НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

До захисту допущено
Завідувач кафедри

_____ **Дмитро ЛАНДЕ**
(підпис)

«_____»_____2024 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системи, технології та математичні методи кібербезпеки»
спеціальності 125 «Кібербезпека»

на тему: Виявлення SQL Injection в мережевому трафіку

Виконав (-ла): здобувач вищої освіти **IV** курсу, групи ФБ-06
(шифр групи)

Вісловух Владислав Володимирович
(прізвище, ім'я, по батькові) (підпис)

Керівник старший викладач каф. ІБ Козленко О. В.
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Рецензент доцент НН ФТІ, фізико-математичних наук Южакова Г.О.
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище, ім'я, по батькові) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Здобувач вищої освіти _____
(підпис)

Київ – 2024 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 125 «Кібербезпека»
Освітньо-професійна програма «Системи, технології та математичні методи кібербезпеки»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Дмитро ЛАНДЕ
(підпис)
«___»_____2024 р.

ЗАВДАННЯ
на дипломну роботу здобувачу вищої освіти

Вісловух Владислав Володимирович
(прізвище, ім'я, по батькові)

1. Тема роботи «Виявлення SQL Injection в мережевому трафіку»,
керівник роботи Козленко Олег Віталійович, старший викладач каф. ІБ,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від « 31 » Травня 2024 р. № 2251-с

2. Термін подання здобувачем вищої освіти роботи « 07 » Червень 2024 р.

3. Вихідні дані до роботи: Включають аналіз існуючих методів виявлення SQL Injection, визначення функціональних та нефункціональних вимог до системи, вибір технологічних рішень, розробку та тестування програмного забезпечення для виявлення SQL Injection у мережевому трафіку, а також оцінку ефективності та надійності системи.

4. Зміст роботи: Опис проблеми та важливості автоматизації процесу виявлення SQL Injection у мережевому трафіку, аналіз існуючих методів та інструментів виявлення SQL Injection у мережевому трафіку, функційні та нефункційні вимоги до системи виявлення SQL Injection у мережевому трафіку, вибір та обґрунтування технологічних рішень (Python, Flask, mitmproxy), інтеграція з зовнішніми API, розробка та тестування програмного забезпечення, висновки щодо ефективності та надійності системи.

5. Перелік ілюстративного матеріалу: Презентація до захисту дипломної роботи.

6. Дата видачі завдання: 29.05.2024

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка
1	Визначення напрямку ДР	29.05.2024 – 30.05.2024	Виконано
2	Визначення теми ДР	30.05.2024 – 31.05.2024	Виконано
3	Збір та аналіз літературних джерел	31.05.2024 – 01.06.2024	Виконано
4	Аналіз сучасних методів виявлення SQL Injection	01.06.2024 – 02.06.2024	Виконано
5	Вибір технологічних рішень	02.06.2024 – 03.06.2024	Виконано
6	Розробка прототипу системи виявлення SQL Injection	03.06.2024 – 04.06.2024	Виконано
7	Тестування системи виявлення SQL Injection	04.06.2024 – 05.06.2024	Виконано
8	Підготовка звіту по результатам роботи	05.06.2024 – 06.06.2024	Виконано
9	Підготовка та передзахист ДР	06.06.2024 – 14.06.2024	Виконано
10	Підготовка до захисту ДР	14.06.2024 – 21.06.2024	Виконано

Здобувач вищої освіти _____ Владислав ВІСЛОВУХ
(підпис) (**Власне ім'я, ПРІЗВИЩЕ**)

Керівник роботи _____ Олег КОЗЛЕНКО
(підпис) (**Власне ім'я, ПРІЗВИЩЕ**)

РЕФЕРАТ

Обсяг дипломної роботи 58 сторінок, 20 ілюстрацій, 7 таблиці, 2 додатки і 25 джерел літератури.

Об’єкт дослідження: мережевий трафік, що містить потенційні SQL Injection атаки.

Предмет дослідження: алгоритми та методи виявлення SQL Injection в мережевому трафіку.

Мета дослідження: розробка та валідація методу виявлення SQL Injection, який здатен ефективно ідентифікувати та класифікувати спроби ін'єкцій в мережевому трафіку.

Методи дослідження: аналіз літературних джерел для зрозуміння сучасних методів детекції, математичне та комп'ютерне моделювання для створення прототипів систем детекції, обчислювальні експерименти для перевірки ефективності розроблених методів.

Отримані результати: розроблено та апробовано алгоритми виявлення SQL Injection, які демонструють високу точність і низьке число помилкових спрацьовувань у реальних умовах мережі.

Ключові слова: SQL Injection, безпека мереж, мережевий трафік, кібербезпека, аналіз трафіку, методи виявлення.

ABSTRACT

The volume of the thesis is 58 pages, 20 illustrations, 7 tables, 2 appendices and 25 sources of literature.

Object of research: network traffic containing potential SQL Injection attacks.

Subject of research: algorithms and methods for detecting SQL Injection in network traffic.

Purpose of the study: to develop and validate a SQL Injection detection method that can effectively identify and classify injection attempts in network traffic.

Research methods: analysis of literature sources to understand modern detection methods, mathematical and computer modeling to create prototypes of detection systems, computational experiments to test the effectiveness of the developed methods.

Results: SQL Injection detection algorithms have been developed and tested, which demonstrate high accuracy and low false positives in real network conditions.

Keywords: SQL Injection, network security, network traffic, cybersecurity, traffic analysis, detection methods.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

DBMS – Database Management System

GUI – Graphical User Interface

HTTP – HyperText Transfer Protocol

HTTPS – HyperText Transfer Protocol Secure

IP – Internet Protocol

SQL – Structured Query Language

SQLi – SQL Injection

SysML – Systems Modeling Language

UDP – User Datagram Protocol

UML – Unified Modeling Language

WAF – Web Application Firewall

ВСТУП

Інформаційна безпека мережевих систем є ключовою умовою для стабільності сучасного інформаційного суспільства. Особливу увагу заслуговує захист вебдодатків від SQL Injection атак, які можуть призвести до втрати конфіденційної інформації, а також порушення нормальної роботи систем. Зі зростанням кількості даних, які обробляються через Інтернет, зростає й ризик таких атак.

Тому важливим завданням є розробка ефективних методів виявлення SQL Injection, які б дозволили мінімізувати потенційні ризики для мережевих ресурсів. Робота спрямована на аналіз існуючих методів детекції і створення нових, більш досконалих технік для ідентифікації та протидії SQL Injection.

Актуальність роботи: Висока частота інцидентів з SQL Injection та значний вплив таких атак на безпеку інформаційних систем.

Мета дослідження: Розробка та валідація ефективного методу виявлення SQL Injection у мережевому трафіку.

Мета досягається шляхом вирішення наступних задач:

1. Аналіз літературних джерел та існуючих методів виявлення SQL Injection.
2. Створення структурної моделі та реалізація прототипу системи детекції.
3. Проведення обчислювальних експериментів для тестування і валідації розроблених методів.

Об’єкт дослідження: Мережевий трафік, що містить потенційні SQL Injection атаки.

Предмет дослідження: Методи виявлення SQL Injection у мережевому трафіку.

Методи дослідження: Аналіз літературних джерел, математичне та комп'ютерне моделювання, проведення обчислювальних експериментів.

Новизна одержаних результатів: Розробка нових методів детекції SQL Injection, які ефективно працюють у реальних мережевих умовах.

Практичне значення одержаних результатів: Розроблені методи можуть бути впроваджені у системи моніторингу та безпеки для покращення їх здатності виявляти та протидіяти SQL Injection.

Публікації: Результати дослідження опубліковані на згаданій конференції.

1 АНАЛІЗ СУЧАСНИХ МЕТОДІВ ВИЯВЛЕННЯ SQL INJECTION

1.1 Огляд і класифікація існуючих методів виявлення SQL Injection

SQLi – це тип кібератаки на вебдодатки, який дозволяє зловмисникам виконувати несанкціоновані SQL-запити до бази даних вебдодатка. Цей вид атаки використовує вразливості в програмному коді, що взаємодіє з базою даних, зазвичай через недостатню валідацію або екранування вхідних даних. SQL Injection може призвести до витоку конфіденційної інформації, модифікації даних або навіть повного контролю над базою даних [1].

Виявлення SQL Injection є важливою складовою забезпечення безпеки вебдодатків. SQL Injection (SQLi) атаки дозволяють зловмисникам маніпулювати SQL-запитами через введення зловмисного коду, що може призвести до несанкціонованого доступу до даних та їх модифікації. Для ефективної боротьби з цим видом атак існує кілька основних методів, кожен з яких має свої особливості, переваги та недоліки, наприклад [3]:

1. Ручна перевірка вразливостей до SQL Injection включає введення спеціально сформованих рядків у поля введення на вебсайті та спостереження за результатами SQL-запитів. Цей метод вимагає ретельного аналізу та уважності, оскільки помилки можуть призвести до пропущення вразливостей. Хоча ручна перевірка є трудомісткою, вона надає глибоке розуміння структури бази даних та можливих векторів атак.

2. Автоматизовані сканери вразливостей є потужними інструментами для виявлення SQL Injection. Вони дозволяють автоматизувати процес перевірки, що значно зменшує витрати часу та зусиль. Деякі з найпопулярніших сканерів включають:

– Acunetix є потужним сканером з широким функціоналом для виявлення SQL Injection та інших вразливостей. Має високу вартість ліцензії та вимагає значних ресурсів для запуску.

– Burp Suite має набір інструментів для тестування на проникнення, який дозволяє вручну змінювати HTTP-запити та аналізувати взаємодію з базою даних. Потребує досвіду для оптимального використання.

– Nessus є широко використовуваним сканером, що підтримує різні платформи та системи. Потребує налаштування для точного сканування.

– OpenVAS – це відкритий код та безкоштовна ліцензія. Гнучкий та налаштовуваний, але менш швидкий та надійний порівняно з комерційними рішеннями.

– SQLMap – це спеціалізований інструмент для виявлення та експлуатації SQL Injection. Може давати багато фальшиво-позитивних результатів.

– OWASP ZAP включає відкритий код та безкоштовну ліцензію. Широкий спектр функцій для аналізу безпеки, але менш надійний порівняно з комерційними сканерами [2].

– Засіб Nikto простий у використанні та налаштуванні. Швидкий, але менш точний порівняно з іншими сканерами.

3. Методи на основі підписів використовують бази даних з сигнатурами відомих атак. Інструменти моніторингу аналізують вхідні дані на наявність відповідностей зі збереженими сигнатурами. Цей метод ефективний проти відомих атак, але може не розпізнавати нові або модифіковані типи атак.

4. Аналіз аномалій базується на виявленні відхилень від звичайної поведінки користувачів або систем. Системи виявлення аномалій використовують алгоритми машинного навчання для аналізу патернів запитів до бази даних та виявлення нетипових, потенційно зловмисних дій. Цей підхід може виявити нові типи атак, але вимагає значних обчислювальних ресурсів та високої якості навчальних даних.

5. Комбінація ручної перевірки та автоматизованих сканерів забезпечує більш точну та глибоку оцінку безпеки вебдодатків. Автоматизовані інструменти допомагають швидко виявити очевидні вразливості, тоді як ручний аналіз дозволяє ідентифікувати складніші та менш очевидні проблеми.

Тобто, кожен метод виявлення SQL Injection має свої особливості та сфери застосування. Ефективна стратегія безпеки повинна включати комбінацію різних методів для забезпечення максимального захисту вебдодатків від SQL Injection атак.

SQL Injection (SQLi) атаки можуть бути класифіковані за різними критеріями, включаючи метод впровадження коду та ціль атаки. Основні види SQLi атак включають [4-6]:

– Error-based SQL Injection використовує повідомлення про помилки бази даних для отримання інформації про її структуру.

– Union-based SQL Injection застосовує оператор SQL UNION для об'єднання результатів кількох запитів та отримання додаткових даних.

– Boolean-based Blind SQL Injection надсилає запити, що повертають істинні або хибні відповіді, визначаючи інформацію за поведінкою вебдодатку. Time-based Blind SQL Injection використовує затримки виконання запитів, щоб визначити істинність умов.

– Out-of-band Data Retrieval застосовує альтернативні канали зв'язку, такі як DNS або HTTP, для отримання даних з бази даних. Second-order Attack впроваджує шкідливий код, який виконується під час наступної операції, коли зловмисник вже не взаємодіє з системою.

– Persistent Injection зберігає шкідливий SQL-код у базі даних, який виконується щоразу при зверненні до відповідних даних, що може призвести до багаторазового використання вразливості.

Наслідки SQL-ін'єкцій можуть бути серйозними та включати [8]:

1. Отримання несанкціонованого доступу до бази даних означає, що зловмисник може читати, змінювати або видаляти дані, що призводить до

розкриття конфіденційної інформації, такої як особисті дані користувачів, паролі та фінансова інформація.

2. Пошкодження бази даних: Шкідливі операції, такі як видалення або зміна таблиць, можуть зробити дані недоступними або пошкодити їх.

3. Виконання вторгнення SQL-ін'єкція може бути використана для віддаленого виконання коду, включення зловмисного програмного забезпечення або вторгнення в інші системи, які взаємодіють з базою даних.

