати тестування на безпечних запитах виявили проблему хибнопозитивних спрацьовувань, коли безпечні запити або текстові повідомлення помилково класифікувалися як потенційно небезпечні. Це може призводити до блокування легітимних запитів у реальних системах, що вимагає додаткової роботи над підвищенням точності та адаптації моделей для практичного використання.

Проведений аналіз часу обробки запитів також показав значну різницю між моделями. Найшвидшою моделлю за середнім часом відповіді стала GPT-4-O, що забезпечує оперативну реакцію системи без значних затримок. Натомість Claude 3 Opus, хоч і демонструвала кращу точність у класифікації безпечних запитів, мала найбільший час обробки, що може бути критичним фактором у системах, орієнтованих на швидкість відповіді.

Таким чином, запропонований метод виявлення SQL-ін'єкцій із застосуванням LLM-моделей є перспективним напрямом підвищення безпеки веб-застосунків, однак для його практичного впровадження необхідно враховувати потенційні хибнопозитивні спрацьовування та оптимізувати час обробки запитів.

РОЗДІЛ 4
ОЦІНКА ТА ПЕРСПЕКТИВИ РОЗВИТКУ РОЗРОБЛЕНОГО МЕТОДУ

4.1. Оцінка ефективності та обмежень розробленого рішення

Метод виявлення SQL-ін'єкцій, що базується на використанні великих мовних моделей, продемонстрував значний потенціал у підвищенні безпеки інформаційних систем. Проведені експерименти виявили як сильні сторони цього підходу, так і низку обмежень, які необхідно враховувати для його вдосконалення та практичного застосування. Нижче наведено детальний аналіз ефективності та обмежень методу, структурований за ключовими аспектами.

Ефективність методу:

- Висока здатність до виявлення зловмисних запитів:

Однією з основних переваг методу є його здатність ефективно ідентифікувати широкий спектр зловмисних SQL-запитів. Як показали результати експериментів, усі шість протестованих моделей – GPT-4, GPT-4-O, GPT-4 Turbo, Claude 3 Haiku, Claude 3 Opus та Claude 3 Sonnet – досягли 100% точності у виявленні зловмисних запитів. Ці запити включали різноманітні типи атак, такі як тавтології (наприклад, 1=1), UNION-атаки, кодовані запити (з використанням URL-кодування або шестнадцяткового формату), а також запити з часовою затримкою (наприклад, SLEEP(5)). Такий результат свідчить про високу здатність LLM до аналізу семантичної структури запитів, що дозволяє їм розпізнавати навіть складні та замасковані атаки.

Ключовим фактором успіху є здатність моделей обробляти контекст і виявляти аномалії в структурі запитів, які не відповідають типовим шаблонам легітимних операцій. Наприклад, моделі успішно ідентифікували запити, що містять коментарі для обходу фільтрів (наприклад, -- або /* */), а також нетипові комбінації ключових слів, які можуть вказувати на атаку. Ця здатність робить метод особливо цінним у

сучасних умовах, коли зловмисники постійно вдосконалюють техніки атак, використовуючи нові методи маскування.

- Гнучкість аналізу:

На відміну від традиційних методів, які покладаються на статичні правила або сигнатурні бази, LLM забезпечують значно більшу гнучкість у виявленні загроз [16]. Статичні правила часто є неефективними проти нових або модифікованих атак, оскільки вони потребують регулярного оновлення сигнатур. Натомість LLM здатні до контекстного аналізу, що дозволяє їм виявляти аномалії навіть у тих запитах, які не відповідають відомим шаблонам атак. Наприклад, моделі можуть розпізнавати нетипові комбінації символів або семантичні конструкції, які не відповідають нормальній поведінці SQL-запитів.

Ця гнучкість також проявляється у здатності моделей адаптуватися до різних діалектів SQL (наприклад, MySQL, PostgreSQL, Oracle), що є важливим для їх використання в гетерогенних системах. Крім того, LLM можуть аналізувати запити, написані різними мовами або з використанням специфічних форматів, що робить їх універсальним інструментом для захисту баз даних у глобальному масштабі.

- Потенціал для масштабування:

Іншою перевагою методу є його потенціал для інтеграції в різноманітні системи безпеки. LLM можуть бути використані як частина комплексних рішень, таких як веб-фільтри (WAF), системи моніторингу або інструменти статичного аналізу коду. Завдяки можливості обробки великих обсягів даних, моделі можуть бути застосовані для аналізу запитів у реальному часі або для ретроспективного аналізу логів, що дозволяє виявляти атаки на ранніх етапах.

