НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці

ДО ЗАХИСТУ ДОПУЩЕНО
В.о. завідувача кафедри
_____ Олександр КОВАЛЬ
«___»_____2025 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем в енергетиці»
спеціальності 121 Інженерія програмного забезпечення

на тему: «Програмне забезпечення для створення захисту конфіденційної інформації баз даних від SQL-атак»

Виконав:
студент IV курсу, групи ТВ-11
Аржанцев Михайло Максимович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник:
доцент кафедри інженерії програмного забезпечення в енергетиці, кандидат технічних наук, доцент Шуклін Г.В. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент:
доцент кафедри цифрових технологій в енергетиці, кандидат технічних наук, доцент Довженко Н.М. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень із праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2025

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем в енергетиці»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Олександр КОВАЛЬ
«___»_____2025р.

ЗАВДАННЯ

на дипломну роботу студенту

Аржанцеву Михайлу Максимовичу
(прізвище, ім'я, по батькові)

1. Тема роботи: «Програмне забезпечення для створення захисту конфіденційної інформації баз даних від SQL-атак»
керівник роботи Шуклін Герман Вікторович, доцент, к.т.н.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «02» червня 2025р. № 1875-с
2. Строк подання студентом роботи «09» червня 2025р.
3. Вихідні дані до роботи: мови програмування PHP, CSS, HTML, середовище розробки VS Code, система керування базами даних MySQL.

4. Зміст (дипломної роботи) пояснювальної записки (перелік завдань, які потрібно розробити): проаналізувати методи захисту конфіденційної інформації баз даних, розробити програмне забезпечення для захисту від SQL-атак, розробити програмний продукт для демонстрації роботи.

5. Перелік ілюстративного матеріалу: діаграма варіантів використання, діаграма послідовностей, демонстрація роботи програми, користувацький інтерфейс.

6. Дата видачі завдання «31» жовтня 2024р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	15.10.2024	Виконано
2.	Вивчення та аналіз задачі	16.10.2024-16.12.2024	Виконано
3.	Розробка архітектури та загальної структури системи	09.03.2025-23.03.2025	Виконано
4.	Розробка структур окремих підсистем	24.03.2025-10.04.2025	Виконано
5.	Програмна реалізація системи	14.04.2025-10.05.2025	Виконано
6.	Оформлення пояснювальної записки	10.05.2025-13.06.2025	Виконано
7.	Захист програмного продукту	13.05.2025	Виконано
8.	Передзахист	02.06.2025	Виконано
9.	Захист	16.06.2025 - 27.06.2025	Виконано

Студент _____ Михайло Аржанцев
(підпис) (ім’я, прізвище)

Керівник роботи _____ Герман Шуклін
(підпис) (ім’я, прізвище)

РЕФЕРАТ

Структура та обсяг дипломної роботи. Робота містить 67 сторінок, 15 рисунків, 2 додаток та 25 посилань.

Метою роботи є дослідження вразливостей баз даних до SQL-ін’єкцій та розробка методики захисту конфіденційної інформації шляхом впровадження надійних механізмів обробки запитів.

Практичне значення одержаних результатів полягає в розробці програмного продукту, створеного для глибокого розуміння механізмів SQL-ін'єкцій та демонстрації ефективних методів захисту. Це сприяє підвищенню обізнаності розробників про важливість безпечних практик кодування та слугує основою для подальших досліджень у сфері кібербезпеки вебдодатків.

Ключові слова: SQL-ін’єкція, база даних, параметризовані запити, PHP, PDO, авторизація, реєстрація.

ABSTRACT

Structure and scope of the thesis. The work contains 67 pages, 15 figures, 2 appendiceses and 25 references.

The purpose of the work is the study of database vulnerabilities to SQL injections and the development of a methodology for protecting confidential information by implementing reliable query processing mechanisms.

The practical value of develop a software product designed to provide a deep understanding of SQL injection mechanisms and demonstrate effective protection methods. This helps to raise awareness among developers about the importance of secure coding practices and serves as a basis for further research in the field of web application cybersecurity.

Keywords: SQL injection, database, parameterized queries, PHP, PDO, authorization, registration.

.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ 7
ВСТУП 8
1 ПОСТАНОВКА ЗАДАЧІ ТА ОСНОВНІ ЗАВДАННЯ 9
1.1 Значення та важливість проблеми захисту від SQL-ін'єкцій 9
1.2 Задача та основні завдання 10
1.3 Сучасні підходи для створення безпечних програмних систем 11
Висновки до розділу 1 12
2 АНАЛІЗ ЛІТЕРАТУРИ ТА ОГЛЯД ІСНУЮЧИХ РІШЕНЬ 144
2.1 Принцип дій та особливості SQL-ін'єкцій 14
2.2 Історія дослідження SQL-ін'єкцій 15
2.3 Класифікація SQL-ін'єкцій за типами та методами експлуатації 17
2.4 Аналіз потенційних наслідків від SQL-ін'єкцій 18
Висновки до розділу 2 19
3 ЗАСОБИ РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 200
3.1 Обгрунтування вибору мови програмування 20
3.2 ХАМРР та його компоненти 21
3.3 Обгрунтування вибору системи керування базами даних 23
3.4 Обгрунтування вибору середовища розробки 24
Висновки до розділу 3 27
4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ 28
4.1 Архітектура розробленої системи 28
4.2 Структура проекту та його основні компоненти 30
4.3 Реалізація захисту від SQL-ін'єкцій 32
4.4 Демонстрація SQL-ін'єкцій у незахищеній авторизації 33
4.5 Демонстрація SQL-ін'єкцій у захищеній авторизації 38
Висновки до розділу 4 42
5 РОБОТА КОРИСТУВАЧА З РОЗРОБЛЕНОЮ СИСТЕМОЮ 43
5.1 Загальний огляд користувацького інтерфейсу 43
5.2 Опис головної сторінки проекту 44
5.3 Опис інформаційної сторінки проекту 45
5.4 Опис сторінок авторизації та реєстрації 48
Висновки до розділу 5 50
ВИСНОВКИ 51
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 52
ДОДАТОК А Лістинг розробленої системи 54
ДОДАТОК Б Презентація 59

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

IDE – інтегроване середовище розробки.

VS Code (англ. Visual Studio Code) — редактор вихідного коду, розроблений Microsoft для Windows, Linux та MacOS.

СКБД — Система керування базами даних.

UI (англ. User Interface) — призначений для користувача інтерфейс.

БД — База даних.

SQL (англ. Structured Query Language) — структурована мова запитів, яка використовується для керування та маніпулювання реляційними базами даних

XAMPP (англ. Cross-platform, Apache, MySQL, PHP, Perl) — безкоштовний програмний пакет, що включає вебсервер Apache, СУБД MySQL/MariaDB, інтерпретатори сценаріїв PHP і Perl, призначений для швидкого розгортання локального вебсервера.

SQLi (англ. SQL Injection) — SQL-ін'єкція, тип атаки, що полягає у впровадженні шкідливого SQL-коду у запити до бази даних через поля вводу користувача.

HTML (англ. HyperText Markup Language) — стандартизована мова гіпертекстової розмітки документів для перегляду вебсторінок у браузері.

CSS (англ. Cascading Style Sheets) — мова таблиць стилів, що використовується для опису зовнішнього вигляду документа, написаного мовою розмітки.

ВСТУП

Питання безпеки інформації набуває особливої актуальності за сучасних умов швидкого розвитку інформаційних технологій і зростання обсягів даних. Бази даних стали обов'язковою складовою більшості інформаційних систем, що використовуються в усіх важливих галузях, зокрема в державному управлінні, економіці, банківській сфері, енергетиці, освіті, охороні здоров'я тощо. Водночас з розвитком технологій удосконалюються й засоби кіберзлочинності, серед яких особливу загрозу становлять SQL-ін'єкції — один із найпоширеніших видів атак на бази даних.

Мета дипломного проекту полягає у зібранні відповідної інформації, створенні програмного забезпечення для захисту конфіденційної інформації баз даних від SQL-атак шляхом впровадження надійних методики перевірки, обробки та контролю введених даних, а також моделювання та імітації типових сценаріїв атак з метою виявлення вразливостей.

Актуальність теми дипломного проекту зумовлена потребою у створенні надійних програмних рішень, здатних ефективно захищати бази даних від подібних атак. Проблема SQL-ін'єкцій залишається актуальною через недосконалу реалізацію або нехтування захисними механізмами на етапі розробки програмного забезпечення, незважаючи на наявність різних засобів безпеки.

Саме проблема недостатньої або неправильної реалізації захисту від SQL-атак залишається невирішеною в багатьох прикладних системах, тому необхідність комплексного підходу до виявлення та нейтралізації таких загроз є основною проблемою, на вирішення якої спрямовано цей проект.

1. ПОСТАНОВКА ЗАДАЧІ ТА ОСНОВНІ ЗАВДАННЯ

1.1 Значення та важливість проблеми захисту від SQL-ін’єкцій

Обсяг даних, що зберігаються, обробляються та передаються через електронні системи, у сучасному світі цифрових технологій постійно зростає; ці дані стосуються значного обсягу інформації – від звичайної довідкової до конфіденційних персональних, зокрема фінансових та медичних, записів. Безпека цих баз стає критично важливою умовою для стабільного функціонування будь-якої сучасної інформаційної системи, оскільки більшість таких даних зберігається саме у базах даних.

Одним з найбільш небезпечних та водночас найпоширеніших типів атак на вебзастосунки є SQL-ін’єкція; суть її полягає у навмисному впровадженні шкідливих SQL-запитів у текстові поля, URL-параметри або інші точки введення користувача з метою модифікації або виконання несанкціонованих операцій з базою даних. Саме такі атаки можуть спричинити критичні наслідків, як-от: витік або знищення інформації; обхід автентифікації; зміни прав доступу або навіть повного контролю над інформаційною системою [1].

Особливої актуальності проблема SQL-ін’єкцій набула за часів глобальної цифровізації сьогодення, коли майже кожне підприємство, установа чи навіть структурний підрозділ має вебпортал або клієнтський інтерфейс, що взаємодіє з базами даних. Ці точки взаємодії з користувачем зазвичай не мають достатнього рівня перевірки введених даних, що дає можливість зловмисникам здійснити атаку навіть без спеціальних технічних знань.

За версіями провідних міжнародних організацій, зокрема Open Web Application Security Project, уразливості, пов’язані із SQL-ін’єкціями, вже не один рік поспіль містяться у списках найбільш небезпечних типів атак. Це свідчить про те, що проблема не лише не втратила своєї актуальності, а є однією з

першочергових саме у сфері інформаційної безпеки. Саме недотримання принципів безпечної розробки програмного забезпечення або ігнорування загальноприйнятих методик валідації даних, а не недостатність технічних засобів захисту, найчастіше є основною причиною успішної атаки [2].

Захист від SQL-ін'єкцій є не тільки суто технічною проблемою, а й питанням загальної культури розробки програмного забезпечення, адже вимагає комплексного підходу, що охоплює етапи аналізу вимог, проектування, реалізації, тестування та супроводу програмної системи. Звичайно, не можна раз і назавжди повністю і одночасно вирішити проблему SQL-ін'єкцій, але її можна ефективно контролювати шляхом дотримання відповідних стандартів, застосування найкращих практик безпечного програмування, використання захищених інтерфейсів доступу до даних, регулярного аудиту коду, підвищення обізнаності розробників тощо.

Отже, проблема захисту від SQL-ін'єкцій є фундаментальною для будь-якої сучасної програмної системи, що має справу з базами даних; а її ігнорування може призвести до серйозних наслідки як для певної організації в цілому, так і для користувачів зокрема [3].

1.2 Задача та основні завдання

Головною метою даної дипломної роботи є розробка методики захисту конфіденційної інформації баз даних від SQL-атак, що реалізується шляхом створення програмного забезпечення, яке дає можливість виявити та ефективно нейтралізувати подібні загрози. Дослідження передбачає вивчення природи SQL-ін'єкцій, їхні різновиди, методи впровадження та наслідки для інформаційної безпеки, а також внесення пропозицій щодо практичного впровадження механізмів протидії з урахуванням сучасних технологій програмної інженерії.

Для досягнення поставленої мети у дипломній роботі визначено такі завдання:

По-перше, провести теоретичне дослідження поняття SQL-ін’єкцій, зокрема розглянути історію їх виникнення, типові механізми реалізації, класифікацію за способами впливу на систему, а також практичні приклади зловмисного впливу. Таке дослідження дозволить глибше зрозуміти загрозу та сформувати основу для наступного проектування захисту.

По-друге, здійснити аналіз сучасних підходів виявлення та запобігання SQL-ін’єкціям, зокрема використання параметризованих запитів, ORM-технологій, інструментів фільтрації вхідних даних та автоматичного тестування на вразливості. Оцінювання ефективності кожного з методів дасть можливість обґрунтувати вибір найбільш доцільного, раціонального підходу для практичного впровадження.

По-третє, розробити прикладну програму або модуль, який буде демонструвати як наявність, так і усунення вразливостей до SQL-ін’єкцій. Цей програмний продукт має включати як небезпечний функціонал, що демонструє ризик атаки, так і реалізацію захищеного функціоналу.

Дана дипломна робота має на меті зробити внесок у підвищення рівня інформаційної безпеки у сфері проектування сучасного програмного забезпечення, а також сформувати навички безпечного програмування з урахуванням актуальних кіберзагроз завдяки комплексному підходу до вивчення SQL-ін’єкцій та розробці практичного рішення.

1.3 Сучасні підходи для створення безпечних програмних систем

Одним із визначальних сучасних підходів для створення безпечних програмних систем є принцип security by design, суть якого полягає в тому, що безпека враховується ще на етапі планування функціоналу системи. Це зумовлює й передбачає попереднє виявлення потенційних векторів атак, оцінку ризиків та інтеграцію захисних механізмів безпосередньо у логіку програми. Наприклад, при

проектуванні бази даних передбачається використання ролей і обмежень доступу, а в інтерфейсах — обробка та фільтрація вхідних даних.