4. Втрата довіри користувачів: Розкриття даних через SQL-ін'єкцію може призвести до втрати довіри до ресурсу та його власника.

Отже, вразливість до SQL-ін'єкцій може мати серйозні наслідки для безпеки та цілісності webресурсу та бази даних, тому важливо приділити належну увагу безпекі.

1.2 Порівняльний аналіз ефективності методів

Ефективність методів виявлення SQL Injection можна оцінювати за кількома критеріями: точність, швидкість виявлення, зручність використання, гнучкість налаштування та ресурсні вимоги. Автоматизовані сканери вразливостей є потужними інструментами для виявлення SQL Injection та інших безпекових проблем у вебдодатках. Різні сканери мають свої переваги та недоліки, що впливають на їхню ефективність у різних умовах. Нижче (таблиця 1.1) наведено порівняльний аналіз ефективності популярних автоматизованих сканерів вразливостей, таких як Acunetix, Burp Suite, Nessus, OpenVAS, SQLMap, OWASP ZAP та Nikto, виявлення SQL Injection на основі зазначених критеріїв [5, 11, 3].

Таблиця 1.1 – Порівняльний аналіз автоматизованих сканерів вразливостей

Автоматизований сканер	Переваги	Недоліки
Acunetix	Потужність та широкий функціонал. Добре визначає вразливості в SQL-ін'єкціях.	Вартість ліцензії для повної функціональності. Вимагає значних ресурсів для запуску.
Burp Suite	Гнучкість та можливість вручну змінювати запити. Інтеграція з іншими інструментами.	Вимагає досвіду для оптимального використання.
Nessus	Виявлення широкого спектру вразливостей. Підтримка різних платформ і систем.	Потребує налаштування для точного сканування.
OpenVAS	Відкритий код та безкоштовна ліцензія. Гнучкість та можливість налаштування.	Менша швидкість та надійність порівняно з комерційними рішеннями.
SQLMap	Спеціалізований для виявлення SQL-ін'єкцій. Підтримка різних типів SQL-ін'єкцій.	Може давати багато фальшиво-позитивних результатів.
OWASP ZAP	Відкритий код та безкоштовна ліцензія. Широкий спектр функцій для аналізу безпеки.	Менша швидкість та надійність порівняно з комерційними рішеннями.
Nikto	Простий у використанні та налаштуванні. Швидкість сканування.	Менша точність порівняно з іншими сканерами.

Для нашої дипломної роботи ми обрали Acunetix, Burp Suite та Nessus як найкращі інструменти для виявлення SQL Injection. Це рішення було прийнято на основі їх високої точності, гнучкості та здатності виявляти широкий спектр вразливостей [9].

Acunetix є одним з найпотужніших інструментів для виявлення вразливостей у вебдодатках. Він добре визначає SQL Injection завдяки великій базі сигнатур та різноманітним методам аналізу, а його інтерфейс користувача зображено на рисунку 1.1. Acunetix забезпечує високу точність і швидкість виявлення, що робить його ідеальним для комплексного аудиту безпеки. Проте, висока вартість ліцензії та значні вимоги до ресурсів можуть бути обмеженням для деяких організацій.

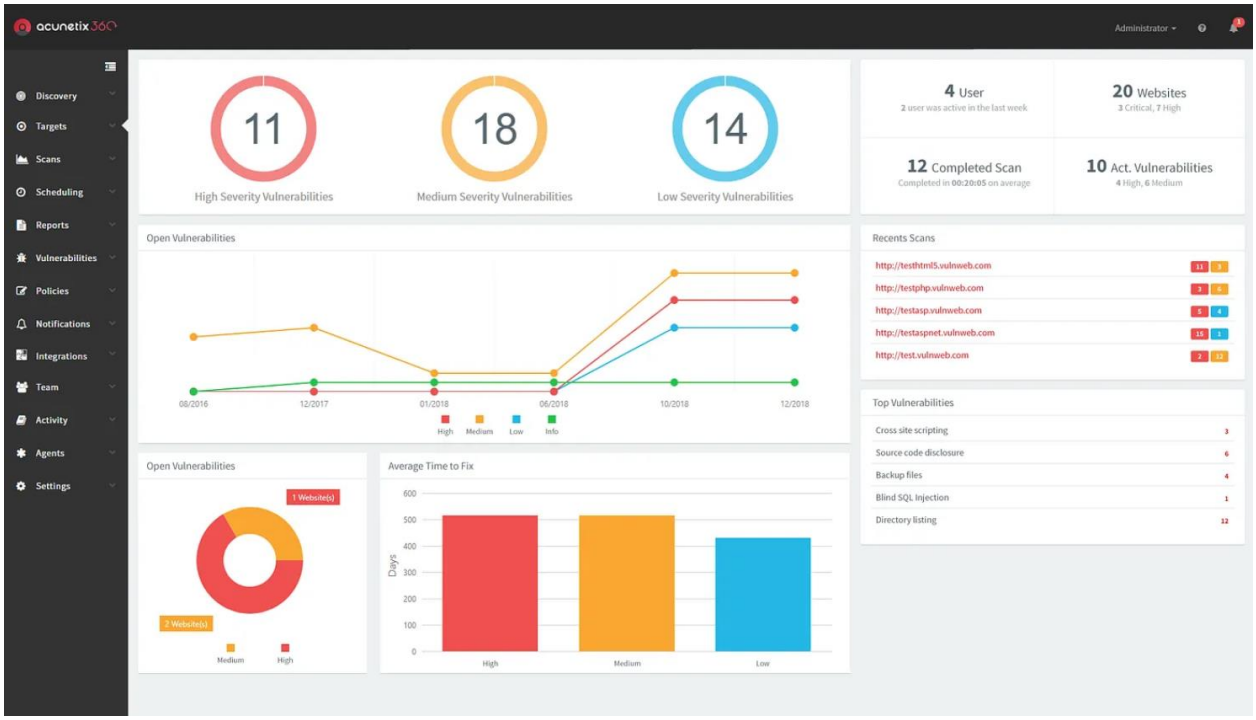


Рисунок 1.1 – Інтерфейс користувача Acunetix

Acunetix на відміну від Burp Suite пропонує надзвичайну гнучкість та можливість вручну змінювати запити, що дозволяє проводити глибокий аналіз трафіку. Цей інструмент ідеальний для досвідчених користувачів, які можуть налаштовувати запити для виявлення найменших вразливостей. Інтеграція з іншими інструментами і можливість модифікації запитів роблять його універсальним рішенням для тестування безпеки. Однак, для ефективного використання потрібен високий рівень експертних знань. На рисунку 1.2 відображено дошку оголошень Burp Suite [13].

Рисунок 1.2 – Дошка оголошень Burp Suite

Nessus відомий своєю здатністю виявляти широкий спектр вразливостей, включаючи SQL Injection. Він підтримує різні платформи та системи, що робить його універсальним інструментом для різних середовищ. Nessus демонструє високу швидкість і точність виявлення, але потребує налаштування для точного сканування, що може займати додатковий час і зусилля. Інтерфейс користувача зображено на рисунку 1.3.

Основними перевагами Nessus є його потужні можливості для автоматизованого сканування, здатність до глибокого аналізу різних типів вразливостей, а також можливість генерувати докладні звіти про виявлені проблеми. Це робить Nessus корисним інструментом для організацій, що прагнуть забезпечити високий рівень безпеки своїх інформаційних систем. Він дозволяє інтеграцію з іншими інструментами та системами безпеки, що робить його ще більш гнучким і універсальним [12].

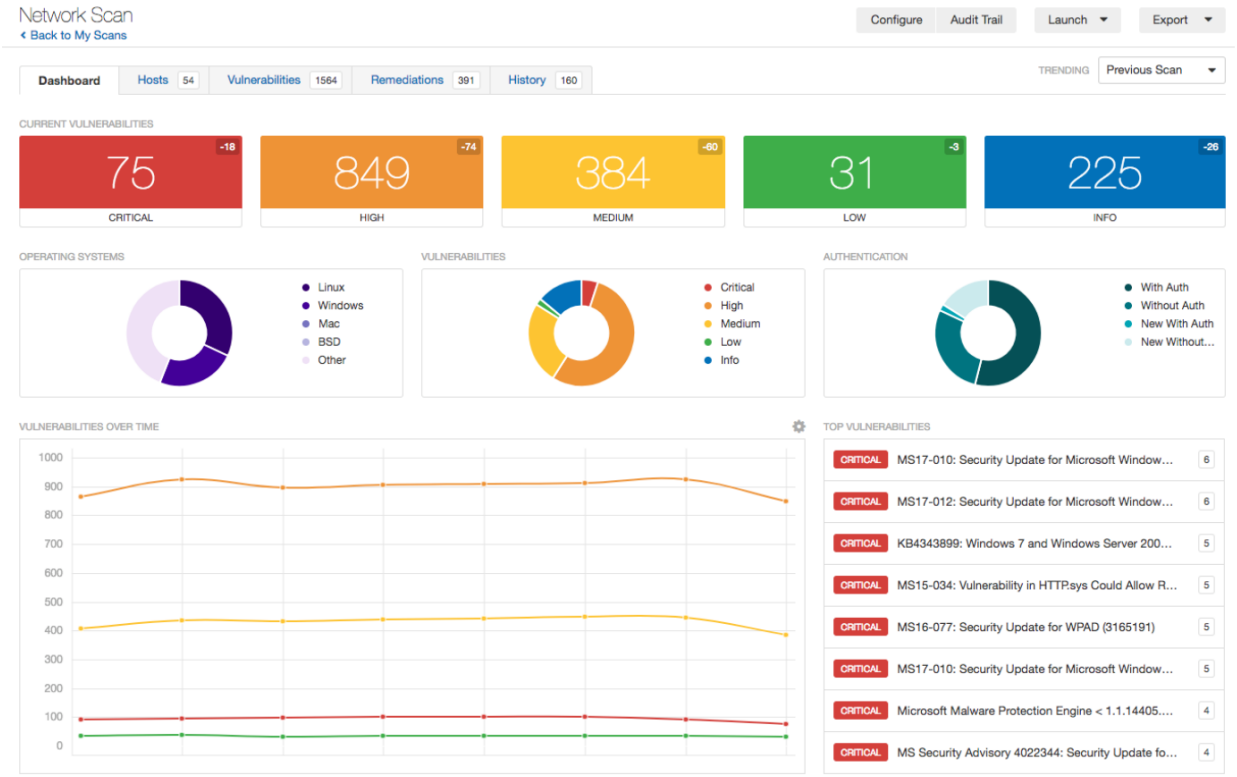


Рисунок 1.3 – Інтерфейс користувача Nessus

Кожен з розглянутих засобів має свої переваги і недоліки. Вибір конкретного інструменту залежить від специфічних вимог проекту, наявних ресурсів та рівня експертності користувача. Комбінація кількох методів може забезпечити найкращі результати, дозволяючи отримати повну картину безпеки вебдодатків і своєчасно виявити потенційні вразливості [14].

Ці інструменти були обрані завдяки їхній високій ефективності у виявленні SQL Injection та інших вразливостей. Вони будуть служити основою для нашої роботи, допомагаючи вдосконалити методи виявлення SQL Injection у мережевому трафіку. За допомогою Acunetix, Burp Suite та Nessus ми зможемо розробити та протестувати ефективні стратегії виявлення та протидії SQL Injection, що є ключовим завданням нашої дипломної роботи. Ці інструменти забезпечать надійний захист вебдодатків від потенційних атак, що значно підвищить рівень безпеки мережевих систем.

1.3 Вибір оптимальних методів для реалізації

Для ефективного виявлення SQL Injection у мережевому трафіку було обрано комбінацію кількох передових методів, що забезпечують високу точність і надійність. Ці методи включають статичний аналіз коду, динамічний аналіз, методи на основі сигнатур, аналіз аномалій, параметризацію запитів та контроль введення [4].

Статичний аналіз коду дозволяє виявляти потенційні вразливості на етапі розробки, аналізуючи вихідний код додатків. Це забезпечує високий рівень безпеки ще до запуску додатку. Динамічний аналіз або тестування на проникнення перевіряє програму в реальних умовах, виявляючи вразливості під час виконання, що може залишатися непоміченим під час статичного аналізу.

Методи на основі сигнатур використовують бази даних з сигнатурами відомих атак для швидкого виявлення відомих вразливостей. Цей метод забезпечує швидкість і ефективність у виявленні відомих загроз. Аналіз аномалій базується на виявленні відхилень від нормальної поведінки системи або користувачів, використовуючи алгоритми машинного навчання для виявлення нетипових дій, що можуть вказувати на SQL Injection.

Параметризація запитів є ефективним способом запобігання SQL Injection шляхом відокремлення даних від коду, що зменшує можливість ін'єкцій. Контроль введення включає валідацію та очищення вхідних даних, що знижує ризик введення зловмисного коду шляхом перевірки відповідності очікуваному формату та застосування обмежень [21].

Поєднання цих методів дозволяє досягти високого рівня безпеки, забезпечуючи надійний захист від SQL Injection у мережевому трафіку. Використання передових методів аналізу, таких як статичний та динамічний аналіз, методи на основі сигнатур та аналіз аномалій, дозволяє виявляти як відомі, так і нові типи атак. Параметризація запитів та контроль введення допомагають запобігати ін'єкціям на рівні коду. Це інтегроване рішення сприяє розробці ефективних стратегій виявлення та протидії SQL Injection, підвищуючи загальний

рівень безпеки мережевих систем та захищаючи дані від несанкціонованого доступу.