Обмеження методу:

- Високий рівень хибнопозитивних спрацьовувань:

Одним із найсерйозніших обмежень методу є схильність моделей до помилкової класифікації безпечних запитів як зловмисних [21]. Як показали результати тестування, рівень хибнопозитивних спрацьовувань варіюється залежно від моделі, але в середньому становить значний відсоток. Наприклад, складні, але

легітимні запити, що містять вкладені підзапити або нестандартні конструкції, часто помилково позначалися як потенційні атаки. У деяких випадках навіть звичайний текст, який не є SQL-запитом, був класифікований як загроза.

Ця проблема може мати серйозні наслідки для практичного застосування методу. Хибнопозитивні спрацьовування можуть призводити до блокування легітимних операцій, що негативно впливає на користувацький досвід і може викликати збої в роботі додатків. Для вирішення цього питання необхідне додаткове навчання моделей на різноманітних наборах даних, що включають як зловмисні, так і легітимні запити, а також використання технік тонкого налаштування (fine-tuning) для підвищення точності класифікації.

- Варіативність часу обробки:

Час, необхідний для аналізу запитів, суттєво варіюється залежно від моделі. Наприклад, GPT-4-O продемонструвала високу швидкість обробки, що робить її придатною для систем із вимогами до аналізу в реальному часі. Натомість Claude 3 Opus виявилася значно повільнішою, що може бути неприйнятним для систем із високою інтенсивністю запитів, таких як веб-додатки з великою кількістю користувачів. Ця варіативність вимагає ретельного вибору моделі залежно від конкретних вимог до продуктивності системи.

Крім того, час обробки може залежати від складності запиту. Складні запити, що містять велику кількість операторів або вкладених конструкцій, потребують більше часу для аналізу, що може створювати затримки в роботі системи. Для вирішення цієї проблеми може бути доцільним використання комбінованого підходу, де швидкі моделі застосовуються для первинного фільтрування, а повільніші, але точніші моделі – для глибшого аналізу підозрілих запитів.

- Обчислювальні та монетарні витрати:

Використання потужних LLM пов’язане зі значними обчислювальними ресурсами, що може бути обмежувальним фактором для організацій із обмеженим бюджетом. Наприклад, розгортання моделей на локальних серверах потребує високопродуктивного обладнання, тоді як використання хмарних API (наприклад,

OpenAI або Anthropic) пов’язане з фінансовими витратами. У контексті систем із великою кількістю запитів ці витрати можуть швидко зростати, що вимагає ретельного планування та оптимізації.

Для зменшення витрат можна розглянути використання легших моделей, таких як Claude 3 Haiku, які мають нижчі вимоги до ресурсів, але можуть поступатися за точністю. Крім того, оптимізація промптів і зменшення кількості запитів до API (наприклад, за допомогою кешування результатів) може допомогти знизити витрати без значної втрати ефективності.

- Залежність від якості промптів:

Ефективність аналізу значною мірою залежить від якості та чіткості промптів, які передаються моделі. Некоректно сформульовані промпти можуть призводити до неправильної інтерпретації запитів або зниження точності класифікації. Наприклад, якщо промпт не містить чітких інструкцій щодо розрізнення легітимних і зловмисних запитів, модель може помилково позначати безпечні запити як загрози.

Розробка оптимальних промптів є складним завданням, яке вимагає глибокого розуміння як роботи LLM, так і особливостей SQL-атак. Для підвищення ефективності необхідно проводити експерименти з різними формулюваннями промптів і використовувати техніки автоматизованого тестування для оцінки їх впливу на результати аналізу.

4.2. Перспективи подальшого розвитку та вдосконалення методу

Метод виявлення SQL-ін'єкцій на основі LLM відкриває широкі можливості для вдосконалення, що дозволить подолати поточні обмеження та розширити його практичне застосування. Подальший розвиток методу може бути зосереджений на підвищенні точності, оптимізації продуктивності, розширенні функціональності та забезпеченні кращої інтеграції з іншими системами безпеки. Нижче наведено детальний аналіз перспектив розвитку за ключовими напрямками.