Не менш важливим підходом є використання шаблонів безпечного програмування: це набір практик і конструкцій, що допомагають уникнути поширених помилок, які призводять до вразливостей. До таких шаблонів належать: застосування параметризованих SQL-запитів замість динамічних, принцип найменших привілеїв при роботі з ресурсами, обмеження довжини та типу даних, обов'язкова валідація введення, перевірка авторизації користувача на кожному рівні доступу до системи [4].

Важливу роль у забезпеченні безпеки відіграє автоматизоване тестування на вразливості. Сучасні інструменти, зокрема OWASP ZAP, SQLMap, Burp Suite, дозволяють сканувати програми на наявність типових загроз, серед яких і SQL-ін'єкції. Виявленню та усуненню помилок ще до того, як продукт потрапить до кінцевого користувача, сприяє регулярне використання таких засобів у процесі розробки [10].

Не менш важливим фактором є також застосування сучасних фреймворків і бібліотек, що вже містять вбудовані засоби захисту. Наприклад, популярні вебфреймворки для PHP, Python або JavaScript часто автоматично обробляють запити у спосіб, який виключає SQL-ін'єкції або XSS. Водночас до хибного відчуття безпеки може призвести надмірна довіра до таких засобів без перевірки їхньої коректної конфігурації.

Впровадження вищенаведених практик дозволяє не лише зменшити ризики SQL-ін'єкцій, а й підвищити загальний рівень стійкості програмного забезпечення до кіберзагроз [5].

Висновки до розділу 1

У першому розділі було розглянуто актуальність проблеми захисту інформаційних систем від SQL-ін’єкцій як одного з найпоширеніших і небезпечних типів атак на бази даних.

Було з’ясовано, що нехтування питаннями безпеки на етапах розробки програмного забезпечення може призвести до серйозних наслідків — витоку конфіденційної інформації, знищення або модифікації даних, а також повної компрометації системи.

Було обґрунтовано значення захисту від SQL-ін’єкцій, зокрема у контексті зростання цифровізації та поширення вебзастосунків, що працюють з базами даних. Підкреслено, що SQL-ін’єкції можуть бути як простими, так і високотехнологічними, але завжди становлять серйозну загрозу для збереження цілісності та конфіденційності інформації.

Сформульовано основну мету і завдання дослідження: основна мета полягає у розробці програмного забезпечення для захисту конфіденційної інформації в базах даних від SQL-атак; серед завдань найважливіші — аналіз типів ін’єкцій, дослідження їх наслідків, огляд методів захисту, а також розробка і реалізація практичного рішення.

2. АНАЛІЗ ЛІТЕРАТУРИ ТА ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

2.1 Принцип дії та особливості SQL-ін’єкцій

SQL-ін’єкція — це тип атаки на вебдодатки, при якому зловмисник вставляє шкідливі SQL-команди в поле вводу користувача з метою несанкціонованого доступу до бази даних, отримання, зміни або знищення її вмісту. Саме такі атаки в контексті сучасних програмних систем стали одними з найпоширеніших і найбільш небезпечних загроз для інформаційної безпеки. Нескладні в реалізації, вони здатні призводити до серйозних наслідків — від розголошення конфіденційної інформації до встановлення повного контролю над сервером.

Принцип роботи SQL-ін’єкції полягає у тому, що зловмисник передає у запит до бази даних спеціально сформовану послідовність символів, яка інтерпретується сервером не як текстове значення, а як частина синтаксису SQL-команди. Це дає можливість зловмиснику впливати на структуру SQL-запиту та змінювати його поведінку. Така атака можлива найчастіше через вебформи URL-параметри, або HTTP-заголовки, якщо на стороні сервера не передбачено належної обробки або фільтрації вхідних даних [6].

Найпростішим прикладом SQL-ін’єкції є ситуація, коли на сервері використовується запит у вигляді: SELECT * FROM users WHERE username = '$username' AND password = '$password'; Якщо користувач замість звичайного логіну введе наступний рядок: ' OR '1'='1, то результуючий SQL-запит набуде вигляду: SELECT * FROM users WHERE username = '' OR '1'='1' AND password = '';

Цей запит завжди повертає істину, оскільки '1'='1' є завжди істинним виразом. У результаті зловмисник може отримати доступ до облікового запису, навіть не знаючи справжнього пароля. У випадках, коли система не обмежує кількість повернених записів або має логіку, що ґрунтується на першому збігу, це може призвести до входу в систему під першим знайденим користувачем [7].

Особливістю SQL-ін’єкцій є те, що вони не завжди очевидні. Іноді вони використовуються не для отримання доступу до облікових записів, а для отримання інформації про структуру бази даних. Також SQL-ін’єкція може бути використана для виконання складних команд, зокрема створення нових облікових записів, завантаження шкідливого коду, зміни або видалення вмісту бази даних.

Принцип дії SQL-ін’єкцій базується на маніпуляції синтаксисом SQL-запитів, тоді як особливості їх застосування охоплюють як прості атаки з миттєвим ефектом, так і складні багаторівневі експлуатації, здатні серйозно вплинути на інформаційну безпеку організації [8].

2.2 Історія дослідження SQL-ін’єкцій

Виникнення SQL-ін'єкцій сягає ранніх етапів розвитку вебзастосунків, що активно використовували бази даних для зберігання та обробки інформації. Перші згадки про цю вразливість та її успішну експлуатацію датуються кінцем 1990-х – початком 2000-х років. Недостатня увага розробників до безпечної обробки даних, що надходили від користувача, стала однією з головних причин їхньої появи. Ці дані часто безпосередньо включалися в SQL-запити, що відкривало зловмисникам можливість маніпулювати логікою запитів та виконувати небажані операції з базою даних, замість належної перевірки та екранування. Експлуатація SQL-ін'єкцій на ранніх етапах зазвичай була відносно нескладною і тому могла призводити до серйозних наслідків, як-от несанкціонований доступ до конфіденційної інформації, її модифікацію або навіть повне видалення [9].

Згодом техніки SQL-ін'єкцій еволюціонували - з'явилися різноманітні типи атак, що використовували різні особливості SQL та вебзастосунків. Зловмисники розробили більш складні методи, окрім базових логічних ін'єкцій, що маніпулювали умовами WHERE, та ін'єкцій на основі об'єднання, що дозволяли отримувати дані з інших таблиць. До них належать сліпі SQL-ін'єкції, де зловмисник не отримує прямих даних у відповідь, а робить висновки на основі

непрямих ознак, як-от час виконання запиту або наявність/відсутність певних елементів у відповіді. Також з'явилися ін'єкції другого порядку, де шкідливий код спочатку зберігається в базі даних, а потім активується пізніше при іншому запиті. Разом з розвитком технік атак почали з'являтися інструменти, що автоматизували процес пошуку та експлуатації SQL-ін'єкцій, значно спрощуючи завдання для зловмисників. Ці інструменти адаптувалися до нових версій SQL та вебтехнологій, підтримуючи широкий спектр технік ін'єкцій [11].

Боротьба з SQL-ін'єкціями також пройшла значний шлях розвитку. На ранніх етапах основні методи захисту зводилися до спроб екранування спеціальних символів у вхідних даних. Однак, через складність передбачення всіх можливих варіантів шкідливих вхідних даних та різницю в синтаксисі SQL у різних СУБД, цей підхід виявився недостатньо надійним. Значним кроком вперед стала поява та широке впровадження підготовлених виразів або параметризованих запитів: ця технологія дозволяє відокремити структуру SQL-запиту від даних, що передаються до нього. Введені користувачем дані розглядаються лише як параметри і не можуть бути інтерпретовані як виконуваний SQL-код, що ефективно запобігає класичним SQL-ін'єкціям.

З розвитком веброзробки важливу роль у забезпеченні безпеки даних почали відігравати вебфреймворки та системи Object-Relational Mapping. Ці інструменти часто мають вбудовані механізми для безпечної взаємодії з базами даних, автоматично обробляючи екранування та використовуючи підготовлені вирази, що значно знижує ризик SQL-ін'єкцій. Почали широко використовуватися на етапах розробки та тестування інструменти статичного аналізу коду та динамічного аналізу безпеки вебзастосунків, які допомагають виявляти потенційні вразливості, включаючи SQL-ін'єкції, ще до етапу розгортання. Важливим елементом захисту на рівні інфраструктури стали між мережеві екрани вебзастосунків, які аналізують HTTP-трафік та можуть блокувати шкідливі запити, що містять ознаки SQL-ін'єкцій [12].

2.3 Класифікація SQL-ін’єкцій за типами та методами експлуатації

Один із найпоширеніших типів атак - пряма SQL-ін’єкція, коли зловмисник у поля введення користувача безпосередньо вставляє шкідливий SQL-код. Такий код виконується на сервері бази даних без попередньої фільтрації, що дозволяє отримати доступ до всієї таблиці або обійти механізми авторизації. Наприклад, anything' OR 'x'='x або ' OR 1=1; -- вираз OR 1=1 завжди повертає істину, дозволяючи авторизуватися без правильного пароля. Цей тип ін’єкцій часто використовується для несанкціонованого входу в систему або перегляду даних усіх користувачів [13].

UNION-based SQL-ін’єкція використовує ключове слово UNION для об’єднання результатів кількох SQL-запитів, що дає можливість зловмиснику додати до початкового запиту свій власний, отримавши додаткову інформацію з бази даних. Наприклад, рядок ' UNION SELECT 1, @@version, "123" # дозволяє зчитати версію СУБД. Цей тип ін’єкції особливо небезпечний, оскільки може надати прямий доступ до внутрішніх даних системи, які не передбачені у звичайному інтерфейсі додатку.

Error-based SQL-ін’єкція дозволяє зловмиснику отримати інформацію через повідомлення про помилки, які генерує база даних. Наприклад, запит ' AND EXTRACTVALUE(1, CONCAT(0x7e, (SELECT database()), 0x7e)) -- спеціально викликає помилку, яка містить назву бази даних у тексті виключення. Цей тип атаки ґрунтується на особливостях поведінки СУБД у разі некоректних запитів і дозволяє отримувати критично важливі дані шляхом навмисного створення помилок [14].

Time-based Blind SQL-ін’єкція використовується в ситуаціях, коли система не виводить жодних повідомлень про помилки або результатів запиту. У таких випадках зловмисник аналізує час відповіді сервера. Наприклад, admin' OR IF(1=1, SLEEP(5), 0) -- виконує затримку на 5 секунд, якщо умова істинна. Якщо відповідь

сервера затримується, це свідчить про успішне виконання ін’єкції. Такий підхід дозволяє поступово витягувати дані з бази даних навіть за повної відсутності виводу, використовуючи часові патерни.

Багато запитна SQL-ін’єкція дає можливість виконувати кілька SQL-операцій за один запит, розділяючи їх крапкою з комою. Це найбільш небезпечний тип SQL-ін’єкції, оскільки може призвести до повної компрометації бази даних. Наприклад, DROP TABLE users; -- видаляє таблицю користувачів після завершення першого запиту. Такий підхід використовують для знищення або змінення даних; він особливо критичний, якщо база даних не захищена від виконання множинних операторів у одному запиті [15].

2.4 Аналіз потенційних наслідків від SQL-ін’єкцій

Через потенційно катастрофічні наслідки, які можуть виникнути в результаті їх успішного використання, SQL-ін’єкції становлять одну з найсерйозніших загроз для інформаційної безпеки програмних систем. Уразливість до SQL-ін’єкцій відкриває можливості для несанкціонованого доступу до конфіденційної інформації, порушення цілісності даних, блокування доступу до сервісів, а в окремих випадках — повного контролю над сервером чи інфраструктурою.

Витік конфіденційної інформації - однин з найбільш поширених наслідків. Зловмисник може отримати доступ через SQL-ін’єкції до особистих даних користувачів, таких як логіни, паролі, адреси електронної пошти, фінансова інформація тощо [16].

Окрім витоку даних, можливе несанкціоноване редагування, видалення або вставлення даних у базу. Це означає, що зловмисник може навмисно змінити баланс на рахунках, підробити транзакції, змінити важливу інформацію або навіть повністю знищити базу даних. Особливо небезпечні подібні атаки для фінансових, медичних і урядових установ, де точність і цілісність даних мають критичне значення.

У деяких випадках SQL-ін’єкції дають можливість захоплення адміністративного доступу, наприклад, через зміну облікових даних адміністратора або ін’єкцію команд для створення нового облікового запису з максимальними правами. Це дає можливість встановлення повного контролю над інформаційною системою, також і можливість інфікування серверу шкідливим ПЗ або організації подальших атак всередині корпоративної мережі.

Отже, потенційні наслідки SQL-ін’єкцій охоплюють широкий спектр проблем. Саме тому критично важливими складовими стратегії інформаційної безпеки є своєчасне виявлення вразливостей, використання сучасних засобів захисту та регулярне тестування безпеки програмного забезпечення [18].

Висновки до розділу 2

У другому розділі було проведено ґрунтовний аналіз SQL-ін’єкцій, її сутності, особливостей реалізації та загроз, які вона несе для інформаційної безпеки програмних систем.

Детально розглянуто принцип дії цієї атаки, прокоментовано, як саме з метою отримання несанкціонованого доступу до бази даних або її модифікації зловмисники можуть впливати на структуру SQL-запитів.

За методами та техніками реалізації наведено класифікацію SQL-ін’єкцій, що дозволяє глибше зрозуміти різноманітність можливих векторів атаки та ступінь їх небезпеки.

Проаналізовано потенційні наслідки успішного здійснення SQL-ін’єкції, серед яких: витік конфіденційних даних, порушення цілісності або доступності інформації, захоплення контролю над системою та репутаційні та юридичні втрати для організацій.

.

3. ЗАСОБИ РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Обґрунтування вибору мови програмування

Одним із найважливіших етапів при розробці будь-якого програмного продукту є вибір мови програмування, оскільки саме він визначає доступні інструменти, фреймворки, а також впливає на швидкість розробки, продуктивність та масштабованість системи. Для реалізації вебдодатку, що ілюструє механізми SQL-ін'єкцій та методи захисту від них, було обрано мову програмування PHP.

