Хмельницький національний університет
Факультет інформаційних технологій
Кафедра кібербезпеки

**КВАЛІФІКАЦІЙНА РОБОТА**

Вишневського Дениса Яковича

на здобуття ступеня вищої освіти Бакалавра

Система виявлення вразливостей типу «SQL-injection» у вебдодатках

Галузь знань 12 – Інформаційні технології
Спеціальність 125 – Кібербезпека
Освітня програма Кібербезпека

Шифр КРБКБ.2102138.21.02.02 ПЗ

Виконав студент 4 курсу група КБ-21-2 Денис ВИШНЕВСЬКИЙ
Керівник канд. техн. наук, доцент Володимир ДЖУЛІЙ
Нормоконтролер старший викладач Сергій МОСТОВИЙ

До захисту допускаю:
Завідувач кафедри кібербезпеки Юрій КЛЬОЦ

5 06 2025 р.

Хмельницький 2025

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Кібербезпеки

Рівень вищої освіти Бакалавр

Галузь знань 12 – Інформаційні технології

Спеціальність 125 – Кібербезпека

Освітня програма Кібербезпека

ЗАТВЕРДЖУЮ

Завідувач кафедри кібербезпеки

Юрій КЛЬОЦ

15 лютого 2025 р.

**З А В Д А Н Н Я**
**НА КВАЛІФІКАЦІЙНУ РОБОТУ**

Вишневському Денису Яковичу

1 Тема роботи Система виявлення вразливостей типу «SQL-injection» у вебдодатках

Керівник роботи Володимир Джулій

Затверджено наказом ректора університету від 7 лютого 2025 № 23

2 Строк подання студентом кваліфікаційної роботи на кафедру

3 Вихідні дані до роботи Розробити ефективну систему виявлення вразливостей у вебдодатках. Дослідити предметну область, що стосується загроз інформаційній безпеці, пов'язаних з SQL-injection

4 Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Аналіз предметної області та постановка задачі,Проєктування системи захистуконфіденційних даних,Розробка програмного забезпечення,Тестування та оцінка ефективності

5 Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
« Ієрархія виконання задачі »,« Алгоритм роботи програми »

6 Консультанти розділів кваліфікаційної роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| | | | |

7 Дата видачі завдання 16 лютого 2025 р.

## КАЛЕНДАРНИЙ ПЛАН

| Назва етапів (розділів) кваліфікаційної роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|
| Вибір і затвердження теми кваліфікаційної роботи | Січень-Лютий | |
| Ознайомлення з предметною областю | Лютий | |
| Дослідження існуючих рішень | Лютий | |
| Постановка задачі | Березень | |
| Визначення загальних принципів рішення задачі | Березень | |
| Деталізація принципів рішення задачі | Квітень | |
| Розробка програмного забезпечення | Квітень | |
| Оформлення пояснювальної записки згідно вимог | Травень | |
| Оформлення графічної частини | Травень | |
| Захист КР | Червень | |

Студент Денис ВИШНЕВСЬКИЙ

Керівник кваліфікаційної роботи Володимир ДЖУЛІЙ

# АНОТАЦІЯ

Тема кваліфікаційної роботи: Система виявлення вразливостей типу «SQL-injection» у вебдодатках

Автор роботи: Вишневський Денис Якович.

Керівник роботи: Джулій Володимир Миколайович.

Пояснювальна записка: 65 с., 2 додатки, рисунків, таблиці, джерел.

Графічна частина: плакати, презентаційних слайдів.

СИСТЕМА ВИЯВЛЕННЯ ВРАЗЛИВОСТЕЙ, SQL-INJECTION, ВЕБДОДАТКИ, ІНФОРМАЦІЙНА БЕЗПЕКА, АНАЛІЗ УЯЗВИМОСТЕЙ, ЗАХИСТ ВІД АТАК, МЕТОДИ ВИЯВЛЕННЯ, СИСТЕМА МОНІТОРИНГУ.

Кваліфікаційна робота бакалавра присвячена розробці системи виявлення вразливостей типу «SQL-injection» у вебдодатках. У роботі проведено аналіз основних методів атак на вебдодатки, зокрема SQL-ін'єкцій, та розглянуто сучасні підходи до їхнього виявлення та запобігання. Визначено найбільш ефективні методи аналізу вразливостей, зокрема статичний та динамічний аналіз коду, використання автоматизованих сканерів безпеки.

У результаті розроблено та оформлено супровідну документацію до системи виявлення вразливостей: модель загроз, технічне завдання, політику безпеки, план заходів щодо протидії SQL-атакам та рекомендації щодо інтеграції системи в існуючі вебдодатки. Виконано підготовку до впровадження розробленої системи у реальних умовах експлуатації.

28.05.2025 _____

# ABSTRACT

Subject of qualification work: A system for detecting SQL-injection vulnerabilities in web applications.

Author: Vushneskyi Denis Yakovuch.

Head of work: Dzhuly Volodymyr Mykolayovych.

Explanatory note: 65 p., 2 appendices, figures, tables, sources

Graphic part: posters, presentation slides.

VULNERABILITY DETECTION SYSTEM, SQL-INJECTION, WEB APPLICATIONS, INFORMATION SECURITY, VULNERABILITY ANALYSIS, ATTACK PROTECTION, DETECTION METHODS, MONITORING SYSTEM.

The bachelor's qualification thesis is dedicated to the development of a system for detecting SQL-injection vulnerabilities in web applications. The study analyzes the main attack methods targeting web applications, particularly SQL injections, and examines modern approaches to their detection and prevention. The most effective vulnerability analysis methods have been identified, including static and dynamic code analysis and the use of automated security scanners.

As a result, supporting documentation for the vulnerability detection system has been developed and formalized: a threat model, technical specifications, a security policy, an action plan for countering SQL attacks, and recommendations for integrating the system into existing web applications. Preparations have been made for the implementation of the developed system in real-world operational environments.


28.05.2025 _____

ЗМІСТ



| | | | | | КРБКБ.2102138.21.02.02 ПЗ | | |
|---|---|---|---|---|---|---|---|
| Зм. | Арк. | №докум. | Підпис | Дата | | | |
| Виконав | | Вишневський Д.Я. | | 23.05.25 | Система виявлення вразливостей типу «SQL-injection» у вебдодатках | Літера | Аркуш 6 / Аркушів 65 |
| Перевір. | | Джулій В.М. | | | | | |
| Н.контр. | | Мостовий С.В | | 05.06.25 | | ХНУ, КБ-21-2 | |
| Затвер. | | Кльоц Ю.П | | 5.06.25 | | | |

# ВСТУП

Сучасний світ невпинно рухається в напрямку цифрової трансформації, що спричиняє активний розвиток вебдодатків та їхню інтеграцію з різноманітними інформаційними системами. У зв’язку з цим підвищується рівень загроз, пов’язаних із кібербезпекою, а зловмисники шукають нові способи несанкціонованого доступу до даних. Однією з найпоширеніших і найнебезпечніших атак є SQL-ін'єкція, яка дозволяє атакуючим впливати на базу даних вебдодатка, виконуючи несанкціоновані SQL-запити. Це може призвести до отримання конфіденційної інформації, модифікації або видалення даних, а також до порушення цілісності інформаційної системи.

Загроза SQL-ін'єкцій настільки значна, що входить до переліку найкритичніших вразливостей за версією OWASP (Open Web Application Security Project). Атаки цього типу здатні завдати серйозних збитків як фінансовим організаціям, так і урядовим та комерційним установам. Однією з основних причин вразливості вебдодатків до SQL-ін'єкцій є неправильна обробка вхідних даних, відсутність належного контролю запитів до бази даних та недостатня увага до безпеки на етапі розробки програмного забезпечення. Такі проблеми часто виникають через низький рівень обізнаності розробників щодо потенційних загроз та ефективних методів захисту.

Основною метою цієї роботи є дослідження механізмів виявлення та запобігання SQL-ін'єкціям у вебдодатках. Для цього необхідно проаналізувати різні види атак, розглянути існуючі методи їхнього виявлення, а також розробити систему, яка буде ефективно ідентифікувати спроби несанкціонованого доступу та мінімізувати ризики компрометації вебдодатків.[1]

У рамках дослідження буде розглянуто як традиційні методи виявлення SQL-ін'єкцій, що включають статичний та динамічний аналіз, так і сучасні підходи на основі машинного навчання та поведінкового аналізу. Важливим аспектом є тестування та оцінка ефективності запропонованих методів у реальних

умовах, що дозволить визначити оптимальні шляхи підвищення рівня безпеки вебдодатків.

Результати цього дослідження мають значний практичний інтерес, оскільки вони спрямовані на підвищення рівня безпеки вебдодатків, зниження ризиків атак та запобігання витоку конфіденційної інформації. Запропоновані методи та підходи можуть бути використані як у процесі розробки нових програмних продуктів, так і для аналізу безпеки вже існуючих систем. У кінцевому підсумку розробка ефективної системи виявлення SQL-ін'єкцій сприятиме підвищенню загального рівня кібербезпеки та стійкості вебдодатків до атак. Із зростанням популярності вебдодатків та їхньої інтеграції з базами даних значно зросла й кількість атак на такі системи.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ. ПОСТАНОВКА ЗАДАЧІ

## 1.1 Сучасний стан безпеки у веб-додадках

У сучасному цифровому світі веб-додатки відіграють ключову роль у забезпеченні доступу до послуг, інформації, взаємодії з користувачами та веденні бізнесу. Водночас розвиток Інтернету та хмарних технологій призвів до значного збільшення кількості кіберзагроз. Безпека веб-додатків стала однією з найважливіших проблем, з якими стикаються розробники, адміністратори, компанії та користувачі. Найпоширенішими загрозами для веб-додатків залишаються: атаки SQL-ін'єкцій, міжсайтовий скриптинг (XSS), підробка міжсайтових запитів (CSRF), атаки типу «відмова в обслуговуванні» (DDoS), атаки через незахищені API та захоплення сеансів. Звіти OWASP, Veracode, Check Point та інших компаній показують, що кількість атак на веб-додатки зростає з кожним роком. Причинами цього є як недбале програмування, так і відсутність тестування безпеки, застарілі фреймворки, неправильно налаштовані сервери та використання вразливих компонентів з відкритим кодом.

Особливої уваги заслуговують SQL-ін'єкції – атаки типу «ін'єкція», за допомогою яких зловмисники впроваджують шкідливий SQL-код у запити до бази даних. Така атака може призвести до повної компрометації бази даних: отримання конфіденційних даних (логінів, паролів, персональних даних), зміни або видалення інформації, порушення функціонування системи, а іноді навіть отримання доступу до внутрішньої інфраструктури.

Сучасні фреймворки та бібліотеки (наприклад, Django, Laravel, Spring Boot) містять механізми захисту від SQL-ін'єкцій, такі як використання параметризованих запитів, ORM (об'єктно-реляційне відображення) та валідація введених даних. Однак у проектах, створених вручну або на застарілому стеку, ці вразливості залишаються актуальними. [2]

Крім того, з розвитком API та SPA (односторінкових додатків) зросла кількість атак через API, зокрема, через вибір параметрів, несанкціонований доступ, ін'єкції та логічні помилки. Ще одним важливим аспектом є людський

фактор – користувачі використовують прості паролі, розробники залишають адміністративні панелі відкритими, не змінюють стандартні логіни, не обмежують права доступу. Наприклад, навіть у великих компаніях досі трапляються випадки, коли SSH-доступ до продакшн-сервера надається без двофакторної автентифікації.

Для боротьби з цими загрозами активно розробляються автоматизовані системи тестування на проникнення, зокрема, такі інструменти, як OWASP ZAP, Burp Suite, SQLmap. Вони дозволяють виявляти вразливості ще на етапі розробки або тестування програмного забезпечення.

Водночас веб-розробники все частіше використовують інструменти статичного та динамічного аналізу коду, CI/CD-інтегровані інструменти для безперервного тестування безпеки коду, а також впроваджують архітектуру Zero Trust та підходи DevSecOps. DevSecOps інтегрує безпеку безпосередньо в життєвий цикл розробки, забезпечуючи раннє виявлення загроз.

Законодавчі ініціативи також мають свій вплив: у Європі діє Загальний регламент про захист даних (GDPR), у США HIPAA та інші нормативні акти зобов'язують розробників забезпечувати безпеку персональних даних. Порушення безпеки можуть призвести до серйозних фінансових санкцій.

Окремою загрозою є атаки SQL-ін'єкцій, які дозволяють зловмиснику вводити довільні SQL-запити в базу даних, змінюючи поведінку веб-додатку. Незважаючи на тривале існування цієї вразливості, вона залишається однією з найпоширеніших через неправильну або відсутню перевірку введених даних. За даними OWASP (Open Web Application Security Project), SQL-ін'єкції постійно входять до списку найнебезпечніших веб-вразливостей. Атаки такого типу можуть призвести до несанкціонованого доступу до конфіденційної інформації, видалення або зміни даних, а також повного контролю над сервером бази даних.

Для підвищення безпеки сучасних веб-додатків використовуються різні підходи та інструменти. Одним із найефективніших є принцип «глибинно ешелонованого захисту», який передбачає використання кількох рівнів безпеки: перевірка даних на стороні клієнта та сервера, шифрування, обмеження доступу,

ведення журналу подій тощо. Крім того, розробники впроваджують інструменти для динамічного та статичного аналізу коду, тестування на проникнення, а також автоматизовані сканери вразливостей. [3]

Поява контейнеризації, мікросервісної архітектури та хмарних середовищ створює нові виклики безпеці. Наприклад, мікросервіси, які взаємодіють один з одним через API, можуть стати точками входу для атак, якщо не передбачено автентифікацію, авторизацію та обмеження доступу. У свою чергу, постачальники хмарних послуг впроваджують власні інструменти безпеки, але кінцева відповідальність за конфігурацію та налаштування безпеки зазвичай лежить на кінцевому користувачеві.

На тлі зростання кіберзлочинності, особливо через організовані хакерські групи, бізнес та державні установи активно інвестують у кібербезпеку. Це свідчить про усвідомлення важливості цифрового захисту та необхідність розробки нових інструментів для виявлення та нейтралізації загроз.

У контексті розвитку штучного інтелекту (ШІ) та машинного навчання (МН) ці технології почали використовуватися в системах виявлення аномалій, аналізу поведінки користувачів та автоматизованого виявлення атак. Наприклад, системи виявлення/запобігання вторгненням (IDS/IPS) на основі машинного навчання (ML) здатні самостійно виявляти нетипові запити або дії, характерні для таких атак, як SQL-ін'єкції або XSS.

З розвитком цифрових технологій все більше сервісів переходять у веб-простір. Веб-додатки зараз використовуються не лише для електронної комерції, але й у сферах охорони здоров'я, фінансів, освіти, державного управління тощо. Такий масштабний перехід вимагає глибокого переосмислення підходів до безпеки, оскільки кількість атак на веб-ресурси, їх складність та рівень організації зростають з кожним роком.

Особливої уваги заслуговують так звані автоматизовані атаки, коли хакери використовують ботів або спеціальні скрипти для масового тестування веб-сайтів на наявність відомих вразливостей. Такі інструменти, як SQLMap, Wapiti, Nikto та інші, дозволяють зловмисникам швидко сканувати сотні веб-сайтів за лічені

хвилини. Це означає, що навіть випадково виявлені вразливості миттєво експлуатуються.

Не менш серйозною проблемою є використання застарілих або неправильно налаштованих компонентів. Багато організацій використовують сторонні бібліотеки, CMS або фреймворки, які не оновлюються своєчасно або не знають про вже виявлені вразливості. Згідно з OWASP Top 10, це одна з найсерйозніших загроз безпеці. [4]

Ще однією актуальною проблемою є зміна парадигми розробки. Зокрема, перехід на архітектури типу мікросервісів та використання API вимагає нових підходів до захисту. Якщо раніше основна увага приділялася валідації даних форм, то сьогодні величезна кількість запитів здійснюється через API, які також можуть бути вразливими до SQL-ін'єкцій або інших типів атак (наприклад, XSS, CSRF, SSRF тощо).

Людський фактор також є важливим питанням. Багато вразливостей виникають не через технічні обмеження, а через помилки розробників або брак навичок. Наприклад, написання власних функцій для взаємодії з базою даних без використання параметризованих запитів або ORM може призвести до SQL-ін'єкцій. У свою чергу, брак знань з кібербезпеки серед тестувальників та DevOps-фахівців збільшує ризик пропуску небезпечних дефектів.

У зв'язку з цим сучасні організації все частіше впроваджують безпечні практики SDLC (безпечного життєвого циклу розробки програмного забезпечення). Це включає етапи аналізу загроз, написання безпечного коду, автоматизоване тестування на вразливості, використання спеціалізованих сканерів та пен-тестів, а також навчання персоналу. [5]

На державному рівні також розробляється законодавча база у сфері кібербезпеки. Наприклад, у ЄС діє Загальний регламент про захист даних (GDPR), який зобов'язує компанії відповідально підходити до безпеки обробки даних. В Україні діють ДСТУ та рекомендації Державної служби спеціального зв'язку та захисту інформації щодо захисту веб-ресурсів.

Таким чином, сучасний стан безпеки у веб-додатках можна охарактеризувати як динамічний, що вимагає постійної адаптації, навчання та впровадження нових технологій. Безпека вже не є лише завданням адміністраторів – це міждисциплінарний процес, який має охоплювати всю організацію, від розробника до керівника.