Покращення точності та зниження хибнопозитивних спрацьовувань:

Спеціалізоване донавчання моделей:

Одним із головних напрямків розвитку є тонке налаштування (fine-tuning) LLM на спеціалізованих наборах даних. Ці набори повинні включати різноманітні приклади зловмисних SQL-запитів, а також великий обсяг легітимних запитів, що відображають реальні сценарії використання баз даних у різних додатках. Наприклад, донавчання на даних, що містять запити з різних діалектів SQL (MySQL, PostgreSQL, SQL Server), а також специфічні конструкції, характерні для певних веб-додатків, може значно підвищити здатність моделей розрізняти безпечні та небезпечні запити. Крім того, включення прикладів звичайного тексту, який не є SQL-запитом, допоможе зменшити помилкову класифікацію такого тексту як загрози.

Удосконалення технік створення промптів:

Розробка більш складних і контекстно орієнтованих промптів є ще одним перспективним напрямком. Використання підходів, таких як "ланцюжок думок" (Chain-of-Thought), може допомогти моделям краще структурувати процес аналізу, розбиваючи завдання на логічні етапи (наприклад, перевірка синтаксису, аналіз семантики, оцінка потенційних ризиків). Додавання до промптів додаткової інформації про контекст додатка, наприклад, тип бази даних або очікувані шаблони запитів, може сприяти точнішій класифікації [26]. Крім того, використання структурованих форматів виводу, таких як JSON, дозволить моделям надавати результати у зрозумілій та легко оброблюваній формі, що полегшить їх подальшу обробку.

Впровадження постпроцесингу:

Для додаткового зниження хибнопозитивних спрацьовувань можна застосовувати постпроцесинг результатів аналізу LLM. Наприклад, створення білих списків легітимних шаблонів запитів, характерних для конкретного додатка, дозволить відфільтровувати безпечні запити, які помилково позначені як зловмисні. Додаткові правила на основі регулярних виразів або статистичних методів можуть допомогти виявляти очевидні помилки класифікації. Такий підхід дозволить зберегти високу чутливість до атак, одночасно зменшивши вплив на легітимні операції.

Підвищення продуктивності та оптимізація витрат:

Розробка гібридних архітектур:

Для забезпечення високої швидкості обробки запитів перспективним є використання гібридних систем, де різні моделі виконують різні етапи аналізу. Наприклад, легкі та швидкі моделі, такі як Claude 3 Haiku, можуть використовуватися для первинного скринінгу запитів, відфільтровуючи очевидно безпечні або зловмисні запити. У разі виявлення підозрілих запитів до аналізу залучаються більш потужні моделі, такі як GPT-4 або Claude 3 Opus, які здатні проводити глибший семантичний аналіз. Такий підхід дозволить оптимізувати продуктивність системи, зменшивши навантаження на дорогі обчислювальні ресурси.

Використання кешування та оптимізації запитів:

Кешування результатів аналізу для ідентичних або схожих SQL-запитів є ефективним способом зниження обчислювальних витрат. Наприклад, якщо певний запит уже був проаналізований і класифікований як безпечний, його результат можна зберегти в базі даних для швидкого доступу в майбутньому. Крім того, групування схожих запитів за допомогою алгоритмів кластеризації може допомогти зменшити кількість викликів API до LLM, що особливо важливо для систем із великою кількістю запитів.

Оптимізація вибору моделей:

Вибір моделі LLM залежно від вимог до швидкості, точності та бюджету є важливим аспектом оптимізації. Наприклад, для систем із низькими вимогами до продуктивності можна використовувати менш ресурсомісткі моделі, тоді як для критичних додатків доцільно застосовувати потужніші моделі. Крім того, дослідження можливостей дистильованих моделей (які зберігають високу точність при значно менших обчислювальних вимогах) може відкрити нові перспективи для економії ресурсів.

Розширення функціональності:

Інтеграція з комплексними системами безпеки:

Для підвищення ефективності методу доцільно інтегрувати його з іншими інструментами безпеки, такими як WAF, SAST DAST. Наприклад, LLM можуть

використовуватися для аналізу запитів, які пройшли попередню фільтрацію WAF, що дозволить зосередити ресурси на потенційно небезпечних запитах. Інтеграція з SAST може допомогти виявляти вразливості в коді додатків, які сприяють SQL-ін'єкціям, тоді як DAST може надавати додатковий контекст про поведінку додатка під час реальних атак.