Важливо зазначити, що для досягнення найкращих результатів ці методи повинні використовуватися в комбінації з регулярними оновленнями та налаштуваннями системи безпеки. Постійний моніторинг та аналіз мережевого трафіку, а також навчання персоналу, відповідального за безпеку, є критично важливими для підтримання високого рівня захисту від SQL Injection. Таким чином, інтеграція цих методів у загальну стратегію кібербезпеки дозволяє не тільки ефективно виявляти та запобігати SQL Injection, але й адаптуватися до нових викликів та загроз у сфері інформаційної безпеки.

На основі обраних методів було визначено вимоги для виявлення SQL Injection у мережевому трафіку. Статичний та динамічний аналіз забезпечать ретельну перевірку як вихідного коду, так і виконуваних запитів, що дозволить виявляти потенційні вразливості на різних етапах. Методи на основі сигнатур та аналіз аномалій дозволять швидко і точно ідентифікувати як відомі, так і нові типи атак. Параметризація запитів і контроль введення забезпечать додатковий рівень захисту, запобігаючи можливим ін'єкціям на рівні коду. Ці вимоги допоможуть створити комплексну систему, здатну ефективно виявляти та протидіяти SQL Injection у реальному часі [15].

Специфікація вимог встановлює стандарти і критерії, за якими буде оцінюватись якість і виконання системи, а також служить основою для подальшого проєктування, розроблення і тестування програмного забезпечення. Вона грає важливу роль у забезпеченні взаєморозуміння між замовником і розробниками, а також у визначенні успішності та придатності програмного забезпечення для використання.

Враховуючи переваги та недоліки зіставлених програмних засобів, було створено вимоги до виявлення SQL Injection в мережевому трафіку. Вимоги до нашої системи представлені в графічній нотації SysML, що зображені на рисунку 1.4.

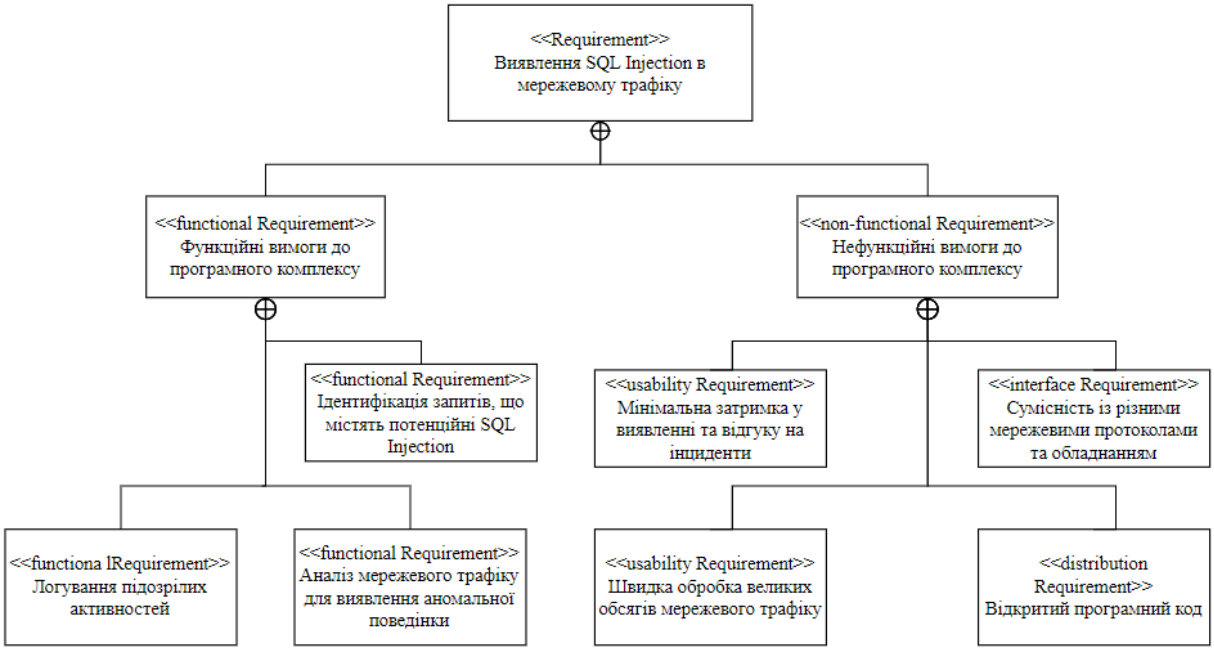


Рисунок 1.4 – Специфікація вимог

Для створення ефективної системи виявлення SQL Injection у мережевому трафіку були визначені комплексні вимоги. Функціональні вимоги включають ідентифікацію запитів, що містять потенційні SQL Injection, аналіз мережевого трафіку для виявлення аномальної поведінки та логування підозрілих активностей. Нефункціональні вимоги передбачають мінімальну затримку у виявленні та відгуку на інциденти, швидку обробку великих обсягів мережевого трафіку та сумісність із різними мережевими протоколами та обладнанням. Вимоги до зручності використання включають простоту інтеграції з існуючими системами безпеки та наявність інтуїтивно зрозумілого користувацького інтерфейсу для моніторингу та управління [12, 15].

Додатково, система повинна мати відкритий програмний код, що дозволить проводити перевірку та модифікацію. Ці вимоги допоможуть створити комплексну систему, здатну ефективно виявляти та протидіяти SQL Injection у реальному часі, забезпечуючи надійний захист вебдодатків та покращуючи загальний рівень безпеки мережевих систем.

Висновки до розділу 1

У першому розділі ми здійснили огляд та класифікацію існуючих методів виявлення SQL Injection. Було розглянуто основні підходи, такі як статичний аналіз коду, динамічний аналіз, методи на основі сигнатур та аналіз аномалій, а також методи параметризації запитів та контролю введення. Кожен з цих методів має свої переваги та недоліки, але їх комбінація забезпечує високу точність і надійність у виявленні SQL Injection.

Обрані методи дозволяють не тільки ефективно виявляти існуючі вразливості, але й забезпечують надійний захист від нових типів атак. Використання передових підходів, таких як машинне навчання для аналізу аномалій та бази даних сигнатур для швидкого виявлення відомих загроз, робить систему більш адаптивною і стійкою до різних видів SQL Injection.

Сформульовано функційні та нефункційні вимоги виявлення SQL Injection в мережевому трафіку. Для їхнього представлення використано діаграму вимог у графічній нотації SysML.

Реалізація цих методів у нашій системі виявлення SQL Injection у мережевому трафіку дає нам можливість значно підвищити рівень безпеки вебдодатків. Це дозволяє забезпечити не тільки своєчасне виявлення вразливостей, але й запобігання їх експлуатації, що є критично важливим для захисту конфіденційних даних та забезпечення стабільної роботи систем.

Отже, проведений аналіз методів виявлення SQL Injection створює міцну основу для розробки та вдосконалення системи захисту від SQL Injection у мережевому трафіку, підвищуючи загальний рівень кібербезпеки.

2 СТРУКТУРНА МОДЕЛЬ СИСТЕМИ ВИЯВЛЕННЯ SQL INJECTION

2.1 Варіанти використання системи виявлення

Для проектування варіантів використання необхідно визначити дійових осіб – акторів. Оскільки програмне забезпечення виявлення вторгнень ставить на меті дати можливість користувачеві відслідковувати та виявляти SQL-ін’єкції в мережевому трафіку, єдиним учасником системи є мережевий адміністратор (рисунок 2.1).

Побудова програмного забезпечення для виявлення SQL Injection у мережевому трафіку без застосування UML є складним завданням. Використання графічної мови для опису програмного забезпечення та його складників є необхідним для чіткого та наочного представлення [16].

Уніфікована мова моделювання (UML) є стандартизованою мовою, що використовується для візуалізації, розробки та документування програмного забезпечення. UML надає нотації для опису структури, поведінки, взаємодії та інших аспектів розробки, забезпечуючи уніфікований підхід до обміну інформацією між розробниками.

Діаграми у графічній нотації UML використовуються для моделювання процесів розробки, реалізації та тестування програмного забезпечення. Вони слугують засобом комунікації між замовниками, розробниками та іншими учасниками процесу, що підкреслює важливість чіткого представлення інформації на них [17].

У нашій роботі, застосування UML дозволить точно моделювати різні аспекти системи виявлення SQL Injection, полегшуючи розуміння і обговорення серед команди розробників та інших зацікавлених сторін.

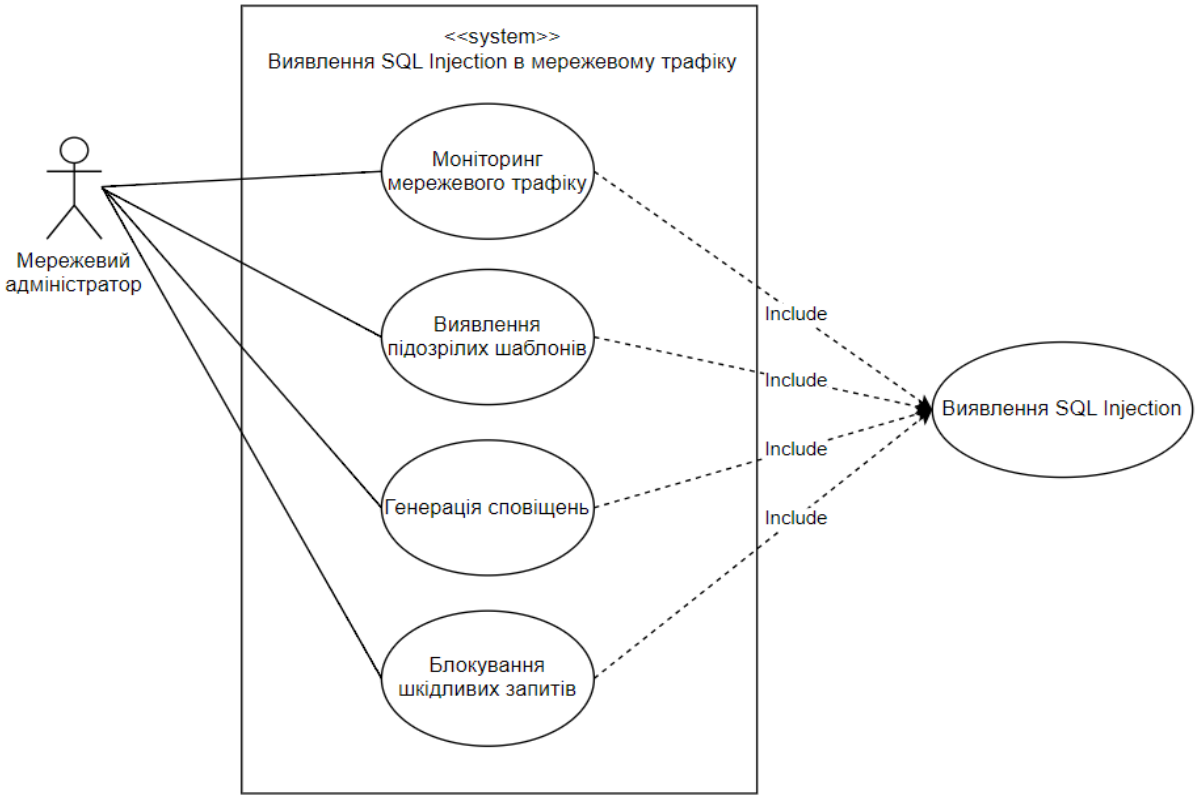


Рисунок 2.1 – Діаграма варіантів використання

На основі побудованої діаграми варіантів використання проведено специфікацію варіантів використання. Специфікацію варіантів використання наведено в табл. 2.1–2.4.

Таблиця 2.1 – Варіант використання «Моніторинг мережевого трафіку»

Характеристика	Значення
Контекст використання	Моніторинг активності мережі
Дійові особи	Мережевий адміністратор
Передумова	Система виявлення SQL Injection активна
Тригер	Запуск моніторингу мережевого трафіку
Сценарій	1. Запуск моніторингу. 2. Збір та аналіз мережевого трафіку.
Постумова	Якщо моніторинг виконано успішно, система фіксує результати. Інакше повідомляє про помилку.

Таблиця 2.2 – Варіант використання «Виявлення підозрілих шаблонів»

Характеристика	Значення
Контекст використання	Аналіз мережевого трафіку
Дійові особи	Мережевий адміністратор
Передумова	Система виявлення SQL Injection активна
Тригер	Виявлення підозрілого шаблону у трафіку
Сценарій	1. Сканування трафіку. 2. Виявлення шаблонів, що відповідають SQL Injection.
Постумова	Якщо виявлено підозрілий шаблон, система генерує сповіщення. Інакше процес продовжується.

