

ПВНЗ «ЄВРОПЕЙСЬКИЙ УНІВЕРСИТЕТ»

КАФЕДРА Комп'ютерних наук та програмної інженерії
СПЕЦІАЛЬНІСТЬ 121 Інженерія Програмного Забезпечення
(шифр та назва)
ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА Фаховий молодший Бакалавр
(назва)

**ВИПУСКНА КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА
РОБОТА**

на тему:

Створення додатку для оцінки безпеки веб-додатків

(назва затвердженої теми)

(на прикладі _____)

Студент _____
(підпис студента)

Тищенко Назарій Андрійович
(прізвище, ім'я по-батькові студента)

Науковий
керівник _____
(підпис наукового керівника)

Яровий Роман Олександрович
(прізвище, ініціали, наукове звання і ступінь)

Допущено до захисту “__” _____ 2025 р.

Завідувач кафедри

(підпис)

Сушинський О.Є
(прізвище, ініціали, наукове звання і ступінь)

Декан факультету

(підпис)

Яровий Р.О
(прізвище, ініціали)

Завідувач
навчальної частини

(підпис)

Козинець А.О
(прізвище, ініціали)

Київ – 2025

ПВНЗ «ЄВРОПЕЙСЬКИЙ УНІВЕРСИТЕТ»

КАФЕДРА Комп'ютерних наук та програмної інженерії
СПЕЦІАЛЬНІСТЬ Інженерія Програмного Забезпечення

ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА Фаховий молодший Бакалавр

ЗАВДАННЯ

на випускню кваліфікаційну бакалаврську роботу студентіві :

Тищенко Назарія Андрійовича
(прізвище, ім'я, по-батькові студента)

1. Тема роботи Створення додатку для оцінки безпеки веб-додатків.

2. Затверджена наказом по університету від “ ” 2025р. № .

3. Цільова установка на вихідні дані до роботи Цільова установка на вихідні дані до роботи полягає у використанні HTTP-запитів, відповідей веб-серверів, структур URL-адрес та вхідних параметрів як об'єкта аналізу для виявлення потенційних вразливостей і генерації звітів щодо безпеки веб-додатків.

4. Перелік графічного матеріалу (з точним визначенням обов'язкових креслень) Схема алгоритму сканування; Структура застосунку для оцінки безпеки веб-додатків; Схема алгоритму фільтрації запитів; Схема алгоритму оцінки й пріоритизації знайдених недоліків; Схема алгоритму кореляції та структурування; Схема бази даних.

5. Консультанти по роботі, із зазначенням розділів, що їх стосуються:

Розділ	Науковий керівник	Підпис, дата	

6. Зміст випускної кваліфікаційної бакалаврської роботи (перелік питань)

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ; 1.1 Принципи безпеки веб-додатків; 1.2 Аналіз аналогічних розробок; 1.2.1 Інструмент Nikto; 1.2.2 Інструмент Wapiti; 1.2.3 Інструмент Wfuzz; 1.3 Постановка задачі; 2 ПРОЕКТУВАННЯ ЗАСТОСУНКУ ДЛЯ ОЦІНКИ БЕЗПЕКИ ВЕБ-ДОДАТКІВ; 2.1 Структура застосунку; 2.2 Вибір та обґрунтування інструментальних засобів; 2.3 Вибір та обґрунтування бази даних; 2.4 Алгоритм роботи застосунку; 3 РОЗРОБКА ТА ТЕСТУВАННЯ ЗАСТОСУНКУ ДЛЯ ОЦІНКИ БЕЗПЕКИ ВЕБ-ДОДАТКІВ; 3.1 Розробка бази даних; 3.2 Розробка інтерфейсу користувача; 3.3 Тестування функціоналу застосунку.

7. Календарний план виконання випускної кваліфікаційної бакалаврської роботи

№	Назва етапів випускної кваліфікаційної бакалаврської роботи	Термін виконання етапів випускної кваліфікаційної бакалаврської роботи	
		За планом	Фактично
1.	Обрання теми випускної кваліфікаційної магістерської роботи (ВКБР)	05.08.24 – 13.09.24	11.09.25
2.	Опрацювання літератури та інших джерел для написання ВКБР	16.09.24 – 01.11.24	25.10.24
3.	Виконання 1 розділу ВКБР	04.11.24 – 28.02.25	24.02.25
4.	Виконання 2 розділу ВКБР	03.03.25 – 21.03.25	18.03.25
5.	Виконання 3 розділу ВКБР, висновки	24.03.25 – 18.04.25	10.04.25
6.	Передзахист та проведення перевірки на плагіат	21.04.24 – 02.05.25	01.05.25
7.	Відгук керівника, рецензія	01.05.25 – 12.05.25	

8. Дата видачі завдання «___» _____ 202_р.