Впровадження адаптивного навчання:

Розробка систем адаптивного навчання, які дозволяють LLM оновлювати свої знання на основі нових даних, є перспективним напрямком [23]. Наприклад, моделі можуть автоматично навчатися на нових типах атак, виявлених у реальному часі, або на основі зворотного зв’язку від адміністраторів безпеки. Такий підхід забезпечить постійне вдосконалення ефективності методу в умовах еволюції кіберзагроз.

Аналіз контексту запитів:

Розширення можливостей моделей для аналізу контексту запитів може значно підвищити точність класифікації. Наприклад, врахування інформації про тип користувача (анонімний, авторизований, адміністратор), історію його активності або частину додатка, з якої надійшов запит, дозволить моделям краще оцінювати потенційні ризики. Це особливо важливо для виявлення цільових атак, які можуть виглядати як легітимні запити в ізольованому контексті.

Пояснюваність результатів:

Розробка інтерпретованих рішень:

Одним із ключових напрямків розвитку є вдосконалення здатності LLM надавати пояснення своїх рішень [22]. Наприклад, моделі можуть генерувати детальні звіти, які вказують, які саме елементи запиту (ключові слова, структура, аномалії) спричинили його класифікацію як зловмисного. Це допоможе адміністраторам безпеки краще розуміти природу загроз і приймати обґрунтовані рішення щодо подальших дій. Крім того, пояснення можуть бути використані для навчання персоналу або для автоматизації процесів реагування на інциденти.

Візуалізація результатів

Іншим аспектом підвищення пояснюваності є створення інструментів для візуалізації результатів аналізу. Наприклад, розробка графічних інтерфейсів, які

відображають структуру запиту, позначені підозрілі елементи та ймовірність атаки, може полегшити роботу аналітиків безпеки. Такі інструменти також можуть бути інтегровані в системи моніторингу для надання реального часу інформації про потенційні загрози.

Висновок за четвертим розділом

У результаті проведеної оцінки розробленого методу виявлення SQL-ін'єкцій на основі LLM було підтверджено його значний потенціал, але водночас виявлено низку суттєвих обмежень, що потребують уваги для практичного впровадження.

Аналіз ефективності показав, що метод демонструє високу здатність до виявлення широкого спектру зловмисних SQL-запитів, включаючи складні та замасковані атаки, що підтверджується 100% точністю виявлення у протестованих моделей. Гнучкість аналізу, яка відрізняє LLM від традиційних методів на основі сигнатур, та потенціал для масштабування є ключовими перевагами підходу.

Водночас, оцінка виявила серйозні обмеження. Найбільш значущим є високий рівень хибнопозитивних спрацьовувань, коли безпечні запити помилково класифікуються як зловмисні, що може призводити до блокування легітимних операцій. Також відзначено суттєву варіативність часу обробки запитів між різними моделями, що впливає на можливість використання методу в системах реального часу. Значні обчислювальні та фінансові витрати, пов'язані з використанням потужних LLM, та сильна залежність ефективності від якості розроблених промптів є додатковими обмежуючими факторами.

Перспективи подальшого розвитку методу зосереджені на подоланні виявлених обмежень. Ключовими напрямками вдосконалення є: підвищення точності та зниження хибнопозитивних спрацьовувань через спеціалізоване донавчання моделей (fine-tuning) та вдосконалення промптів; оптимізація продуктивності та витрат за допомогою гібридних архітектур, кешування та вибору оптимальних моделей; розширення функціональності шляхом інтеграції з іншими системами безпеки та

впровадження адаптивного навчання; а також покращення пояснюваності результатів для кращого розуміння та реагування на загрози.

Таким чином, хоча метод виявлення SQL-ін'єкцій на базі LLM є перспективним інструментом для підвищення безпеки інформаційних систем, його успішне практичне застосування вимагає подальших досліджень та розробок, спрямованих на мінімізацію хибнопозитивних спрацьовувань, оптимізацію продуктивності та витрат, а також підвищення надійності та інтерпретованості результатів.

ВИСНОВКИ

У представленій дипломній роботі було досліджено актуальну проблему виявлення вразливостей типу SQL-ін'єкцій у веб-застосунках із використанням технологій штучного інтелекту, зокрема LLM. SQL-ін'єкції залишаються однією з найпоширеніших та найнебезпечніших кібератак, що здатні призвести до витоку конфіденційних даних, несанкціонованого доступу до баз даних і повної компрометації систем. Через обмежену ефективність традиційних методів захисту перед сучасними складними атаками постає потреба у впровадженні нових, гнучкіших та адаптивних підходів.