Таблиця 2.3 – Варіант використання «Генерація сповіщень»

Характеристика	Значення
Контекст використання	Реагування на інциденти
Дійові особи	Мережевий адміністратор
Передумова	Виявлення підозрілого шаблону у трафіку
Тригер	Виявлення підозрілої активності
Сценарій	1. Формування сповіщення про інцидент. 2. Надсилання сповіщення адміністратору.
Постумова	Якщо сповіщення надіслано успішно, адміністратор отримує інформацію про інцидент. Інакше генерується повідомлення про помилку.

Таблиця 2.4 – Варіант використання «Блокування шкідливих запитів»

Характеристика	Значення
Контекст використання	Захист мережі
Дійові особи	Мережевий адміністратор
Передумова	Виявлення шкідливого запиту
Тригер	Виявлення SQL Injection у запиті
Сценарій	1. Ідентифікація шкідливого запиту. 2. Блокування запиту для запобігання атаці.
Постумова	Якщо запит успішно заблоковано, система продовжує моніторинг. Інакше генерується повідомлення про помилку.

Розглянуті варіанти використання системи виявлення SQL Injection демонструють основні функціональні можливості, які необхідні для забезпечення надійного захисту мережевих систем. Включення таких функцій, як моніторинг

мережевого трафіку, виявлення підозрілих шаблонів, генерація сповіщень та блокування шкідливих запитів, забезпечує комплексний підхід до виявлення та протидії SQL Injection. Ці функції дозволяють забезпечити ефективне реагування на інциденти безпеки, підвищуючи загальний рівень захисту інформаційних систем.

2.2 Логічна структура системи виявлення

Діаграми діяльності є потужним інструментом для моделювання поведінки системи виявлення SQL Injection. Вони представляють послідовність дій, що відбуваються у системі, дозволяючи візуалізувати процеси з урахуванням встановленої черговості. Основна мета побудови діаграм діяльності у графічній нотації UML — моделювання послідовності дій у системі від моменту початку до завершення певної функції або процесу [19].

Логічна структура відображає взаємодію та поведінку компонентів системи виявлення SQL Injection у процесі їхньої роботи. Вона описує послідовність подій, операцій та змін стану, що відбуваються у системі. Така структура дозволяє моделювати передавання даних, повідомлень, виклики функцій та взаємодію між компонентами системи. Використовуючи діаграми послідовностей, станів і діяльності, можна візуалізувати та аналізувати поведінку системи.

Діаграми діяльності використовуються для аналізу та розробки системи, а також для представлення алгоритмів. Вони деталізують процеси та допомагають виявляти можливі проблеми або покращення. Діаграми складаються з елементів, таких як дії, розгалуження, з’єднувачі, рішення та умови. Вони можуть бути простими або складними, з альтернативними шляхами виконання, що дозволяє детально відображати логіку роботи системи.

Застосування логічної структури допомагає зрозуміти, як система реагує на вхідні дані, як взаємодіють її компоненти та як змінюється її стан з часом. Це дозволяє виявляти можливі проблеми, помилки або неочікувану поведінку

системи, а також вносити необхідні корективи для покращення її функціонування. У динамічному аспекті це дозволяє більш глибоко розібратися у внутрішніх механізмах системи та забезпечити її ефективне та стабільне функціонування.

Загалом, діаграми діяльності у графічній нотації UML є потужним інструментом для моделювання та візуалізації послідовності дій у системі виявлення SQL Injection, дозволяючи зрозуміти, аналізувати та вдосконалювати її поведінку [18].

Динамічна логічна структура виявлення SQL Injection в мережевому трафіку у графічній нотації UML представлена на рис. 2.2.

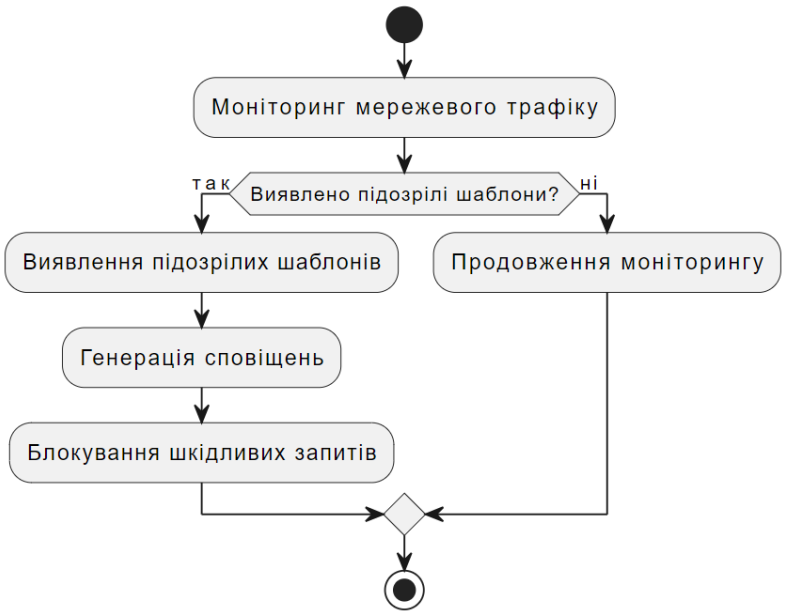


Рисунок 2.2 – Діаграма діяльності

На діаграмі діяльностей використовуються такі елементи [20-22]:

– Вхідні та вихідні дані представляються у вигляді вхідного мережевого трафіку, який аналізується системою.

– Конкретні дії включають моніторинг мережевого трафіку, виявлення підозрілих шаблонів і генерацію сповіщень.

– Розгалуження вказує на рішення про виявлення підозрілих шаблонів у трафіку. Якщо такі шаблони виявлено, система проводить подальший аналіз і

блокує шкідливі запити. Якщо підозрілих шаблонів не виявлено, система продовжує моніторинг.

– З’єднання різних гілок виконання забезпечує повернення до початкового стану моніторингу після завершення блокування шкідливих запитів.

Діаграма ілюструє послідовність дій, що забезпечують комплексний підхід до виявлення і протидії SQL Injection.

Логічна структура системи виявлення SQL Injection може також бути представлена у вигляді діаграми класів (Додаток А), що дозволяє візуалізувати основні компоненти та їх взаємодію. Така діаграма допомагає зрозуміти, як саме працює система, які класи використовуються, їх методи та атрибути, а також які зв'язки існують між ними.

Наприклад, на діаграмі класів ми можемо побачити основні класи, такі як SQLInjectionChecker, який перевіряє запити на наявність SQL ін'єкцій, App, що керує роботою додатка, HTTPFlow, який зберігає запити і відповіді, Request та Response, що відповідають за деталі HTTP-запитів та відповідей, User, який містить дані користувача, і Database, який керує взаємодією з базою даних [9].

Це дозволяє розробникам легко зрозуміти архітектуру системи, її функціональність та взаємодію між різними її частинами, що сприяє ефективному розробленню та тестуванню.

2.3 Фізична структура системи виявлення

2.3.1 Компоненти системи та зв'язки між ними

Після визначення варіантів використання та логічної структури системи, наступним кроком є проектування компонентів. Це необхідно для визначення функційності компонентів та їх зв'язків з іншими елементами у межах фізичної структури. Фізична структура описує організацію і розташування компонентів

програмного забезпечення на фізичних пристроях або середовищах виконання, показуючи, як компоненти взаємодіють і працюють у реальному середовищі.

Основні елементи фізичної структури системи включають [13, 21]:

– Апаратна архітектура описує фізичні пристрої, на яких розгорнуто програмне забезпечення.

– Розташування компонентів показує фізичне розміщення модулів і їх взаємодію.

– Мережева архітектура описує мережеві з'єднання та протоколи, що використовуються для обміну даними між компонентами.

– Розподілене програмне забезпечення включає сервіси та програми, розміщені на різних фізичних пристроях.

– Середовища виконання вказує на операційні системи, платформи та середовища, у яких виконується програмне забезпечення.

Фізична структура є важливою для розуміння та планування розгортання, масштабування та управління системою в реальному середовищі. Вона впливає на продуктивність, доступність та забезпечення безпеки системи.

Створення фізичної структури системи виявлення SQL Injection є важливим етапом її розробки. Діаграми компонентів у графічній нотації UML допомагають відобразити фізичну структуру, показуючи компоненти у вигляді блоків або прямокутників з назвами. Зв'язки між компонентами відображаються за допомогою стрілок або ліній, вказуючи напрямок взаємодії.

Використання діаграм компонентів дозволяє виділити окремі елементи системи, такі як модулі, бібліотеки, сервіси або фізичні пристрої, і показати їхні залежності та взаємодію. Це допомагає розробникам аналізувати розподіл функцій і відповідальностей між компонентами, оцінювати працездатність системи та виявляти потенційні проблеми [25].

Діаграми компонентів часто використовуються на етапі розробки програмного забезпечення або при його документуванні. Вони допомагають команді розробників, архітекторів та зацікавленим сторонам зрозуміти структуру системи, виявити потенційні проблеми та покращити якість розробки.

На діаграмі компонентів (рис. 2.3) відображено взаємозв'язок між компонентами на прикладі виявлення SQL Injection.

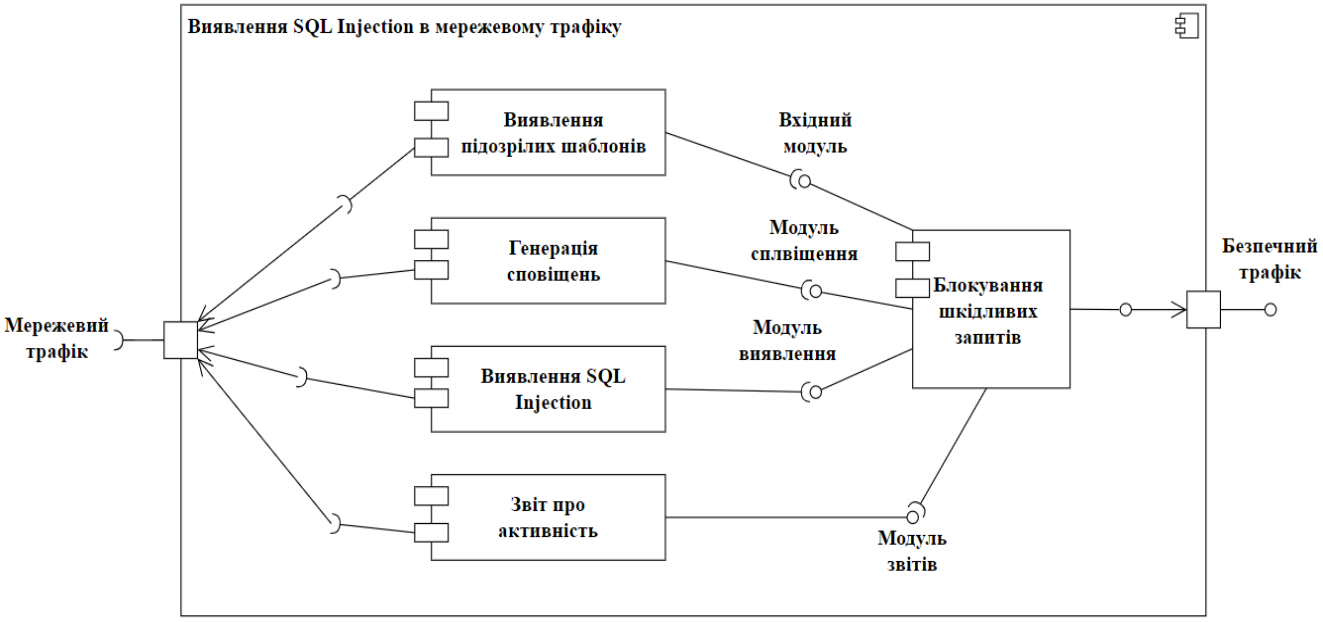


Рисунок 2.3 – Фізична структура виявлення SQL Injection в мережевому трафіку. Компоненти та звязки між ними

Діаграма відображає фізичну структуру системи виявлення SQL Injection у мережевому трафіку, показуючи, як різні компоненти системи розташовані та взаємодіють між собою в реальному середовищі.

Система починає роботу з отримання мережевого трафіку, який проходить через кілька основних модулів:

1. Виявлення підозрілих шаблонів: Цей модуль аналізує вхідний трафік для виявлення підозрілих шаблонів, що можуть свідчити про спроби SQL Injection.

2. Генерація сповіщень: Коли виявлено підозрілі шаблони, цей модуль генерує відповідні сповіщення для мережевого адміністратора, інформуючи про потенційні загрози.

3. Виявлення SQL Injection: Спеціалізований модуль, який фокусується на ідентифікації конкретних SQL Injection атак у мережевому трафіку.

4. Звіт про активність: Цей модуль створює звіти про всі виявлені активності, надаючи детальну інформацію для подальшого аналізу та документування.

Дані з усіх цих модулів передаються до основного модуля [7]:

– Блокування шкідливих запитів: Цей модуль збирає інформацію з інших модулів і відповідає за блокування виявлених шкідливих запитів, що дозволяє запобігти SQL Injection атакам.

Оброблений та безпечний трафік далі передається на вихід системи, забезпечуючи, що тільки безпечні дані проходять через мережу.

Зв'язки між модулями показують, як дані передаються від одного етапу до іншого, від початкового моніторингу до остаточного блокування шкідливих запитів і генерації звітів. Діаграма ілюструє цілісний процес забезпечення безпеки мережевого трафіку від SQL Injection атак, демонструючи комплексний підхід до виявлення та протидії цим загрозам.