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

АНОТАЦІЯ

У роботі досліджено проблему забезпечення безпеки веб-додатків у контексті зростаючої кількості кіберзагроз і підвищеної уразливості цифрових сервісів, що працюють з конфіденційними даними. Метою дослідження стало створення функціонального програмного застосунку, який реалізує наскрізний підхід до оцінювання рівня безпеки веб-рішень, охоплюючи етапи валідації вхідних даних, аналізу вразливостей та генерації аналітичних звітів. У межах роботи проведено аналіз сучасних інструментів (Nikto, Wapiti, Wfuzz), розроблено архітектуру програмного комплексу, побудовано інтерфейс користувача, реалізовано модулі сканування, обробки результатів та зберігання даних. Тестування довело ефективність запропонованого рішення для виявлення найпоширеніших типів атак, зокрема SQL-ін'єкцій та XSS. Отримані результати підтверджують доцільність використання розробленого інструменту як основи для інтеграції в CI/CD-конвеєри безпеки або як окремого модуля в системах аудиту веб-додатків.

Ключові слова: веб-безпека, аналіз вразливостей, сканер, SQL-ін'єкція, XSS, автоматизоване тестування.

ANNOTATION

The paper addresses the issue of web application security in the context of increasing cyber threats and the heightened vulnerability of digital services that process confidential data. The aim of the research was to develop a functional software application implementing an end-to-end approach to assessing the security level of web systems, covering the stages of input data validation, vulnerability analysis, and analytical report generation. The study includes a review of existing tools (Nikto, Wapiti, Wfuzz), the design of the application architecture, development of the user interface, and implementation of scanning, result processing, and data storage modules. Testing confirmed the effectiveness of the proposed solution in detecting the most common attack types, including SQL injection and XSS. The results demonstrate the feasibility of using the developed tool as a foundation for integration into CI/CD security pipelines or as a standalone module in web application audit systems.

Keywords: web security, vulnerability analysis, scanner, SQL injection, XSS, automated testing.

ЗМІСТ

ВСТУП	7
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ	9
1.1 Принципи безпеки веб-додатків	9
1.2 Аналіз аналогічних розробок	17
1.2.1 Nikto	17
1.2.2 Wapiti	20
1.2.3 Wfuzz	23
1.3 Постановка задачі	26
2 ПРОЕКТУВАННЯ ЗАСТОСУНКУ ДЛЯ ОЦІНКИ БЕЗПЕКИ ВЕБ-ДОДАТКІВ	28
2.1 Структура застосунку	28
2.2 Вибір та обґрунтування інструментальних засобів	32
2.3 Вибір та обґрунтування бази даних	34
2.4 Алгоритм роботи застосунку	35
3 РОЗРОБКА ТА ТЕСТУВАННЯ ЗАСТОСУНКУ ДЛЯ ОЦІНКИ БЕЗПЕКИ ВЕБ-ДОДАТКІВ	44
3.1 Розробка бази даних	44
3.2 Розробка інтерфейсу користувача	45
3.3 Тестування функціоналу застосунку	47
ВИСНОВКИ	54
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	56
ДОДАТКИ	59
Додаток А «Лістинг програмного коду»	59

ВСТУП

З активним поширенням інформаційно-комунікаційних технологій та стрімким зростанням кількості веб-сервісів, які обробляють надзвичайно великі масиви персональних та комерційних даних, питання надійного захисту конфіденційної інформації набуває дедалі більшої актуальності. Зловмисні дії, пов'язані з використанням уразливостей у мережевих застосунках, можуть спричиняти відчутні збитки як для організацій, так і для індивідуальних користувачів, ставлячи під загрозу не лише майнові інтереси, а й репутацію компаній та стабільність цифрових платформ. У такому контексті обґрунтоване й систематичне виявлення слабких ланок у схемах безпеки інтернет-ресурсів стає першочерговою необхідністю, оскільки оперативність реагування на виявлені ризики безпосередньо впливає на зменшення ймовірності успішного атакуючого впливу. Розвиток методів моніторингу та аналізу даних надає широкий спектр можливостей для перевірки коректності функціонування веб-протоколів, процедур аутентифікації та зберігання ключових відомостей, водночас створюючи нові технологічні виклики, пов'язані з різноманітністю шкідливих сценаріїв і динамічним характером сучасних кібератак. Унаслідок цього постає завдання розробити спеціалізований інструмент, що здатен комплексно аналізувати наявні елементи й механізми захисту, швидко виявляти найкритичніші загрози та пропонувати чітку стратегію подальших дій для усунення вразливостей.