Переваги цієї мови програмування, які роблять її оптимальним рішенням для даного типу проекту, такі:

По-перше, PHP - одна з найпопулярніших серверних мов програмування, що активно використовується для розробки динамічних вебсайтів. За даними W3Techs, PHP працює на 77,5% усіх вебсайтів, серверна частина яких відома. Така значна поширеність зумовлює велику кількість доступних бібліотек, фреймворків, документації та активну спільноту розробників, що значно спрощує пошук рішень та підтримку проекту. Для даного проекту це важливо, оскільки дозволяє створити максимально реалістичне середовище, в якому потенційно можуть виникати вразливості.

По-друге, PHP має відносно низький поріг входження, що робить його доступним для швидкого освоєння та ефективного використання у проектах різної складності. Його синтаксис інтуїтивно зрозумілий і має схожість з C-подібними мовами, що спрощує розробку та налагодження коду. Простота мови є значною перевагою для створення додатку, де головним є ясність представлення концепцій SQL-ін'єкцій.

По-третє, PHP розроблявся з орієнтацією на веброзробку і на взаємодію з базами даних. Він має вбудовану підтримку для роботи з великою кількістю СУБД, включаючи MySQL та MariaDB, які є центральними для даного проекту. Це здійснюється за допомогою спеціальних розширень, зокрема MySQL Improved

Extension або PHP Data Objects. Останній, PDO, є особливо важливим для цієї роботи, оскільки він надає уніфікований інтерфейс для доступу до баз даних і, що найважливіше, підтримує підготовлені вирази – ключовий механізм для захисту від SQL-ін'єкцій. Ця можливість PDO дозволяє наочно продемонструвати різницю між вразливим та захищеним кодом.

По-четверте, PHP виконується на стороні сервера, що є типовим для більшості вебдодатків, які взаємодіють з базами даних. Це дозволяє моделювати реальні сценарії атак, оскільки SQL-ін'єкції виникають саме в процесі формування та виконання запитів на сервері.

По-п’яте, PHP є невід'ємною частиною багатьох популярних пакетів локальних вебсерверів, таких як XAMPP, WAMP, LAMP. Даючи можливість зосередитися на логіці програми, а не на конфігурації інфраструктури, це спрощує налаштування середовища розробки та його розгортання.

Отже, вибір PHP як основної мови програмування для даного проекту обумовлений його поширеністю у веброзробці, простотою інтеграції з базами даних, потужними інструментами для забезпечення безпеки, а також широкою доступністю ресурсів та середовищ розробки. Це дозволяє ефективно вирішити поставлені задачі та наочно ілюструвати принципи роботи SQL-ін'єкцій та методів протидії їм [17].

3.2 XAMPP та його компоненти

Розробка вебзастосунків потребує стабільного, безпечного та зручного середовища для написання, тестування та налагодження коду до моменту розгортання системи на продуктивному сервері. Одним із найбільш популярних рішень для таких завдань є XAMPP — крос-платформний програмний пакет, який містить все необхідне для локального хостингу PHP-застосунків з використанням бази даних MySQL або MariaDB.

XAMPP (від англ. Cross-platform, Apache, MySQL, PHP, Perl) – це повністю безкоштовний, крос-платформний пакет програмного забезпечення, розроблений компанією Apache Friends. Його основне призначення полягає у наданні розробникам готового до використання локального, комплексного вебсервера. Назва XAMPP є акронімом, де "X" позначає "крос-платформність", а інші літери – ключові компоненти: вебсервер Apache, система управління базами даних MariaDB, мова програмування PHP та мова сценаріїв Perl [19].

XAMPP також має графічний інтерфейс користувача – XAMPP Control Panel, який дозволяє легко керувати основними компонентами: запускати, зупиняти, перезавантажувати сервіси, а також отримувати доступ до їх конфігураційних файлів та логів. Ця панель управління робить роботу з локальним сервером інтуїтивно зрозумілою навіть для початківців.

Вибір XAMPP як середовища розробки для даного дипломного проекту є обґрунтованим з кількох причин. По-перше, його простота встановлення та налаштування значно прискорює початок роботи, усуваючи необхідність окремо інсталювати та конфігурувати кожен компонент, що особливо важливо для проектів, де увага зосереджується на функціоналі самого додатку, а не на складнощах налаштування інфраструктури. По-друге, XAMPP надає інтегроване середовище, яке максимально імітує реальні виробничі сервери, що робить розробку та тестування більш ефективними та зменшує вірогідність виникнення проблем при розгортанні проекту на хостингу. По-третє, його крос-платформність забезпечує гнучкість та дозволяє працювати над проектом незалежно від операційної системи [20].

XAMPP має такі програмні компоненти, як Apache, MariaDB, PHP, phpMyAdmin. Надаємо їх стислі характеристики:

Apache HTTP Server є ключовим елементом XAMPP. Це найбільш поширений у світі вебсервер, який забезпечує підтримку основних стандартів протоколу HTTP. У рамках локального середовища Apache обробляє запити до

PHP-файлів, які розміщені в папці htdocs, та повертає результати у вигляді HTML-сторінок через браузер.

Початково XAMPP постачався з MySQL, однак у новіших версіях він використовує MariaDB — розгалуження MySQL із відкритим вихідним кодом. Обидві системи є сумісними з SQL-запитами, тож їх використання в проекті не потребує змін у логіці взаємодії з базою даних. Особливо важливо в контексті цієї дипломної роботи те, що система забезпечує збереження, обробку та захист даних.

PHP є основною мовою програмування, яку підтримує XAMPP. Саме PHP використовується для реалізації серверної логіки, взаємодії з базами даних, обробки форм користувачів, перевірки прав доступу тощо. Середовище XAMPP дозволяє працювати з будь-якою версією PHP та легко змінювати її в налаштуваннях [21].

3.3 Обґрунтування вибору системи керування базами даних

Система керування базами даних є фундаментальним компонентом будь-якого вебдодатку, що працює з даними. Вибір СУБД має прямий вплив на архітектуру програми, її продуктивність, масштабованість, на можливості взаємодії та демонстрації вразливостей. Для реалізації даного проекту було обрано MySQL та MariaDB**.**

MySQL є однією з найпопулярніших у світі реляційних систем управління базами даних з відкритим вихідним кодом. Вона широко використовується у вебдодатках, а також у великих корпоративних системах. MySQL була придбана компанією Oracle Corporation 2010 року.

Побоюючись можливих змін у ліцензуванні та стратегії розвитку, після придбання MySQL компанією Oracle спільнота розробників та частина оригінальних творців MySQL створили форк проекту, який отримав назву MariaDB. MariaDB зберігає повну бінарну та програмну сумісність з MySQL, що означає: програми, розроблені для MySQL, можуть легко перейти на MariaDB без

необхідності внесення суттєвих змін у код. MariaDB активно розвивається спільнотою і включає низку покращень щодо продуктивності, безпеки та нових функцій, яких немає в MySQL [22].

MySQL та MariaDB є реляційними СУБД. Це означає, що дані зберігаються в таблицях, які пов'язані між собою визначеними зв'язками. Така модель є найбільш поширеною та зрозумілою для більшості розробників, що спрощує проектування структури бази даних для проекту. Більшість SQL-ін'єкцій також спрямовані саме на реляційні бази даних, тому вибір MySQL/MariaDB дозволяє максимально точно імітувати реальні сценарії атак.

Обидві СУБД є високопродуктивними та надійними, здатними обробляти великі обсяги даних та одночасні запити. Для проекту, де потрібно багаторазово виконувати запити та спостерігати за їхньою поведінкою, стабільність системи є важливою.

Отже, вибір MySQL/MariaDB як системи керування базами даних для даного проекту є рішенням, що дозволяє не лише ефективно зберігати та керувати даними, але й повноцінно продемонструвати механізми SQL-ін'єкцій, а також ефективність впроваджених механізмів захисту, використовуючи стандартні та широко застосовувані у веброзробці інструменти [23].

3.4 Обґрунтування вибору середовища розробки

Без використання відповідних інструментів, що забезпечують зручність написання коду, його налагодження, управління проектом та взаємодію з базами даних, ефективна розробка програмного забезпечення неможлива. Для реалізації проекту були обрані Visual Studio Code як основне інтегроване середовище розробки та phpMyAdmin для адміністрування бази даних.

VS Code

Visual Studio Code — це потужне, безкоштовне та крос-платформне інтегроване середовище розробки, створене компанією Microsoft. З моменту свого

запуску 2015 року воно стало одним із найпопулярніших редакторів коду серед розробників у світі. VS Code ідеально підходить як для початківців, так і для професійних програмістів завдяки своїй гнучкості, розширюваності та підтримці великої кількості мов програмування та технологій.

Модульна архітектура є однією з ключових особливостей VS Code. Базовий функціонал редактора може бути легко розширений за допомогою великої кількості доступних розширень, які додають підтримку мов програмування, інструментів для налагодження, інтеграцію з системами контролю версій, зокрема Git, а також інтерфейси для роботи з базами даних, терміналами, серверами та іншими сервісами.

Visual Studio Code побудований на базі платформи Electron і використовує HTML, CSS та JavaScript. Завдяки цьому, він може працювати на всіх основних операційних системах, включаючи Windows, Linux та macOS.

VS Code особливо зручний для розробки вебдодатків та взаємодії з базами даних, що розташовані на хостингу або локальному сервері, завдяки можливості віддаленої роботи з серверами за допомогою SSH. У поєднанні з XAMPP, редактор дозволяє безперешкодно працювати над проектами, написаними на PHP, HTML, JavaScript та інших мовах, що використовуються у веброзробці.

Інтерфейс VS Code відзначається простотою, мінімалізмом та високим рівнем персоналізації. Користувачі можуть обирати теми оформлення, змінювати комбінації клавіш, налаштовувати панелі та встановлювати необхідні модулі для оптимізації робочого процесу.

Завдяки своїй легкості, відкритості, розширюваності та регулярним оновленням, Visual Studio Code є надійним вибором для проектів будь-якого масштабу, включаючи дипломні роботи, де необхідно ефективно працювати з кодом, базами даних та серверами [24].

phpMyAdmin

phpMyAdmin — це вебзастосунок з відкритим вихідним кодом, створений для зручного адміністрування баз даних MySQL та MariaDB за допомогою

вебінтерфейсу. Він був розроблений в 1998 році; з того часу став одним із найпопулярніших інструментів для керування реляційними базами даних через браузер.

Можливість працювати з базами даних без необхідності використовувати командний рядок або писати складні SQL-запити вручну - головна перевага phpMyAdmin. Інтерфейс програми інтуїтивно зрозумілий, що робить її придатною навіть для користувачів-початківців.

phpMyAdmin розроблений на мові PHP і запускається як частина стеку вебсерверів; це дає можливість швидко розгорнути систему для локальної розробки або на віддаленому сервері. Після запуску phpMyAdmin надає вебінтерфейс, через який можна підключитися до будь-якої бази даних MySQL, переглянути її вміст, змінити структуру, додати або видалити записи.

Інтерфейс програми підтримує понад 80 мов, включаючи українську, що полегшує її використання широким колом користувачів. Окрім того, phpMyAdmin підтримує тему оформлення, що дозволяє адаптувати вигляд панелі керування під власні уподобання.

phpMyAdmin також підтримує складні SQL-операції, транзакції, збережені процедури та тригери, що важливо більш досвідчених користувачів. Це робить інструмент зручним не лише для адміністративних завдань, але й для аналітичної роботи, створення звітів, аналізу зв'язків між таблицями та профілювання запитів.

Дозволяючи розробникам швидко перевірити структуру та вміст бази даних, налагодити зв’язки між таблицями, виконати тестові запити та забезпечити надійне збереження інформації, за умов розробки вебзастосунків phpMyAdmin є незамінним інструментом.

phpMyAdmin — це зручний, потужний і доступний інструмент для адміністрування баз даних, який значно спрощує розробку, тестування та обслуговування інформаційних систем [25].

Висновки до розділу 3

У третьому розділі було розглянуто інструменти та засоби, що використовуються для розробки програмного забезпечення, що демонструє вразливості до SQL-ін’єкцій та способи їх захисту. Описано мову програмування PHP як основний засіб створення серверної логіки, а також HTML, CSS і JavaScript — для реалізації інтерфейсу користувача.

Для реалізації проекту було обрано середовище розробки Visual Studio Code, що забезпечує зручну роботу з кодом, підтримує розширення для різних мов програмування, а також інтеграцію з системами контролю версій. Для управління базою даних обрано систему phpMyAdmin, яка надає вебінтерфейс для адміністрування MySQL і дозволяє ефективно працювати з таблицями, запитами, індексами та іншими об'єктами бази даних.

У роботі використовуються Apache HTTP Server та MySQL як основа для створення повноцінного локального середовища розробки за допомогою XAMPP, що спрощує налаштування серверної частини та тестування вебдодатків.

Отже, набір інструментів, обраний для реалізації програмної частини, дозволяє створити гнучке, масштабоване та зручне середовище для розробки, тестування та демонстрації як потенційних SQL-атак, так і засобів їх запобігання.

4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

4.1 Архітектура розробленої системи

Архітектура вебдодатку спроектована таким чином, щоб забезпечити чітке розмежування компонентів, що відповідають за вразливий та захищений функціонал, а також наочно проілюструвати взаємодію між користувачем, системою та базою даних. Система має класичну трирівневу архітектуру: клієнтський рівень, серверний рівень та рівень даних.

Для візуалізації функціональних вимог та динамічної поведінки системи були використані діаграма варіантів використання та діаграма послідовностей.

Діаграма варіантів використання

Діаграма варіантів використання або Use Case Diagram є одним з інструментів UML, що дозволяє візуалізувати функціональні вимоги до системи з точки зору її взаємодії з зовнішніми акторами. Вона показує, хто може використовувати систему і що саме вони можуть робити. На рисунку 4.1 зображено діаграму варіантів використання.