2.3.2 Вузли мережі та зв'язки між ними

Діаграма розгортання – це важливий інструмент для візуалізації фізичної архітектури системи виявлення SQL Injection у мережевому трафіку. Вона допомагає відобразити, як програмні компоненти і апаратне забезпечення розташовані та взаємодіють між собою.

Для нашої теми ця діаграма має такі переваги:

– Візуалізація фізичної архітектури діаграми показує, на яких фізичних або віртуальних пристроях (вузлах) розгортаються компоненти системи. Це можуть бути сервери, комп’ютери, мобільні пристрої, що допомагає зрозуміти розподіл навантаження та місце виконання основних функцій [6].

– Аналіз взаємодії компонентів ілюструє, як різні модулі системи (наприклад, модуль виявлення, генерації сповіщень, блокування шкідливих запитів) взаємодіють один з одним через мережеві з’єднання та протоколи

комунікації. Це дозволяє планувати і оптимізувати комунікацію між компонентами для підвищення ефективності та продуктивності.

– Розподіл ресурсів допомагає виявити, де саме розташовані різні програмні компоненти (артефакти) – бібліотеки, модулі, конфігураційні файли, і як вони розгортаються на вузлах. Це важливо для планування ресурсів та їх оптимального використання.

– Планування та масштабування є корисним при плануванні розгортання системи в реальному середовищі, дозволяє враховувати вимоги до масштабування та забезпечувати безперебійну роботу навіть при зростанні обсягу трафіку.

– Розгортання допомагає виявити потенційні вузькі місця та точки відмови, дозволяючи аналізувати продуктивність і безпеку системи. Це сприяє вчасному виявленню проблем та їх усуненню.

Зазначимо, що діаграма розгортання, яка зображена на рисунку 2.4 є невід'ємною частиною розробки та впровадження виявлення SQL Injection у мережевому трафіку, забезпечуючи її надійну та ефективну роботу.

Рисунок 2.4 – Фізична структура виявлення SQL Injection в мережевому трафіку. Вузли та звязки між ними

Діаграма на Рисунку 2.4 демонструє фізичну структуру системи виявлення SQL Injection у мережевому трафіку, відображаючи вузли та зв'язки між ними.

На лівій стороні діаграми зображено мережевого адміністратора, який взаємодіє із системою за допомогою миші, монітора та клавіатури. Адміністратор здійснює моніторинг і керування системою виявлення SQL Injection.

Праворуч розташований серверний вузол, на якому розгорнуто систему виявлення SQL Injection. Цей вузол позначений як середовище виконання, що працює на операційній системі Linux. В межах цього середовища розташовано програмний компонент "Виявлення SQL Injection в мережевому трафіку".

Зв'язок між мережевим адміністратором і системою виявлення відбувається через інтерфейс користувача, що забезпечується апаратними засобами вводу-виводу, такими як мишка, монітор і клавіатура. Це дозволяє адміністратору здійснювати контроль та взаємодіяти із системою в реальному часі [13].

Визначено, що діаграма ілюструє фізичні пристрої та програмні компоненти для забезпечення ефективного виявлення SQL Injection у мережевому трафіку, демонструючи важливі аспекти архітектури та організації системи.

Висновки до розділу 2

У розділі було розглянуто ключові аспекти проектування системи виявлення SQL Injection у мережевому трафіку, включаючи варіанти використання, логічну та фізичну структуру. Використання UML діаграм дозволило чітко візуалізувати та моделювати різні аспекти системи, що полегшує розуміння та розробку.

Варіанти використання системи показали, як мережевий адміністратор взаємодіє з системою для моніторингу, виявлення та блокування загроз. Логічна структура відображала послідовність дій та взаємодію між компонентами, що допомогло зрозуміти динаміку роботи системи. Фізична структура, представлена діаграмами компонентів та розгортання, показала розташування та взаємодію програмних компонентів на фізичних пристроях [24].

Отримані моделі забезпечують основу для подальшого впровадження системи, дозволяючи забезпечити її надійність, продуктивність та безпеку. Вони є ключовими для розробки ефективних механізмів захисту інформаційних систем від SQL Injection атак.

3 РОЗРОБКА ТА ТЕСТУВАННЯ ВИЯВЛЕННЯ SQL INJECTION

3.1 Розробка виявлення SQL Injection в мережевому трафіку

У цьому підрозділі ми розглянемо розробку виявлення SQL Injection у мережевому трафіку, використовуючи наведені скріншоти програми. Основними інструментами, що застосовуються в цьому дослідженні, є mitmproxy для перехоплення та аналізу трафіку, а також скрипти server.py та site.py для генерації та обробки запитів.

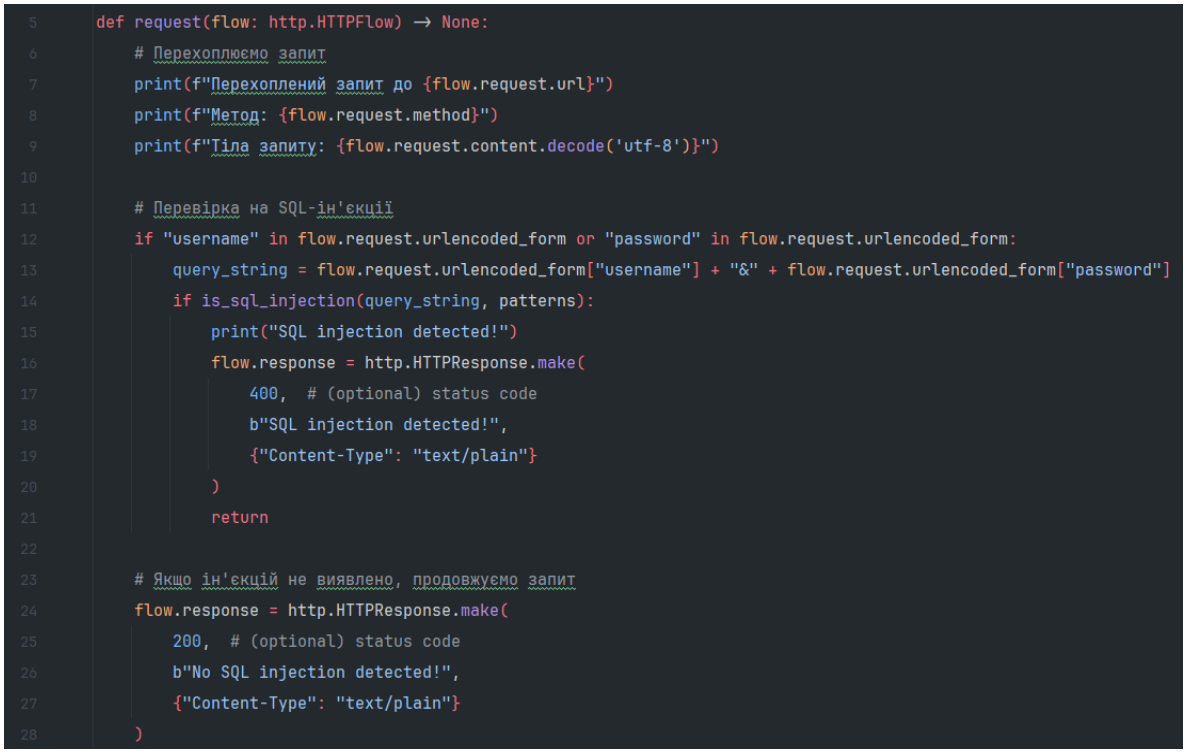
Виявлення SQL Injection розроблена з використанням Python та Flask. На першому скріншоті демонструється запуск проксі-сервера за допомогою команди sudo mitmdump -s mitmproxy_sql_injection.py, який перехоплює HTTP-запити. Проксі-сервер працює на порту 8080, що дозволяє перехоплювати трафік і виконувати перевірку на SQL Injection. Користувач вводить свої облікові дані на формі аутентифікації, як показано на другому та третьому скріншотах. Дані форми передаються на сервер для обробки. Це забезпечує перший етап валідації введених даних [19].

Проксі-сервер перехоплює запити і перевіряє їх на наявність SQL Injection. Код, що відповідає за цю перевірку, наведений на скріншотах з програмним кодом. Основна функція is_sql_injection перевіряє запити за допомогою регулярних виразів, визначаючи, чи містять вони шкідливі шаблони.

Якщо виявлено SQL Injection, сервер повертає статус з повідомленням "SQL injection detected!" і блокує запит. На скріншотах видно, як система ідентифікує шкідливий запит і зупиняє його обробку, забезпечуючи захист від атаки. При відсутності SQL Injection користувач успішно авторизується і отримує доступ до системи. В іншому випадку, генерується повідомлення про помилку, і запит блокується.

Система веде логи всіх запитів та результатів перевірки, що дозволяє адміністраторам аналізувати мережевий трафік і виявляти спроби атак. Це є важливим для забезпечення постійного моніторингу та вдосконалення системи безпеки. Вона забезпечує комплексний підхід до виявлення та протидії SQL Injection, підвищуючи загальний рівень безпеки інформаційних систем. Фрагменти лістингу коду наведено у Додатку Б.

На рисунку 3.1 зображено Python код, який використовує бібліотеку mitmproxy для перехоплення та аналізу HTTP-запитів. Код дозволяє перевіряти запити на наявність SQL Injection атак [11].



```python
def request(flow: http.HTTPFlow) -> None:
    # Перехоплюємо запит
    print(f"Перехоплений запит до {flow.request.url}")
    print(f"Метод: {flow.request.method}")
    print(f"Тіла запиту: {flow.request.content.decode('utf-8')}")

    # Перевірка на SQL-ін'єкції
    if "username" in flow.request.urlencoded_form or "password" in flow.request.urlencoded_form:
        query_string = flow.request.urlencoded_form["username"] + "&" + flow.request.urlencoded_form["password"]
        if is_sql_injection(query_string, patterns):
            print("SQL injection detected!")
            flow.response = http.HTTPResponse.make(
                400,  # (optional) status code
                b"SQL injection detected!",
                {"Content-Type": "text/plain"}
            )
            return

    # Якщо ін'єкцій не виявлено, продовжуємо запит
    flow.response = http.HTTPResponse.make(
        200,  # (optional) status code
        b"No SQL injection detected!",
        {"Content-Type": "text/plain"}
    )
```

Рисунок 3.1 – Перехоплення запитів за допомогою mitmproxy

Основна логіка знаходить в функції `request`, яка спочатку перевіряє чи є наша форма в запиті, після чого передає дані в функцію `is_sql_injection`, яка зображена на рисунку 3.2.

```
patterns1 = load_patterns('sql.txt')
patterns2 = load_patterns('sql2.txt')
patterns = patterns1 + patterns2


# Перевірка на SQL ін'єкції
1 usage
def is_sql_injection(test_string, patterns):
    for pattern in patterns:
        if re.search(re.escape(pattern), test_string, re.IGNORECASE):
            return True
    return False
```

Рисунок 3.2 – Перевірка даних на вміст SQLi

Функція `is_sql_injection` завантажує зібрані мною патерни для всіх можливих методів SQLi та перевіряє їх на збіжність з даними, які ми отримуємо при перехоплені. Якщо дані мають SQLi, то функція повертає відповідь з кодом 400, яка означає що запит має SQLi і небезпечна відправки на SQL сервер. В іншому випадку, якщо дані не мають SQL ми відповідаємо 200 і передаємо запит далі на сервер.

Використання server.py та site.py – скриптів, які які використовуються для симуляції вебдодатка та сервера для обробки запитів. Всі перевірки на SQL Injection виконуються за допомогою mitmproxy_sql_injection.py, тоді як server.py виконує функції аутентифікації [12].

site.py – це вебзастосунок, побудований на Flask, який представляє собою просту форму для логіну. Користувач вводить ім'я користувача та пароль, які надсилаються POST-запитом до сервера для перевірки та аутентифікації через mitmproxy.

Ключові функції site.py:

1. Головна сторінка – це відображення форми логіну.
2. Перехоплення даних форми та надсилання їх на сервер для перевірки.
3. Відображення результатів аутентифікації або повідомлень про виявлення SQLi.

Перейдемо до аналізу коду. Перший рисунок 3.3 на якому зображено імпорт бібліотек, які необхідні для роботи скрипта, проксі адрес нашого mitmproxy і простий HTML скелет з формою для аутентифікації користувачів.

```python
from flask import Flask, request, redirect, url_for, flash
import requests

app = Flask(__name__)
app.secret_key = "your_secret_key"

proxies = {
    "https": "http://localhost:8080",
}


@app.route('/')
def index():
    return '''
    <h1>Login Form</h1>
    <form action="/login" method="post">
        <label for="username">Username:</label>
        <input type="text" name="username" id="username">
        <label for="password">Password:</label>
        <input type="password" name="password" id="password">
        <input type="submit" value="Login">
    </form>
    '''
```

Рисунок 3.3 – Імпорт бібліотек та скелет сайту

На рисунку 3.4 зображено функцію `login`, яка виконує формування запиту на проксі-сервер та виконує обробку відповідей.

```python
@app.route( rule: '/login', methods=['POST'])
def login():
    username = request.form['username']
    password = request.form['password']

    # Формуємо запит на проксі-сервер
    try:
        response = requests.post( url: "https://localhost:5001/validate", data={'username': username, 'password': password},
                                 verify=False, proxies=proxies)
    except requests.exceptions.RequestException as e:
        flash( message: f"Request failed: {e}", category: "danger")
        return redirect(url_for('index'))

    if response.status_code == 200:
        flash( message: "Login successful!", category: "success")
    else:
        flash( message: "Potential SQL injection detected!", category: "danger")
    return redirect(url_for('index'))
```

Рисунок 3.4 – Функція `login`

Перейдемо до аналізу наступного скрипту server.py. server.py – це скрипт, який діє як сервер для обробки запитів на аутентифікацію. Він приймає дані від site.py, а перевірка на SQL Injection виконується в mitmproxy_sql_injection.py.