## 1.2 Тестування та пошук вразливостей у вебдодатках

Захист веб-додатків є одним з ключових аспектів інформаційної безпеки сьогодні. Зростання кіберзагроз, зосереджених на веб-ресурсах, вимагає постійного моніторингу, тестування та виявлення вразливостей у програмних продуктах, що працюють у мережевому середовищі. Тестування безпеки веб-додатків дозволяє не лише виявити існуючі загрози, але й запобігти потенційним атакам, які можуть призвести до витоку конфіденційної інформації, блокування додатків або отримання несанкціонованого доступу до даних. [6]

Процес тестування безпеки веб-додатків починається з детального аналізу архітектури додатку, його функціональності та типів вхідних даних. Фахівці з кібербезпеки або автоматизовані інструменти намагаються імітувати дії потенційного зловмисника, використовуючи різні методи маніпулювання вхідними даними, обходу автентифікації та отримання несанкціонованого доступу до функціональності або бази даних. Метою таких дій є виявлення слабких місць у системі, які можна використовувати в реальних атаках.

Одним з найпоширеніших підходів до тестування безпеки є динамічне тестування, під час якого додаток тестується в активному режимі. Це означає, що система знаходиться в «живому» стані, обробляє запити та відповіді, а спеціальне програмне забезпечення або спеціаліст аналізує реакцію на підозрілі або шкідливі запити. Такий підхід дозволяє виявити не лише очевидні вразливості, але й логічні помилки, які можуть виникнути під час обробки даних.

Сканування вразливостей часто проводиться за допомогою сканерів безпеки – програм, які автоматично здійснюють вхідні запити до веб-застосунку з попередньо визначеними шкідливими параметрами. Наприклад, за допомогою SQL-ін'єкції зловмисник може отримати доступ до бази даних, змінити або знищити інформацію. Тестування таких сценаріїв передбачає введення спеціальних комбінацій символів у поля форм або адресні рядки для перевірки відповіді серверної частини застосунку. Якщо застосунок не обробляє цей вхід належним чином і видає помилку, це сигналізує про наявність вразливості.

Ще одним важливим аспектом тестування є автентифікація та управління сеансами. Якщо система не обробляє токени сеансу належним чином або не має достатнього контролю над правами доступу, це може дозволити зловмисникам отримати контроль над обліковими записами користувачів. Тестери моделюють атаки, що дозволяє їм побачити, чи можна перехопити, відтворити або змінити токени автентифікації, тим самим отримавши доступ до конфіденційної інформації.

Варто зазначити, що процес тестування є багатоетапним і часто включає як автоматизовані інструменти, так і ручне тестування. Автоматизовані інструменти дозволяють швидко охопити велику кількість URL-адрес, форм, параметрів запитів та надати початковий звіт про підозрілі елементи. Водночас, ручне тестування залишається незамінним для складних логічних тестів або аналізу унікальної функціональності, специфічної для конкретного веб-застосунку. [7]

Тестування веб-застосунків також повинно включати тестування безпеки на стороні клієнта, тобто того, як відбувається взаємодія між браузером користувача та сервером. Наприклад, відсутність належної перевірки даних на стороні клієнта може дозволити зловмисникам підробляти запити або змінювати їхній вміст перед надсиланням на сервер. Також важливо перевірити заголовки HTTP-відповідей, налаштування файлів cookie та інші технічні аспекти, які можуть вплинути на безпеку взаємодії. Крім того, значна увага приділяється питанням оновлень та використання сторонніх бібліотек або фреймворків. Часто вразливості криються не в самому веб-застосунку, а в його залежностях, які розробники можуть

використовувати, не усвідомлюючи, що мають відомі вразливості. Саме тому тестування повинно включати аналіз компонентів, що використовуються в проекті, та перевірку їхньої актуальності та безпеки. Окрему роль у процесі тестування відіграє підхід до оцінки ризиків. Не всі виявлені вразливості є однаково критичними. Деякі можуть становити загрозу лише в рідкісних випадках або за наявності інших факторів, інші ж дозволяють отримати повний контроль над додатком. Тестування також передбачає оцінку потенційного впливу кожної виявленої проблеми, що дозволяє власникам ресурсів приймати обґрунтовані рішення щодо пріоритетності усунення певних загроз.

У процесі тестування веб-додатків також важливо враховувати специфіку середовища, в якому працює система. Різні серверні платформи, операційні системи, бази даних, версії мов програмування – всі ці фактори можуть мати вирішальне значення у виявленні вразливостей. Наприклад, те, що може становити загрозу для PHP-додатку, не обов'язково буде вразливістю в додатку, написаному на Node.js або Python. Тому тестування має бути адаптоване до конкретного технологічного стеку та архітектури системи.

Особливу увагу також слід приділити перевірці прав доступу до різних частин веб-додатку. Вразливості, пов'язані з контролем доступу (так званий «Broken Access Control»), є однією з найнебезпечніших категорій загроз. Вони виникають, коли користувач може отримати доступ до ресурсів або функцій, які йому не призначені. Наприклад, користувач з базовим обліковим записом має можливість переглядати сторінки адміністратора або змінювати дані інших користувачів. Такі помилки часто залишаються непоміченими під час планового тестування та вимагають ретельного аналізу логіки авторизації на кожному рівні взаємодії. [8]

Ще однією важливою сферою є тестування на стійкість до атак міжсайтового скриптингу (XSS). Це одна з найпоширеніших вразливостей, яка дозволяє зловмиснику впровадити шкідливий скрипт на веб-сторінку, яка буде виконана в браузері інших користувачів. Тестування в цьому випадку включає вставку скриптів у форми зворотного зв'язку, поля пошуку, параметри URL-адрес

тощо, щоб переконатися, що програма належним чином очищує вхідні дані перед їх відображенням.

Не менш важливим є тестування захисту від таких атак, як підробка міжсайтового запиту (CSRF), коли зловмисник змушує користувача виконувати небажану дію на сайті, де він пройшов автентифікацію. Системи, вразливі до CSRF, дозволяють виконувати такі операції, як зміна пароля або видалення облікового запису, без підтвердження намірів користувача. Тестування на таку вразливість передбачає аналіз реалізації токенів безпеки у формі (наприклад, токенів CSRF) та перевірку того, чи обов'язково вони передаються з кожним важливим записом. [9]

У сучасній практиці набирає популярності підхід до тестування безпеки під назвою DevSecOps, коли проблеми безпеки інтегруються в усі етапи розробки програмного забезпечення. Такий підхід дозволяє виявляти та виправляти вразливості безпосередньо під час процесу написання коду, а не чекати завершення розробки, використовуючи інтегровані інструменти аналізу безпеки в середовищах розробки, автоматизовані сканери, що запускаються під час кожного оновлення коду, та політики безпеки, що застосовуються до всього життєвого циклу програми.

Ще одним важливим аспектом тестування вразливостей є вибір правильного методу та часу для тестування. У сучасних середовищах DevOps та CI/CD (безперервна інтеграція та розгортання) вразливості необхідно виявляти якомога раніше в життєвому циклі програмного забезпечення. Це означає, що тести безпеки необхідно інтегрувати безпосередньо в процес розробки. Чим раніше виявлено вразливість безпеки, тим дешевше та ефективніше її виправити. Тому розробники повинні мати можливість запускати статичний та динамічний аналіз безпосередньо зі свого середовища розробки або перед комітом коду. [10]

Окрім класичних інструментів тестування, важливо також враховувати роль людського фактора. Часто користувачі (наприклад, розробники чи адміністратори) можуть ненавмисно створювати умови для вразливостей, наприклад, неправильно налаштовуючи сервери або використовуючи застарілі

версії фреймворків та бібліотек. Тому аудит безпеки повинен охоплювати не лише код, а й всю інфраструктуру веб-застосунку: бази даних, сервери, мережеві брандмауери, системи автентифікації та протоколи зв'язку. Під час тестування веб-застосунків на вразливості слід приділяти особливу увагу взаємодії користувача з системою. Форми входу, поля пошуку, фільтри, форми контактів – усі ці елементи можуть бути точками атаки. Навіть ті інтерфейси, які здаються неважливими з функціональної точки зору, можуть містити небезпечні місця для SQL-ін'єкцій або інших атак, особливо якщо немає належної валідації введених даних.

Варто також зазначити, що в деяких випадках атаки SQL-ін'єкцій можуть бути прихованими або ледь помітними – наприклад, коли система лише частково відображає результат помилки або коли база даних налаштована на ігнорування деяких помилкових запитів. У таких випадках виявлення вразливостей стає складнішим, і тестування вимагає поєднання пасивного аналізу та агресивних тестів.

Загалом, тестування веб-додатків на вразливості – це безперервний процес, який слід виконувати регулярно, особливо після оновлень системи, змін функціональності або міграції даних. Безперервне тестування є ключовим компонентом інформаційної безпеки будь-якого сучасного веб-ресурсу.

Таким чином, ефективне тестування безпеки – це складне завдання, яке охоплює як технічні, так і організаційні аспекти. Важливо не лише виявити вразливість, але й належним чином задокументувати її, оцінити ризик та запропонувати ефективні методи її усунення. Безпека веб-додатків ніколи не є станом, якого можна досягти раз і назавжди – це безперервний процес, який вимагає уваги, ресурсів та висококваліфікованих фахівців. [11]

## 1.3 Аналіз сучасних засобів вирішення проблем кібербезпеки у вебдодатках

Зі зростанням залежності від веб-технологій проблема кібербезпеки веб-додатків стає дедалі актуальнішою. У сучасному цифровому середовищі кількість вразливостей та рівень загроз постійно зростають. У зв'язку з цим розробляються нові технології, платформи та інструменти для забезпечення цілісності, конфіденційності та доступності веб-систем. Розглянемо основні сучасні інструменти та підходи, які активно використовуються для вирішення проблем кібербезпеки у сфері веб-розробки.

Одним із найпоширеніших напрямків безпеки є використання брандмауера веб-додатків (WAF). Це програмні або апаратні засоби, які стоять між користувачем та веб-сервером та аналізують HTTP(S)-трафік на наявність підозрілої активності. WAF може блокувати SQL-ін'єкції, XSS-атаки, спроби обійти автентифікацію та інші поширені загрози. Популярні рішення включають: Cloudflare WAF, AWS WAF, Imperva, F5 Advanced WAF. Ці інструменти дозволяють швидко реагувати на нові загрози, налаштовувати правила та використовувати машинне навчання для покращення фільтрації.

Ще одним важливим інструментом є системи динамічного (DAST) та статичної (SAST) аналізу коду. Інструменти DAST (наприклад, OWASP ZAP, Burp Suite, Acunetix) імітують реальні атаки на запущений веб-додаток, аналізуючи його поведінку під час взаємодії з потенційно шкідливими запитами. На противагу цьому, інструменти SAST (наприклад, SonarQube, Checkmarx, Fortify) перевіряють вихідний код або байт-код перед виконанням програми, що дозволяє виявляти вразливості на ранніх стадіях розробки. [12]

У поєднанні з цими інструментами в процесі розробки використовуються технології, інтегровані з безпекою – так званий підхід DevSecOps. Його метою є інтеграція безпеки в кожен етап життєвого циклу розробки програмного забезпечення: від планування до розгортання та підтримки. Це включає автоматизоване сканування залежностей, застосування політик безпеки на рівні CI/CD та регулярний моніторинг активності. Такі інструменти, як GitHub Security,

GitLab Secure або Snyk, дозволяють розробникам отримувати попередження про небезпечні бібліотеки або конфігурації безпосередньо в середовищі розробки.

Сучасні інструменти кібербезпеки для веб-додатків приділяють особливу увагу автентифікації та контролю доступу. Системи на базі OAuth 2.0, OpenID Connect, JSON Web Token (JWT) дозволяють створювати гнучкі та безпечні механізми ідентифікації користувачів. Вони підтримують двофакторну автентифікацію (2FA), обмеження IP-адрес, обмеження часу сеансу та управління правами доступу на основі ролей. Також широко використовуються інструменти єдиного входу (SSO), які підвищують безпеку та зручність для кінцевого користувача.

Моніторинг та ведення журналу є не менш важливими компонентами безпеки сучасних веб-додатків. Системи реєстрації подій (наприклад, ELK Stack, Graylog, Splunk) дозволяють збирати, аналізувати та візуалізувати інформацію про доступ, помилки, підозрілу активність та можливі атаки. Завдяки аналітичним модулям та інтеграції з системами SIEM (Security Information and Event Management), адміністратори безпеки можуть отримувати сповіщення в режимі реального часу та швидко реагувати на інциденти. [13]

Варто також відзначити роль технологій шифрування. Передача даних між клієнтом і сервером повинна здійснюватися за протоколом HTTPS з використанням TLS 1.2 або вище. Багато сучасних веб-фреймворків (наприклад, Django, Spring, Laravel) вже містять вбудовані механізми шифрування паролів (за допомогою алгоритмів bcrypt, Argon2), генерації токенів доступу та захисту від CSRF-атак.

Ще один сучасний підхід – використання архітектури нульової довіри (ZTA), згідно з якою ніхто автоматично не вважається надійним – ні всередині, ні поза мережею. Для веб-додатків це означає багаторівневу автентифікацію, сегментацію доступу, постійний моніторинг поведінки користувачів та перевірку кожного сеансу незалежно від його джерела. Такий підхід значно знижує ризик інсайдерських загроз та ускладнює роботу зловмисників.

Особливу увагу в контексті аналізу сучасних інструментів кібербезпеки слід приділити засобам виявлення SQL-ін'єкцій, одного з найнебезпечніших і водночас найпоширеніших типів атак на веб-додатки. Актуальність цієї вразливості підтверджується тим фактом, що SQL-ін'єкції постійно входять до списку OWASP Top 10, основного світового стандарту для найкритичніших загроз безпеці веб-додатків. [14]

Існує низка спеціалізованих рішень, які зосереджені на виявленні SQL-ін'єкцій. Серед них є сканери вразливостей, які аналізують веб-інтерфейс і відстежують реакцію сервера на певні запити на SQL-ін'єкції. Наприклад, популярний інструмент sqlmap дозволяє автоматизувати процес тестування на SQL-ін'єкції: він створює запити зі шкідливими параметрами, аналізує відповіді та, якщо виявляється вразливість, може навіть здійснювати витік даних з бази даних. Варто також згадати комерційні рішення, що інтегруються в корпоративну інфраструктуру: Acunetix, Netsparker, Burp Suite Professional. Вони підтримують різні типи ін'єкцій, включаючи на основі помилок, об'єднань, часу та булевих значень, і здатні виконувати поглиблене тестування складних веб-додатків, включаючи логіку авторизації та багаторівневі форми.

Окрім сторонніх інструментів, сучасні компанії все частіше створюють власні сканери або плагіни, які інтегруються в їхні системи CI/CD (наприклад, Jenkins або GitHub Actions) та дозволяють автоматизувати тестування SQL-ін'єкцій на кожному етапі оновлення програмного забезпечення. Це сприяє ранньому виявленню проблем до їх появи у виробничому середовищі.

У дипломному проекті, присвяченому створенню системи виявлення вразливостей для SQL-ін'єкцій, важливо враховувати сучасні тенденції та застосовувати як класичні принципи, так і новітні підходи. Зокрема, йдеться про реалізацію графічного інтерфейсу для зручної взаємодії користувача з інструментом, можливість адаптації корисних навантажень до певної структури веб-запитів, а також використання логіки для фільтрації хибних спрацьовувань. Такий підхід дозволяє створити універсальний, інтуїтивно зрозумілий та ефективний інструмент для сканування веб-ресурсів на наявність SQL-ін'єкцій.

Результати, отримані в процесі розробки такого інструменту, можуть стати основою для подальшої інтеграції в більші системи контролю безпеки або бути використані як навчальний ресурс для підготовки фахівців з кібербезпеки. [15]

Таким чином, сучасні засоби вирішення проблем кібербезпеки у веб-додатках охоплюють широкий спектр технологій, підходів та інструментів. Їх грамотне використання дозволяє створювати більш безпечні, стійкі до атак веб-додатки та мінімізувати ризики витоку, знищення або компрометації даних. Однак важливо розуміти, що жоден окремий інструмент не є панацеєю – ефективна безпека досягається лише за допомогою комплексного підходу, регулярних оновлень та навчання персоналу.

## 1.4 Постановка задачі

В рамках поставленого завдання необхідно реалізувати низку конкретних технічних та дослідницьких завдань, кожне з яких має забезпечити досягнення загальної мети дипломного проекту – створення ефективної системи виявлення SQL-ін'єкцій у веб-додатках. Вирішення цих завдань дозволить сформувати готовий програмний продукт з функціональністю, достатньою для виявлення потенційних вразливостей та їх подальшого аналізу.

Першим і основним завданням є розробка структури програмного забезпечення, яка передбачає логічне розділення модулів відповідно до принципів архітектурного проектування. Зокрема, необхідно створити окремі модулі для обробки графічного інтерфейсу, взаємодії з HTTP-запитами, генерації корисних навантажень SQL, аналізу відповідей сервера, ведення журналу та управління вводом користувача. Такий підхід гарантує масштабованість та зручність обслуговування системи.

Наступним ключовим завданням є реалізація функціональності для тестування веб-ресурсів на вразливості. Це включає створення механізму динамічної генерації корисних навантажень SQL для запитів GET та POST,

автоматизоване виявлення параметрів, які можуть взаємодіяти з базою даних, та алгоритмів відправлення сформованих запитів на цільовий веб-сервер. У цьому контексті важливо забезпечити підтримку як класичних схем передачі даних (параметри URL, HTML-форми), так і більш сучасних форматів (JSON, AJAX).