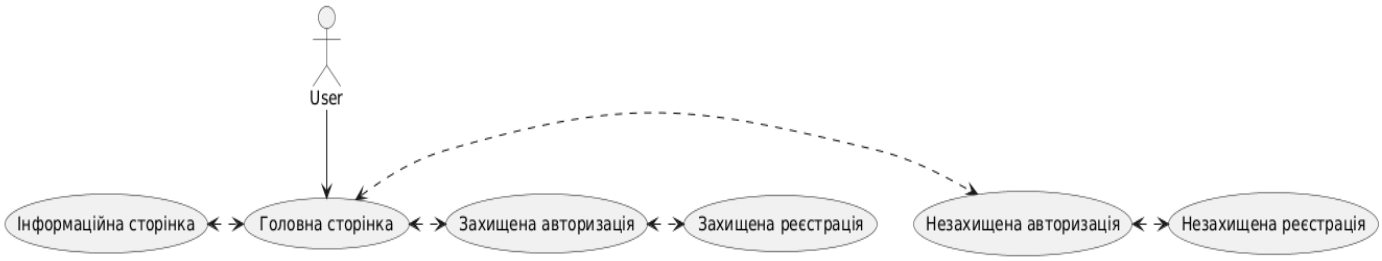


Рисунок 4.1 – Діаграма варіантів використання

Основним актором є користувач, який може взаємодіяти із системою для виконання різних функціональних завдань. До цих завдань належать: Реєстрація

користувача та Авторизація користувача, які представлені у двох варіантах – незахищена та захищена. Користувач також може перейти на інформаційну сторінку, де можна більше дізнатись про SQL-ін'єкції.

Діаграма послідовностей

Діаграма послідовностей або Sequence Diagram є іншим інструментом UML, призначеним для моделювання динамічної поведінки системи. Вона показує порядок взаємодії між об'єктами або компонентами системи у часі, відображаючи послідовність повідомлень, якими вони обмінюються для виконання певного варіанта використання. На Рисунку 4.2 представлена діаграма послідовностей.

Рисунок 4.2 – Діаграма послідовностей

Процес починається з того, що користувач вводить облікові дані. Браузер надсилає HTTP POST-запит до вебсервера Apache, який, у свою чергу, перенаправляє запит відповідному PHP-скрипту авторизації, що знаходиться на сервері.

Далі, у залежності від того, чи це вразливий чи захищений сценарій, PHP-скрипт формує запит до бази даних. У незахищеному сценарії запит формується шляхом прямої конкатенації рядків, що робить його чутливим до SQL-ін'єкцій. У захищеному сценарії, натомість, використовується механізм підготовлених виразів. Скрипт спочатку готує запит з плейсхолдерами, потім прив'язує дані користувача до цих плейсхолдерів і лише після цього виконує запит. Цей метод гарантує, що будь-який ввід користувача буде інтерпретований як дані, а не як частина SQL-коду. Після виконання запиту, база даних повертає результат до PHP-скрипта, який, після обробки формує відповідь і відправляє до вебсервера для відображення користувачеві. Ця послідовність наочно демонструє, як архітектурні рішення впливають на безпеку взаємодії з даними.

4.2 Структура проекту та його основні компоненти

Проста, але функціональна структура розробленого вебдодатку дозволяє легко ідентифікувати різні компоненти системи та їхню роль. Ця структура була спроектована так, щоб наочно відокремити вразливий функціонал від захищеного, а також забезпечити зручність навігації та адміністрування. На рисунку 4.3 зображено структуру проекту.

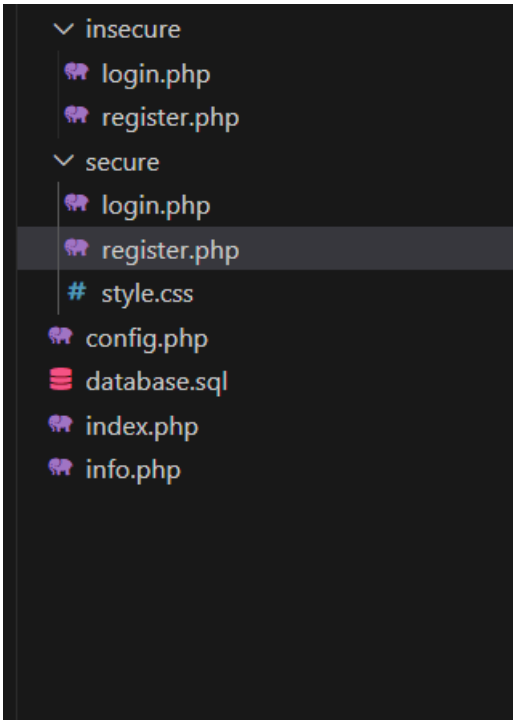


Рисунок 4.3 – Структура проекту

Опишемо кожну частину проекту детальніше:

1. insecure/login.php: відповідає за форму авторизації. Тут обробляються дані, уведені користувачем, — логін та пароль, і формується SQL-запит для перевірки облікових даних без використання підготовлених виразів. Саме через цю сторінку буде демонструватися обхід автентифікації за допомогою SQL-ін'єкцій.

2. insecure/register.php: містить форму реєстрації нового користувача. Уведені користувачем дані для створення облікового запису також обробляються без належного захисту, що робить цю сторінку вразливою для ін'єкцій під час вставки даних.

3. secure/login.php: реалізує форму авторизації, яка є захищеною від SQL-ін'єкцій. Унеможливлюючи впровадження шкідливого коду, вхідні дані користувача коректне прив'язуються до параметрів запиту,.

4. secure/register.php: містить форму реєстрації користувача, яка також захищена від SQL-ін'єкцій, забезпечуючи безпечну вставку даних до бази.

5. style.css: цей файл містить каскадні таблиці стилів, які відповідають за візуальне оформлення всіх сторінок вебдодатку. Забезпечуючи єдиний та привабливий зовнішній вигляд інтерфейсу, він визначає кольори, шрифти, розміри елементів, відступи, розташування блоків та інші візуальні властивості.

6. config.php: цей файл містить налаштування для підключення до бази даних: ім'я хоста БД, ім'я користувача, пароль та назва самої бази даних.

7. database.sql цей файл є SQL-скриптом, який містить команди для створення бази даних та необхідних таблиць для проекту. Його можна імпортувати через phpMyAdmin або виконати через командний рядок для швидкого розгортання структури бази даних.

8. index.php: це головна сторінка вебдодатку, яка слугує точкою входу для користувача та містить посилання на інші розділи проекту такі, як інформаційна сторінка, а також на вразливі та захищені форми авторизації та реєстрації.

9. info.php: цей файл містить інформаційну сторінку з детальним поясненню SQL-ін'єкцій. Слугуючи навчальним посібником для користувача проекту, вона містить визначення, класифікацію, наслідки атак та методи протидії.

4.3 Реалізація захисту від SQL-ін’єкцій

Для захисту вебзастосунку від SQL-ін’єкцій у даній роботі реалізовано безпечну обробку вхідних даних користувача з використанням підготовлених виразів у PHP разом із бібліотекою PDO**.** Це дозволяє уникнути прямого включення користувацького введення у SQL-запити, що є головною причиною SQL-ін’єкцій.

Було реалізовано дві ключові функції: реєстрація нового користувача та авторизація. Розглянемо кожну з них більш детально.

Під час обробки запиту на реєстрацію система спочатку перевіряє коректність введених даних, а саме: наповненість обидвох полів, довжину імені

користувача та пароля, а також допустимість символів у логіні. Усі перевірки виконуються на стороні сервера, що знижує ризики обману клієнтської валідації. У разі успішної перевірки, за допомогою підготовленого виразу дані передаються у базу даних.

Змінні :username та :password є плейсхолдерами, що замінюються під час виконання запиту. Такий підхід унеможливлює вставку довільного SQL-коду користувачем. Також для забезпечення захист паролів навіть у разі компрометації бази даних було реалізовано хешування пароля за допомогою функції password_hash().

Під час входу користувача використовується аналогічний захищений механізм. Якщо користувач знайдений, введений пароль перевіряється через password_verify(), що дозволяє порівняти хеш без розкриття оригінального значення.

Отже, реалізований код демонструє ефективні практики захисту від SQL-ін’єкцій при роботі з формами автентифікації користувачів. Цей підхід значно підвищує рівень безпеки вебзастосунку.

4.4 Демонстрація SQL-ін’єкцій у незахищеній авторизації

Цей підрозділ детально описує процес демонстрації різних типів SQL-ін'єкцій на сторінці незахищеної авторизації та аналізує отримані результати, підкреслюючи вразливість системи до цих поширених атак.

Класична SQL-ін’єкція зображена на рисунку 4.4

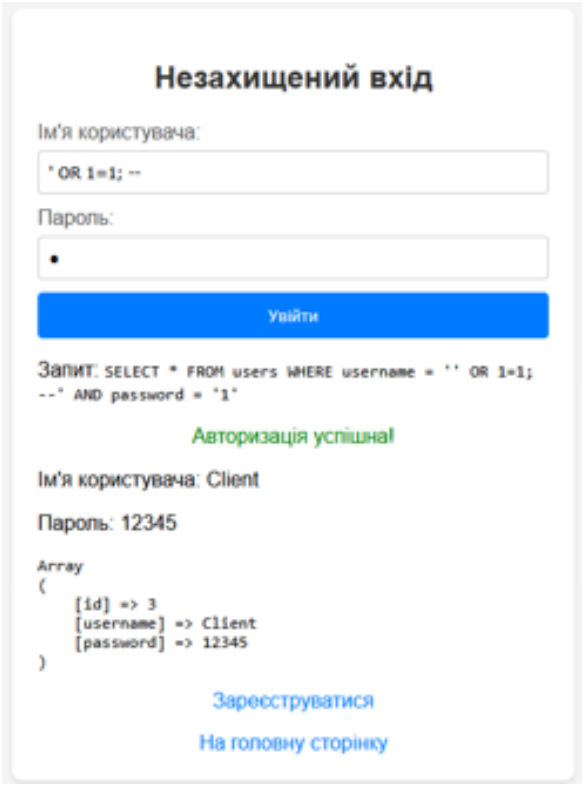


Рисунок 4.4 – Пряма SQL-ін’єкція незахищеної авторизації

Рисунок 4.4 ілюструє результат успішної класичної (прямої) SQL-ін'єкції на сторінці незахищеної авторизації. На зображенні представлено інтерфейс форми входу, де у поле "Ім'я користувача" введено шкідливий SQL-код: ' OR 1=1; --.

Після натискання кнопки "Увійти", введений SQL-код без належної обробки потрапляє безпосередньо в тіло SQL-запиту до бази даних. База даних повертає перший знайдений запис з таблиці users, оскільки умова WHERE завжди істинна.

UNION-based SQL-ін'єкція зображена на рисунку 4.5

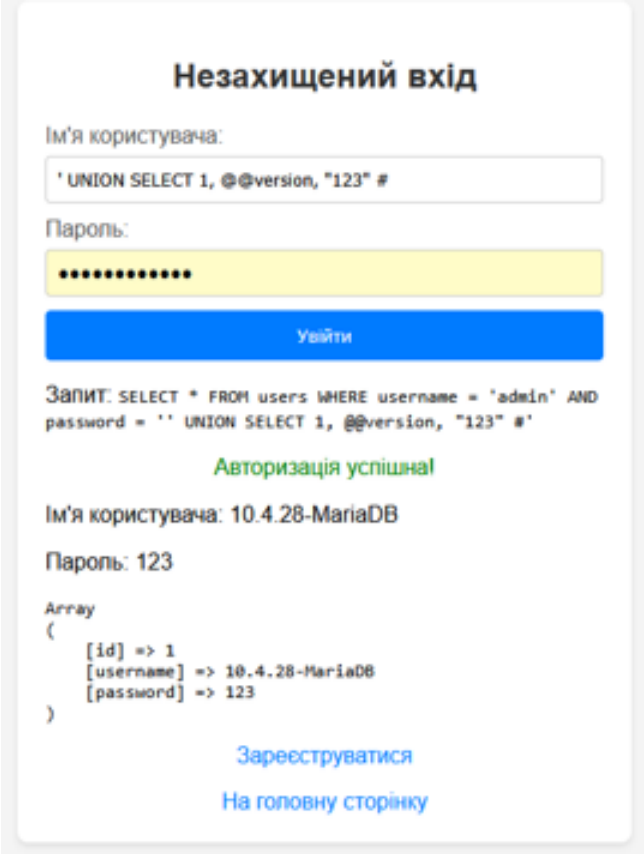


Рисунок 4.5 – UNION-based SQL-ін’єкція незахищеної авторизації

Рисунок 4.5 ілюструє результат успішної SQL-ін'єкції з використанням оператора UNION на сторінці незахищеної авторизації. На зображенні представлено інтерфейс форми входу, де у поле "Ім'я користувача" введено шкідливий SQL-код: ' UNION SELECT 1, @@version, "123" #. Це дозволяє витягувати інформацію, яка не призначена для публічного перегляду, таку як версія використовуваної СКБД. Успішність авторизації в цьому контексті є менш значущою, ніж факт успішного виведення сторонніх даних у місцях, де очікувалися дані користувача. Ця техніка може бути використана для збору інформації про систему з метою подальших атак.

Error-based SQL-ін'єкція зображена на рисунку 4.6

Fatal error: Uncaught PDOException: SQLSTATE[42000]: Syntax error or access violation: 1064 You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near '12345" at line 1 in C:\xampp\htdocs\diplom1\insecure\login.php:14 Stack trace: #0 C:\xampp\htdocs\diplom1\insecure\login.php(14): PDO->query('SELECT * FROM u...') #1 {main} thrown in **C:\xampp\htdocs\diplom1\insecure\login.php** on line **14**

Рисунок 4.6 – Error-based SQL-ін’єкція незахищеної авторизації

Рисунок 4.6 ілюструє результат помилкової SQL-ін’єкції, яка була виконана на сторінці незахищеної авторизації. У полі «Ім’я користувача» було введено спеціально сформований SQL-запит: ' AND EXTRACTVALUE(1, CONCAT(0x7e, (SELECT database()), 0x7e)) --

Цей шкідливий код використовує функцію EXTRACTVALUE(), яка, вставляючи в повідомлення помилки результату виконання підзапиту, навмисно викликає помилку при обробці. У даному випадку підзапит SELECT database() витягує назву активної бази даних, а спеціальні символи (0x7e — це символ ~) використовуються для візуального відокремлення отриманої інформації.