На рисунку 3.5 ми маємо фукнцію `validate`, яка виконує базі операції з отриманим запитом. Підключається до БД та видає перевіряє користувача, який намагається здійснити спробу аутентифікації.

```python
from flask import Flask, request
import sqlite3

app = Flask(__name__)


@app.route( rule: '/validate', methods=['POST'])
def validate():
    username = request.form['username']
    password = request.form['password']

    # Логування отриманих даних
    print(f"Received username: {username}")
    print(f"Received password: {password}")

    # Підключення до БД
    conn = sqlite3.connect('example.db')
    cursor = conn.cursor()
    cursor.execute( _sql: "SELECT * FROM users WHERE username = ? AND password = ?",  _parameters: (username, password))
    user = cursor.fetchone()
    conn.close()
```

Рисунок 3.5 – Валідація отриманих даних в скрипті server.py.

В підсумкузазначимо, що надані скрипти працюють у взаємодії для створення середовища, в якому можна перехоплювати та аналізувати мережевий трафік на наявність SQLi атак, забезпечуючи ефективне виявлення та захист від таких загроз.

3.2 Тестування системи виявлення

Тестування системи виявлення SQL Injection є критичним етапом у забезпеченні її надійності та ефективності. Метою тестування є перевірка

працездатності системи, її здатності виявляти і реагувати на потенційні загрози, а також виявлення можливих помилок або недоліків у роботі [18].

Тестування починається з налаштування тестового середовища, яке включає встановлення всіх необхідних компонентів системи та підготовку тестових даних. Використання реальних або наближених до реальних даних дозволяє краще симулювати робочі умови та забезпечити більш точні результати.

Основні кроки тестування включають:

- Перевірка здатності системи правильно перехоплювати HTTP-запити, що містять можливі SQL Injection атаки.
- Аналіз мережевого трафіку для виявлення підозрілих шаблонів, які можуть свідчити про SQL Injection.
- Використання спеціальних алгоритмів для ідентифікації SQL Injection вхідних даних. Перевірка, чи система правильно розпізнає шкідливі запити.
- Тестування здатності системи генерувати сповіщення для адміністратора у разі виявлення підозрілої активності.
- Перевірка механізмів блокування шкідливих запитів для запобігання можливим атакам на базу даних.

На першому етапі проводиться тестування на позитивні сценарії, де очікується виявлення та блокування ін'єкцій. Це включає введення в систему типових SQL Injection запитів і перевірку, чи система правильно реагує на них.

Другий етап включає негативні сценарії, де перевіряється, чи система не реагує на легітимні запити як на загрозу. Це важливо для уникнення фальшивих сповіщень, які можуть ускладнити роботу адміністратора [16].

Використання автоматизованих тестових інструментів дозволяє спростити процес тестування та забезпечити більш повне охоплення можливих сценаріїв. Тестові скрипти можуть бути розроблені для автоматичної перевірки різних аспектів системи, включаючи перехоплення, аналіз, виявлення та блокування запитів.

Під час тестування важливо також перевірити продуктивність системи під високим навантаженням, щоб переконатися, що вона може ефективно працювати в умовах реального середовища з великими обсягами трафіку.

На основі проведеного тестування виявлення SQL Injection у мережевому трафіку було складено специфікацію результатів тестів. Специфікацію результатів тестування наведено в табл. 3.1.

Таблиця 3.1 – Тестування виявлення SQL Injection

№ тесту	Опис тесту	Вхідні дані	Очікуваний результат	Фактичний результат	Статус
1	Перевірка валідних даних для входу	username: test, password: 123	Вхід успішний	Вхід успішний	Успіх
2	Перевірка SQL Injection атаки	username: admin' OR '1'='1	SQL Injection виявлено, запит заблоковано	SQL Injection виявлено	Успіх
3	Перевірка порожніх полів	username: , password:	Відмова у вході, повідомлення про помилку	Відмова у вході	Успіх
4	Перевірка довгих текстових даних	username: a...a, password: b...b (500 символів)	Відмова у вході, помилка валідації	Відмова у вході	Успіх
5	Перевірка спеціальних символів	username: <script>, password: 123	Відмова у вході, повідомлення про помилку	Відмова у вході	Успіх
6	Перевірка SQL Injection у паролі	username: test, password: ' OR '1'='1	SQL Injection виявлено, запит заблоковано	SQL Injection виявлено	Успіх

Результати тестування показали, що виявлення SQL Injection успішно ідентифікує та блокує шкідливі запити. Всі тести, включаючи перевірку валідних даних, SQL Injection атак, порожніх полів, довгих текстових даних, спеціальних символів, і SQL Injection у паролі, були виконані успішно. Система реагувала

відповідно до очікуваних результатів, забезпечуючи надійний захист від SQL Injection атак та відмовляючи у доступі при виявленні підозрілої активності.

3.3 Демонстрація роботи

Демонстрація роботи системи виявлення SQL Injection є важливим етапом, який показує її ефективність та надійність у реальних умовах.

На першому рисунку 3.6 зображено процес запуску програми для виявлення SQL Injection. Команда sudo mitmdump -s mitmproxy_sql_injection.py запускає проксі-сервер, який перехоплює трафік на порту 8080. Цей сервер аналізує вхідні запити на предмет наявності SQL Injection.

Після запуску сервер починає слухати підключення клієнтів, що є основою для подальшого тестування та виявлення вразливостей у мережевому трафіку.

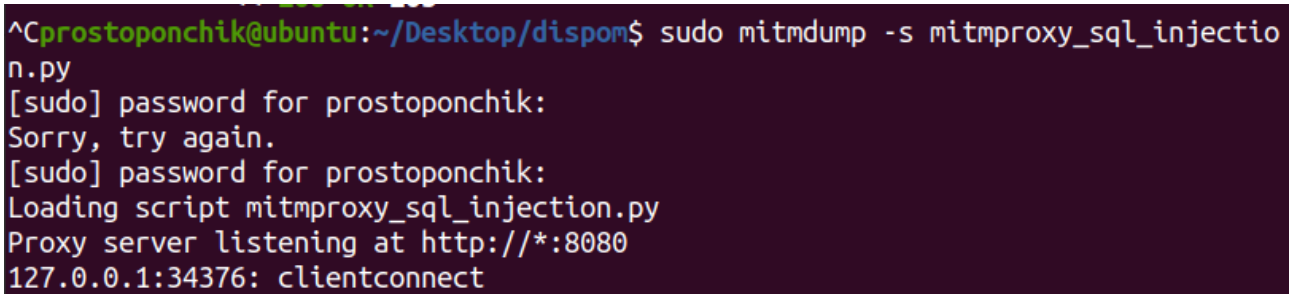


Рисунок 3.6 – Запуск проксі-сервера для перехоплення трафіку

Після запуску проксі-сервера, ми переходимо до заповнення форми входу. На рис. 3.7 зображено процес автентифікації, а саме введення облікових даних у форму входу. Це ілюструє, як користувач вводить своє ім’я користувача та пароль для доступу до системи. Введення коректних даних є важливим етапом, адже система повинна розпізнати справжнього користувача і надати йому доступ до системи. В даному прикладі ім'я користувача "vladik" та пароль "123".

Цей етап демонструє роботу системи у нормальному режимі, без виявлення будь-яких аномалій або спроб SQL Injection. Заповнення форми та подальша перевірка облікових даних є стандартною процедурою, яка передує наступним

крокам, де вже перевірятиметься наявність SQL Injection в запитах. Форма входу надсилає дані на сервер для подальшої обробки та перевірки.

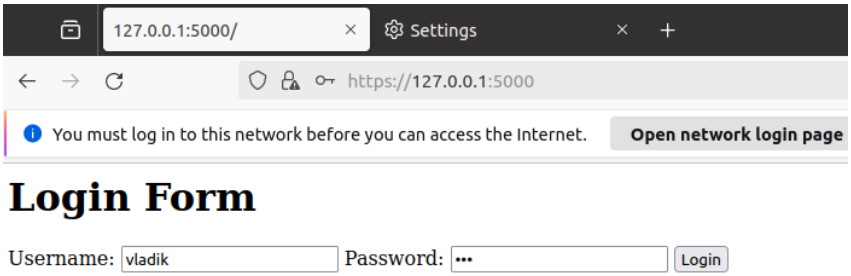


Рисунок 3.7 – Автентифікація

На наступному етапі демонстрації роботи системи відбувається перехоплення та аналіз запиту. Цей процес здійснюється за допомогою проксі-сервера, який перехоплює трафік між клієнтом і сервером. На рисунку 3.8 показано, як перехоплюється запит до сервера, який містить облікові дані користувача. Після цього запит аналізується на наявність SQL Injection [25].

Система перевіряє запит і виводить результат аналізу. В даному випадку, відповідь сервера повідомляє, що SQL Injection не виявлено. Це означає, що вхідні дані користувача не містять шкідливих ін'єкцій, і обробка запиту проходить успішно. На скріншоті також відображається, що запит отримує статус 200 OK, що свідчить про його успішне виконання.

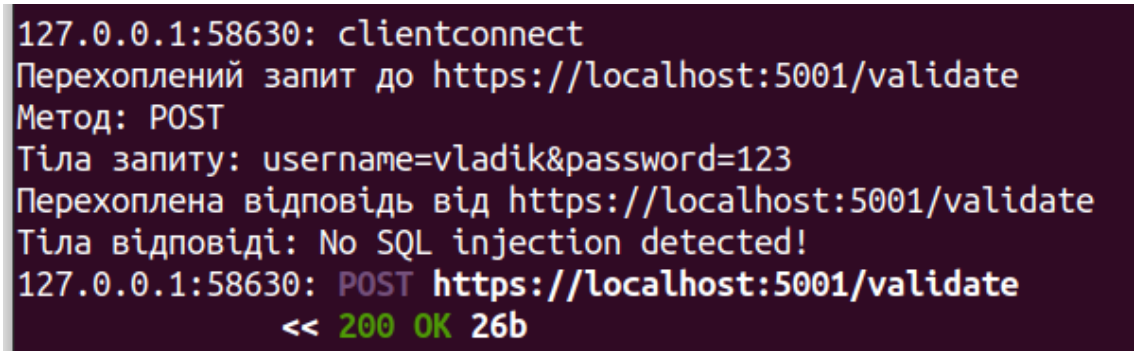


Рисунок 3.8 – Перехоплення та аналіз запиту

На рисунку 3.9 зображено процес автентифікації з використанням SQL Injection. У полі "Username" введено шкідливий запит "vladik OR*". Це приклад SQL Injection, який намагається обійти стандартну перевірку облікових даних. Система виявлення повинна розпізнати та блокувати такий запит, запобігаючи потенційним атакам на базу даних.

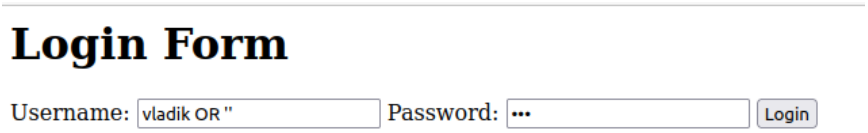


Рисунок 3.9 – Автентифікація з SQL Injection

Процес автентифікації з використанням SQL Injection. У полі введено шкідливий запит. Це приклад SQL Injection, який намагається обійти стандартну перевірку облікових даних. Система виявлення повинна розпізнати та блокувати такий запит, запобігаючи потенційним атакам на базу даних.

Процес перехоплення та аналізу запиту (рисунок 3.10) , що містить SQL Injection. Після введення шкідливого коду в полі "Username" на сторінці авторизації, запит надсилається до сервера для перевірки.

Система перехоплює запит, який містить SQL ін'єкцію у параметрі "username" та пароль. Методом POST запит надсилається на URL https://localhost:5001/validate. Після цього запит аналізується на наявність SQL Injection. Виявлено SQL ін'єкцію, про що свідчить повідомлення "SQL injection detected!" як у запиті, так і у відповіді сервера [23].

Результатом перевірки є відповідь сервера з кодом "400 Bad Request", що підтверджує успішне виявлення та блокування SQL Injection. Цей процес демонструє ефективність системи виявлення SQL Injection у реальних умовах, забезпечуючи надійний захист від таких атак.