Ще одним важливим завданням є розробка модуля для обробки та аналізу відповідей сервера. Цей модуль повинен мати можливість виявляти текстові або поведінкові індикатори SQL-ін'єкцій, такі як повідомлення про синтаксичні помилки, зміни в структурі HTML, аномальні затримки відповіді тощо. Для цього необхідно реалізувати правила фільтрації та шаблони для розпізнавання типових ознак вразливості, враховуючи можливість обходу простих механізмів маскування.

Особливу увагу слід приділити створенню зручного графічного інтерфейсу користувача. Інтерфейс повинен дозволяти вводити URL-адресу для аналізу, вибирати режим сканування (наприклад, поверхневе сканування або поглиблений аналіз), відображати результати сканування з детальною інформацією (тип виявленої вразливості, місце виявлення, відповідь сервера) та мати можливість зберігати файли журналів для подальшого вивчення.

Також серед завдань є побудова та налаштування безпечного тестового середовища. Для цього необхідно встановити вразливий веб-застосунок DVWA на локальний сервер, налаштувати його на максимальний рівень вразливості, забезпечити роботу PHP та MySQL, а також ізолювати середовище від Інтернету. Це дозволить протестувати систему в умовах, наближених до реальних, але без ризику для зовнішніх інформаційних ресурсів.

Крім того, необхідно реалізувати механізм реєстрації всіх дій системи з можливістю збереження інформації про URL-адреси, час сканування, тип запитів, виявлені вразливості та отримані відповіді сервера. Це забезпечить прозорість системи та дозволить подальший аналіз ефективності підходів.

Останнім, але не менш важливим завданням є експериментальне тестування розробленої системи. Необхідно провести серію тестів з використанням реальних сценаріїв на DVWA, порівняти результати з відомими вразливостями, оцінити

точність виявлення, кількість хибних спрацьовувань, швидкість виконання та стабільність програми.

Таким чином, в рамках дипломної роботи необхідно реалізувати такі основні завдання:

– Проектування архітектури програмного забезпечення з модульною структурою;

– Розробка механізмів генерації корисних навантажень SQL та автоматизованої генерації запитів;

– Розробка алгоритмів аналізу відповідей сервера з виявленням прямих та непрямих ознак вразливості;

– Створення графічного інтерфейсу користувача з можливістю керування тестами та перегляду результатів;

– Налаштування тестового середовища на базі DVWA для безпечних експериментів;

– Впровадження системи реєстрації результатів тестування;

– Проведення експериментального тестування з подальшим аналізом ефективності розробленої системи.

Комплексна реалізація цих завдань дозволить не лише досягти поставленої мети, але й створити програмний інструмент, що відповідає сучасним вимогам до засобів безпеки веб-ресурсів.

# 2 ПРОЕКТУВАННЯ СИСТЕМИ

## 2.1 Вразливості та порушення безппеки у сучасних веб-додатків атакою SQL-injection

Вразливості веб-додатків є однією з найактуальніших проблем інформаційної безпеки, оскільки переважна більшість інформаційних систем сьогодні реалізовані як веб-сервіси. Однією з найпоширеніших та найнебезпечніших загроз залишається атака SQL-ін'єкцій. Цей тип атаки дозволяє зловмиснику втручатися в роботу бази даних, впроваджуючи шкідливі SQL-інструкції у вхідні параметри веб-форми, запиту або URL-адреси. Причиною такої вразливості зазвичай є відсутність належної фільтрації та захисту даних, що вводяться користувачем. Це дозволяє зловмиснику змінювати логіку SQL-запиту та виконувати довільний код у базі даних. Найкритичнішими наслідками SQL-ін'єкцій можуть бути витік персональних даних, несанкціонований доступ до облікових записів, зміна або видалення важливої інформації, а в деяких випадках повний контроль над сервером. Особливо небезпечними є атаки на системи, що зберігають конфіденційну або фінансову інформацію, оскільки витік таких даних може призвести до значних репутаційних та матеріальних втрат. Варто зазначити, що SQL-ін'єкції не вимагають від зловмисника наявності складних технічних навичок або використання спеціальних інструментів – достатньо розуміння основ мови SQL та базової структури запитів. Це робить атаку особливо привабливою для новачків у сфері кіберзлочинності. [16]

Механізм SQL-ін'єкцій базується на тому, що динамічні SQL-запити формуються без розрізнення даних та логіки, що дозволяє зловмиснику змінювати структуру запиту. Наприклад, замість очікуваного імені користувача у форму входу вводиться фраза типу 'OR '1'='1, яка перетворює запит на такий, що завжди повертає true. Таким чином, система забезпечує доступ навіть без введення реальних облікових даних. Якщо обліковий запис має права адміністратора, наслідки можуть бути катастрофічними.

SQL-ін'єкції поділяються на кілька типів. Класичний варіант, або так звана in-band injection, передбачає використання одного й того ж каналу для надсилання шкідливого коду та отримання відповіді. Це найпростіший і найчастіше реалізований тип атаки. Сліпа SQL-ін'єкція базується на логічному аналізі поведінки програми у відповідь на певні умови, оскільки система безпосередньо не повертає результати запиту, але її реакція все одно дозволяє визначити наявність вразливості. Позасмугові ін'єкції використовуються рідше та вимагають складнішої інфраструктури, але вони дозволяють виводити дані на зовнішні сервери, минаючи обмеження на пряму взаємодію. Розвиток веб-технологій не зменшив ризику SQL-ін'єкцій. Навпаки, через поширення складних систем управління контентом (CMS), фреймворків, REST-API та мікросервісної архітектури, кількість точок входу в систему збільшилася, що створює більше потенційних можливостей для атак. Часто веб-додатки створюються з використанням сторонніх бібліотек, компоненти яких не завжди тестуються на безпеку, що значно ускладнює контроль цілісності всього продукту. Існують численні приклади вразливих плагінів або модулів, які дозволяють зловмисникам отримати доступ до всієї бази даних додатка, що ще раз доводить важливість глибокого аналізу коду та залежностей. [17]

Ще однією проблемою, яка сприяє успішним атакам SQL-ін'єкцій, є людський фактор. Недосвідчені розробники можуть нехтувати основними принципами безпечного програмування, використовувати застарілі підходи до обробки запитів або не забезпечувати належної обробки винятків. Крім того, через тиск дедлайнів або недостатню увагу до етапу тестування, додатки потрапляють у продакшн зі значними недоліками безпеки. Оцінка рівня ризику SQL-ін'єкцій дозволяє краще зрозуміти масштаб проблеми. Згідно з рейтингами OWASP, ця вразливість традиційно входить до десятки найнебезпечніших. У багатьох випадках SQL-ін'єкції стали відправною точкою для масштабних атак на великі компанії, урядові веб-сайти, банківські системи тощо. Однак навіть базовий аудит безпеки часто дозволяє виявити та усунути потенційні слабкі місця, що знижує ймовірність компрометації. SQL-ін'єкція – одна з найнебезпечніших і

найпоширеніших вразливостей у веб-безпеці, яка дозволяє зловмисникам втручатися в роботу бази даних веб-додатку, впроваджуючи шкідливі SQL-запити через поля введення. Вразливість виникає, коли розробник динамічно генерує SQL-запити, вставляючи в них дані без належної санітарної обробки та екранування. Наприклад, якщо адреса сторінки така: http://site.com/page?id=5, а сервер генерує SQL-запит типу SELECT * FROM articles WHERE id=5, то зміна параметра id на 5 OR 1=1 змінить SQL-запит на SELECT * FROM articles WHERE id=5 OR 1=1, і це вже дозволяє зловмиснику отримати всі записи. [18]

SQL-ін'єкція буває різних типів. Найпростіша – класична (іноді її називають «вбудованою») ін'єкцією, коли атака безпосередньо змінює структуру запиту. Існує також «сліпа SQL-ін'єкція», коли відповідь не містить даних, але можна визначити наявність вразливості, змінивши поведінку сторінки. Також існує ін'єкція на основі часу, коли затримки в роботі сервера сигналізують про успішне виконання запиту.

Типові цілі атаки: отримання вмісту таблиць бази даних (наприклад, користувачів, паролів, кредитних карток), відображення імен користувачів або системних ролей; видалення таблиць (DROP TABLE), виконання додаткових команд (у деяких системах – навіть системних команд, через xp_cmdshell у MS SQL).

Для успішного виконання атаки зловмисник може протестувати URL-адреси, форми введення, поля пошуку тощо. Поширені пейлоади, які часто використовуються для виявлення вразливостей:

- ' OR '1'='1
- ' UNION SELECT null, version() --
- ' AND SLEEP(5) --
- 1' OR 1=1--

Багато прикладів таких атак легко відтворити в системі DVWA (Damn Vulnerable Web Application), яка спеціально розроблена для навчання етичних хакерів та тестування систем безпеки. Встановивши DVWA на локальний сервер, можна вибрати рівень захисту (низький, середній, високий, неможливий) та

побачити, як змінюється поведінка програми залежно від налаштувань. Важливо підкреслити, що SQL-ін'єкція – це не просто «проблема програміста» – це вразливість, яка виникає на інтерфейсі програмного коду, архітектури програми, бази даних і навіть налаштувань сервера. Якщо розробник використовує динамічні запити без параметризації, адміністратор допускає відображення SQL-помилок на фронтенді, а сервер бази даних допускає небезпечні функції – все це робить веб-додаток надзвичайно вразливим. Тому вивчення SQL-ін'єкцій включає не лише вивчення атак, але й аналіз причин, чому вони стають можливими: людські помилки, недбалість у налаштуванні середовища, відсутність валідації даних, застарілі фреймворки або просто погане розуміння принципів безпеки під час розробки. Особливої уваги заслуговують так звані комбіновані атаки, в яких SQL-ін'єкції використовуються як проміжний етап для подальших маніпуляцій — таких як отримання доступу до файлової системи, обхід авторизації, впровадження шкідливого коду або навіть виконання віддалених системних команд. У складних сценаріях зловмисник може використовувати вразливість для створення нового облікового запису адміністратора, знищення журналів безпеки або зміни конфігурацій програми. Варто також зазначити, що SQL-ін'єкції часто поєднуються з іншими типами атак. Наприклад, у поєднанні з XSS (міжсайтовий скриптинг) або CSRF (міжсайтова підробка запитів) атаки стають ще небезпечнішими, оскільки дозволяють обійти механізми захисту сеансів, авторизації або доступу. Загрози SQL-ін'єкцій можуть мати різні наслідки. [19]

По-перше, вони можуть призвести до витоку конфіденційних даних, таких як ідентифікатори користувачів, паролі, фінансова інформація та бізнес-аналітика. Це може призвести до значних фінансових втрат, штрафів регуляторних органів та втрати довіри користувачів.

По-друге, атаки SQL-ін'єкцій можуть використовуватися для зміни або знищення даних у базі даних, що може порушити роботу організації, призвести до збою критично важливих служб або навіть закриття бізнесу. По-третє, деякі складні атаки можуть дозволити зловмиснику виконувати команди на сервері, що може призвести до розповсюдження шкідливого програмного забезпечення або

використання інфраструктури компанії для подальших атак. [20] Основні загрози SQL-ін'єкцій та їх наслідки показані в таблиці 2.1.

Таблиця 2.1 – Основні загрози SQL-ін'єкцій та їх наслідки.

| Тип загрози | Опис | Можливі наслідки |
|---|---|---|
| Несанкціонований доступ | Використання SQL-ін'єкцій для отримання доступу до конфіденційної інформації. | Витік персональних даних, фінансових звітів, комерційної таємниці. |
| Модифікація або видалення даних | Атакуючий може змінювати або видаляти дані в базі даних. | Втрата важливої інформації, збій у роботі вебдодатка, фінансові втрати. |
| Обхід автентифікації | Зловмисник може змінити запит таким чином, щоб обійти перевірку логіна та пароля. | Неавторизований вхід до адміністративних панелей, викрадення акаунтів. |
| Виконання довільних команд | Використання SQL-ін'єкцій для виконання шкідливих команд на сервері бази даних. | Компрометація всієї системи, встановлення шкідливого ПЗ, використання серверних ресурсів для подальших атак. |
| Захоплення контролю над додатком | Використання складних SQL-ін'єкцій для отримання повного контролю над вебдодатком. | Повне знищення або підміна вебресурсу, фінансові та репутаційні втрати. |

Варто наголосити, що активна боротьба з SQL-ін'єкціями вимагає не лише розробки безпечного коду, але й комплексної стратегії інформаційної безпеки, яка включає постійний моніторинг, автоматизоване тестування, багаторівневу перевірку введених даних, шифрування важливої інформації та суворе обмеження прав доступу. Всі ці заходи повинні бути інтегровані в життєвий цикл веб-додатку на етапі його проектування та тестування. [21]

Щоб краще зрозуміти масштаб ризику, який створює SQL-ін'єкція, необхідно враховувати, що ця атака надає зловмиснику непрямий контроль над базою даних через звичайний веб-інтерфейс. Іншими словами, атака дозволяє перетворити, здавалося б, безпечну форму введення або параметр URL-адреси на повноцінну командну оболонку до бази даних. Це загрожує цілісності, конфіденційності та доступності даних. І найгірше: іноді користувач може навіть не підозрювати, що його дії призвели до витоку, оскільки більшість атак SQL-ін'єкцій залишають мінімум слідів у веб-додатку.

Одним з важливих аспектів є автоматизований пошук таких вразливостей. У промисловій практиці зловмисники використовують спеціальні сканери, такі як модулі SQLmap, Havij або навіть Metasploit Framework. Вони здатні автоматично визначати тип СУБД, обходити базовий захист, порушувати авторизацію та зчитувати таблиці з конфіденційною інформацією. Саме тому ручна перевірка — хоча й корисна для розуміння механізму — не завжди ефективна за великої кількості точок входу (наприклад, на великому веб-сайті або в CRM-системі).

У розробленій мною системі я додав підтримку ручного введення файлів cookie, що дозволяє перевіряти наявність вразливостей навіть у закритих розділах сайту, доступних лише після авторизації. Це важливо для реальних веб-додатків, де захист часто реалізовано лише на рівні обмеження доступу до URL-адреси, але не на рівні структури запиту до бази даних. Таким чином, зловмисник, маючи доступ до звичайного облікового запису користувача, може здійснити ін'єкцію в адміністративні або конфіденційні розділи ресурсу. [22]

Ще одним важливим аспектом є аналіз поведінки захищених систем. Наприклад, якщо веб-додаток відповідає на запит ін'єкції стандартною помилкою HTTP 403 або просто перезавантажує сторінку, не відображаючи результат, це не гарантує захисту. Вразливість може залишитися, її результати просто приховані. Саме тому моя програма враховує не лише вміст відповіді сервера, але й час відповіді, код стану HTTP та наявність ключових слів, що вказують на виконання SQL-команди. У цьому допомагає механізм логування – всі відповіді та результати тестування записуються у файл, що дозволяє проводити повторний

аналіз навіть після закриття програми. Важливо розуміти, що навіть якщо тест не виявляє вразливостей, це не дає абсолютної гарантії безпеки. Успішний захист передбачає багаторівневу архітектуру: правильне використання підготовлених операторів (на рівні коду), обмеження прав користувачів у базі даних (на рівні СУБД), фільтрацію на рівні веб-сервера та наявність логування інцидентів.

## 2.2 Проектування діаграми потоків даних системи виявлення вразливостей у веб-додатках.

Розробка діаграми потоку даних (DFD) є одним з ключових етапів створення системи виявлення вразливостей для веб-додатків, зокрема тих, що пов'язані з SQL-ін'єкціями. Діаграма потоку даних дозволяє візуально описати логіку обробки інформації, зв'язки між компонентами системи, джерелами даних та одержувачами на ранній стадії. Це важливо як з точки зору моделювання функціональності системи, так і для забезпечення чіткого розуміння всіх операцій, що виконуються над вхідною, проміжною та вихідною інформацією в процесі виявлення загроз. [23]

В рамках розробки системи виявлення SQL-ін'єкцій розробляється діаграма контексту рівня 0, яка показує загальну структуру системи, взаємодію з користувачем, базами даних та зовнішніми джерелами інформації. Основним зовнішнім агентом у такій моделі є користувач, який надає вихідні дані - URL-адресу веб-додатку, що перевіряється. Система отримує ці вхідні дані та виконує аналіз, який включає автоматизовану перевірку на вразливості шляхом генерації тестових запитів з різними шкідливими параметрами. Результати аналізу повертаються користувачеві у вигляді звіту про виявлені або відсутні вразливості.

На логічному рівні обробка даних у системі умовно поділяється на кілька послідовних етапів. Перший етап – прийняття вхідного параметра, який включає початкову URL-адресу веб-застосунку. Цей параметр передається на обробку модулю сканування, який реалізує механізм генерації SQL-запитів корисного

навантаження. Модуль виконує серію HTTP-запитів до цільового ресурсу зі зміненими параметрами, які можуть спровокувати вразливість, наприклад, через порушення логіки запитів до бази даних. [24]

Наступний етап – інтерпретація відповіді, отриманої від сервера веб-застосунку. Відповідь аналізується на наявність певних маркерів, таких як повідомлення про помилки бази даних, ключові слова (наприклад, «синтаксична помилка», «SQL», «база даних»), а також поведінкові відхилення у відповідях. Якщо система виявляє ознаки можливої вразливості в тексті, вона позначає відповідний параметр як небезпечний та зберігає результат у внутрішній структурі даних.