У процесі дослідження було проведено ґрунтовний аналіз SQL-ін'єкцій: вивчено їхню сутність, класифікацію (In-band, Inferential, Out-of-band), механізми реалізації та наслідки для інформаційної безпеки. Окрему увагу приділено існуючим методам виявлення й протидії цим атакам.

Подальший акцент зроблено на потенціал штучного інтелекту, зокрема великих мовних моделей, у боротьбі з SQL-ін'єкціями. Було розглянуто здатність LLM аналізувати великі обсяги даних, виявляти аномалії та навчатися на прикладах, що робить їх ефективним інструментом у сфері кібербезпеки. На основі цього запропоновано власний метод виявлення SQL-ін'єкцій у веб-застосунках із використанням LLM, що забезпечує вищу точність та гнучкість у порівнянні з класичними рішеннями.

З метою реалізації методу створено веб-додаток, який використовує моделі GPT-4, GPT-4.o, GPT-4 Turbo, Claude-3 Sonnet, Opus та Haiku. Проведене тестування на реальних кейсах SQL-ін'єкцій засвідчило високу ефективність розробленого рішення – всі моделі продемонстрували 100% точність у виявленні зловмисних запитів. Водночас було виявлено проблему хибнопозитивних спрацьовувань під час аналізу безпечних запитів. Найшвидшою в обробці виявилася GPT-4-O, а модель

Claude 3 Opus забезпечила кращу класифікацію безпечних запитів за рахунок довшого часу обробки.

Результати роботи підтвердили високий потенціал LLM у виявленні SQL-ін'єкцій, однак також виявили низку обмежень: наявність хибнопозитивів, варіативність часу обробки, обчислювальні та фінансові витрати, а також критичну залежність від якості формування запитів. Це створює необхідність подальших досліджень, зокрема у напрямах донавчання моделей, вдосконалення промптів, впровадження постпроцесингу, оптимізації вибору моделей, створення гібридних архітектур та підвищення пояснюваності результатів.

Отже, запропонований метод виявлення SQL-ін'єкцій з використанням штучного інтелекту є перспективним напрямком підвищення кібербезпеки веб-застосунків. Він демонструє високу ефективність і практичну цінність, але водночас потребує подальшого вдосконалення для забезпечення стабільної роботи, мінімізації хибнопозитивних спрацьовувань та оптимізації ресурсних витрат. Його успішне впровадження у реальні системи можливе за умови проведення додаткових досліджень та інтеграції з комплексними рішеннями у сфері інформаційної безпеки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What is SQL Injection | SQLI Attack Example & Prevention Methods | Imperva. URL: https://www.imperva.com/learn/application-security/sql-injection-sqli/ (дата звернення: 12.05.2025).

2. SQL Injection Attack | Bright Security. URL: https://brightsec.com/blog/sql-injection-attack/ (дата звернення: 12.05.2025).

3. Error-Based SQL Injection | Bright Security. URL: https://brightsec.com/blog/error-based-sql-injection/ (дата звернення: 12.05.2025).

4. Union SQL Injection | Bright Security. URL: https://brightsec.com/blog/union-sql-injection/ (дата звернення: 12.05.2025).

5. Blind SQL Injection | Bright Security. URL: https://brightsec.com/blog/blind-sql-injection/ (дата звернення: 12.05.2025).

6. Out-of-Band SQL Injection (OOB SQLi) | Invicti. URL: https://www.invicti.com/learn/out-of-band-sql-injection-oob-sqli/ (дата звернення: 12.05.2025).

7. SQL Injection | Acunetix. URL: https://www.acunetix.com/websitesecurity/sql-injection2/ (дата звернення: 12.05.2025).

8. SQL Injection Prevention Cheat Sheet | OWASP Cheat Sheet Series. URL: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html (дата звернення: 12.05.2025).

9. Best SQL Injection Detection Tools | Zenarmor. URL: https://www.zenarmor.com/docs/network-security-tutorials/best-sql-injection-detection-tools (дата звернення: 12.05.2025).

10. A Comprehensive Comparison of All LLMs | AI Pro. URL: https://ai-pro.org/learn-ai/articles/a-comprehensive-comparison-of-all-llms/ (дата звернення: 12.05.2025).

11. LLM Comparison and Leaderboard | YourGPT.ai. URL: https://yourgpt.ai/tools/llm-comparison-and-leaderboard (дата звернення: 12.05.2025).