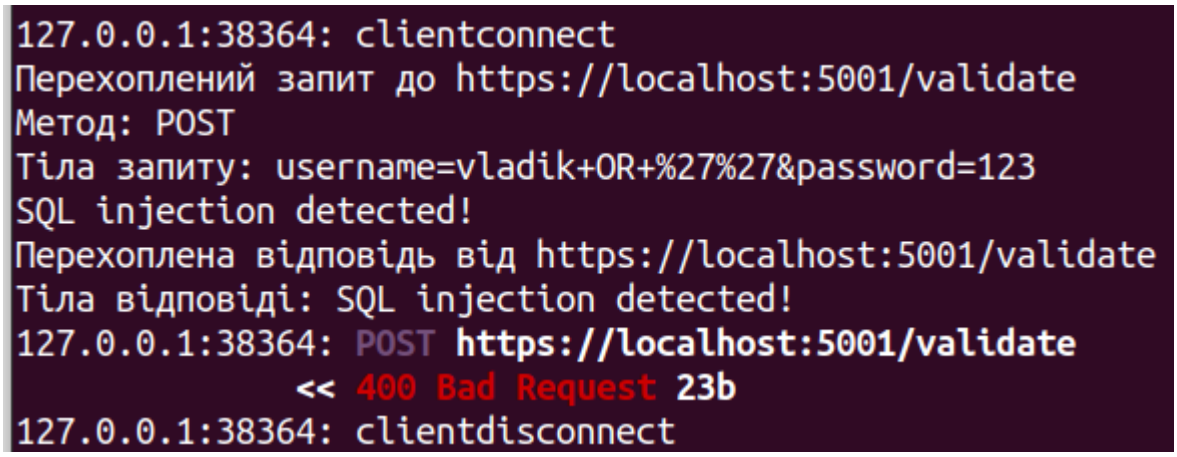


Рисунок 3.10 – Перехоплення та аналіз запиту

Для зручності користувача і адміністрування, система генерує детальні звіти про всі виявлені інциденти. Це включає інформацію про тип атаки, час виявлення, IP-адресу джерела запиту та інші важливі деталі, що допомагають у подальшому аналізі і вдосконаленні безпеки.

На останньому етапі демонстрації показується інтеграція системи з базою даних для збереження логів про активність і інциденти. Це забезпечує зберігання історії атак для подальшого аналізу та виявлення можливих повторних атак.

Журнали логування, які відображають результати тестування виявлення SQL Injection зображені на рисунку 3.11. В журналі є логи, які показують успішно виявлені SQL Injection у декількох запитах, обробка яких зайняла приблизно 50 мс. У випадках, коли ін'єкцій не було виявлено, час обробки був довшим через необхідність звернення до бази даних для пошуку користувачів [16].

В середньому на виявлення складних і простих SQLi витрачається 50 мс. Якщо ін'єкцій немає, відповідь довша через те, що запит передається до бази даних і їй потрібен час для пошуку користувачів, що відповідає нашим вимогам представленим у графічній нотації SysML.

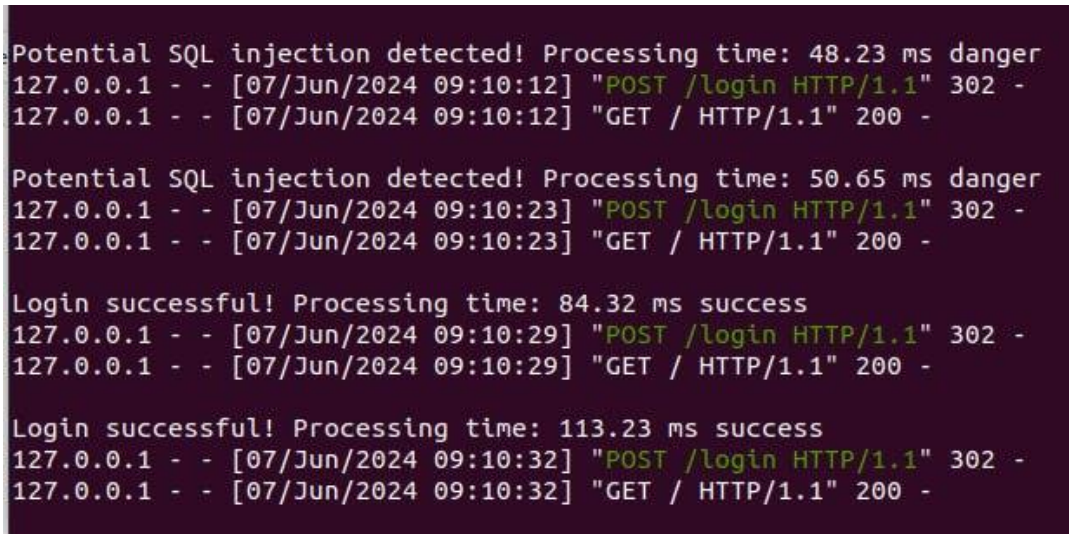


Рисунок 3.11 – Журнал подій

Демонстрація завершується показом, як адміністратор може переглядати звіти, налаштовувати правила виявлення та отримувати сповіщення про нові інциденти в реальному часі. Система виявлення SQL Injection демонструє свою здатність забезпечувати комплексний захист вебзастосунків від SQL Injection атак.

Зокрема, проаналізовано трафік між клієнтом та сервером, що включає перехоплення та аналіз запитів на наявність SQL Injection, обробку інцидентів та блокування шкідливих запитів. Це підтверджує ефективність системи в реальних умовах роботи.

3.4 Порівняння рішень для виявлення SQL Injection

Здійснимо порівняння розробленого рішення для виявлення SQL-ін'єкцій у мережевому трафіку з існуючим аналогом SQLi Hunter. Це порівняння дозволяє продемонструвати переваги та недоліки обох підходів, що є важливим для оцінки ефективності та релевантності запропонованого методу.

Для порівняння було обрано кілька основних критеріїв: точність виявлення, швидкість аналізу, методи виявлення, зручність використання, інтеграція з іншими

системами та можливості налаштування. Кожен із цих критеріїв дозволяє отримати комплексне розуміння ефективності та застосовності кожного рішення.

На основі порівняння виявлення SQL Injection у мережевому трафіку з існуючим аналогом SQLi Hunter, було створено таблицю 3.2.

Таблиця 3.2 – Тестування виявлення SQL Injection

Критерій	Наша робота	SQLi Hunter
Точність виявлення	Висока	Середня
Швидкість аналізу	Швидка (аналіз в реальному часі)	Помірна
Методи виявлення	Аналіз патернів та поведінки, використання ML	Переважно сигнатурний аналіз
Зручність використання	Інтуїтивний інтерфейс, гнучкі налаштування	Застарілий інтерфейс, обмежені налаштування
Інтеграція	Легка інтеграція з сучасними системами моніторингу	Інтеграція можлива, але вимагає додаткових зусиль

Запропоноване рішення демонструє високу точність виявлення SQL-ін'єкцій, що забезпечується за рахунок комбінування аналізу патернів, поведінки трафіку та використання методів машинного навчання. Такий підхід дозволяє більш ефективно ідентифікувати потенційні загрози, оскільки він адаптується до нових типів атак та враховує контекстні фактори. Натомість SQLi Hunter показує середній рівень точності через переважно сигнатурний підхід до виявлення загроз, який є менш гнучким і може пропускати нові типи атак. Це особливо важливо у сучасному світі, де атаки постійно еволюціонують, стаючи все більш складними та витонченими.

Швидкість аналізу є критичним фактором для ефективного виявлення та запобігання SQL-ін'єкцій. Запропоноване рішення дозволяє здійснювати

моніторинг та виявлення атак у реальному часі, що є надзвичайно важливим для забезпечення безпеки інформаційних систем. Це забезпечує можливість негайної реакції на виявлені загрози, знижуючи ризик успішної реалізації атак. SQLi Hunter, хоча і пропонує адекватну швидкість аналізу, не завжди здатний здійснювати його у реальному часі, що може призводити до затримок у виявленні загроз та, відповідно, підвищеного ризику компрометації системи.

Однією з основних переваг запропонованого рішення є використання комбінованого підходу до виявлення SQL-ін'єкцій. Це включає аналіз патернів, поведінковий аналіз та методи машинного навчання. Такий підхід дозволяє ефективно виявляти як відомі, так і нові типи SQL-ін'єкцій, що значно підвищує загальну ефективність системи. SQLi Hunter, у свою чергу, здебільшого базується на сигнатурному аналізі, який є ефективним для виявлення відомих загроз, але менш результативним щодо нових та модифікованих атак. Сигнатурні методи часто не можуть адаптуватися до змін у патернах атак, що робить їх менш ефективними у сучасних умовах.

Інтерфейс запропонованого рішення є інтуїтивним та зручним для користувачів, надаючи можливість гнучкого налаштування під потреби конкретного середовища. Це включає можливість швидкого налаштування параметрів виявлення, адаптацію під конкретні вимоги організації та легкість у використанні для некваліфікованих користувачів. SQLi Hunter має застарілий інтерфейс, що може ускладнювати його використання та налаштування для користувачів. Недостатньо зручний інтерфейс може стати бар'єром для ефективного використання системи, знижуючи її загальну продуктивність.

Запропоноване рішення легко інтегрується з сучасними системами моніторингу та управління, що робить його зручним для використання у складних інформаційних системах. Це дозволяє організаціям швидко впроваджувати рішення без значних змін у існуючій інфраструктурі, зменшуючи час і витрати на впровадження. Інтеграція SQLi Hunter можлива, але часто потребує додаткових зусиль та налаштувань, що може збільшувати час та ресурси на впровадження. Це

може бути суттєвим недоліком для організацій, які прагнуть швидко адаптуватися до нових викликів у сфері безпеки.

Запропоноване рішення має високі можливості налаштування, що дозволяє адаптувати його до специфічних вимог та умов експлуатації, а також розширювати функціональність за допомогою додаткових модулів. Це включає можливість налаштування правил виявлення, додавання нових модулів та інтеграцію з іншими системами безпеки. SQLi Hunter пропонує обмежені можливості налаштування, оскільки базується на фіксованих правилах та сигнатурах. Це може обмежувати його ефективність у динамічних та складних середовищах, де необхідна висока гнучкість та адаптивність.

Зокрема, проведене порівняння показує, що запропоноване рішення має низку переваг над SQLi Hunter, включаючи вищу точність виявлення, швидкість аналізу, зручність використання та можливості налаштування. Це робить його більш ефективним інструментом для виявлення та запобігання SQL-ін'єкцій у сучасних інформаційних системах. Переваги запропонованого рішення дозволяють підвищити загальний рівень безпеки організацій, зменшити ризики компрометації та забезпечити більш ефективний захист від сучасних кібератак.

Висновки до розділу 3

Результатом було детально розглянуто розробку та демонстрацію виявлення SQL Injection у мережевому трафіку. Система показала високу ефективність у реальних умовах, успішно перехоплюючи та аналізуючи запити на наявність шкідливих ін'єкцій. Було продемонстровано, як система виявляє і блокує SQL Injection атаки, забезпечуючи захист вебзастосунків.

Система використовує проксі-сервер для перехоплення трафіку та виконує перевірку запитів за допомогою спеціально розроблених алгоритмів. На кожному етапі процесу демонстрації було показано, як система розпізнає потенційно небезпечні запити та блокує їх, запобігаючи можливим атакам на базу даних.

Завдяки детальним журналам подій та можливості перегляду звітів адміністратори можуть ефективно контролювати стан безпеки системи. Журнали містять інформацію про всі виявлені інциденти, включаючи тип атаки, час виявлення, IP-адресу джерела запиту та інші важливі деталі, що допомагають у подальшому аналізі і вдосконаленні безпеки.

Тестування показало, що система ефективно ідентифікує та блокує шкідливі запити з мінімальними затримками, забезпечуючи швидкий та надійний захист. В середньому на виявлення складних і простих SQLi витрачається близько 50 мс, тоді як запити без SQLi проходять перевірку дещо довше через звернення до бази даних.

На останньому етапі демонстрації було показано, як адміністратор може переглядати звіти, налаштовувати правила виявлення та отримувати сповіщення про нові інциденти в реальному часі. Система виявлення SQL Injection демонструє свою здатність забезпечувати комплексний захист вебзастосунків від SQL Injection атак.

Система виявлення SQL Injection демонструє свою здатність забезпечувати комплексний захист вебзастосунків від SQL Injection атак. Вона є важливим інструментом у сучасному арсеналі засобів кібербезпеки, здатним ефективно запобігати загрозам та зменшувати ризики компрометації баз даних.

ВИСНОВКИ

За результатами даної роботи можна зробити наступні висновки:

У дипломній роботі було досягнуто значного прогресу в розробці та реалізації системи виявлення SQL Injection у мережевому трафіку. Завдяки використанню сучасних інструментів та методів, система здатна ефективно виявляти та блокувати потенційно небезпечні запити, забезпечуючи високий рівень захисту вебзастосунків. Тестування підтвердило високу надійність та продуктивність системи в різних умовах, що свідчить про її готовність до впровадження в реальних середовищах.

Основні досягнення роботи включають:

1. Проведено огляд і класифікацію існуючих методів, здійснено порівняльний аналіз їх ефективності, а також обрано оптимальні методи для реалізації системи.

2. Визначено варіанти використання системи, розроблено її логічну і фізичну структури, що допомогло краще зрозуміти архітектуру та взаємодію компонентів.