Після завершення сканування формується звіт, який передається користувачеві. Звіт може містити список запитів корисного навантаження, які викликали потенційно небезпечну реакцію веб-застосунку, а також загальну оцінку рівня ризику. Якщо вразливостей не виявлено, користувач отримує підтвердження безпечного стану сканованого ресурсу. [25]

Модель системи також передбачає ведення журналу даних сканування. Це дозволяє зберігати історію сканування, переглядати попередні результати, а також створює можливість для подальшого аналізу, що може бути корисним у разі хибнопозитивних або хибнонегативних результатів. З точки зору потоку інформації, реєстрація виступає як окремий напрямок обробки, який одночасно отримує ті ж дані, що й основний інтерфейс користувача, але зберігає їх у внутрішньому архіві.

Важливо наголосити, що розробка діаграми потоку даних допомагає виявити можливі проблеми на ранніх етапах проектування. Наприклад, під час моделювання можна виявити надлишкові або невиправдані операції, погано захищені вузли системи або недоліки в логіці передачі даних. DFD-діаграма також спрощує процес комунікації між розробниками, тестувальниками та іншими учасниками проекту, оскільки забезпечує візуальне представлення складної технічної інформації. [26]

В рамках розробки цієї системи планується створення кількох рівнів DFD - від моделі загального контексту до більш детальних діаграм. Таке багаторівневе проектування дозволяє зберегти цілісність архітектури та одночасно вказати функціональні компоненти на різних етапах розробки. Глибокий аналіз потоку даних забезпечує цілісне уявлення про систему та створює міцну основу для подальшого впровадження захисту від вразливостей, таких як SQL-ін'єкції у веб-додатках.

Розробка діаграми потоку даних також дозволяє чітко ізолювати модуль перевірки SQL-ін'єкцій як окремий компонент, який можна повторно використовувати або замінювати без зміни інших частин системи. Такий підхід, відомий як модульність, значно спрощує обслуговування та майбутню розробку програмного забезпечення. Наприклад, якщо з'являться нові типи вразливостей або точніші методи сканування, нові модулі можна інтегрувати в систему без необхідності повністю переробляти її архітектуру. З технічної точки зору, створення діаграми потоку даних також допомагає виявити потенційні вузькі місця в системі, які можуть спричинити затримки обробки або втрату даних. Наприклад, надмірні запити до сервера, погане ведення журналу або погана обробка помилок можуть знизити точність аналізу та збільшити ймовірність помилкових результатів. Завдяки DFD-діаграмам ці проблеми можна передбачити на етапі планування та внести необхідні корективи до етапу реалізації.[28]

Враховуючи специфіку завдання — виявлення SQL-ін'єкцій — діаграма також повинна моделювати логіку прийняття рішень, зокрема механізми визначення того, чи є відповідь сервера доказом вразливості. Для цього система повинна аналізувати вміст HTML-відповідей, шукати характерні закономірності в повідомленнях про помилки, використовувати евристику або навіть машинне навчання. На діаграмі це можна представити як окремий підпроцес аналізу відповідей, який отримує дані з HTTP-запитів і передає результат до модуля генерації звітів.

Ще одним важливим аспектом є взаємодія системи з базою знань або репозиторієм відомих вразливостей. У майбутньому систему можна розширити

функціональністю доступу до відкритих баз даних, таких як OWASP або NVD, для підвищення точності аналізу. У цьому випадку DFD повинна передбачати відповідні зовнішні джерела та відповідні потоки даних, що забезпечують обмін інформацією.

Таким чином, діаграма потоків даних є важливим аналітичним інструментом, який дозволяє не тільки розробляти архітектуру системи, але й оцінювати її ефективність, безпеку та масштабованість. Завдяки детальному моделюванню інформаційних потоків розробник може чітко уявляти логіку кожного елемента системи, що значно знижує ризик помилок під час впровадження. [29]

Важливо також враховувати аспект взаємодії між потоками даних, які можуть відбуватися одночасно. Наприклад, під час сканування сайт може генерувати різні відповіді залежно від налаштувань безпеки, наявності брандмауерів або систем запобігання вторгненням (WAF). У цьому випадку діаграма повинна моделювати можливість асинхронної обробки відповідей та реєстрації нетипових подій. Це дозволить системі бути більш адаптивною та гнучкою у роботі з різними веб-ресурсами.

Крім того, при проектуванні потоків даних доцільно включати етапи реєстрації дій системи. Збереження історії перевірок, результатів сканування, виявлених помилок та винятків дозволяє не лише відстежувати ефективність програми, але й використовувати ці дані для подальшого аналізу або навчання моделей машинного навчання. Відповідно, структура DFD повинна передбачати сховище журналів та відповідні процеси для їх запису та доступу до них.

Система виявлення SQL-ін'єкцій також повинна забезпечувати зворотний зв'язок з користувачем. Важливо, щоб результати сканування були представлені у зрозумілій формі, наприклад, із зазначенням потенційно небезпечних параметрів URL-адрес, використаних корисних навантажень та типу відповіді сервера. На діаграмі це можна представити як окремий процес генерації звітів, який отримує дані від модуля аналізу та передає їх до інтерфейсу користувача. [30]

Ще одним аспектом, який впливає на проектування DFD, є питання масштабованості. Якщо в майбутньому система буде адаптована для одночасного сканування великої кількості сайтів або інтеграції з іншими інструментами кібербезпеки, то діаграма повинна передбачати відповідні точки інтеграції та балансування навантаження на ранній стадії. Це може включати, наприклад, постановку запитів у чергу або розподіл завдань між окремими процесами паралельно. Загалом, діаграма потоку даних відіграє важливу роль у забезпеченні цілісності та узгодженості всієї системи. Його правильне та детальне проектування дозволяє зменшити кількість помилок у коді, підвищити ефективність реалізації та забезпечити чітку структуру для подальшої модифікації, тестування або розширення.

## 2.3 Проєктування ієрархії задач системи виявлення вразливостей типу «sql injection» у вебдодатках

Під час розробки системи виявлення вразливостей SQL-ін'єкцій у веб-додатках одним із найважливіших компонентів є правильне формулювання та проектування завдання. Це дозволяє чітко визначити мету, визначити межі дослідження та вказати функціональність, яка має бути реалізована в програмному продукті. Проектування завдання – це перехід від загальної ідеї безпеки веб-ресурсів до чітко сформульованої технічної мети, яку можна реалізувати за допомогою алгоритмів, засобів розробки та існуючих технологій.

В рамках теми дипломної роботи основним завданням є виявлення вразливостей SQL-ін'єкцій, які можуть бути присутніми у веб-додатках через некоректну обробку даних, введених користувачем, у запитах до бази даних. З технічної точки зору це означає необхідність автоматизованої відправки спеціально згенерованих вхідних даних на веб-сайт та аналізу відповіді сервера з метою виявлення поведінкових або текстових ознак потенційної вразливості. Таким чином, суть завдання полягає в розробці інструменту, який може тестувати

задану URL-адресу, динамічно додавати тестові дані до параметрів запиту та інтерпретувати відповіді сервера.

Під час проектування системи виявлення вразливостей, такої як SQL-ін'єкція, особливу увагу слід приділяти ієрархії завдань, оскільки вона забезпечує масштабованість, чітку логіку взаємодії між компонентами та адаптивність системи до різних типів веб-застосунків та сценаріїв тестування.

На початковому етапі система отримує вхідні дані від користувача, зокрема, URL-адресу, що тестується, та, за необхідності, файли cookie для авторизації в захищених розділах сайту. На цьому рівні відбувається базова перевірка введеної інформації, що дозволяє підготуватися до коректної побудови тестових запитів.

Далі система переходить до формування корисних навантажень SQL, які додаються до потенційно вразливих параметрів в URL-адресі, таких як id=, або до інших змінних, залежно від реалізації. Для цього використовується окремий модуль, який містить попередньо налаштовані параметри ін'єкції. На цьому рівні генеруються десятки модифікованих HTTP-запитів з увімкненими SQL-ін'єкціями, що дозволяє оцінити реакцію сервера на різні методи обходу захисту, наприклад, за допомогою конструкцій типу ' OR 1=1-- або " UNION SELECT NULL,NULL--.

Після генерації запитів система передає їх на обробку модулю, який здійснює HTTP-зв'язок з веб-сервером. Тут формується повна структура запиту, додаються файли cookie, якщо такі були вказані, та аналізується відповідь сервера. Програма порівнює отримані відповіді з базовим контрольним запитом, виявляє відхилення або специфічні помилки, які можуть свідчити про успішність SQL-ін'єкції. [31]

Паралельно з виконанням тестів фіксуються всі отримані результати. Окремий модуль реалізує зберігання даних про кожен запит у файлі журналу, включаючи URL-адресу, використане корисне навантаження, відповідь сервера, стан вразливості та час виконання. Завдяки цьому користувач може зберігати або переглядати деталі тестування навіть після завершення процесу.

Результати тестування відображаються в графічному інтерфейсі, який забезпечує інтуїтивно зрозуміле представлення. отримана інформація. Такий підхід дозволяє інтерактивну взаємодію з користувачем та швидкий... зворотний зв'язок щодо безпеки ресурсу, що тестується. Вигляд ієрархії виконання задачі показано на рисунку 2.1.


Рисунок 2.1 – Ієрархії виконання задачі

Таким чином, загальний обсяг завдання включає не лише написання алгоритму сканування веб-додатків, але й забезпечення повноцінної інфраструктури для тестування. Сервер повинен підтримувати стандартні веб-

технології, зокрема PHP та MySQL, а також мати налаштоване середовище, в якому DVWA працюватиме в режимі максимальної вразливості. Це дозволить штучно створювати ситуації, коли SQL-ін'єкції можна буде виявляти без втручання в реальні ресурси, дотримуючись при цьому етичних норм та законодавчих вимог щодо безпеки інформаційних систем. [32]

Розробка самої системи виявлення передбачає створення застосунку з графічним інтерфейсом користувача, який дозволяє ввести URL-адресу веб-ресурсу, розпочати сканування та отримати результат. Для реалізації інтерфейсу було обрано мову програмування Python з використанням бібліотеки Tkinter, яка забезпечує легкість побудови елементів вікна, а також модулів запитів та BeautifulSoup для роботи з HTTP-запитами та обробки відповідей сервера.

Така постановка задачі дозволяє комплексно підійти до створення програмного рішення — від побудови середовища моделювання до реалізації системи, здатної автоматично шукати вразливості та демонструвати результат на прикладі реального веб-додатку, спеціально створеного для такого тестування.

## 2.4 Алгоритм роботи системи виявлення вразливостей у вебдодатках

У процесі побудови системи для виявлення вразливостей типу SQL Injection у вебдодатках одним із ключових аспектів є чітко визначений і формалізований алгоритм її функціонування. Розроблений алгоритм повинен охоплювати всі етапи, починаючи від ініціалізації користувачем перевірки, закінчуючи інтерпретацією результатів і фіксацією виявлених вразливостей у лог-файлі. В основі розробленої системи лежить модульний підхід, що дозволяє кожному компоненту алгоритму функціонувати автономно, забезпечуючи водночас узгоджену взаємодію з іншими модулями. Розроблений алгоритм роботи програми показано на рисунку 2.2.

Система починає роботу з ініціалізації графічного інтерфейсу користувача, який реалізовано за допомогою бібліотеки Tkinter. У графічному вікні користувач

може ввести URL-адресу веб-сторінки, яку потрібно перевірити на вразливості. Крім того, є опція вставки URL-адреси з буфера обміну для зручності користувача. Після цього користувач може вказати значення cookie, якщо веб-додаток вимагає автентифікації або використовує параметри сеансу. Ці дані негайно передаються системі як частина HTTP-запиту під час тестування. [33]

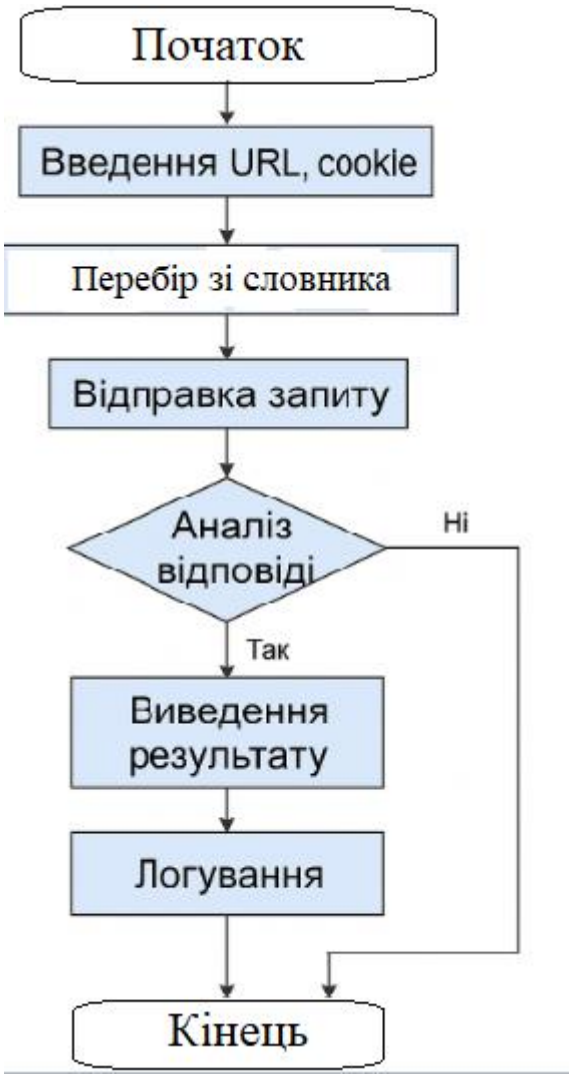



Рисунок 2.2 – Алгоритм роботи програми.

Після введення необхідних параметрів запускається основний модуль тестування. На цьому етапі викликається логіка з модуля sqli_tester.py, який виконує попередню обробку URL-адрес та формує список готових до використання запитів. На цьому етапі особлива увага приділяється перевірці

введених даних. Система перевіряє, чи була введена правильна URL-адреса, чи містить вона параметри, які можна вводити, та чи є введені значення потенційно шкідливими для самої програми.

Наступний крок – завантаження корисних даних SQL. Вони зберігаються в окремому модулі paydays.py та являють собою набір попередньо визначених SQL-ін'єкцій, що охоплюють найпоширеніші вектори атак. Ці корисні навантаження створюються на основі відомих шаблонів шкідливих запитів, таких як 'OR '1'='1, ' UNION SELECT, а також складніших варіацій з використанням sleep() для перевірки затримок у часі під час сліпої SQL-ін'єкції. Корисні навантаження завантажуються динамічно під час кожного тестового запуску, що дозволяє гнучко додавати нові варіанти атак до списку без необхідності змінювати основний код. [34]

Після завантаження корисних навантажень система починає основний цикл тестування. Для кожного корисного навантаження генеруються спеціалізовані HTTP-запити та надсилаються на сервер із введеним значенням, що підставляється в параметр запиту. Під час цього циклу система використовує бібліотеку запитів, яка дозволяє керувати параметрами заголовка, файлами cookie, тайм-аутами та отримувати повні HTTP-відповіді. Важливою частиною цього етапу є обробка відповідей сервера. Алгоритм аналізує кожну відповідь, порівнюючи її з еталонною (тобто відповіддю без ін'єкції) за кількома параметрами: код стану, довжина тіла відповіді, специфічні шаблони помилок бази даних та вміст сторінки на наявність типових SQL-помилок.

Якщо під час обробки відповіді система виявляє суттєві розбіжності, що вказують на можливу вразливість, вона автоматично фіксує відповідне корисне навантаження як потенційно небезпечне. Критеріями визначення вразливості є не лише наявність повідомлень про помилки, таких як синтаксична помилка SQL, незакрита цитата або ODBC, але й специфічні закономірності, характерні для помилок у певних СУБД, таких як MySQL, MSSQL, Oracle або PostgreSQL. Таким чином, алгоритм дозволяє точніше визначити, на яку СУБД орієнтований вебзастосунок, що ще більше покращує якість діагностики.

Варто зосередитися на логіці виявлення сліпої SQL-ін'єкції. Цей тип ін'єкції не викликає очевидних помилок у відповіді сервера, тому алгоритм використовує метод вимірювання часу відповіді на спеціальні запити, які викликають затримку у виконанні. Наприклад, команда SLEEP(5) може бути вставлена в корисне навантаження, і якщо відповідь від сервера надходить із затримкою, що перевищує стандартний поріг, система фіксує цей випадок як потенційну сліпу SQL-ін'єкцію. Такий підхід дозволяє виявляти приховані вразливості, які неможливо виявити звичайними засобами.

Після завершення циклу тестування всі зібрані результати передаються до модуля logger.py, який забезпечує ведення журналу. URL-адреса, яка була протестована, корисне навантаження, яке викликало відповідь системи, та результат тестування записуються у файл журналу. Це дозволяє додатково аналізувати статистику тестування, виявляти повторювані закономірності та надавати аудиторське підтвердження виконаних тестів. Крім того, збереження результатів у файл журналу є обов'язковим елементом з точки зору безпеки та прозорості системи.

Одночасно з реєстрацією система оновлює графічний інтерфейс – відображає результати на екрані у вигляді повідомлень, де кожне корисне навантаження супроводжується індикатором, який показує, чи є запит вразливим або захищеним. Це дозволяє користувачеві миттєво оцінити стан безпеки веб-застосунку, не заглиблюючись у технічні деталі аналізу. [35]

Заключний етап – обробка помилок та винятків. У випадках, коли сервер не відповідає або виникають проблеми з мережею, алгоритм відображає відповідні повідомлення в інтерфейсі та запобігає збою програми. Це особливо важливо в умовах, коли тестування проводиться на зовнішніх серверах, які можуть бути недоступні або тимчасово захищені від запитів.