На зображенні видно, що система не тільки не обробляє помилку, а й відображає її прямо в інтерфейсі, тим самим розкриваючи критично важливі дані. Такий підхід демонструє, наскільки небезпечним є вивід необроблених системних помилок у публічних частинах додатка.

Цей тип ін’єкції дає можливість зловмиснику отримувати технічні подробиці про структуру бази даних, таблиці, стовпці або привілеї, що є відправною точкою для наступних етапів атак — таких, як витік облікових даних або повний контроль над системою.

Time-based Blind SQL-ін'єкція зображена на рисунку 4.7

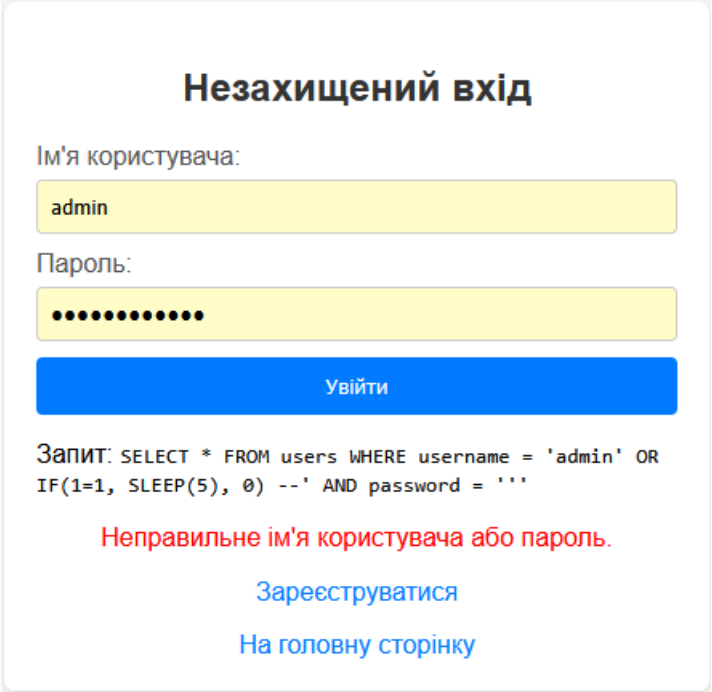


Рисунок 4.7 – Time-based SQL-ін’єкція незахищеної авторизації

Рисунок 4.7 ілюструє результат time-based SQL-ін’єкції на сторінці незахищеної авторизації. У поле «Ім’я користувача» введено шкідливий SQL-код: admin' OR IF(1=1, SLEEP(5), 0) --

Цей тип ін’єкції належить до так званих "сліпих" атак, коли система не виводить жодної помилки або повідомлення, з якого можна було б отримати інформацію напряму. Натомість зловмисник аналізує поведінку серверу, зокрема час відповіді на запит. У цьому прикладі, якщо умова 1=1 є істинною, виконується команда SLEEP(5), яка навмисно затримує відповідь сервера на 5 секунд.

На зображенні видно, що після введення шкідливого запиту сторінка авторизації реагує із помітною затримкою, що є індикатором успішного виконання частини SQL-коду. Така поведінка свідчить про вразливість до time-based атак, навіть попри відсутність явних повідомлень про помилку чи витоку даних.

За умов, коли традиційні методи ін’єкцій не дають результату, використовуються подібні атаки. Змінюючи умови та вимірюючи час відповіді,

time-based SQL-ін’єкція дозволяє зловмиснику поступово "вгадувати" структуру бази даних, значення полів або облікові дані, що становить серйозну загрозу для конфіденційності системи.

4.5 Демонстрація SQL-ін’єкцій у захищеній авторизації

На відміну від попередніх прикладів, де SQL-ін’єкції були можливі через пряме включення введених користувачем даних у SQL-запити, у цьому випадку реалізовано захищену авторизацію з використанням підготовлених виразів та прив’язки параметрів.

Оскільки дані не включаються до SQL-запиту безпосередньо, а передаються окремим параметром, обробленим СКБД як звичайне значення, а не як частину синтаксису, спроба застосування SQL-ін’єкції не матиме жодного результату. Подібний захист є обов’язковим для всіх точок взаємодії з базою даних у вебдодатках, що працюють з конфіденційною інформацією.

Пряма SQL-ін’єкція в захищеній авторизації зображена на рисунку 4.8

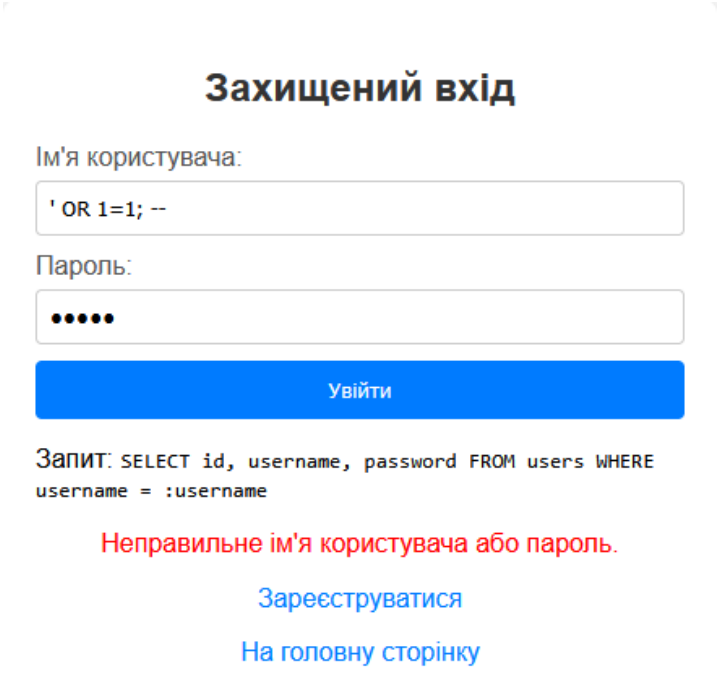


Рисунок 4.8 – Пряма SQL-ін’єкція захищеної авторизації

Рисунок 4.8 ілюструє результат спроби прямої SQL-ін'єкції на сторінці захищеної авторизації. У полі "Ім’я користувача" було введено класичний шкідливий SQL-код: ' OR 1=1 --. Проте система не виконала шкідливий запит, оскільки використовуються підготовлені вирази з прив’язкою параметрів. Завдяки цьому, введені дані не інтерпретуються як частина SQL-запиту, а лише як значення, що захищає від ін’єкцій. У результаті авторизація не відбулася - і зловмисник не отримав доступу до системи.

UNION-based SQL-ін'єкція в захищеній авторизації зображена на рисунку 4.9

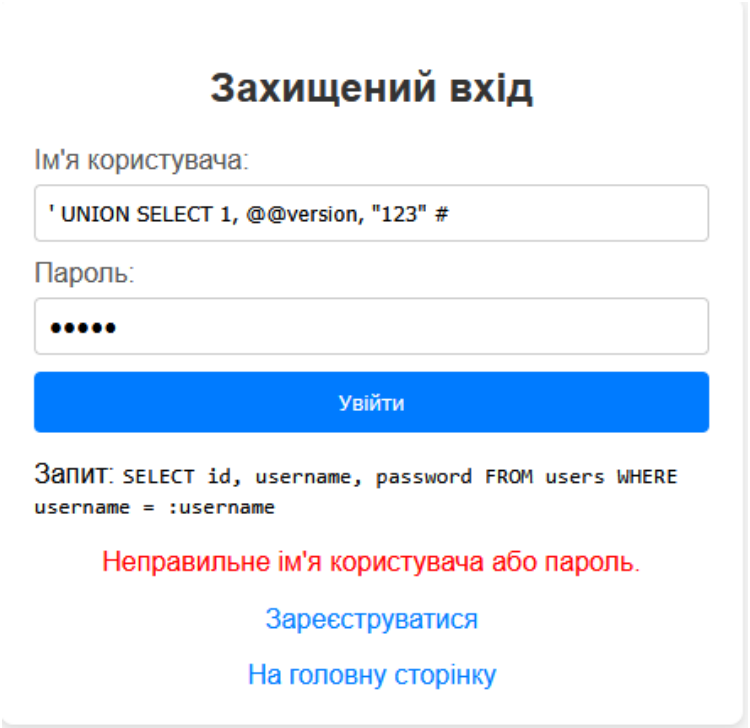


Рисунок 4.9 – UNION-based SQL-ін’єкція захищеної авторизації

На рисунку 4.9 продемонстровано спробу SQL-ін’єкції з використанням оператора UNION: ' UNION SELECT 1, @@version, "123" #. Цей підхід дозволяє уразливим системам об'єднати запити та вивести критичну інформацію, наприклад, версію СКБД. Однак у даній реалізації запит до бази даних побудовано за допомогою підготовленого виразу з параметром :username, який не дозволяє

змінити структуру SQL-запиту. У результаті ін'єкція не спрацювала - і система залишилася захищеною.

Error-based SQL-ін'єкція в захищеній авторизації зображено на рисунку 4.8

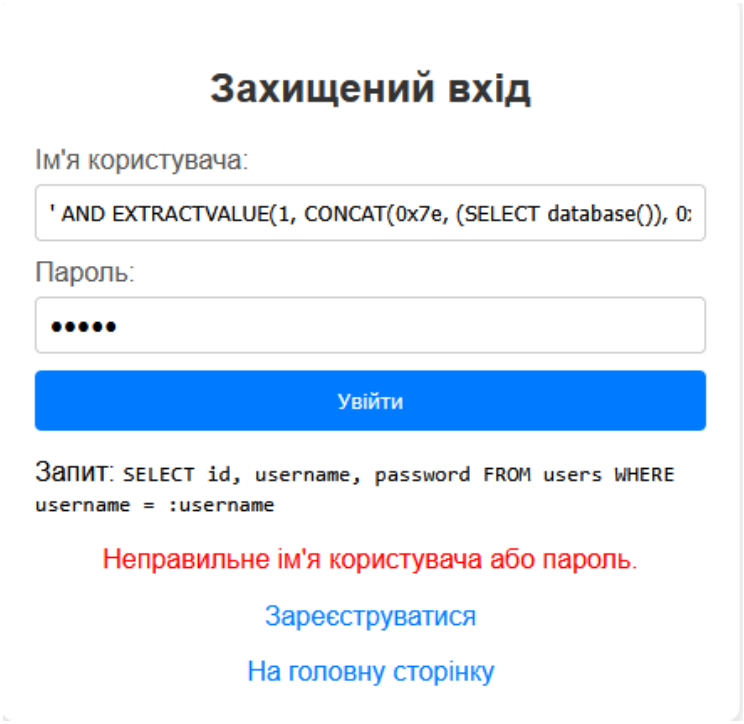


Рисунок 4.10 – Error-based SQL-ін’єкція захищеної авторизації

На рисунку 4.10 наведено приклад невдалої спроби Eerror-based SQL-ін'єкції на захищеній сторінці авторизації. Зловмисник ввів код: ' AND EXTRACTVALUE(1, CONCAT(0x7e, (SELECT database()), 0x7e)) --, який уразливі системи могли б інтерпретувати як запит, що викликає помилку та виводить назву бази даних. Проте через використання підготовлених виразів цей рядок не вплинув на структуру запиту, і жодна помилка не була викликана чи відображена. Це підтверджує, що система не розкриває внутрішніх даних через обробку помилок.

Time-based Blind SQL-ін'єкція в захищеній авторизації зображено на рисунку 4.9

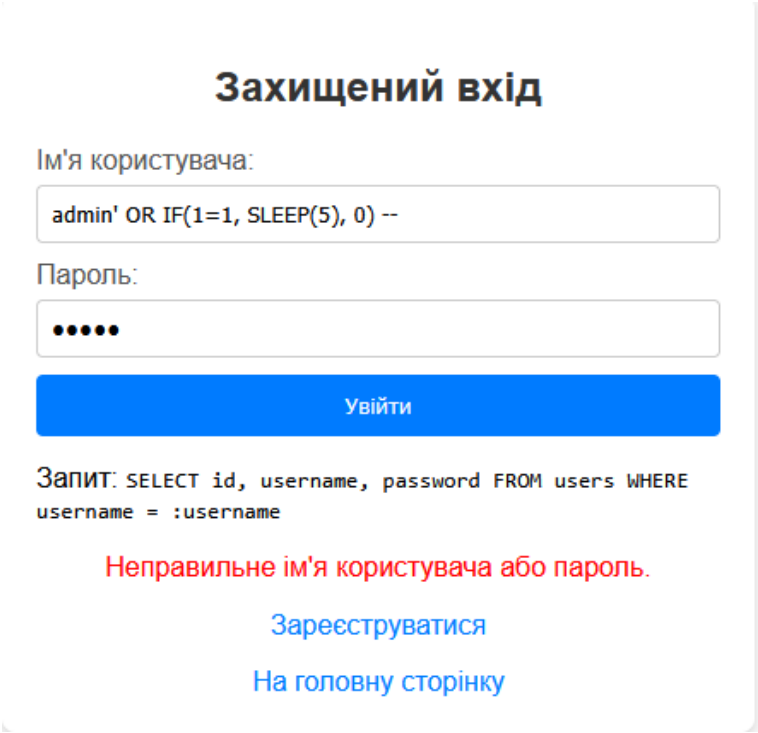


Рисунок 4.11 – Time-based SQL-ін’єкція захищеної авторизації

На рисунку 4.11 показано результат спроби time-based SQL-ін'єкції, де у полі логіну було введено: ' OR IF(1=1, SLEEP(5), 0) --. Якщо система уразлива, така атака може викликати затримку відповіді сервера. Проте підготовлені вирази не дозволяють SQL-інструкціям змінювати логіку запиту. Запит виконується без помітної затримки, а отже, ін'єкція не має ефекту. Це демонструє стійкість системи навіть до складних типів ін'єкцій, що не потребують прямого виводу інформації.