12. CVE Database | MITRE. URL: https://cve.mitre.org/ (дата звернення: 12.05.2025).

13. SQL Injection Cheat Sheet | OWASP Cheat Sheet Series. URL: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Cheat_Sheet.html (дата звернення: 12.05.2025).

14. Erickson, J. Hacking: The Art of Exploitation. 2nd ed. San Francisco: No Starch Press, 2008. 488 p.

15. Сілін Є. С., Кадубовський О. А. Основи кібербезпеки : навчальний посібник. Дніпро, 2023. 200 с. URL: https://ddpu.edu.ua/fmk/publications/manuals/manual_18.pdf (дата звернення: 12.05.2025).

16. Alazab, M., Shalaginov, A. AI and Machine Learning in Cybersecurity: A Review // Intelligent Automation & Soft Computing. 2023. Vol. 36, No. 2. P. 1211–1230.

17. OWASP Top 10:2021 | OWASP Foundation. URL: https://owasp.org/Top10/ (дата звернення: 12.05.2025).

18. NIST Special Publication 800-53 Rev. 5. Security and Privacy Controls for Information Systems and Organizations | National Institute of Standards and Technology. 2020. URL: https://csrc.nist.gov/publications/detail/sp/800-53/rev-5/final (дата звернення: 12.05.2025).

19. ДСТУ ISO/IEC 27001:2023. Інформаційна безпека, кібербезпека та захист конфіденційності. Системи менеджменту інформаційної безпеки. Вимоги (ISO/IEC 27001:2022, IDT). Київ : ДП «УкрНДНЦ», 2023.

20. Stuttard, D., Pinto, M. The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws. 2nd ed. Wiley, 2011. 912 p.

21. Bishop, M. Introduction to Computer Security. Addison-Wesley Professional, 2004. 768 p.

22. Горбенко І. Д., Горбенко Ю. І. Прикладна криптологія. Теорія. Практика. Застосування : монографія. Харків : Форт, 2012. 870 с.

23. Штучний інтелект: досягнення, виклики та ризики: матеріали XV Міжнародної науково-практичної конференції (Київ, 15 березня 2024 р.) / Інститут

проблем штучного інтелекту МОН України і НАН України. Київ, 2024. URL: https://www.ipai.net.ua/docs/ai_conf_15.03.2024.pdf (дата звернення: 12.05.2025).

24. Великі мовні моделі (LLM) у кібербезпеці | ITpedia. 2024. URL: https://uk.itpedia.nl/2024/05/25/large-language-models-llms-in-cyber-security/ (дата звернення: 12.05.2025).

25. Що таке великі мовні моделі (LLM) - найкращі випадки використання, набори даних, майбутнє | Shaip. URL: https://uk.shaip.com/blog/a-guide-large-language-model-llm/ (дата звернення: 12.05.2025).

26. Shostack, A. Threat Modeling: Designing for Security. Wiley, 2014. 624 p.

27. Бондар О. І., Левченко О. В. Безпека Web-додатків: актуальні проблеми та їх аналіз // Сучасні інформаційні системи. 2021. Т. 5, № 2. С. 85-91. URL: http://repository.vsau.org/getfile.php/17100.PDF (дата звернення: 12.05.2025).

28. Peltier, T. R. Information Security Risk Analysis. 3rd ed. CRC Press, 2016. 408 p.

29. Anderson, R. Security Engineering: A Guide to Building Dependable Distributed Systems. 3rd ed. Wiley, 2020. 1168 p.

30. Joint Task Force Transformation Initiative. NIST Special Publication 800-30 Revision 1. Guide for Conducting Risk Assessments | National Institute of Standards and Technology. 2012. URL: https://csrc.nist.gov/publications/detail/sp/800-30/rev-1/final (дата звернення: 12.05.2025).

31. SQL Injection Detection Using Machine Learning Techniques and Deep Learning | San José State University. URL: https://scholarworks.sjsu.edu/cgi/viewcontent.cgi?article=1649&context=etd_projects (дата звернення: 12.05.2025).

32. Deep Learning Architecture for Detecting SQL Injection Attacks Using RNN Autoencoder | MDPI Mathematics. URL: https://www.mdpi.com/2227-7390/11/15/3286 (дата звернення: 12.05.2025).