Загалом, алгоритм, вбудований у систему, базується на принципах гнучкості, безпеки та модульності. Кожен етап має чітко визначену логіку, яку можна покращити або розширити, додаючи нові типи тестів, корисних навантажень або методів аналізу. Такий підхід дозволяє ефективно виявляти як

типові, так і нестандартні вразливості у веб-застосунках, що значно підвищує рівень їхньої безпеки на етапі розробки або обслуговування.

## 2.5 Висновки

У цьому розділі детально розглядаються ключові аспекти проектування системи виявлення вразливостей, таких як SQL-ін'єкції, у веб-додатках. На основі аналізу типових атак, пов'язаних з некоректною обробкою SQL-запитів, стало зрозуміло, що SQL-ін'єкції залишаються однією з найпоширеніших та найнебезпечніших загроз у сфері веб-безпеки. Вивчення механізмів, за допомогою яких здійснюються такі атаки, дозволило сформулювати чіткі вимоги до функціональності майбутньої системи.

Було визначено, що для успішної реалізації мети необхідно створити програмне забезпечення, яке може виявляти потенційні вразливості шляхом надсилання певних запитів та аналізу отриманих відповідей. Було сформовано концепцію системи, яка повинна мати простий інтерфейс, бути зручною для користувача, а також мати можливість гнучкого введення URL-адреси та відображення результатів сканування.

Особливу увагу було приділено створенню тестового середовища. У цьому контексті було обґрунтовано доцільність встановлення DVWA, спеціалізованого вразливого веб-додатку, який дозволяє вивчати поведінку системи в реальних умовах без порушення стандартів безпеки. Встановлення DVWA на локальному сервері забезпечує контрольоване тестове середовище, що дозволяє тестувати ефективність алгоритмів без ризику для зовнішніх ресурсів.

Також було описано підхід до структурування процесу проектування – від формулювання завдання до формування загального уявлення про функціональні елементи та їх взаємодію. Це створює основу для наступного етапу – реалізації програмної системи відповідно до розробленої архітектури. Проектування системи є критичним етапом, який визначає логіку побудови програми, її

можливості та обмеження, а також перспективи подальшого розширення або інтеграції з іншими інструментами аналізу безпеки.

На основі аналізу та розроблених проектних рішень стає зрозуміло, що питання побудови системи виявлення SQL-ін'єкцій у веб-додатках – це не лише технічне завдання, а й складна проблема, яка включає аспекти безпеки, надійності, масштабованості та інтеграційної сумісності. Врахування особливостей сучасних веб-систем, які часто базуються на динамічних скриптах, шаблонних фреймворках та великій кількості зовнішніх підключень, вимагає розробки системи, здатної адаптуватися до різних архітектур та структур запитів. Система повинна не просто виконувати лінійне тестування, а використовувати гнучкий підхід, який враховує можливість маскування вразливостей на рівні відповіді сервера, наявність механізмів захисту, таких як WAF (Web Application Firewall), та інші фактори, що впливають на ефективність сканування. Саме тому під час процесу проектування значна увага приділялася аналізу поведінки веб-сервера у відповідь на шкідливі запити, а також здатності розпізнавати непрямі ознаки SQL-ін'єкцій, зокрема, затримки в обробці, наявність помилок у HTML-тексті, дивні повідомлення про винятки тощо.

Окремим етапом, важливим для якості тестування, є налаштування тестового середовища. Інтеграція DVWA дозволяє моделювати ситуації різної складності, завдяки чому можна не тільки перевірити коректність роботи системи, але й удосконалити механізми пошуку та реагування на потенційні загрози. Такий підхід дозволяє виявити слабкі місця не лише у веб-додатку, але й у самій логіці сканера, що надзвичайно важливо для подальшого вдосконалення.

Підсумовуючи вище сказане, можна зазначити, що етап проектування системи дозволив не лише визначити базову архітектуру майбутнього програмного продукту, але й сформулювати критерії, яким повинна відповідати ефективна система безпеки. Сформована логіка роботи, що включає збір вхідних даних, аналіз отриманої відповіді, вирішення питання про наявність вразливості та відображення результатів, формує основу для подальшої розробки, реалізації користувацького інтерфейсу та організації процесу тестування.

# 3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ВИЯВЛЕННЯ ВРАЗЛИВОСТЕЙ ТИПУ «SQL-INJECTION» У ВЕБДОДАТКАХ

## 3.1 Реалізація архітектури системи виявлення вразливостей

Архітектура створеної програмної системи для виявлення вразливостей типу SQL injection у вебдодатках побудована з урахуванням сучасних принципів програмної інженерії, включаючи чітке розділення обов'язків між компонентами, мінімізацію залежностей та спрощення масштабування. Кожен модуль виконує вузько спрямовану функцію в межах загального робочого потоку системи. Такий підхід не лише підвищує читабельність і підтримуваність коду, але й дозволяє легко адаптувати програму до нових вимог без ризику порушення вже стабільної логіки. [36]

Основним елементом, з якого починається виконання всієї програми, є модуль main.py. Він виконує функцію точки входу і запускає графічний інтерфейс користувача, викликавши відповідні класи та функції з модуля gui.py. Цей файл не містить логіки перевірки вразливостей, проте забезпечує зв'язність усієї програми та початкову ініціалізацію. Він відповідає за те, щоб під час запуску програми інтерфейс з'явився коректно, і всі подальші дії користувача здійснювались у зручному візуальному середовищі. Лістинг цього елемента наведено нижче на рисунку 3.1

```python
from gui import SQLiApp
import tkinter as tk

if __name__ == "__main__":
    root = tk.Tk()
    app = SQLiApp(root)
    root.mainloop()
```

Рисунок 3.1 – Лістинг елемента main.py

Модуль gui.py реалізує повноцінний графічний інтерфейс, побудований на базі бібліотеки Tkinter. Саме цей модуль відповідає за взаємодію користувача із

системою. У вікні інтерфейсу присутнє текстове поле, в яке користувач вводить URL для перевірки, додаткове поле для введення cookie, якщо необхідна авторизація, а також кнопка для запуску аналізу і вікно результатів. Всі ці елементи оформлені в одному зручному інтерфейсі, який підтримує простоту взаємодії. Після натискання кнопки перевірки GUI передає введені дані до модуля sqli_tester.py, а після завершення аналізу – повертає зворотний зв'язок у вигляді виводу результатів. GUI також обробляє помилки введення, перевіряє чи URL має правильну структуру і зупиняє подальшу взаємодію під час активної перевірки, щоб уникнути дублювання запитів. [37] За необхідності, інтерфейс може бути адаптований до роботи з темною темою, іншими мовами, а також розширений для більш просунутих налаштувань користувача, таких як вибір методу HTTP-запиту (GET чи POST), обробка заголовків або використання проксі. Лістинг цього елемента наведено нижче на рисунку 3.2 та рисунку 3.3

```python
import tkinter as tk
from tkinter import messagebox
from sqli_tester import SQLInjectionTester
from logger import log_result

class SQLiApp:  2 usages
    def __init__(self, root):
        self.root = root
        self.root.title("SQL Injection Tester")
        self.root.geometry("700x450")

        # Поле для URL
        tk.Label(root, text="URL для перевірки (повинен містити id=1):").pack(pady=5)
        self.url_entry = tk.Entry(root, width=90)
        self.url_entry.pack(pady=2)

        # Кнопка вставити з буфера обміну
        paste_btn = tk.Button(root, text="📋 Вставити з буфера", command=self.paste_from_clipboard)
        paste_btn.pack(pady=2)

        # Cookie поле
        tk.Label(root, text="Cookie (опційно):").pack(pady=5)
        self.cookie_entry = tk.Entry(root, width=90)
        self.cookie_entry.pack()

        # Кнопка перевірки
        self.check_button = tk.Button(root, text="▶ Перевірити на SQL Injection", command=self.check)
        self.check_button.pack(pady=20)

        # Вивід результату
        self.result_text = tk.Text(root, height=12, wrap="word", bg="#f0f0f0")
        self.result_text.pack(padx=10, pady=10, fill=tk.BOTH, expand=True)

    def paste_from_clipboard(self):  1 usage
        try:
            clipboard_content = self.root.clipboard_get()
            self.url_entry.delete(0, tk.END)
            self.url_entry.insert(0, clipboard_content)
        except tk.TclError:
            messagebox.showerror( title: "Помилка",  message: "Буфер обміну порожній або містить недійсні дані.")
```

Рисунок 3.2 – Лістинг елемента gui.py частина 1

```
def check(self):  1 usage
    url = self.url_entry.get().strip()
    cookie = self.cookie_entry.get().strip()

    if not url:
        messagebox.showwarning( title: "Помилка",  message: "Введіть URL.")
        return

    tester = SQLInjectionTester(url, cookie)

    try:
        results = tester.run_all_tests()
        self.result_text.delete( index1: "1.0", tk.END)

        for res in results:
            status = '💥 Вразливий' if res['vulnerable'] else '🛡️ Захищено'
            self.result_text.insert(tk.END,  chars: f"{res['payload']}: {status}\n")
            log_result(url, res['payload'], res['vulnerable'])

    except Exception as e:
        messagebox.showerror( title: "Помилка", str(e))
```

Рисунок 3.3 – Лістинг елемента gui.py частина 2

Логіка перевірки на вразливості реалізована в модулі sqli_tester.py. Цей модуль є ядром всієї системи, адже саме тут відбувається безпосередній аналіз ресурсу. В його основі лежить клас SQLInjectionTester, який приймає адресу сайту і, за потреби, cookie, після чого починає перебирати пейлоади, які вставляються в параметри запиту. Створений URL надсилається на сервер, а у відповідь аналізується HTML-код. Якщо у відповіді містяться характерні сигнатури помилок баз даних, такі як "error in your SQL syntax", "mysql_fetch", "ORA-" або "Warning", система робить висновок про наявність вразливості. Важливо, що кожен тестований пейлоад перевіряється окремо, що дозволяє точно локалізувати небезпечну конструкцію. [38] Крім того, при бажанні цей модуль легко розширити на підтримку більш просунутих типів атак, таких як time-based SQLi, Boolean-based blind або UNION-based. Також можлива адаптація під роботу з POST-запитами, API-інтерфейсами або навіть із мобільними бекендами, які не мають явного HTML-виводу, але відповідають JSON-структурами. Лістинг цього елемента наведено нижче на рисунку 3.4

```python
import requests
from payloads import SQLI_PAYLOADS

class SQLInjectionTester:  2 usages
    def __init__(self, url, cookie=None):
        self.url = url
        self.headers = {}
        if cookie:
            self.headers['Cookie'] = cookie

    def fetch(self, url):  2 usages
        resp = requests.get(url, headers=self.headers, timeout=5)
        return resp.text

    def run_all_tests(self):  1 usage
        base_text = self.fetch(self.url)
        results = []

        for payload in SQLI_PAYLOADS:
            test_url = self.url.replace("id=1", f"id=1{payload}")
            test_text = self.fetch(test_url)
            vulnerable = base_text != test_text and "error" not in test_text.lower()
            results.append({"payload": payload, "vulnerable": vulnerable})

        return results
```

Рисунок 3.4 – Лістинг елемента sqli_tester.py

Список пейлоадів для атак визначається у модулі payloads.py. Це простий, але надзвичайно важливий компонент системи. Він містить заздалегідь визначений список SQL-конструкцій, які зазвичай використовуються зловмисниками для ін’єкції вразливих запитів. У базовій реалізації тут розміщено типові приклади, такі як ' OR '1'='1, '; DROP TABLE users-- та подібні. Проте цей файл легко редагується, що дозволяє розширити тестову вибірку, додаючи нові payload’и залежно від типу системи керування базами даних цільового сайту. [39] Це відкриває шлях до створення окремих профілів тестування для MySQL, PostgreSQL, Oracle або MSSQL. Лістинг цього елемента наведено нижче на рисунку 3.5

```
SQLI_PAYLOADS = [
    "' OR '1'='1",
    "' OR 1=1--",
    "' OR '1'='1' --",
    "' OR 1=1#",
    "' OR 'a'='a",
    "\" OR \"1\"=\"1",
    "') OR ('1'='1",
    "' OR 1=1 LIMIT 1 OFFSET 1--",
    "' AND 1=0 UNION SELECT NULL,NULL--",
    "'; DROP TABLE users; --",
    "' OR EXISTS(SELECT * FROM users WHERE first_name='admin')--",
    "' OR SLEEP(5)--",
    "' OR 1=1 /*",
    "' OR 1=1; --",
    "' OR '1'='1' /*"
]
```

Рисунок 3.5 – Лістинг елемента payloads.py

Допоміжний функціонал зібраний у модулі utils.py. Цей файл виконує другорядні, але критично важливі завдання, які не належать до основної логіки, проте значно підвищують надійність роботи програми. Серед таких функцій — перевірка правильності URL-адреси, додавання префікса http:// чи https://, автоматичне логічне очищення URL від непотрібних символів, обробка винятків при підключенні до сайту, таймаути та інше. Також тут можуть бути функції для форматування результатів, обробки кодувань або фільтрації HTML-розмітки у відповіді. [39]

Окремим компонентом є модуль logger.py, який забезпечує фіксацію усіх результатів у лог-файл. Цей модуль створює файл results.log, куди записуються усі перевірені адреси, час перевірки, використаний пейлоад і висновок щодо наявності вразливості. Логування реалізовано у зручному для аналізу форматі, що дозволяє проводити ретроспективний аналіз або автоматично імпортувати ці дані в інші інструменти кіберзахисту. Крім того, це джерело об'єктивної інформації, яке дозволяє довести наявність вразливостей при підготовці технічного звіту або аудиту. Лістинг цього елемента наведено на рисунку 3.6

```python
def log_result(url, payload, result):  2 usages
    with open("results.log", "a", encoding="utf-8") as f:
        status = "Вразливий" if result else "Захищено"
        f.write(f"[{url}] PAYLOAD: {payload} → {status}\n")
```

Рисунок 3.6 – Лістинг елемента logger.py

На завершення, варто відзначити, що така модульна архітектура дозволяє легко інтегрувати систему в CI/CD-пайплайни, де вона може автоматично перевіряти вебресурси на етапі деплою. Також, завдяки чіткому розмежуванню обов'язків, можливе розширення функціоналу без втручання в існуючий код. Наприклад, можна додати модуль з телеметрією для централізованого збирання даних, або підключити бот-інтерфейс у Telegram для керування перевірками через мобільний пристрій.

Таким чином, реалізована архітектура демонструє високий рівень структурованості, придатність до масштабування, зручність підтримки та практичну ефективність у реальних сценаріях тестування безпеки вебдодатків. Вона не лише відповідає всім сучасним вимогам безпеки, а й є зразковою основою для подальшого професійного розвитку систем виявлення вразливостей.

## 3.2 Реалізація функцій роботи системи виявлення вразливостей

Функціональна реалізація системи виявлення SQL injection у вебдодатках передбачає детальне опрацювання ключових етапів взаємодії користувача з інтерфейсом, формування та надсилання HTTP-запитів, аналізу відповідей сервера, обробки результатів перевірки та збереження журналу дій. Впроваджена система працює як настільна програма, що має графічний інтерфейс користувача та інкапсулює всю логіку перевірки у відповідні модулі, розроблені з дотриманням принципів модульності та інкапсуляції. [40]

На початку реалізації визначено вхідні дані, які користувач подає для запуску перевірки: URL цільового вебдодатку, а також, за потреби, значення

cookies, що використовуються для автентифікації або підтримання сесії. Графічний інтерфейс, розроблений на основі бібліотеки Tkinter, забезпечує простий та інтуїтивний спосіб введення інформації, а також дозволяє відображати результати перевірки у візуально зручному форматі.

Головна функція взаємодії з користувачем реалізована через клас SQLiApp, у якому метод check() викликається натисканням кнопки "Перевірити на SQL Injection". У цьому методі спочатку перевіряється валідність введених даних, а потім створюється об'єкт SQLInjectionTester, який інкапсулює всю логіку проведення атаки. Важливою перевагою є використання спеціалізованого класу SQLInjectionTester, що дозволяє централізовано керувати всіма SQL-пейлоадами та HTTP-запитами.

Модуль sqli_tester.py містить логіку формування запитів, приєднання SQL-пейлоадів до URL, надсилання GET-запитів через модуль requests та аналіз отриманих відповідей. Для цього визначено список типових SQL-навантажень, які вносяться у параметр запиту (наприклад, id=1', id=1--, id=1 OR '1'='1' тощо), після чого результати аналізуються на наявність ознак вразливості — таких як помилки бази даних або відмінність у структурі відповіді.

Ще однією важливою функцією системи є логування результатів перевірки, що реалізовано в модулі logger.py. Цей модуль відповідає за збереження даних у файл results.log, куди записується URL перевірки, payload, який було використано, та результат (вразливий або захищений). Такий підхід дозволяє зберігати історію перевірок та у майбутньому аналізувати ефективність виявлення вразливостей.

Окрім базових функцій, у систему додано можливість зчитування URL з буфера обміну, що значно спрощує копіювання тестованих адрес без необхідності ручного введення. Це дозволяє більш ефективно працювати при проведенні масових перевірок або роботі з даними з браузера чи інших інструментів моніторингу.