Проаналізовані приклади демонструють ефективність використання підготовлених виразів у захисті системи авторизації від різних типів SQL-ін’єкцій.

Завдяки чіткому розмежуванню структури SQL-запиту та вхідних параметрів, навіть складно сформовані шкідливі рядки не впливають на логіку запиту й не виконуються як частина SQL-коду. Окрім того, унеможливлює витік технічної інформації про базу даних належна обробка помилок.

Висновки до розділу 4

У четвертому розділі було розглянуто практичну реалізацію захисту від SQL-ін’єкцій у межах створеного програмного продукту. Розділ включає аналіз структури проекту, розробку захищеного механізму автентифікації користувачів, а також демонстрацію атак SQL-ін’єкцій у захищених і незахищених сценаріях.

Було детально описано архітектуру проекту, основні його компоненти та роль кожного модуля в загальній логіці системи. За допомогою використання підготовлених виразів у поєднанні з валідацією введених даних та хешуванням паролів було реалізовано ефективний захист від SQL-ін’єкцій.

Також було наведено практичні приклади щодо використання SQL-ін’єкцій. Вдалося успішно здійснити атаку в незахищеній авторизації, що призвела до несанкціонованого доступу до системи. Натомість у захищеній авторизації ті самі ін’єкційні спроби не мали успіху, що підтверджує ефективність обраного методу захисту.

Отже, реалізований програмний продукт не лише демонструє вразливість баз даних до SQL-ін’єкцій, а й демонструє практично, як можна шляхом дотримання сучасних принципів безпечної розробки успішно їм запобігти.

5. РОБОТА КОРИСТУВАЧА З РОЗРОБЛЕНОЮ СИСТЕМОЮ

5.1 Загальний огляд користувацького інтерфейсу

Ключовим елементом взаємодії між системою та кінцевим користувачем є користувацький інтерфейс програмного забезпечення. Основна мета інтерфейсу - забезпечення інтуїтивно зрозумілого доступу до функціональності системи, демонстрації вразливості до SQL-ін’єкцій та можливості захисту від неї. У рамках даного вебзастосунку користувачу доступні кілька основних сторінок, кожна з яких виконує окрему функцію в межах демонстраційного процесу.

Користувач після запуску вебзастосунку потрапляє на головну сторінку, яка містить короткий опис проекту, інформацію про його мету та призначення. Ця сторінка є своєрідним навігаційним центром, забезпечуючи доступ до інших функціональних частин системи.

Інформаційна сторінка надає теоретичні відомості про SQL-ін’єкції, їхні типи, принципи дії та можливі наслідки й орієнтована на ознайомлення користувача з проблематикою; має освітній характер.

У системі реалізовано два типи форм реєстрації та авторизації — незахищена та захищена. Незахищені сторінки демонструють приклад про можливе виконання SQL-ін’єкцій через відсутність обробки вхідних даних або використання небезпечних SQL-запитів; натомість захищені сторінки реалізовані з дотриманням рекомендацій з безпеки, зокрема із використанням підготовлених виразів, перевіркою введених даних та хешуванням паролів.

У цілому інтерфейс розроблений так, аби користувач мав можливість самостійно взаємодіяти з різними реалізаціями авторизації та реєстрації, порівнювати їхню поведінку при введенні коректних і некоректних даних, а також дослідити, як саме SQL-ін’єкції можуть впливати на безпеку вебзастосунків.

5.2 Опис головної сторінки проекту

Головна сторінка вебзастосунку, виконуючи роль вступного інтерфейсу для користувача, забезпечує зручний доступ до основної функціональності системи, тому оформлення має бути інформативним, лаконічним і зрозумілим; вона є першим елементом взаємодії, з яким стикається користувач після відкриття вебзастосунку.

На головній сторінці розміщено назву проекту та подається стислий опис його призначення. У даному випадку головна ідея полягає в демонстрації SQL-ін’єкцій, їхнього впливу на безпеку баз даних, а також методів протидії цим атакам. У цілому сторінка інформує користувача про актуальність теми інформаційної безпеки в контексті веброзробки.

Інтерфейс головної сторінки реалізовано у простому, оптимальному стилі з акцентом на зручність користування та швидку орієнтацію; у вигляді кнопок або меню винесено основний функціонал, що забезпечує швидкий перехід до демонстраційної частини системи.

Для зручності реалізації сторінка створена з використанням стандартних технологій веброзробки — HTML і CSS; надалі її можна розширити шляхом додавання інструкцій для користувача, додаткової інформації про автора проекту або інтеграції з аналітичними модулями.

Інтерфейс головної сторінки зображений на рисунку 5.1.

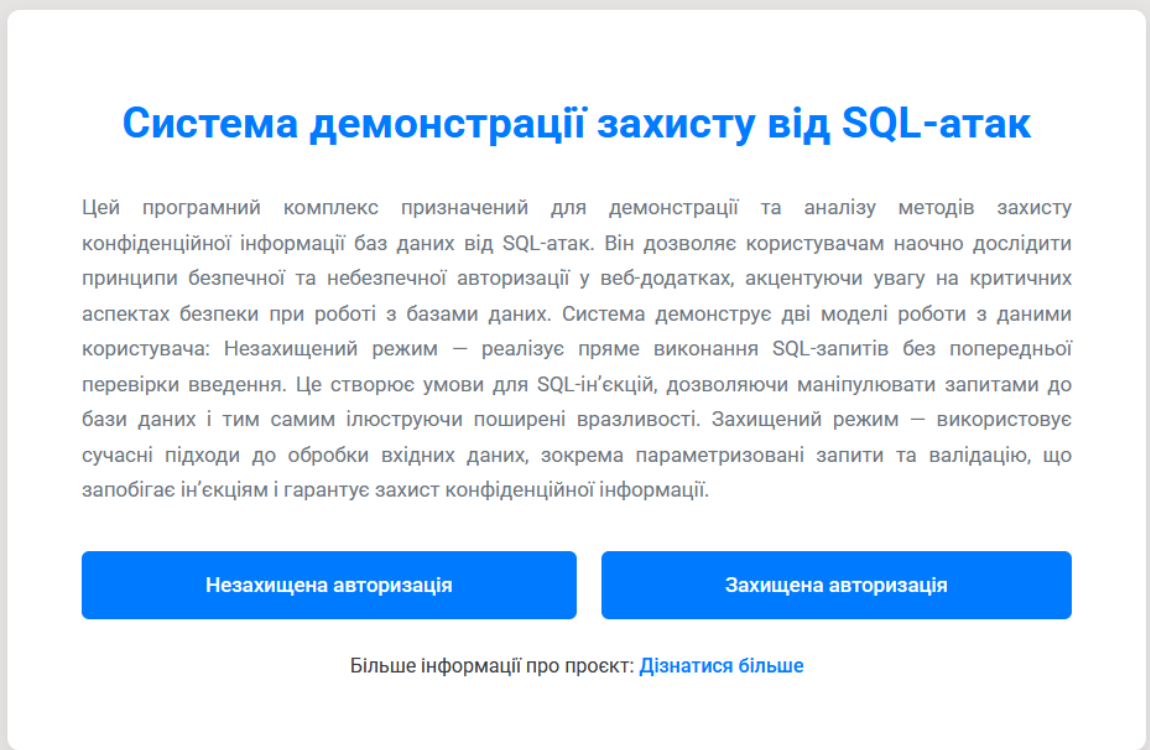


Рисунок 5.1 – Інтерфейс головної сторінки

5.3 Опис інформаційної сторінки проекту

З метою ознайомлення користувача з поняттям SQL-ін’єкцій, принципом їхньої дії, типами атак та методами захисту, створена інформаційна сторінка проєкту. Ця сторінка має теоретичний характер і слугує підґрунтям для кращого розуміння практичної частини, яка представлена у вигляді демонстрації незахищеної та захищеної авторизації.

Надається пояснення, як через введення шкідливого SQL-коду у поля вводу користувача можливо впливати на роботу бази даних у випадку недостатнього захисту системи. Розглянуто принцип дії такої атаки, а саме: від моменту формування запиту з даними користувача до можливого виконання шкідливих інструкцій, що змінюють логіку роботи програми.

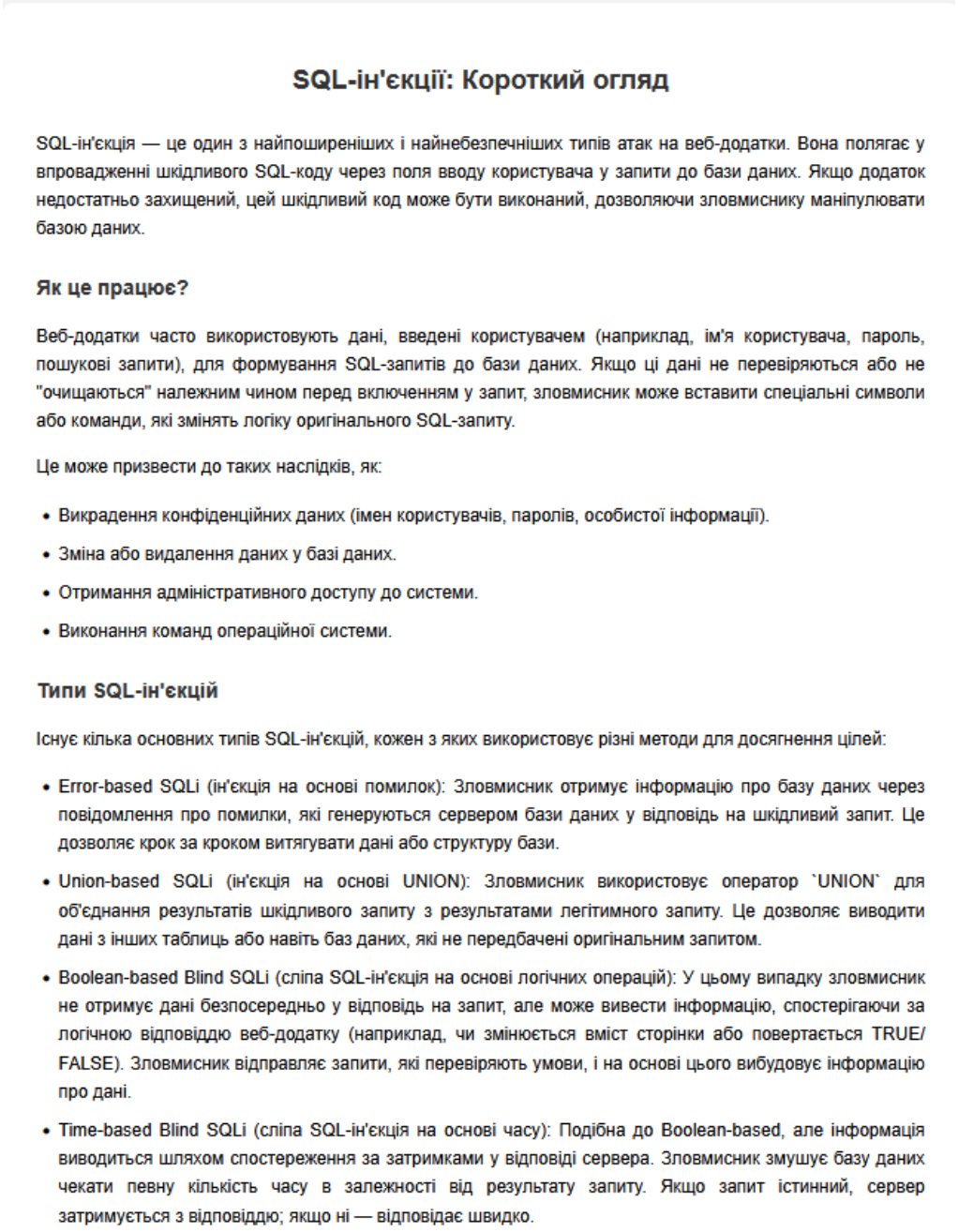
Наведено приклади потенційних наслідків SQL-ін’єкцій, зокрема: несанкціонований доступ до даних, їхнє викрадення, зміна або видалення інформації, а також виконання сторонніх команд на сервері.

Також окремо описано найпоширеніші варіанти реалізації SQL-ін’єкцій, серед яких — варіанти, що базуються на отриманні помилок, використанні оператора для об’єднання результатів, а також сліпі ін’єкції, які дозволяють отримувати інформацію через логічні реакції системи або аналіз часу відповіді.

На завершення наведено основні підходи щодо захисту від подібних атак; наголошено на важливості використання параметризованих запитів як основного механізму безпеки, а також додаткових заходів, таких як фільтрація введених даних, обмеження доступу до бази даних на рівні прав користувача і своєчасне оновлення програмного забезпечення.

Підвищуючи рівень обізнаності щодо потенційних загроз та шляхів їхнього усунення, інформаційна сторінка виконує роль вступної теоретичної довідки та є важливою складовою загального користувацького досвіду.

Інтерфейс головної сторінки зображений на рисунку 5.2.



SQL-ін'єкції: Короткий огляд

SQL-ін'єкція — це один з найпоширеніших і найнебезпечніших типів атак на веб-додатки. Вона полягає у впровадженні шкідливого SQL-коду через поля вводу користувача у запити до бази даних. Якщо додаток недостатньо захищений, цей шкідливий код може бути виконаний, дозволяючи зловмиснику маніпулювати базою даних.

Як це працює?

Веб-додатки часто використовують дані, введені користувачем (наприклад, ім'я користувача, пароль, пошукові запити), для формування SQL-запитів до бази даних. Якщо ці дані не перевіряються або не "очищаються" належним чином перед включенням у запит, зловмисник може вставити спеціальні символи або команди, які змінять логіку оригінального SQL-запиту.