33. Securing Web Applications Against XSS and SQLi Attacks Using Hybrid Deep Learning Networks | Nature Scientific Reports. URL: https://www.nature.com/articles/s41598-023-48845-4 (дата звернення: 12.05.2025).

34. Application of Artificial Intelligence in Detecting SQL Injection Attacks | Journal of Intelligent Informatics and Virtual Systems. URL: https://joiv.org/index.php/joiv/article/download/3631/1136 (дата звернення: 12.05.2025).

35. A Deep Learning Approach for Detection of SQL Injection Attacks Using Convolutional Neural Networks | ResearchGate. URL: https://www.researchgate.net/publication/356440209_A_Deep_Learning_Approach_for_Detection_of_SQL_Injection_Attacks_Using_Convolutional_Neural_Networks (дата звернення: 12.05.2025).

36. Defending Against SQL Injection Attacks in Web Applications Using Machine Learning and Natural Language Processing | ResearchGate. URL: https://www.researchgate.net/publication/358318689_Defending_against_SQL_Injection_Attacks_in_Web_Applications_using_Machine_Learning_and_Natural_Language_Processing (дата звернення: 12.05.2025).

37. Detection and Prevention of SQL Injection Attacks and Developing Secure Web Applications | Journal of Big Data. URL: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-022-00678-0 (дата звернення: 12.05.2025).

38. Detection of SQL Injection Attack Using Machine Learning Techniques | MDPI Information. URL: https://www.mdpi.com/2624-800X/2/4/39 (дата звернення: 12.05.2025).

39. A Deep Learning Approach Based on Multi-View Consensus for SQL Injection Detection | Springer International Journal of Information Security. URL: https://link.springer.com/article/10.1007/s10207-023-00791-y (дата звернення: 12.05.2025).

40. Detecting and Mitigating SQL Injection in .NET Applications Using AI-Based Anomaly Detection | International Journal of Innovative Science and Research Technology. URL: https://www.ijisrt.com/assets/upload/files/IJISRT25MAR1676.pdf (дата звернення: 12.05.2025).

41. Deep Learning-Based Detection Technology for SQL Injection Attacks Using Enhanced TextCNN and LSTM | MDPI Applied Sciences. URL: https://www.mdpi.com/2076-3417/13/16/9466 (дата звернення: 12.05.2025).

42. Detection SQL Injection Attacks Against Web Application by Using Support Vector Machine Algorithm | AIP Conference Proceedings. URL: https://pubs.aip.org/aip/acp/article/3009/1/020008/3265140/Detection-SQL-injection-attacks-against-web (дата звернення: 12.05.2025).

43. SQL Injection Attack: Detection, Prioritization & Prevention | ScienceDirect. URL: https://www.sciencedirect.com/science/article/pii/S221421262400173X (дата звернення: 12.05.2025).

44. An Effective SQL Injection Detection Model Using LSTM for Web Applications | ScienceDirect. URL: https://www.sciencedirect.com/science/article/abs/pii/S016740482500080X (дата звернення: 12.05.2025).

45. DeepSQLi: Deep Semantic Learning for Testing SQL Injection | arXiv. URL: https://arxiv.org/abs/2005.11728 (дата звернення: 12.05.2025).

46. Enhancing SQL Injection Detection and Prevention Using Generative Models | arXiv. URL: https://arxiv.org/abs/2502.04786 (дата звернення: 12.05.2025).

47. ModSec-AdvLearn: Countering Adversarial SQL Injections with Robust Machine Learning | arXiv. URL: https://arxiv.org/abs/2308.04964 (дата звернення: 12.05.2025).

48. Подус О. Мирутеко Л. Метод виявлення SQL-ін’єкцій у веб-застосунках з використанням штучного інтелекту. VIII Міжнародна науково-практична конференція «Проблеми кібербезпеки інформаційно-комунікаційних систем» (PCSICS). 2025. С. 121-122