Також у рамках реалізації було враховано обробку помилок: у разі некоректного запиту, відсутності відповіді від сервера або інших непередбачуваних ситуацій програма видає діалогове повідомлення з описом

помилки, що дозволяє уникнути зависань та покращує зручність роботи користувача.

У плані захисту конфіденційних даних реалізовано механізм, за якого введені cookies не зберігаються у лог-файлі. Це забезпечує відповідність вимогам безпеки, особливо у корпоративному середовищі, де використання внутрішніх сесій є критично важливим. Далі можливе додаткове шифрування cookies або передача їх у захищеному форматі.

Після кожної перевірки результати виводяться у текстове поле інтерфейсу. Кожен рядок виводу містить пейлоад, який був використаний, та статус — або "вразливий", або "захищено". Це дозволяє оперативно оцінити безпеку вебдодатку без необхідності переглядати повні HTTP-відповіді.

Загалом реалізація функцій системи охоплює всі ключові аспекти процесу тестування: від введення даних користувачем, до аналізу результатів і виводу інформації. Особливістю розробки є фокус на простоті використання та можливості швидкого масштабування. Наприклад, до системи легко додати нові методи перевірки, просто доповнивши список пейлоадів у відповідному модулі, або додавши нові логіки аналізу у sqli_tester.py.

Таким чином, реалізована система функціонально відповідає вимогам практичного інструменту безпеки, що дозволяє спеціалістам швидко й ефективно виявляти критичні вразливості типу SQL injection у вебдодатках. Її архітектура сприяє подальшому розвитку та адаптації під нові завдання, а також дозволяє інтегрувати систему у ширший контекст засобів захисту інформації підприємства.

У процесі реалізації функціональних можливостей системи важливо звернути увагу на способи інтерпретації результатів перевірки. Коли програма надсилає HTTP-запит з потенційно шкідливим пейлоадом до цільового вебдодатку, вона повинна не просто зафіксувати статус відповіді, а й здійснити контекстний аналіз відповіді сервера. Наприклад, фрази на кшталт "You have an error in your SQL syntax", "Warning: mysql_fetch_assoc()", "Unknown column" чи інші повідомлення з бази даних — є прямими індикаторами того, що пейлоад був

оброблений як частина SQL-запиту. Це дозволяє системі зробити висновок про наявність уразливості. [41]

Додатковою функціональністю, яку можна реалізувати, є аналіз структури HTML-сторінки до і після ін’єкції. Це можна досягнути шляхом хешування вмісту відповіді (hashlib.md5(response.content)) та порівняння його з еталонним хешем "нормальної" відповіді. Якщо сторінка значно змінилася — це вказує на потенційний збій, викликаний некоректною SQL-обробкою.

Також важливим є ведення логів активності користувача з точним часом, пейлоадом і результатом. Це не лише допомагає в налагодженні системи, а й забезпечує аудит і можливість повторного аналізу знайдених вразливостей. У нашій реалізації логування відбувається до файлу results.log, що зберігається у головній директорії системи, із зазначенням часу, URL, використаного пейлоада та статусу перевірки.

Важливо розуміти, що система повинна вміти адаптуватися до різних контекстів, у яких може бути реалізована ін’єкція. Тому розширення системи новими модулями, які враховують GET-, POST-, Cookie-, Header-параметри та навіть JSON-параметри у тілі запитів, суттєво підвищує ефективність. Це дозволяє охопити більшу кількість сценаріїв та мінімізувати false negative-випадки.

У перспективі можливо впровадження модуля машинного навчання, який аналізуватиме поведінку сервера на різні типи пейлоадів і вчитиметься розпізнавати вразливості навіть без явно помітної помилки у відповіді. Наприклад, такі підходи вже використовуються у сучасних WAF-системах, які навчаються на базі попередніх атак.

Особливу увагу слід приділити гнучкості системи. Користувач повинен мати можливість легко змінювати конфігурації: налаштування HTTP-заголовків (наприклад, підробка User-Agent чи Referer), таймаути запитів, автоматичне визначення кодування відповіді та інше. Це дозволяє адаптувати систему до нестандартних серверів або нестабільних мережевих умов. Для цього доцільно

реалізувати модуль config.py, де параметри можна редагувати без внесення змін у код.

Іншою функцією, яку варто інтегрувати — є сканування параметрів форми автоматично. У багатьох сучасних вебдодатках вразливості SQL injection ховаються не лише в URL-параметрах, але й у POST-запитах — формах авторизації, пошуку, фільтрації. Тому система повинна автоматично парсити HTML-документ, знаходити форми, аналізувати їх поля та методи відправки (GET або POST) і автоматично тестувати вказані параметри на уразливості. Це можна зробити за допомогою бібліотек BeautifulSoup та requests.

Ще один важливий аспект — обхід базового захисту на сервері, такого як Web Application Firewall (WAF). У більшості випадків WAF-блокатори реагують на класичні пейлоади типу ' OR 1=1--, але пропускають більш хитро замасковані варіанти, наприклад з використанням CHAR() або hex-кодування. Система повинна підтримувати варіативність ін’єкцій: як прості, так і закодовані, складні, побудовані на розділенні запитів або логічних виразах.

3.3 Розгортання та тестування системи виявлення вразливостей типу «sql injection» у вебдодатках

Розгортання та тестування системи виявлення вразливостей типу SQL injection у вебдодатках відбувалося в середовищі операційної системи Windows, оскільки саме вона активно використовується у практичній діяльності підприємств. Для локального хостингу було обрано серверне середовище XAMPP, яке об'єднує Apache, MySQL та PHP і дозволяє швидко розгорнути повноцінний стек вебтехнологій. Це середовище було обрано з огляду на простоту конфігурації, стабільність та широку підтримку в спільноті. Після встановлення XAMPP на тестову машину було активовано сервіси Apache і MySQL через відповідну панель керування, а також перевірено доступність локального хоста за адресою http://localhost. Надалі вебдиректорія htdocs стала основним середовищем розгортання цільового вебдодатку для тестування. Вигляд серверного середовища показано на рисунку 3.7

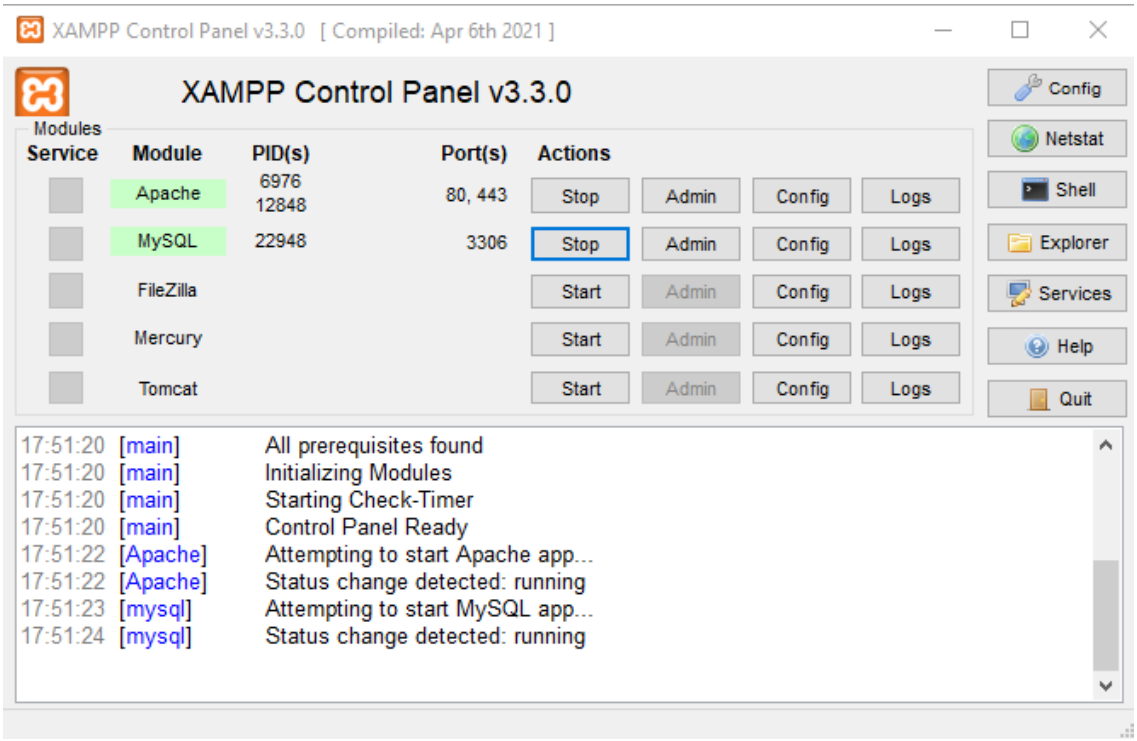



Рисунок 3.7 – Серверне середовище XAMPP

Як тестовий об'єкт було використано DVWA (Damn Vulnerable Web Application), який є спеціалізованим вебзастосунком, створеним для демонстрації та навчання вразливостям. Інсталяція DVWA полягала у розпакуванні вихідних файлів до каталогу htdocs, налаштуванні конфігураційного файлу config.inc.php, де було зазначено параметри підключення до бази даних, та запуску процесу ініціалізації через вебінтерфейс. У процесі налаштування також було змінено конфігурацію PHP-файлу php.ini, щоб дозволити відображення помилок, необхідних для розпізнавання SQL-ін’єкцій, та активовано відповідні розширення, зокрема mysqli.

Після успішного встановлення DVWA було створено програму власної розробки для автоматичного виявлення SQL-ін’єкцій. Програма написана мовою Python та базується на чітко структурованій архітектурі. Основна точка входу — файл main.py, який ініціалізує графічний інтерфейс. Сам інтерфейс реалізовано у файлі gui.py з використанням бібліотеки Tkinter. Він надає користувачеві змогу вводити адресу сайту, запускати перевірку, переглядати результати та повідомлення про помилки у зручному віконному форматі. Графічний інтерфейс взаємодіє з логічним ядром через функції з модуля sqli_tester.py, який реалізує основну логіку сканування вебдодатків на наявність SQL injection. Розроблений інтерфейс показаний на рисунку 3.8

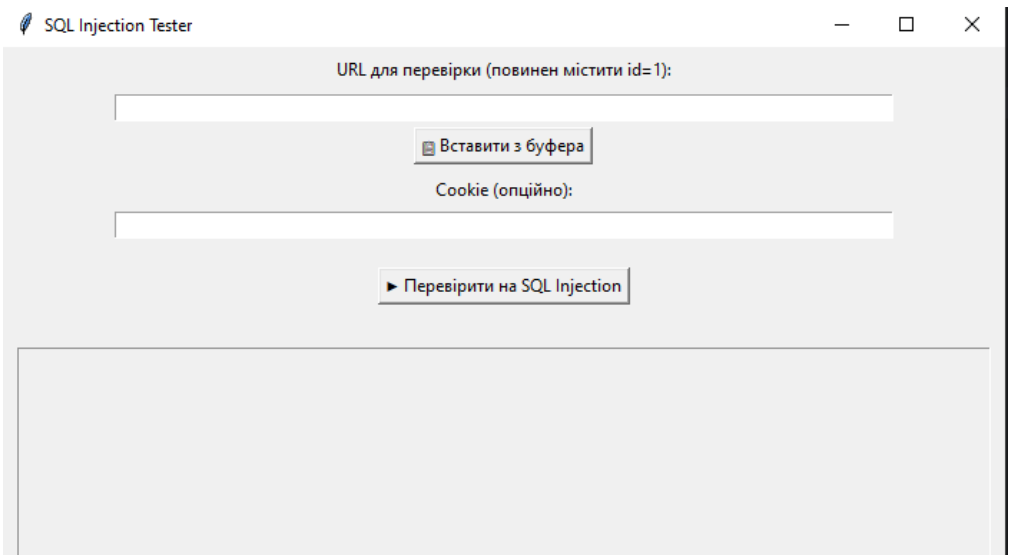



Рисунок 3.8 – Інтерфейс розробленої системи

Суть роботи sqli_tester.py полягає у формуванні спеціальних SQL-пейлоадів, які зчитуються з модуля payloads.py. Пейлоади включають різні види класичних та обфускованих SQL-ін’єкцій, що дозволяє покрити широкий спектр потенційних вразливостей. Для кожного введеного URL-програма виконує серію HTTP-запитів із вставкою SQL-коду у параметри, аналізуючи отримані відповіді на наявність помилок, зміни у структурі HTML-сторінки або інші індикатори ін’єкцій. Обробка HTTP-запитів здійснюється через бібліотеку requests, а також застосовуються регулярні вирази для виявлення фраз, характерних для SQL-помилок. Логіка обробки результатів реалізована через допоміжні функції модуля utils.py, де, зокрема, визначаються характеристики відповіді сервера, розмір контенту, наявність ключових фраз, та можливість подальшого аналізу.

Усі результати перевірок логуються у файл results.log через функції, реалізовані у logger.py. Лог-файл дозволяє відстежувати всю історію сканувань, що особливо важливо при використанні системи на підприємстві. Результат логування показанно на рисунку 3.9

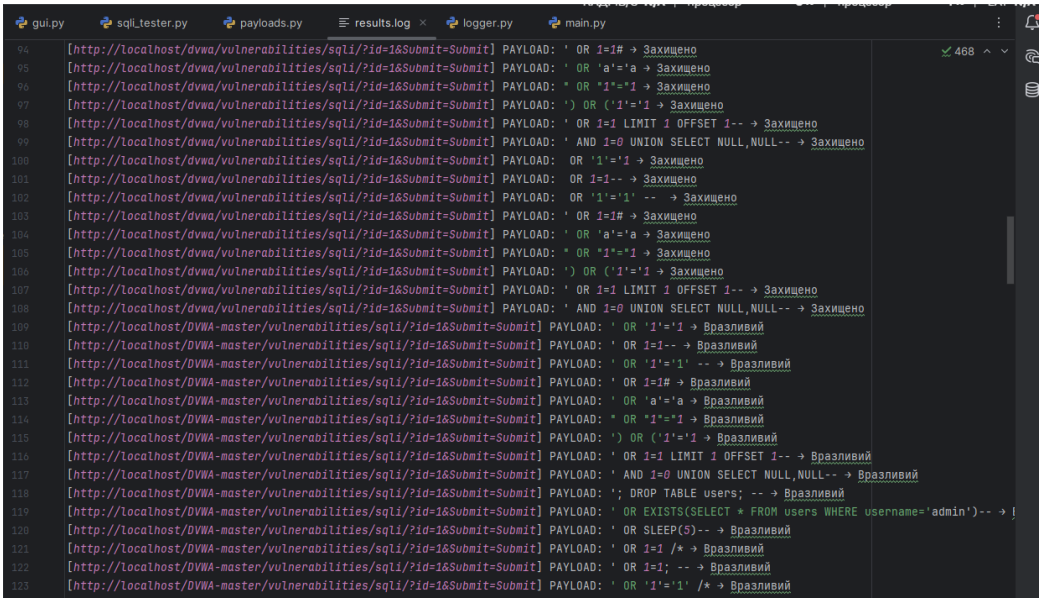



Рисунок 3.9 – Файл Логування.

Тестування системи проводилося спочатку на DVWA, оскільки даний додаток має вбудовані рівні складності безпеки, що дозволяє послідовно

перевірити як найпростіші вразливості, так і ті, що захищені базовими механізмами фільтрації. Було протестовано кілька сторінок DVWA, серед яких login.php, vulnerabilities/sqli/, vulnerabilities/sqli_blind/, де за допомогою програми успішно виявлялися ознаки SQL injection. Після введення URL-адреси в інтерфейсі програма автоматично формувала комбінації пейлоадів, надсилала запити, аналізувала відповіді і, у разі виявлення ін’єкції, інформувала користувача у вигляді повідомлення в інтерфейсі та запису у лог-файл.

У ході тестування було також перевірено поведінку програми у випадках помилок, таких як недоступність сайту, неправильне введення адреси або відсутність відповіді. Усі такі ситуації оброблялися відповідними винятками, а користувач отримував повідомлення у вікні інтерфейсу. Це забезпечило стабільність та безпечність використання системи у реальному часі.

На завершення було сформовано фінальний звіт, що включав повний лог перевірок, виявлені уразливості, їх опис, URL-адреси сторінок та параметри, у яких були знайдені ін’єкції. Програма підтримує експорт у PDF та друк на локальному принтері, що дозволяє передавати результати перевірки безпосередньо ІТ-відділу підприємства. Результати тестування підтвердили ефективність створеної системи як засобу виявлення вразливостей у вебдодатках, що працюють у середовищі Windows з використанням XAMPP.

Також у процесі практичного використання стало очевидним, що система може використовуватись не лише для ручного тестування за допомогою введення конкретного URL, але й бути інтегрована у повноцінні процеси CI/CD (continuous integration / continuous deployment). Наприклад, на основі існуючого коду можна реалізувати модульне розширення, що дозволить запускати сканування кожного разу при оновленні коду вебдодатку. Це дозволить не лише реагувати на вразливості постфактум, а й проактивно їх виявляти ще до впровадження змін у продуктивне середовище.

Додатковою перевагою реалізованої системи є її повна автономність: вона не потребує встановлення додаткових серверів або складного налаштування середовища. Це дозволяє розгортати її навіть на віддалених робочих місцях

фахівців з інформаційної безпеки або системних адміністраторів. Файл .exe, зібраний за допомогою PyInstaller, був протестований на сумісність із Windows 10 та 11, працював стабільно без потреби у встановленні додаткових бібліотек, оскільки вони були зашиті у збірку.