Це може призвести до таких наслідків, як:

- Викрадення конфіденційних даних (імен користувачів, паролів, особистої інформації).
- Зміна або видалення даних у базі даних.
- Отримання адміністративного доступу до системи.
- Виконання команд операційної системи.

Типи SQL-ін'єкцій

Існує кілька основних типів SQL-ін'єкцій, кожен з яких використовує різні методи для досягнення цілей:

- Error-based SQLi (ін'єкція на основі помилок): Зловмисник отримує інформацію про базу даних через повідомлення про помилки, які генеруються сервером бази даних у відповідь на шкідливий запит. Це дозволяє крок за кроком витягувати дані або структуру бази.
- Union-based SQLi (ін'єкція на основі UNION): Зловмисник використовує оператор `UNION` для об'єднання результатів шкідливого запиту з результатами легітимного запиту. Це дозволяє виводити дані з інших таблиць або навіть баз даних, які не передбачені оригінальним запитом.
- Boolean-based Blind SQLi (сліпа SQL-ін'єкція на основі логічних операцій): У цьому випадку зловмисник не отримує дані безпосередньо у відповідь на запит, але може вивести інформацію, спостерігаючи за логічною відповіддю веб-додатку (наприклад, чи змінюється вміст сторінки або повертається TRUE/FALSE). Зловмисник відправляє запити, які перевіряють умови, і на основі цього вибудовує інформацію про дані.
- Time-based Blind SQLi (сліпа SQL-ін'єкція на основі часу): Подібна до Boolean-based, але інформація виводиться шляхом спостереження за затримками у відповіді сервера. Зловмисник змушує базу даних чекати певну кількість часу в залежності від результату запиту. Якщо запит істинний, сервер затримується з відповіддю; якщо ні — відповідає швидко.

Рисунок 5.2 – Інтерфейс інформаційної сторінки

5.4 Опис сторінок авторизації та реєстрації

Підсистема незахищеної авторизації містить скрипти, що імітують типову, але вразливу реалізацію авторизації та реєстрації користувачів.

Сторінка реєстрації надає користувачеві можливість створити новий обліковий запис (рис 5.3).

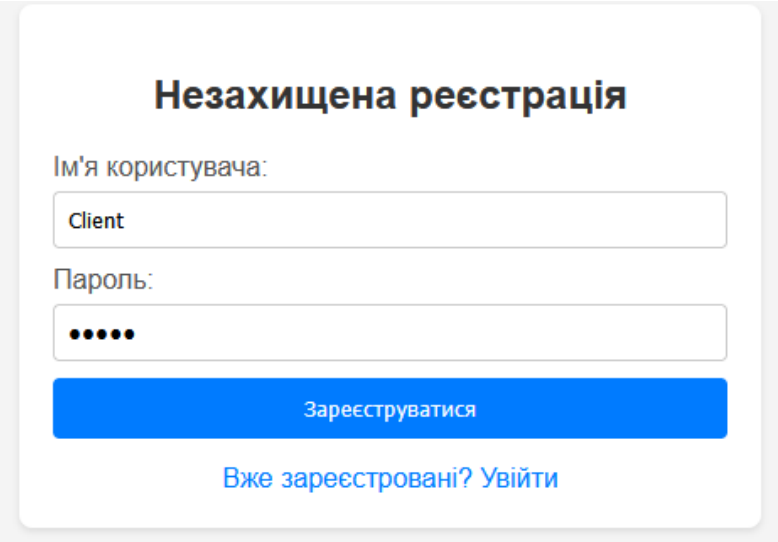


Рисунок 5.3 – Екран незахищеної реєстрації

Сторінка входу - це стандартна форма для введення імені користувача та пароля. Основна вразливість цього скрипту полягає у безпосередньому включенні введених користувачем даних у SQL-запит без будь-якої попередньої обробки або екранування. Саме на цій сторінці демонструються різні техніки SQL-ін'єкцій, що можуть призвести до несанкціонованого доступу або витоку інформації. Користувачеві після спроби входу відображається результат авторизації або повідомлення про помилку. Також є посилання для повернення на головну сторінку та переходу на сторінку реєстрації.

У рамках цієї реалізації не застосовуються жодні механізми захисту, зокрема підготовлені вирази або екранування вхідних даних, що створює крайню вразливість, яка дозволяє зловмиснику вводити SQL-код безпосередньо в поля

форми для виконання небажаних запитів. Такий підхід є прикладом небезпечної практики, яку в реальних вебзастосунках необхідно уникати. (рис. 5.4).

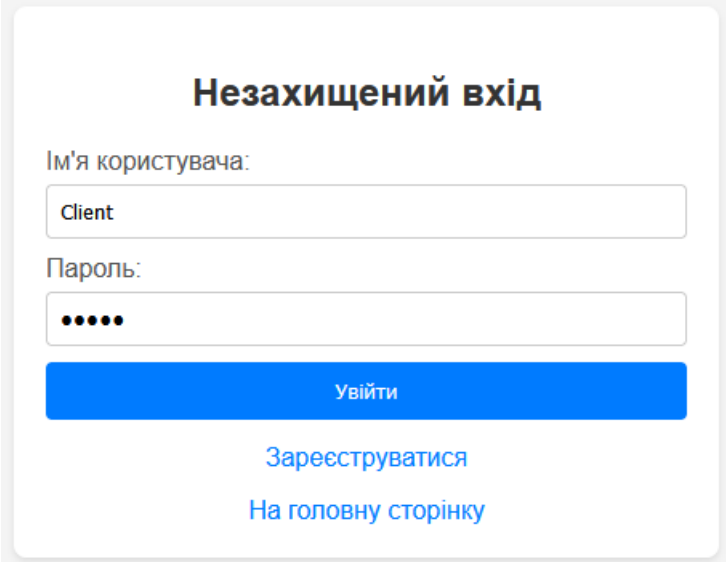


Рисунок 5.4 – Екран незахищеної авторизації

Підсистема захищеної авторизації містить аналоги скриптів з підсистеми незахищеної авторизації, але з реалізованими механізмами захисту від SQL-ін'єкцій. Інтерфейс аналогічний до незахищеної авторизації.

Сторінка реєстрації аналогічна до захищеної сторінки входу; реєстрація нових користувачів здійснюється з використанням підготовлених виразів для запису даних у базу даних. Найважливішим аспектом є хешування паролів перед їх збереженням за допомогою надійних алгоритмів, що робить зберігання паролів безпечним, оскільки навіть у випадку компрометації бази даних зловмисник не зможе отримати доступ до паролів у вигляді відкритого тексту.

Сторінка входу захищеної авторизації візуально і функціонально схожа на незахищену, однак ключова відмінність полягає в обробці введених даних. Використовуються підготовлені вирази з використанням PDO, замість прямого формування SQL-запитів; це гарантує, що введені користувачем дані розглядаються лише як параметри запиту, а не як виконуваний SQL-код. Окрім

того, за допомогою функції password_verify(), яка порівнює введений пароль з попередньо хешованим значенням, що зберігається в базі даних, паролі користувачів, отримані з бази даних, проходять безпечну перевірку

Висновки до розділу 5

У п’ятому розділі було розглянуто особливості користувацького інтерфейсу створеного вебзастосунку, його загальну структуру та функціональні можливості; надано опис кожної з доступних сторінок, зокрема головної інформаційної сторінки, форм авторизації та реєстрації як у захищеному, так і в незахищеному варіантах.

Користувацький інтерфейс було реалізовано з урахуванням як функціональних потреб, так і наочності для демонстрації вразливостей SQL-ін’єкцій та способів їх запобігання в реальних умовах взаємодії з вебзастосунком.

ВИСНОВКИ

У даній дипломній роботі було розроблено програмне забезпечення для захисту конфіденційної інформації баз даних від SQL-ін’єкцій. Забезпечуючи надійний захист облікових даних користувачів від потенційного несанкціонованого доступу, система реалізує ключові принципи безпечної обробки запитів до бази даних. Основна увага приділялася впровадженню сучасних методів протидії SQL-ін’єкціям, зокрема використанню підготовлених запитів через бібліотеку PDO.

Розроблений вебзастосунок демонструє роботу як захищеної, так і вразливої системи авторизації, що дозволяє оцінити ефективність впроваджених заходів безпеки. У процесі реалізації було досліджено основні типи SQL-ін’єкцій і реалізовано механізми, які дадуть можливість запобігати їхньому виконанню.

Результатом роботи стала функціональна система, що як компонент безпечної авторизації та зберігання конфіденційних даних може бути інтегрована в будь-який вебдодаток. Програмне забезпечення є актуальним рішенням у сфері інформаційної безпеки та може бути використане в реальних проектах для підвищення захисту баз даних від SQL-атак.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Clarke J. SQL Injection Attacks and Defense : монографія. 2-ге вид. Waltham : Syngress, 2012. 512 с.
2. Stuttard D., Pinto M. The Web Application Hacker’s Handbook: Finding and Exploiting Security Flaws. 2-ге вид. Indianapolis : Wiley, 2011. 912 с.
3. Howard M., LeBlanc D. Writing Secure Code. 2-ге вид. Redmond : Microsoft Press, 2003. 800 с.
4. Halfond W. G. J., Orso A., Manolios P. WASP: Protecting Web Applications Using Positive Tainting and Syntax-Aware Evaluation : монографія. New York : Springer, 2008. 250 с.
5. Anley C., Heasman J., Lindner F. SQL Injection: Examples, Real Life Attacks & Defensive Measures. Boston : Addison-Wesley, 2007. 320 с.
6. Fogie S., Grossman J., Hansen R. XSS Attacks: Cross Site Scripting Exploits and Defense. Burlington : Syngress, 2007. 480 с.
7. Andrews M., Whittaker J. A. How to Break Web Software: Functional and Security Testing of Web Applications and Services. Boston : Addison-Wesley, 2006. 240 с.
8. Daswani N., Kern C., Kesavan A. Foundations of Security: What Every Programmer Needs to Know. New York : Apress, 2007. 304 с.
9. Späth P. Pro PHP Security: From Application Security Principles to the Implementation of XSS Defenses. 2-ге вид. New York : Apress, 2010. 368 с.
10. Williams J., Wichers D. OWASP Top Ten Project: Understanding and Mitigating Web Application Vulnerabilities. San Francisco : OWASP Press, 2017. 200 с.
11. Shiflett C. Essential PHP Security. Sebastopol : O’Reilly Media, 2005. 128 с.
12. Scambray J., Liu V., Sima C. Hacking Exposed Web Applications. 3-тє вид. New York : McGraw-Hill, 2010. 480 с.

13. Shema M. Seven Deadliest Web Application Attacks. Burlington : Syngress, 2010. 192 с.

14. Zalewski M. The Tangled Web: A Guide to Securing Modern Web Applications. San Francisco : No Starch Press, 2011. 320 с.

15. Meier J. D., Mackman A., Dunner M. Improving Web Application Security: Threats and Countermeasures. Redmond : Microsoft Press, 2003. 960 с.

16. Hope P., Walther B. Web Security Testing Cookbook: Systematic Techniques to Find Problems Fast. Sebastopol : O'Reilly Media, 2008. 312 с.

17. Snyder C., Southwell M., Myer T. Pro PHP: Patterns, Frameworks, Testing and More. New York : Apress, 2008. 360 с.

18. Fogie S., Grossman J. Cross-Site Scripting Attacks: XSS Exploits and Defense : монографія. Burlington : Syngress, 2007. 432 с.

19. Burns D., Manion M., McDonald A. Security Power Tools. Sebastopol : O'Reilly Media, 2007. 864 с.

20. Dowd M., McDonald J., Schuh J. The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities. Boston : Addison-Wesley, 2006. 1200 с.

21. Wysopal C., Nelson L., Zovi D. D., Dustin E. The Art of Software Security Testing: Identifying Software Security Flaws. Boston : Addison-Wesley, 2006. 288 с.

22. Curphey M., Arawo R. Web Application Security Assessment Tools: A Practical Guide to Implementation. New York : McGraw-Hill, 2006. 400 с.

23. Skoudis E., Liston T. Counter Hack Reloaded: A Step-by-Step Guide to Computer Attacks and Effective Defenses. 2-ге вид. Upper Saddle River : Prentice Hall, 2006. 784 с.

24. Hoglund G., McGraw G. Exploiting Software: How to Break Code. Boston : Addison-Wesley, 2004. 512 с.

25. Engebretson P. The Basics of Hacking and Penetration Testing: Ethical Hacking and Penetration Testing Made Easy. 2-ге вид. Waltham : Syngress, 2013. 224 с.