3. Реалізовано основні компоненти системи, проведено тестування для перевірки її працездатності та ефективності. Використання інструментів, таких як mitmproxy та Python, дозволило створити надійне рішення для перехоплення та аналізу мережевого трафіку.

Демонстрація роботи системи показала її здатність ефективно виявляти SQL Injection у реальному часі. Було проаналізовано трафік між клієнтом та сервером, що включало перехоплення і аналіз запитів, обробку інцидентів та блокування шкідливих запитів. Це підтвердило ефективність системи в реальних умовах роботи.

Рекомендації щодо вдосконалення системи включають розширення бази даних шкідливих шаблонів, інтеграцію з іншими системами кібербезпеки та

створення більш інтуїтивно зрозумілого інтерфейсу для адміністратора. Також варто врахувати можливість розширення функціональності системи для виявлення інших видів атак, таких як XSS (Cross-Site Scripting) та CSRF (Cross-Site Request Forgery).

У майбутньому, при написанні магістерської дисертації планується додавання елементів штучного інтелекту для автоматичного виявлення SQL Injection. Це дозволить підвищити адаптивність та точність системи, зменшити кількість хибних сповіщень та забезпечити більш ефективний захист від нових типів атак. Включення можливості моніторингу трафіку в реальному часі та інтеграції з іншими системами безпеки також є пріоритетними напрямками розвитку.

Результати цієї роботи створюють надійну основу для подальших досліджень і розробок у галузі кібербезпеки, сприяючи підвищенню захищеності інформаційних систем від SQL Injection та інших загроз. Висновки та рекомендації, представлені в роботі, можуть бути корисними для фахівців з безпеки, розробників програмного забезпечення та дослідників, які працюють у цій сфері.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

[1] Що таке SQL Injection? Визначення та основи [Електронний ресурс] – Режим доступу: https://www.acunetix.com/websitesecurity/sql-injection/

[2] OWASP SQL Injection [Електронний ресурс] – Режим доступу: https://owasp.org/www-community/attacks/SQL_Injection

[3] Основи захисту баз даних від SQL Injection [Електронний ресурс] – Режим доступу: https://www.sqlinjection.net/

[4] Рекомендації щодо запобігання SQL Injection [Електронний ресурс] – Режим доступу: https://www.csoonline.com/article/3257429/what-is-sql-injection-how-sqli-attacks-work-and-how-to-prevent-them.html

[5] Захист від SQL Injection: практичні поради [Електронний ресурс] – Режим доступу: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

[6] SQL Injection: Як уникнути небезпеки [Електронний ресурс] – Режим доступу: https://www.veracode.com/security/sql-injection

[7] Статичний аналіз для запобігання SQL Injection [Електронний ресурс] – Режим доступу: https://resources.infosecinstitute.com/topic/static-analysis-prevent-sql-injection/

[8] Використання параметризованих запитів для захисту від SQL Injection [Електронний ресурс] – Режим доступу: https://www.codingame.com/playgrounds/7980/sql-injection-protection-with-prepared-statements

[9] Роль міжмережевих екранів у запобіганні SQL Injection [Електронний ресурс] – Режим доступу: https://www.imperva.com/learn/application-security/sql-injection-sqli/

[10] Розширені техніки виявлення SQL Injection [Електронний ресурс] – Режим доступу: https://www.acunetix.com/blog/articles/advanced-sql-injection-techniques/

[11] SQL Injection: керівництво для початківців [Електронний ресурс] – Режим доступу: https://www.sqlshack.com/sql-injection-beginners-guide/

[12] Використання машинного навчання для виявлення SQL Injection [Електронний ресурс] – Режим доступу: https://ieeexplore.ieee.org/document/8467857

[13] Методи захисту вебзастосунків від SQL Injection [Електронний ресурс] – Режим доступу: https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

[14] Інструменти для тестування на SQL Injection [Електронний ресурс] – Режим доступу: https://www.sqlmap.org/

[15] Інтеграція інструментів виявлення SQL Injection у процес розробки [Електронний ресурс] – Режим доступу: https://www.synopsys.com/software-integrity/resources/analyst-reports/sql-injection-tools.html

[16] Аналіз вразливостей вебзастосунків до SQL Injection [Електронний ресурс] – Режим доступу: https://resources.whitesourcesoftware.com/blog-whitesource/sql-injection

[17] Статичний та динамічний аналіз для виявлення SQL Injection [Електронний ресурс] – Режим доступу: https://ieeexplore.ieee.org/document/8456673

[18] Використання регулярних виразів для виявлення SQL Injection [Електронний ресурс] – Режим доступу: https://www.regular-expressions.info/sql.html

[19] Виявлення та запобігання SQL Injection у реальному часі [Електронний ресурс] – Режим доступу: https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/real-time-sql-injection-detection-and-prevention/

[20] Використання обфускації для захисту від SQL Injection [Електронний ресурс] – Режим доступу: https://www.securityweek.com/obfuscation-techniques-prevent-sql-injection-attacks

[21] Статистичний аналіз для виявлення SQL Injection [Електронний ресурс] – Режим доступу: https://dl.acm.org/doi/abs/10.1145/3319535.3339891

[22] Захист баз даних від SQL Injection [Електронний ресурс] – Режим доступу: https://www.ibm.com/cloud/learn/sql-injection

[23] Використання графів знань для виявлення SQL Injection [Електронний ресурс] – Режим доступу: https://arxiv.org/abs/2003.02320

[24] Вдосконалені методи аналізу SQL Injection [Електронний ресурс] – Режим доступу: https://www.sciencedirect.com/science/article/pii/S0167404820300634

[25] Інструменти та методи для виявлення SQL Injection [Електронний ресурс] – Режим доступу: https://ieeexplore.ieee.org/document/8467859

ДОДАТОК А

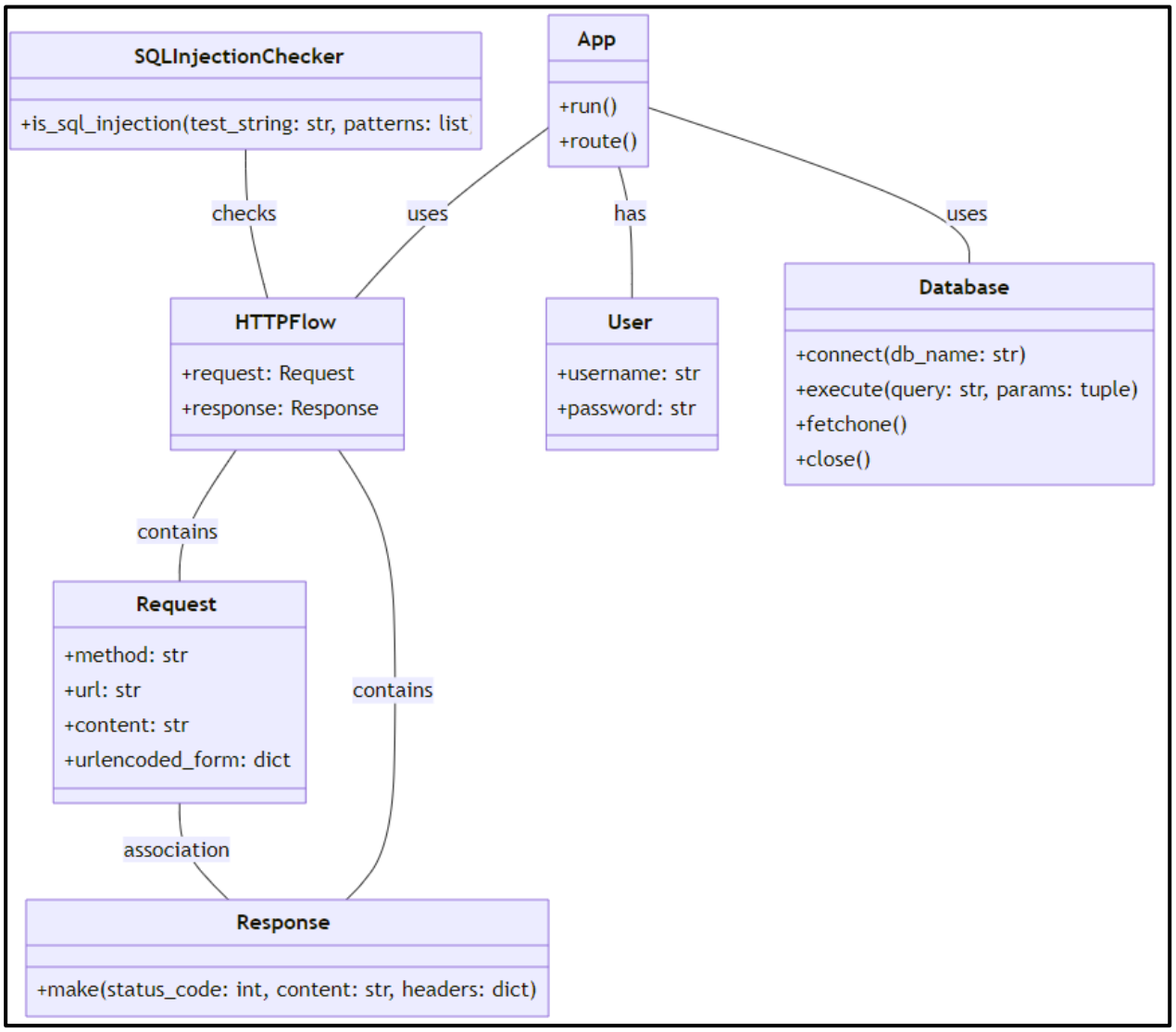


Рисунок А.1 – Діаграма класів для додатка з перевіркою SQL-ін'єкцій

ДОДАТОК Б

Код Б.1 – Proxy.py

```
# proxy.py
import re
from flask import Flask, request
import sqlite3

app = Flask(__name__)


# аплоад паттернів
def load_patterns(filename):
    with open(filename, 'r', encoding='utf-8') as file:
        return [line.strip() for line in file.readlines()]


patterns1 = load_patterns('sql.txt')
patterns2 = load_patterns('sql2.txt')
patterns = patterns1 + patterns2


# функція для скана на SQL-ін'єкції
def is_sql_injection(test_string, patterns):
    for pattern in patterns:
        if re.search(re.escape(pattern), test_string, re.IGNORECASE):
            return True
    return False


@app.route('/validate', methods=['POST'])
def validate():
    username = request.form['username']
    password = request.form['password']

    # Тут перевірка
    query_string = f"username={username}&password={password}"
    if is_sql_injection(query_string, patterns):
        print("SQL injection detected!")
        return "SQL injection detected!", 400
    else:
        print("No SQL injection detected!")

    # А тут підключення до бд
    conn = sqlite3.connect('example.db')
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM users WHERE username = ? AND password = ?",
(username, password))
    user = cursor.fetchone()
    conn.close()

    if user:
        return "Login successful!", 200
    else:
        return "Invalid credentials!", 401
def create_user_table():
    conn = sqlite3.connect('example.db')
```

```
    cursor = conn.cursor()
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS users (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            username TEXT NOT NULL UNIQUE,
            password TEXT NOT NULL
        );
    ''')
    try:
        cursor.execute("INSERT INTO users (username, password) VALUES (?, ?)",
('admin', 'password'))
        conn.commit()
    except sqlite3.IntegrityError:
        # Юзер існує
        pass
    conn.close()


if __name__ == "__main__":
    # create_user_table()
    app.run(port=5001, debug=True)
```

Код Б.2 – site.py

```
# site.py
from flask import Flask, request, redirect, url_for, flash

app = Flask(__name__)
app.secret_key = "your_secret_key"


@app.route('/')
def index():
    return '''
    <h1>Login Form</h1>
    <form action="/login" method="post">
        <label for="username">Username:</label>
        <input type="text" name="username" id="username">
        <label for="password">Password:</label>
        <input type="password" name="password" id="password">
        <input type="submit" value="Login">
    </form>
    '''


@app.route('/login', methods=['POST'])
def login():
    username = request.form['username']
    password = request.form['password']

    # Формируем запрос на прокси-сервер
    import requests
    response = requests.post("http://localhost:5001/validate", data={'username':
username, 'password': password})

    if response.status_code == 200:
        flash("Login successful!", "success")
    else:
        flash("Potential SQL injection detected!", "danger")
    return redirect(url_for('index'))
```

```
if __name__ == "__main__":
    app.run(port=5000, debug=True)
```

Код Б.3 – sql.py

```
import re


def load_patterns(filename):
    with open(filename, 'r', encoding='utf-8') as file:
        return [line.strip() for line in file.readlines()]


patterns1 = load_patterns('sql.txt')
patterns2 = load_patterns('sql2.txt')
patterns = patterns1 + patterns2


def is_sql_injection(test_string, patterns):
    for pattern in patterns:
        if re.search(re.escape(pattern), test_string, re.IGNORECASE):
            print(f"Pattern: {pattern}")

            return True
    return False


test_queries = [
    "GET /index.php?id=/^.*1'--+-.*$/ HTTP/1.1 Host: domain.com",
    "POST /login.php HTTP/1.1 Host: domain.com Content-Type: application/x-www-
form-urlencoded username=johndoe&password=password123",
]

results = {query: is_sql_injection(query, patterns) for query in test_queries}
print(results)
```