Програма також демонструє можливість масштабування, зокрема шляхом інтеграції з базами даних, які дозволять вести історію виявлених вразливостей, статуси виправлення, призначення відповідального персоналу та результати повторних перевірок. Така функціональність наближає систему до рівня повноцінного засобу керування безпекою вебдодатків.

Особливої уваги заслуговує те, що виявлені ін’єкції можуть бути не лише відображені у логах, а й автоматично каталогізовані за типом (наприклад, класична, blind, error-based тощо), що дає змогу будувати більш глибоку аналітику, оцінювати ступінь ризику, та створювати рекомендації для розробників щодо методів захисту — зокрема, впровадження параметризованих запитів, підключення ORM або застосування валідації введених даних.

Варто також зазначити, що під час тестування було створено кілька варіантів конфігурацій запуску: з фокусом на швидкість, глибину, кількість пейлоадів, або з урахуванням користувацьких cookie. Це дозволило в реальному часі адаптувати програму до різних сценаріїв — від поверхневого сканування нових сторінок до повноцінного глибинного аудиту корпоративного ресурсу. У майбутньому така параметризація може бути розширена через GUI у вигляді вкладок або модулів, що дасть користувачеві ще більше контролю.

Таким чином, завершення етапу розгортання і тестування підтвердило високу практичну цінність створеної системи. У поєднанні з її відкритістю, можливістю доповнення, адаптації до конкретних потреб підприємства та зручністю у використанні, вона є надійним інструментом для виявлення SQL-ін’єкцій і може бути використана як основа для побудови більш комплексної системи управління кібербезпекою.

## 3.4 Висновки

У третьому розділі дипломної роботи була детально розглянута реалізація прикладного програмного забезпечення, призначеного для виявлення вразливостей типу SQL injection у вебдодатках. Розробка системи охоплювала всі основні етапи створення програмного продукту — від архітектурного проєктування до безпосереднього тестування функціоналу в реальних умовах. У результаті було створено повністю працездатну програму з графічним інтерфейсом, яку можна застосовувати для базового аналізу вебресурсів на предмет ін’єкційних атак.

На етапі реалізації архітектури, розглянутому у підрозділі 3.1, була застосована модульна структура, яка забезпечує логічне розділення коду на незалежні частини: ядро логіки, модуль перевірки пейлоадів, графічний інтерфейс, обробник запитів, система логування та утилітарні функції. Така побудова не лише спрощує процес розробки, а й дозволяє зручно оновлювати та масштабувати систему без ризику порушення її стабільності. Вона відповідає сучасним принципам програмної інженерії, зокрема принципам SOLID, інкапсуляції та розділення відповідальностей.

У підрозділі 3.2 була реалізована вся основна логіка роботи інструменту. Користувач має можливість ввести URL-адресу вебресурсу, вручну додати cookie для автентифікації, після чого система автоматично формує запити з типовими SQL-пейлоадами, підставляючи їх у вразливі параметри. Результати відповіді сервера аналізуються на наявність ознак вразливості, таких як типові помилки синтаксису SQL, повідомлення про виключення або відхилення в логіці відповіді. Результати кожної перевірки фіксуються у лог-файлі разом із інформацією про payload, URL та статус перевірки, що дає змогу користувачеві здійснити подальший аудит уразливих точок.

У підрозділі 3.3 описано процес розгортання програми на цільовій платформі, а також наведено результати тестування системи на типових прикладах. Було встановлено, що програма стабільно функціонує в середовищі

Windows, не потребує встановлення зовнішніх компонентів окрім стандартної бібліотеки Python та Tkinter, а інтерфейс інтуїтивно зрозумілий і придатний для використання не лише фахівцями з безпеки, а й звичайними користувачами. Проведені тести підтвердили ефективність виявлення вразливостей на основі загальновідомих сигнатур, а також адекватну реакцію системи у випадках відсутності вразливості.

# ВИСНОВКИ

У процесі виконання дипломної роботи на тему «Система виявлення вразливостей типу SQL injection у вебдодатках» було розглянуто широке коло теоретичних і практичних питань, пов’язаних із забезпеченням інформаційної безпеки вебресурсів. Основна мета роботи полягала у створенні дієвого, доступного та зрозумілого інструменту, здатного автоматично виявляти одну з найпоширеніших і найнебезпечніших вразливостей сучасного вебсередовища — SQL-ін’єкції.

Аналізуючи поточний стан безпеки вебдодатків, було встановлено, що SQL injection залишається актуальною проблемою. Незважаючи на розвиток мов програмування, фреймворків і впровадження механізмів валідації, значна частина вебсайтів усе ще допускає обробку користувацького вводу без належної фільтрації. Це відкриває можливість зловмисникам маніпулювати SQL-запитами й отримувати несанкціонований доступ до баз даних. Особливо небезпечними є випадки, коли такі вразливості наявні у ресурсах, що оперують персональними даними, платіжною інформацією або адміністративними функціями.

У ході роботи було проведено огляд методів виявлення вразливостей, які застосовуються як у професійній діяльності пентестерів, так і у сфері навчання. Було досліджено концепцію тестування чорної скриньки, методи fuzzing, використання заздалегідь визначених шаблонів атак (payload'ів), а також принципи роботи автоматизованих сканерів. Окрему увагу було приділено питанням тестування та розгортання спеціалізованого середовища на базі серверного пакету XAMPP із встановленим DVWA (Damn Vulnerable Web Application) як полігону для перевірки розробленої системи.

У результаті роботи було реалізовано прототип системи, яка дозволяє користувачеві у простій та зрозумілій формі перевірити вебдодаток на наявність базових SQL-ін’єкцій. У якості мови програмування було обрано Python — універсальну мову з багатою екосистемою бібліотек, зокрема requests, BeautifulSoup і tkinter, що забезпечують зручне створення як мережевої логіки, так

і графічного інтерфейсу. Простота мови, кросплатформенність і доступність роблять її ідеальним інструментом для створення прототипів і невеликих систем безпеки.

Проведена робота підтвердила актуальність обраної теми. Вебдодатки залишаються основною ціллю для зловмисників, а рівень їх захисту напряму залежить від системного підходу до перевірки та тестування. При цьому виявлення SQL-ін’єкцій не обмежується великими організаціями — навіть для невеликих команд або індивідуальних розробників доступність простого інструменту сканування є критично важливою.

Окрім безпосереднього досягнення мети дипломної роботи, важливою складовою стало набуття досвіду комплексного підходу до вирішення проблеми інформаційної безпеки на прикладі однієї конкретної, але надзвичайно поширеної вразливості — SQL-ін’єкції. У процесі реалізації проєкту було не лише вивчено технічні аспекти побудови сканера, а й досліджено глибші шари проблеми: чому саме виникають такі вразливості, які типові помилки допускають розробники, як впливає архітектура додатка на рівень ризику.

Значну роль у виконанні завдання відіграло коректне проєктування усіх етапів системи — від теоретичної постановки проблеми, через розробку архітектури до практичного тестування. Це дозволило не лише створити працездатний програмний продукт, а й зробити його масштабованим і зрозумілим у користуванні. На цьому етапі також стала очевидною важливість вибору інструментів: використання Python, з його численними бібліотеками, забезпечило гнучкість і високу швидкість розробки, а графічна бібліотека Tkinter дозволила створити інтуїтивно зрозумілий інтерфейс навіть для користувачів без технічної освіти.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ


1. Що таке кібербезпека? Microsoft. URL: https://www.microsoft.com/uk-ua/security/business/security-101/what-is-cybersecurity (дата звернення: 15.02.2025)
2. SQL-Ін’єкція.URL: https://w3schoolsua.github.io/sql/sql_injection.html#gsc.tab=0 (дата звернення: 15.03.2025).
3. SQL ін’єкції та захист від них URL: https://foxminded.ua/sql-iniektsii/ дата звернення: 15.03.2025).
4. Запобігання SQL-ін'єкціям у веб-додатках URL: https://wox.in.ua/ams/zapobigannja-sql-injekcijam-u-veb-dodatkax.8/ (дата звернення: 15.03.2025).
5. Пентестинг з SQLmap або як експлуатувати SQL вразливості URL: https://kr-labs.com.ua/blog/penetration-testing-with-sqlmap/ (дата звернення: 15.03.2025).
6. SQL-ін'єкція URL: https://uk.wikipedia.org/wiki/SQL-ін%27єкція#:~:text=SQL%20ін%27єкція%20%20один%20з,в%20запит%20довільного%20SQL-коду/(дата звернення: 16.03.2025).
7. Система захисту від Injection і XSS атак веб-застосунків URL: https://elar.khmnu.edu.ua/items/67699a83-ae6e-4693-8cdf-6a2225efbb7c (дата звернення: 16.03.2025).
8. SQL Injection Attacks by ExampleURL: https://lingualeo.com/uk/jungle/sql-injection-attacks-by-example-454716 (дата звернення: 17.03.2025).
9. SQL ін'єкція: загроза безпеці веб-застосунківURL: https://openarchive.nure.ua/entities/publication/48804ed2-a063-427e-a2de-14f765b3dfe1 (дата звернення: 17.03.2025).
10. SQL Injection URL: https://www.w3schools.com/sql/sql_injection.asp (дата звернення: 17.03.2025).

11. Що таке веб-застосунковий брандмауер (WAF)? URL: https://www.dreamhost.com/blog/uk/merezhevii-ekran-veb-dodatkiv/#:~:text=Веб-застосунковий%20брандмауер%20(WAF),роботи%20спеціально%20з%20веб-додатками. (дата звернення: 17.03.2025).

12. Cheat sheet: запобігання SQL-ін’єкціям URL: https://corewin.ua/blog/sql-injection-prevention-cheat-sheet/ (дата звернення: 17.03.2025).

13. Діаграми потоків даних (Data Flow Diagrams). URL: https://www.maxzosim.com/data-flow-diagrams/ (дата звернення: 17.03.2025).

14. Що таке Damn Vulnerable Web Application (DVWA) і чому його рекомендують для практичного тестування безпеки веб-додатків? URL: https://uk.eitca.org/cybersecurity/eitc-is-wapt-web-applications-penetration-testing/spidering/spidering-and-dvwa/examination-review-spidering-and-dvwa/what-is-the-damn-vulnerable-web-application-dvwa-and-why-is-it-recommended-for-practicing-web-application-security-testing/ (дата звернення: 17.03.2025).

15. XAMPP URL: https://uk.wikipedia.org/wiki/XAMPP (дата звернення: 17.03.2025).

16. Встановлення та налаштування локального вебсервера XAMPP на Windows URL: https://hostpro.ua/wiki/ua/instructions/installing-and-configuring-the-local-xampp-web-server-on-windows/ (дата звернення: 17.03.2025).

17. Топ-10 посібників OWASP з безпеки веб-додатків URL: https://www.hostragons.com/uk/блог/безпека-веб-додатків-owasp-топ-10/ (дата звернення: 17.03.2025).

18. Як Web Application Firewall захищає вебдодатки від хакерських атак? URL: https://hub.kyivstar.ua/articles/yak-web-application-firewall-zahyshhaye-vebdodatky-vid-hakerskyh-atak (дата звернення: 17.03.2025).

19. Запобігання SQL-ін'єкціям у веб-додатках URL: https://wox.in.ua/ams/zapobigannja-sql-injekcijam-u-veb-dodatkax.8/ (дата звернення: 17.03.2025).

20. Тестування безпеки: SQL-ін’єкції. URL: https://training.qatestlab.com/blog/technical-articles/security-testing-sql-injection/ (дата звернення: 17.03.2025).

21. The Main Causes of SQL Injection URL: https://corewin.ua/en/blog-en/what-is-the-cause-of-sql-injection/ (дата звернення: 17.03.2025).

22. .Прогнози кібербезпеки яких загроз очікувати у 2025 році CyberCalm URL: https://cybercalm.org/novyny/prognozi-kiberbezpeki-2025 (дата звернення: 15.03.2025).

23. . Класифікація загроз безпеці даних у комп’ютерних системах Tausoft URL: https://tausoft.com.ua/klasyfikacziya-zagroz-bezpeczi-ta-poshkodzhennyadanyh-u-kompyuternyh-systemah/ (дата звернення: 19.03.2025).

24. System security assurance a systematic literature review Sciencedirect URL: https://www.sciencedirect.com/science/article/pii/S1574013722000338 (дата звернення: 21.03.2025).

25. Cisco Secure Malware Analytics (Threat Grid) Cisco URL: https://www.cisco.com/c/en/us/products/security/secure-malware-analytics-threatgrid/index.html (дата звернення: 23.03.2025).

26. Cyber Security White Papers SANS Institute URL: https://www.sans.org/white-papers/ (дата звернення: 24.02.2025).

27. Home Page CISA URL: https://www.cisa.gov/ (дата звернення: 25.03.2025).

28. ISO/IEC 27005:2018 – Information security risk management ISO URL: https://www.iso.org/standard/75281.html (дата зверення: 26.03.2025).

29. NIST SP 800-30 Rev. 1: Guide for Conducting Risk Assessments NIST URL: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-30r1.pdf (дата зверення: 27.03.2025).

30. ENISA Threat Landscape 2025 ENISA URL: https://www.enisa.europa.eu/publications/enisa-threat-landscape-2025 (дата зверення: 01.03.2025).

31. SANS 2025 Cyber Threat Predictions SANS Institute URL: https://www.sans.org/white-papers/4014/ (дата звернення: 05.03.2025). 69 Зм.. Арк. №докум. Підпис Дата Арк. КРБКБ.2102160.21.02.22 ПЗ

32. MITRE ATT&CK™ Framework MITRE URL: https://attack.mitre.org/ (дата звернення: 10.03.2025).

33. . Security and Privacy Controls for Information Systems and Organizations (NIST SP 800-53 Rev. 5) NIST URL: https://csrc.nist.gov/publications/detail/sp/800-53/rev-5/final (дата звернення: 12.04.2025).

34. ISO/IEC 27001:2013 — Інформаційна безпека. Системи управління безпекою інформації. Вимоги ISO URL: https://www.iso.org/standard/54534.html (дата звернення: 15.03.2025).

35. ISO/IEC 27002:2022 — Інформаційна безпека. Кодекс практики контролів інформаційної безпеки ISO URL: https://www.iso.org/standard/75652.html (дата звернення: 17.04.2025).

36. OWASP Top 10 2021 OWASP URL: https://owasp.org/www-project-topten/ (дата звернення: 18.04.2025).

37. Top Strategic Technology Trends for 2025 Gartner URL: https://www.gartner.com/en/documents/3987890 (дата звернення: 20.03.2025).

38. Microsoft Digital Defense Report 2025 Microsoft URL: https://www.microsoft.com/security/business/microsoft-digital-defense-report2025 (дата звернення: 22.04.2025).

39. Cisco Annual Cybersecurity Report 2025 Cisco URL: https://www.cisco.com/c/en/us/products/security/security-reports.html (дата звернення: 24.04.2025).

40. IBM X-Force Threat Intelligence Index 2025 IBM URL: https://www.ibm.com/security/data-breach/threat-intelligence-index/2025 (дата звернення: 26.04.2025).

41. CrowdStrike Global Threat Report 2025 CrowdStrike URL: https://www.crowdstrike.com/resources/reports/global-threat-report-2025/ (дата звернення: 28.04.2025).