ДОДАТОК А

Захист конфіденційної інформації баз даних

Лістинг розробленої системи

УКР.НТУУ”КПІ”_ІАТЕ_ІПЗЕ_ТВ-11

Аркушів 4

Київ — 2025

index.php

```
<!DOCTYPE html>
<html>
<head>
   <title>Головна сторінка</title>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <link href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500;700&display=swap"
rel="stylesheet">
   <style>
      :root {
         --primary-color: #007bff;
         --primary-hover-color: #0056b3;
         --background-light-gray:rgb(230, 227, 227);
         --text-dark: #343a40;
         --text-muted: #6c757d;
         --card-background: #ffffff;
         --card-shadow: 0 4px 12px rgba(0, 0, 0, 0.08);
         --border-radius: 10px;
      }

      body {
         font-family: 'Roboto', sans-serif;
         margin: 0;
         background-color: var(--background-light-gray);
         display: flex;
         flex-direction: column;
         align-items: center;
         justify-content: center;
         min-height: 100vh;
         color: var(--text-dark);
         line-height: 1.6;
      }

      .main-container {
         background-color: var(--card-background);
         padding: 40px 60px;
         border-radius: var(--border-radius);
         box-shadow: var(--card-shadow);
         text-align: center;
         max-width: 800px;
         width: 90%;
         margin-bottom: 30px;
      }

      h1 {
         color: var(--primary-color);
         margin-bottom: 25px;
         font-weight: 700;
         font-size: 2.2em;
      }

      p.description {
         color: var(--text-muted);
         margin-bottom: 35px;
         font-size: 1.05em;
         text-align: justify;
         line-height: 1.7;
      }
```

```
        .button-group {
            display: flex;
            justify-content: space-between;
            gap: 20px;
            margin-top: 30px;
            width: 100%;
        }

        .button {
            background-color: var(--primary-color);
            color: white;
            padding: 14px 28px;
            border-radius: 6px;
            cursor: pointer;
            text-decoration: none;
            font-size: 1.05em;
            font-weight: 500;
            transition: background-color 0.3s ease, transform 0.2s ease;
            flex-grow: 1;
            text-align: center;
        }

        .button:hover {
            background-color: var(--primary-hover-color);
            transform: translateY(-2px);
        }

        .info-link {
            margin-top: 25px;
            font-size: 1em;
        }

        .info-link a {
            color: var(--primary-color);
            text-decoration: none;
            font-weight: 500;
            transition: color 0.3s ease;
        }

        .info-link a:hover {
            color: var(--primary-hover-color);
            text-decoration: underline;
        }

    </style>
</head>
<body>
    <div class="main-container">
        <h1>Система демонстрації захисту від SQL-атак</h1>
        <p class="description">
            Цей програмний комплекс призначений для демонстрації та аналізу методів захисту конфіденційної
інформації
            баз даних від SQL-атак. Він дозволяє користувачам наочно дослідити принципи безпечної та небезпечної
            авторизації у вебдодатках, акцентуючи увагу на критичних аспектах безпеки при роботі з базами даних.
            Система демонструє дві моделі роботи з даними користувача:
            Незахищений режим — реалізує пряме виконання SQL-запитів без попередньої перевірки введення.
            Це створює умови для SQL-ін’єкцій, дозволяючи маніпулювати запитами до бази даних і тим самим
            ілюструючи поширені вразливості.
            Захищений режим — використовує сучасні підходи до обробки вхідних даних,
```

зокрема параметризовані запити та валідацію, що запобігає ін’єкціям і гарантує захист конфіденційної інформації.

Незахищена авторизація
Захищена авторизація

Більше інформації про проєкт: Дізнатися більше

</body>
</html>

secure/login.php

```
<?php
require '../config.php';

$error = '';
$login_successful = false;
$sql_query = '';
$user_data = [];

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $username = trim($_POST['username']);
    $password = $_POST['password'];

    $sql = "SELECT id, username, password FROM users WHERE username = :username";
    $stmt = $pdo->prepare($sql);
    $stmt->bindParam(':username', $username);
    $stmt->execute();
    $user = $stmt->fetch(PDO::FETCH_ASSOC);

    $sql_query = $stmt->queryString;

    if ($user && password_verify($password, $user['password'])) {
        $login_successful = true;
        $user_data = $user;
    } else {
        $error = 'Неправильне ім\'я користувача або пароль.';
    }
}

$registration_success = isset($_GET['registration']) && $_GET['registration'] === 'success';
?>

<!DOCTYPE html>
<html>
<head>
    <title>Захищений вхід</title>
    <link rel="stylesheet" type="text/css" href="../secure/style.css">
</head>
<body>
    <div class="container">
        <h2>Захищений вхід</h2>
        <?php if ($registration_success): ?>
            <p class="success">Реєстрація успішна! Тепер ви можете увійти.</p>
        <?php endif; ?>
```

```
        <form method="post">
            <label for="username">Ім'я користувача:</label>
            <input type="text" id="username" name="username" required>

            <label for="password">Пароль:</label>
            <input type="password" id="password" name="password" required>

            <button type="submit">Увійти</button>
        </form>

        <?php if ($sql_query): ?>
            <p>Запит: <code><?php echo htmlspecialchars($sql_query); ?></code></p>
        <?php endif; ?>

        <?php if ($login_successful): ?>
            <p class="success">Авторизація успішна!</p>
            <p>Ім'я користувача: <?php echo htmlspecialchars($user_data['username'] ?? ''); ?></p>
        <?php elseif ($error): ?>
            <p class="error"><?php echo $error; ?></p>
        <?php endif; ?>

        <div class="links">
            <a href="../secure/register.php">Зареєструватися</a>
        </div>
        <div class="links">
            <a href="../index.php">На головну сторінку</a>
        </div>
    </div>
</body>
</html>
```

ДОДАТОК Б

Захист конфіденційної інформації баз даних

Презентація

УКР.НТУУ”КПІ”_ІАТЕ_ІПЗЕ_ТВ-11

Аркушів 8

Київ — 2025

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут атомної та теплової енергетики

Кафедра інженерії програмного забезпечення в енергетиці

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ СТВОРЕННЯ ЗАХИСТУ КОНФІДЕНЦІЙНОЇ ІНФОРМАЦІЇ БАЗ ДАНИХ ВІД SQL-АТАК

виконав: студент групи ТВ-11 Аржанцев Михайло Максимович

керівник: доцент Шуклін Герман Вікторович, к.т.н.

2025

Актуальність теми

Проблема: SQL-ін’єкції є однією з найпоширеніших і найнебезпечніших видів атак на бази даних, що призводять до витоку конфіденційної інформації, порушення цілісності даних або повної компрометації системи.

Значення для галузі: Захист баз даних від SQL-атак є критично важливим для інформаційної безпеки вебдодатків, особливо в умовах зростання цифровізації в енергетиці, фінансах, охороні здоров’я тощо.

Мета роботи: Дослідження вразливостей до SQL-ін’єкцій та розробка програмного забезпечення для захисту конфіденційної інформації баз даних шляхом впровадження надійних механізмів обробки запитів.

Постановка задачі

Формулювання задачі: Розробити програмне забезпечення для демонстрації вразливостей до SQL-ін’єкцій та реалізації захисту конфіденційної інформації баз даних.

Перелік завдань:

1.Провести теоретичне дослідження SQL-ін’єкцій, їх типів і механізмів реалізації.

2.Проаналізувати сучасні методи захисту від SQL-ін’єкцій.

3.Розробити програмний продукт, що реалізує захист інформації.

4.Створити користувацький інтерфейс для демонстрації роботи системи.

5.Провести тестування ефективності захисту від SQL-атак.

Аналіз предметної області

Огляд існуючих рішень:

• Параметризовані запити PDO ефективно запобігають SQL-ін’єкціям.

• Інструменти автоматичного тестування, такі як OWASP ZAP та SQLMap виявляють вразливості.

• Вебфреймворки з вбудованим захистом зменшують ризики, але потребують коректної конфігурації.

Типи SQL-атак

• Пряма SQL-ін’єкція: Введення шкідливого коду (наприклад, ' OR 1=1 --) для обходу авторизації чи доступу до даних.

• UNION-based: Використання UNION для отримання даних з інших таблиць (наприклад, ' UNION SELECT 1, @@version, "123" #).

• Error-based: Виклик помилок для витоку інформації (наприклад, ' AND EXTRACTVALUE(1, CONCAT(0x7e, (SELECT database()), 0x7e)) --).

• Time-based Blind: Аналіз затримки відповіді сервера (наприклад, admin' OR IF(1=1, SLEEP(5), 0) --).

Проєктування системи

Методи вирішення:

• Використання підготовлених виразів PDO для безпечної обробки даних.

• Хешування паролів за допомогою password_hash().

• Валідація вхідних даних на сервері.

Архітектура системи:

• Модулі: головна сторінка, інформаційна сторінка, незахищена та захищена авторизація, реєстрація.

• Використання MySQL/MariaDB для бази даних.

Реалізація системи

Процес розробки:

• Створення структури проєкту з модулями для вразливої та захищеної авторизації.

• Розробка користувацького інтерфейсу.

• Налаштування локального сервера через XAMPP.

Використані технології:

• Мови: PHP, HTML, CSS.

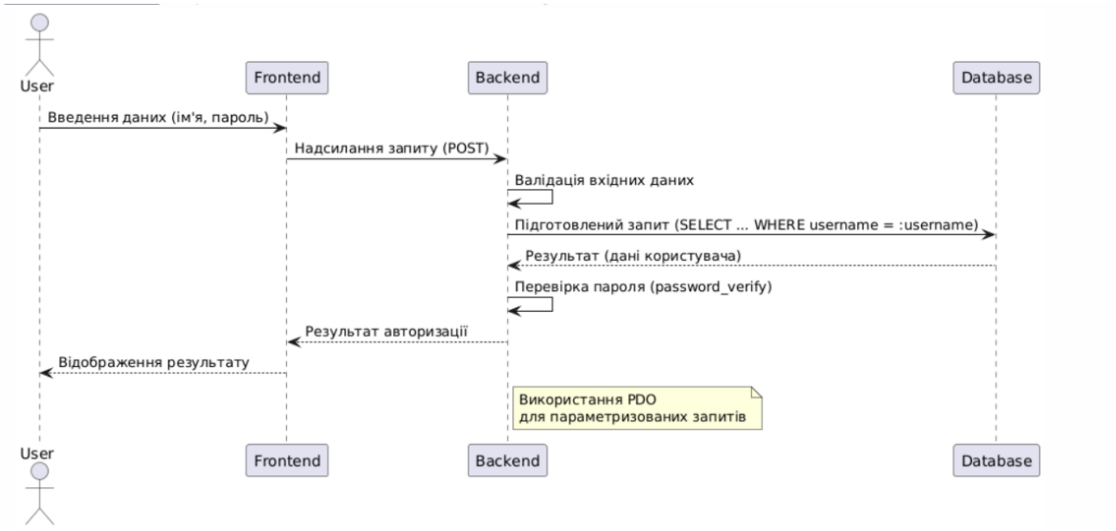
• Середовище розробки: Visual Studio Code.

• СУБД: MySQL/MariaDB, адміністрування через phpMyAdmin.

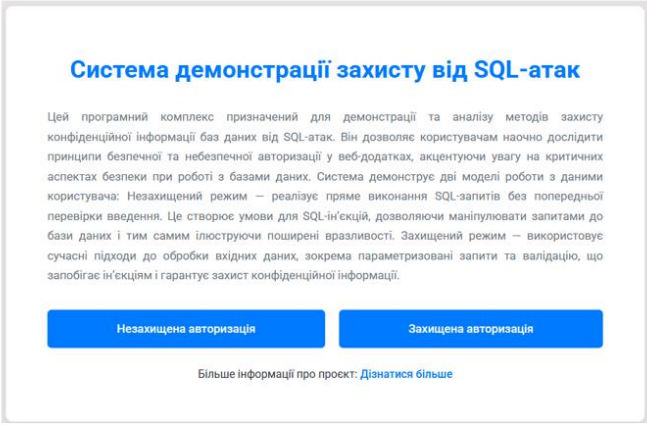
• Бібліотека: PDO для параметризованих запитів.

Архітектура системи: Use Case Diagram

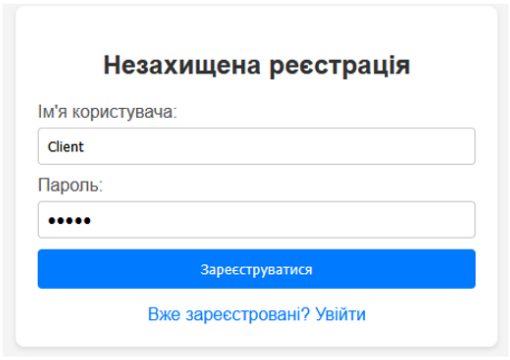
Архітектура системи: Sequence Diagram

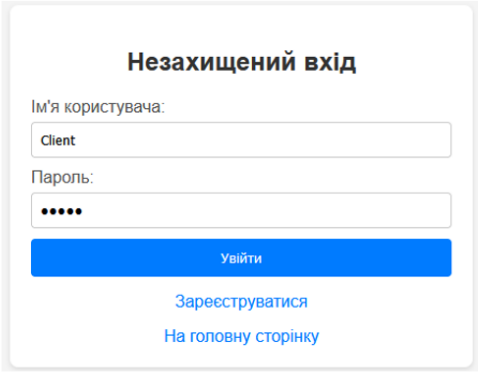


Інтерфейс користувача: Головна та інформаційні сторінки

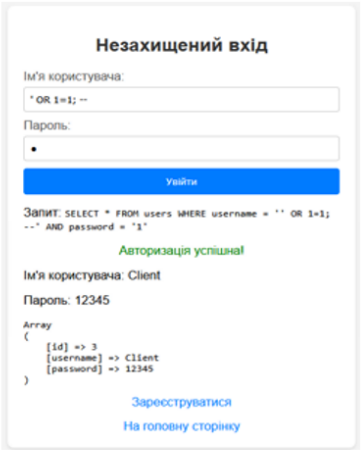


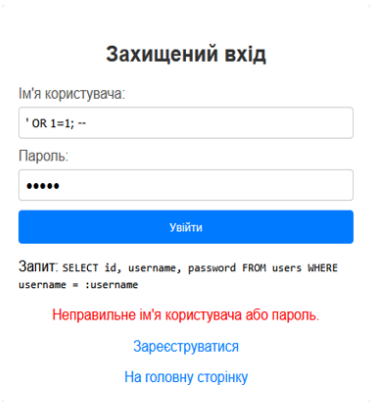
Інтерфейс користувача: Авторизація та реєстрація





Тестування системи





Впровадження та супровід

Процес розгортання:

• Встановлення XAMPP для локального сервера.

• Імпорт database.sql через phpMyAdmin для створення структури бази даних.

• Копіювання проєктних файлів у папку htdocs.

Плани розвитку:

• Додавання автоматичних тестів безпеки SQLMap.

• Розширення функціоналу для демонстрації інших типів атак.

Висновки

Основні результати роботи:

1.Провели теоретичне дослідження SQL-ін’єкцій, їх типів і механізмів реалізації.

2.Проаналізували сучасні методи захисту від SQL-ін’єкцій.

3.Розробили програмний продукт, що реалізує захист інформації.

4.Створили користувацький інтерфейс для демонстрації роботи системи.

5.Провели тестування ефективності захисту від SQL-атак.


Національний технічний університет України
«Київський політехнічний інститутімені Ігоря Сікорського»
National Technical University of Ukraine
“Igor Sikorsky Kyiv Polytechnic Institute”