ДОДАТОК А

Лістинг частини коду програмного модуля для відображення UI.

```
 "use client";
import {
 Card,
 CardContent,
 CardFooter,
 CardHeader,
 CardTitle,
} from "@/components/ui/card";
import { Input } from "@/components/ui/input";
import { Button } from "@/components/ui/button";
import { Label } from "@/components/ui/label";
import SelectBasic from "./selectBasic";
import { useState } from "react";
import { useMutation } from "@tanstack/react-query";
import axios from "axios";
import LoadingAnimation from "./loading-animation";

const models = [
 {
 value: "gpt-4",
 label: "GPT-4",
 },
 {
 value: "gpt-4-o",
 label: "GPT-4-0",
 },
```

```
{
value: "gpt-4-turbo",
label: "GPT-4-turbo",
},
{
value: "claude-3-sonnet-20240229",
label: "Claude 3.0 Sonnet",
},
{
value: "claude-3-opus-20240229",
label: "Claude 3.0 Opus",
},
{
value: "claude-3-haiku-20240307",
label: "Claude 3.0 Haiku",
},
];

export default function FormCard({ revalidate }: { revalidate: VoidFunction }) {
 const [selectedModel, setSelectedModel] = useState<string | null>(null);

 const [attackText, setAttackText] = useState<string>("");

 const { mutate, isPending } = useMutation({
 mutationFn: (data: Record<string, any>) =>
 axios.post(`/api/example-route?model=${selectedModel}`, data),

 onSuccess: () => {
 revalidate();
 setAttackText("");
```

```
},
});

return (
<Card className="w-full max-w-md max-h-fit col-span-2">
<CardHeader>
<CardTitle>Форма для атаки</CardTitle>
</CardHeader>
<form
action={async () => {
if (!attackText?.length) return;

mutate({
input: attackText,
});
}}
>
<CardContent className="space-y-4">
<div className="space-y-2">
<Label>Виберіть модель</Label>
<SelectBasic
placeholder="Модель"
options={models}
setValue={setSelectedModel}
/>
</div>

<div className="space-y-2">
<Label>Текст атаки</Label>
<Input
```

```
value={attackText}
onChange={(e) => setAttackText(e.target.value)}
placeholder="..."
/>
</div>
</CardContent>
<CardFooter className="flex flex-col gap-2">
<Button
type="submit"
className="w-full"
disabled={isPending || selectedModel === null || !attackText.length}
>
Відправити
</Button>
{isPending && (
<LoadingAnimation
text={`${
models.find((model) => model.value === selectedModel)?.label
} думає`}
/>
)}
</CardFooter>
</form>
</Card>
);
}
```

ДОДАТОК Б

Лістинг коду для виконання основної серверної логіки додатку

```
import { NextRequest } from "next/server";
import { openai } from "@ai-sdk/openai";
import { generateObject } from "ai";
import * as z from "zod";
import { db } from "@/db";
import { attackTable } from "@/db/schema";
import { anthropic } from "@ai-sdk/anthropic";
import { format } from "date-fns";

const getModel = (model: string) => {
 switch (model) {
 case "gpt-4":
 return openai("gpt-4");
 case "gpt-4-0":
 return openai("gpt-4o");
 case "gpt-4-turbo":
 return openai("gpt-4-turbo");
 case "claude-3-sonnet-20240229":
 return anthropic("claude-3-sonnet-20240229");
 case "claude-3-opus-20240229":
 return anthropic("claude-3-opus-20240229");
 case "claude-3-haiku-20240307":
 return anthropic("claude-3-haiku-20240307");
 default:
 return openai("gpt-4");
 }
```

```
};

export const maxDuration = 60;
export const dynamic = "force-dynamic";

export async function POST(request: NextRequest) {
 const startTime = Date.now();

 const data = await request.json();

 const searchParams = request.nextUrl.searchParams;

 const searchParamsModel = searchParams.get("model");

 if (!searchParamsModel)
 return new Response(JSON.stringify({ error: "model not found" }), {
 status: 400,
 });

 const model = getModel(searchParamsModel);

 const prompt =
 "Ти експерт в кібербезпеці, як ти думаєш це текст є SQL ін'єкцією, яка може нести загрозу? Поясни свою відповідь. Текст атаки: " +
 data.input;

 const result = await generateObject({
 model: model,
 schema: z.object({
 isSqlInjection: z.boolean(),
```

```
explanation: z
.string()
.describe(
"Чому ти думаєш що це потенційна SQL ін'єкція яка може бути небезпечною
?"
),
}),
prompt,
});

const endTime = Date.now();
const processingTime = format(new Date(endTime - startTime), "mm:ss");

await db.insert(attackTable).values({
isAttack: result.object.isSqlInjection,
model: searchParamsModel,
explanation: result.object.explanation,
input: data.input,
prompt,
processingTime,
createdAt: new Date().toISOString(),
});

return new Response(JSON.stringify(true), {
status: 200,
});
}
```