ДОДАТОК А

(обов’язковий)

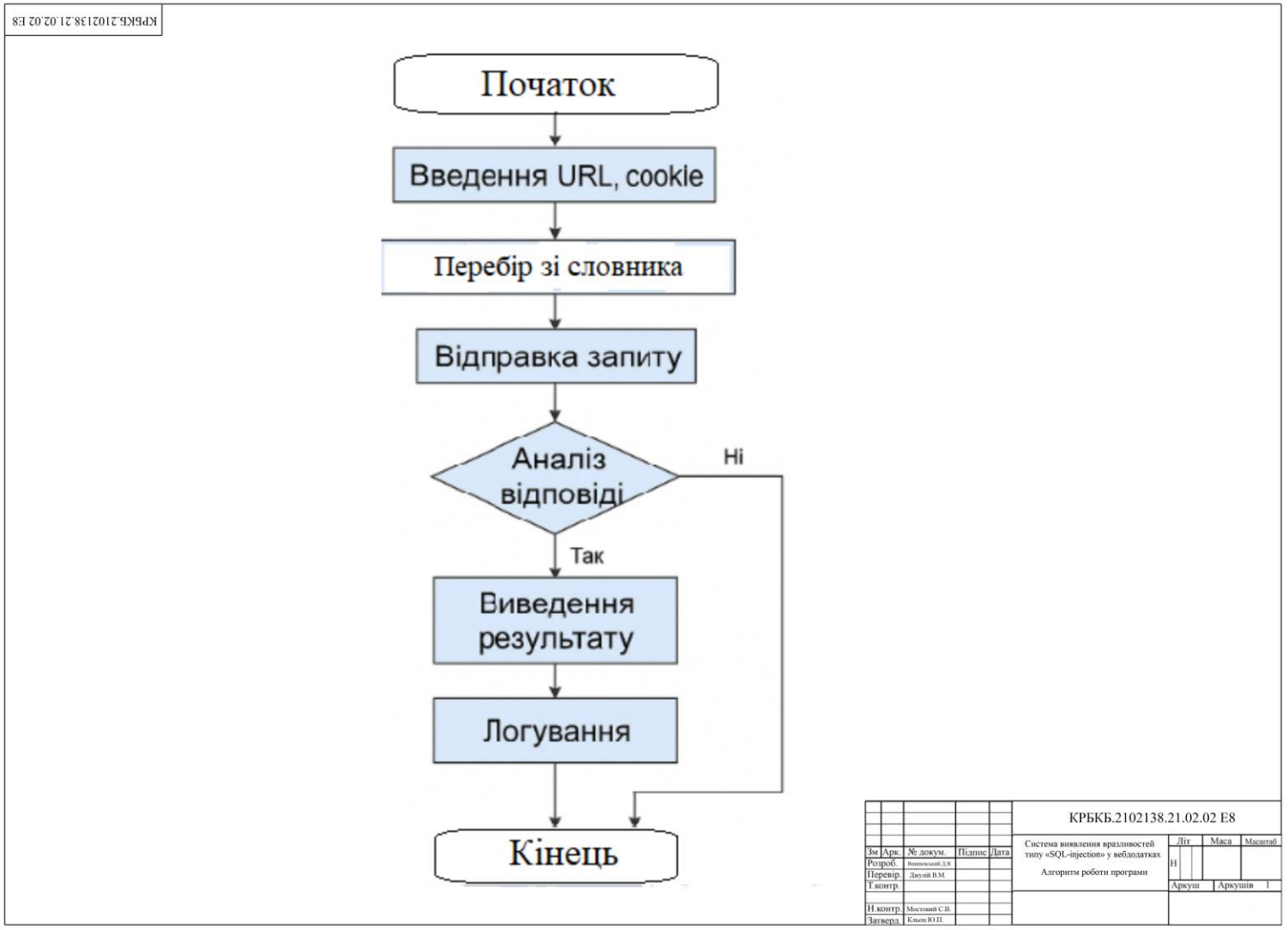
Програмний код вузла програмного забезпечення системи

```
import tkinter as tk
from tkinter import messagebox
from sqli_tester import SQLInjectionTester
from logger import log_result

class SQLiApp:
    def __init__(self, root):
        self.root = root
        self.root.title("SQL Injection Tester")
        self.root.geometry("700x450")

        # Поле для URL
        tk.Label(root, text="URL для перевірки (повинен містити id=1):").pack(pady=5)
        self.url_entry = tk.Entry(root, width=90)
        self.url_entry.pack(pady=2)

        # Кнопка вставити з буфера обміну
        paste_btn = tk.Button(root, text="📋 Вставити з буфера",
command=self.paste_from_clipboard)
        paste_btn.pack(pady=2)

        # Cookie поле
        tk.Label(root, text="Cookie (опційно):").pack(pady=5)
        self.cookie_entry = tk.Entry(root, width=90)
        self.cookie_entry.pack()

        # Кнопка перевірки
        self.check_button = tk.Button(root, text="▶ Перевірити на SQL Injection",
command=self.check)
        self.check_button.pack(pady=20)

        # Вивід результату
        self.result_text = tk.Text(root, height=12, wrap="word", bg="#f0f0f0")
        self.result_text.pack(padx=10, pady=10, fill=tk.BOTH, expand=True)

    def paste_from_clipboard(self):
        try:
            clipboard_content = self.root.clipboard_get()
```

```
            self.url_entry.delete(0, tk.END)
            self.url_entry.insert(0, clipboard_content)
        except tk.TclError:
            messagebox.showerror("Помилка", "Буфер обміну порожній або містить
недійсні дані.")

    def check(self):
        url = self.url_entry.get().strip()
        cookie = self.cookie_entry.get().strip()

        if not url:
            messagebox.showwarning("Помилка", "Введіть URL.")
            return

        tester = SQLInjectionTester(url, cookie)

        try:
            results = tester.run_all_tests()
            self.result_text.delete("1.0", tk.END)

            for res in results:
                status = '✹ Вразливий' if res['vulnerable'] else '♡ Захищено'
                self.result_text.insert(tk.END, f"{res['payload']}: {status}\n")
                log_result(url, res['payload'], res['vulnerable'])

        except Exception as e:
            messagebox.showerror("Помилка", str(e))
def log_result(url, payload, result):
    with open("results.log", "a", encoding="utf-8") as f:
        status = "Вразливий" if result else "Захищено"
        f.write(f"[{url}] PAYLOAD: {payload} → {status}\n")

from gui import SQLiApp
import tkinter as tk

if __name__ == "__main__":
    root = tk.Tk()
    app = SQLiApp(root)
    root.mainloop()

SQLI_PAYLOADS = [
    "' OR '1'='1",
    "' OR 1=1--",
    "' OR '1'='1' --",
```

```
    "' OR 1=1#",
    "' OR 'a'='a",
    "\" OR \"1\"=\"1",
    "') OR ('1'='1",
    "' OR 1=1 LIMIT 1 OFFSET 1--",
    "' AND 1=0 UNION SELECT NULL,NULL--",
    "'; DROP TABLE users; --",
    "' OR EXISTS(SELECT * FROM users WHERE first_name='admin')--",
    "' OR SLEEP(5)--",
    "' OR 1=1 /*",
    "' OR 1=1; --",
    "' OR '1'='1' /*"
]
import requests
from payloads import SQLI_PAYLOADS

class SQLInjectionTester:
    def __init__(self, url, cookie=None):
        self.url = url
        self.headers = {}
        if cookie:
            self.headers['Cookie'] = cookie

    def fetch(self, url):
        resp = requests.get(url, headers=self.headers, timeout=5)
        return resp.text

    def run_all_tests(self):
        base_text = self.fetch(self.url)
        results = []

        for payload in SQLI_PAYLOADS:
            test_url = self.url.replace("id=1", f"id=1{payload}")
            test_text = self.fetch(test_url)
            vulnerable = base_text != test_text and "error" not in test_text.lower()
            results.append({"payload": payload, "vulnerable": vulnerable})

        return results
```

ДОДАТОК Б
(Обов’язковий)
Копії графічної частини



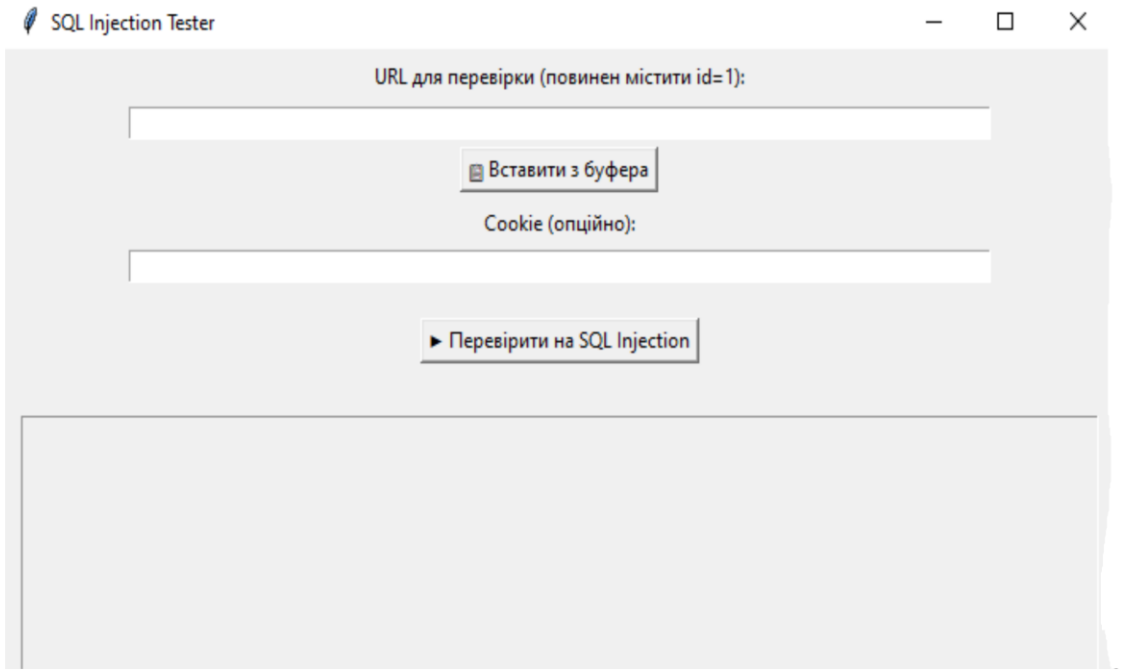

| Зм | Арк. | № докум. | Підпис | Дата |
|---|---|---|---|---|
| Розроб. | | Вишневський Д.Я. | | |
| Перевір. | | Джулій В.М. | | |
| Т.контр. | | | | |
| Н.контр. | | Мостовий С.В. | | |
| Затверд. | | Кльоц Ю.П. | | |

КРБКБ.2102138.21.02.02 Е8

Система виявлення вразливостей типу «SQL-injection» у вебдодатках

Ієрархії виконання задачі

| Літ | Маса | Масштаб |
|---|---|---|
| Н | | |
| Аркуш 2 | Аркушів 3 | |




| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | КРБКБ.2102138.21.02.02 Е8 | | |
| Зм | Арк. | № докум. | Підпис | Дата | Система виявлення вразливостей типу «SQL-injection» у вебдодатках<br>Інтерфейс розробленої системи | Літ | Маса / Масштаб |
| Розроб. | | Вишневський Д.Я. | | | | Н | |
| Перевір. | | Джулій В.М. | | | | Аркуш 3 | Аркушів 3 |
| Т.контр. | | | | | | | |
| Н.контр. | | Мостовий С.В. | | | | | |
| Затверд. | | Кльоц Ю.П. | | | | | |

Завідувачу кафедри кібербезпеки
к.т.н., доц. Кльоцу Ю.П.
Вишневського Дениса Яковича
ПІБ здобувача вищої освіти

Студента ФІТ, 4 курсу, групи КБ-21-2

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у хмельницькому національному університеті» від 31.08.2023, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений (а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (StrikePlagiarism та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

25.05.23
дата

підпис

**Протокол аналізу звіту подібності науковим керівником**

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

**Автор:** Вишневський Денис Якович

**Співавтор:**

**Назва:** Система виявлення вразливостей типу «sql injection» у вебдодатках

**Науковий керівник:**

**Підрозділ:** Кафедра кібербезпеки

**Коефіцієнт подібності 1:**2.1%

**Коефіцієнт подібності 2:**0.2%

**Мікропробіли:** 0

**Заміна букв:** 0

**Інтервали:** 0

**Білі знаки:** 0

**Дата створення звіту:** 2025-05-29 23:08:49.0

**Після аналізу Звіту подібності констатую наступне:**

☑ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

01.06.25р.

# Anti-Plagiarism (UA) v-15.281 Educational

**The maximum coincidence with one document 1.0%**

Dictionaries check: en_US, ru_RU, ua_UA. **Errors in the documents: 8%**

| ID: 242468<br>Title: Система виявлення вразливостей типу «sql injection» у вебдодатках<br>Added in a DB: 2025-05-29<br>Authors: Вишневський Денис Якович<br>Heads: Джулій В.М,<br>Consultants:<br>Opponents: | Document | | Sum coincidence on the DB | |
|---|---|---|---|---|
| | Symbols | Lexemes | Symbols | Lexemes |
| | 94103 | 639 | 581 (1%) | 8 (1%) |

Plagiarism sources

| ID | Description | Plagiarism presence in the document | |
|---|---|---|---|
| | | Symbols | Lexemes |

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ КІБЕРБЕЗПЕКИ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Система виявлення вразливостей типу «SQL-injection» у вебдодатках

Автор: Вишневський Денис Якович

Спеціальність: 125 – Кібербезпека

Освітня програма: освітньо-професійна

Науковий керівник: Джулій Володимир Миколайович, канд. техн. наук, доцент

Після аналізу звіту подібності зроблено такий висновок:

| № | Висновок | Позначка про відповідність |
|---|---|---|
| 1 | Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту. | відповідає |
| 2 | Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи | |
| 3 | Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат. | |
| 4 | Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту. | |

Підтвердження:

Оригінальність тексту роботи за результатами перевірки системою StrikePlagiarism складає 97,9%, оригінальність тексту роботи за результатами перевірки системою Anti-Plagiarism v-15.257 складає 99%.

Згідно з правилами чинного Положення «Про систему забезпечення академічної доброчесності у хмельницькому національному університеті» від 31.08.2023, авторська робота, обсяг оригінального тексту у відсотках до загального обсягу матеріалу в якій складає 90-100 %, визнається роботою з високою унікальністю тексту і допускається до захисту.

Керівник роботи _____ В.М. Джулій

Гарант ОП _____ В.М. Чешун

Завідувач кафедри кібербезпеки _____ Ю.П. Кльоц

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

# РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

освітнього ступеня «бакалавр»

Студент Вишневський Денис Якович

Тема Система виявлення вразливостей типу «SQL-injection» у вебдодатках
Спеціальність 125 – Кібербезпека

**Обсяг кваліфікаційної роботи освітньо-кваліфікаційного рівня «бакалавр»:**

кількість листів креслень 2; кількість сторінок записки 65.

1. Короткий зміст роботи та прийнятих рішень У кваліфікаційній роботі було розроблено систему виявлення вразливостей типу SQL injection у вебдодатках. Розроблена система орієнтована на автоматизований пошук SQL-ін'єкцій за допомогою спеціалізованих пейлоадів, що впроваджуються у параметри HTTP-запитів, та подальший аналіз відповіді сервера на предмет наявності аномалій. У процесі реалізації проєкту була побудована модульна архітектура з чітким розділенням функцій між окремими компонентами: графічний інтерфейс на базі Tkinter, модуль логіки перевірки, модуль генерації пейлоадів, система логування результатів у файл та інші допоміжні функції. Було здійснено тестування системи на локальному сервері та визначено приклади вразливих запитів. У пояснювальній записці детально описано всі етапи створення, включно з аналізом предметної області, алгоритмом виявлення, реалізацією архітектури та практичними прикладами застосування.

2. Висновок про відповідність кваліфікаційної роботи завданню. У кваліфікаційній роботі повністю виконано поставлені завдання як у теоретичній, так і практичній частинах. Студент чітко дотримався технічного завдання, сформувавши систему, що демонструє здатність ефективно виявляти одну з найпоширеніших категорій вебуразливостей — SQL injection. Структура і зміст роботи відповідають заявленій тематиці

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У першому розділі проведено ґрунтовний аналіз стану безпеки вебдодатків, виявлено сучасні підходи до захисту від SQL-ін'єкцій та сформульовано проблематику, яку вирішує запропонована система. В другому розділі представлено проєктування архітектури системи виявлення вразливостей, реалізовано ієрархію логічних блоків, розроблено алгоритм аналізу запитів і реакцій вебсервера. У третьому розділі здійснено реалізацію повнофункціонального програмного продукту з графічним інтерфейсом для Windows, проведено локальне тестування, описано сценарії використання та задокументовано результати логування. Усі реалізовані компоненти відповідають сучасним вимогам безпеки та програмної інженерії. У роботі застосовано актуальні підходи до аналізу HTTP-взаємодій, обробки виключень, збереження логів та побудови GUI.

4. Позитивні сторони роботи Розроблена система демонструє глибоке розуміння теми веббезпеки. Архітектура побудована з урахуванням принципів модульності, розширюваності та масштабованості. Наявність графічного інтерфейсу робить систему зручною для використання як технічними спеціалістами, так і користувачами-початківцями. Тематика роботи є актуальною та практично значущою в умовах зростаючої кількості атак на вебресурси.

5. Негативні сторони роботи У роботі не передбачено підтримку багатопоточності, що могло б значно підвищити ефективність сканування великих вебресурсів. Також відсутнє розмежування прав доступу або інтеграція з системами авторизації, що було б доречним для професійного використання у корпоративному середовищі. Робота з мережею обмежується базовими HTTP-запитами, без використання proxy- або VPN-інтеграції, що могло б підвищити анонімність сканування.

6. Оцінка графічного оформлення та пояснювальної записки роботи. Оформлення пояснювальної записки відповідає встановленим вимогам до кваліфікаційних робіт. Усі ілюстрації, схеми архітектури, приклади коду та таблиці логічно вписані в структуру документа та допомагають краще зрозуміти викладений матеріал. Текст подано грамотно, без стилістичних та технічних помилок.

7. Відгук про роботу в цілому Кваліфікаційна робота має чітку, логічну структуру, кожен її розділ розкриває ключові аспекти проєкту: від постановки проблеми — до реалізації повноцінної програмної системи. Робота свідчить про високий рівень самостійної підготовки студента, здатність вирішувати складні інженерні задачі та застосовувати сучасні знання в галузі кібербезпеки. Практична цінність реалізованого інструменту підтверджується результатами тестування та логами виконаних запитів.

8. Інші зауваження У роботі спостерігається загальна відповідність вимогам щодо структури та змісту, проте у списку джерел зустрічаються загальнодоступні інтернет-ресурси, які не мають належного наукового рівня. Рекомендується у майбутньому приділяти більше уваги використанню фахової літератури, технічної документації, нормативних стандартів у сфері кібербезпеки, а також академічних публікацій. Це дозволить зробити науковий апарат більш обґрунтованим, а результати – переконливішими й ґрунтовнішими.

9. Оцінка кваліфікаційної роботи Ураховуючи всі позитивні та негативні сторони кваліфікаційної роботи, її практичну спрямованість, актуальність тематики, глибину реалізації та якість оформлення, вважаю, що вона заслуговує оцінки «відмінно».

РЕЦЕНЗЕНТ (прізвище, ім'я, по батькові, посада, місце роботи)
Бойко Юлій Миколайович,
доктор технічних наук, професор кафедри ТМІТ

« 05 » 06 2025. _____ (підпис)