Об'єктом дослідження слугують технології створення й удосконалення програмного забезпечення для виявлення недоліків, які можуть бути використані зловмисниками з метою несанкціонованого доступу до даних або порушення штатного режиму роботи веб-додатків. Предметом у цьому випадку визначаються алгоритми та інструментальні засоби, спрямовані на систематизовану оцінку чинників ризику й формування чіткого переліку виявлених проблем, з якого можна зробити висновки щодо комплексності загроз. Мета роботи полягає у створенні функціонального застосунку, що

здатен інтегруватися із різними веб-середовищами, зберігаючи при цьому можливість оперативного розширення та модифікації логіки аналізу відповідно до розвитку методів атак. Новизна підходу зумовлюється реалізацією наскрізної концепції оцінювання: від початкової валідації й нормалізації даних, надходження яких може бути потенційним джерелом шкідливих впливів, до побудови інтерактивних візуальних панелей та висновків, що дають змогу проранжувати знайдені недоліки за критичністю й розробити точкові рекомендації. Завдання, поставлені в межах цієї роботи, передбачають розроблення концепції архітектури, налаштування процесу ідентифікації та фільтрації типових загроз, забезпечення наочного уявлення про стан безпеки шляхом використання зрозумілих звітних форматів і узгодження з вимогами надійного зберігання та обробки вхідних відомостей. Залучення таких методологічних і практичних підходів створює комплексну платформу, яка сприяє своєчасному виявленню критичних точок, узгодженню дій між усіма учасниками процесу гарантування безпеки й підвищує стійкість веб-додатків до найпоширеніших видів кібератак.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Принципи безпеки веб-додатків

Конфіденційність у веб-додатках – це аспект, який не лише потребує постійної уваги, але й є викликом для сучасного розробника. Захист персональних даних стає основою довіри користувачів до сервісів, і саме на цій довірі тримається функціонування багатьох цифрових платформ. Водночас зростання кількості веб-додатків спричиняє нові загрози, які не завжди очевидні на перший погляд.

Розробник з України, зважаючи на поточні реалії, може сприймати це питання як особистий обов'язок — адже національна кібербезпека та міжнародні стандарти, такі як GDPR чи CCPA, накладають чіткі обмеження та відповідальність. Наприклад, навіть звичайний вхід у систему зберігає більше, ніж здається: IP-адреси, cookie-файли, поведінкові патерни[1]. І ці дані мають оброблятися з максимальною обережністю.

У той самий час важко ігнорувати компроміс між зручністю користувача та дотриманням конфіденційності. Функції автоматичного заповнення форм або рекомендації товарів спираються на збір та аналіз великих масивів даних.

Часто обговорюється проблема витоків даних через неналежне тестування коду. Наприклад, неправильно налаштовані доступи до баз даних або недостатній захист API можуть стати причиною масштабного інциденту. Іронія в тому, що навіть невелика помилка може перетворити ретельно створену систему на загрозу для тисяч користувачів. В Україні, де багато стартапів орієнтовані на західні ринки, це питання стає ще гострішим через суворіші регуляції у Європі та США.

Питання забезпечення конфіденційності в розподілених системах є не менш захопливим і водночас складним. Особливо це актуально для веб-додатків, побудованих за мікросервісною архітектурою, де дані постійно циркулюють між різними серверами. Використання протоколів на кшталт

HTTPS або TLS[2], які відповідають за шифрування передачі даних, є важливим елементом, але його часто недостатньо. Ефективна система також потребує додаткових заходів безпеки, таких як ретельний контроль доступу чи впровадження журналів аудиту для моніторингу взаємодії з даними.

Зрештою, постійне навчання і розвиток інструментів є невіддільною частиною забезпечення конфіденційності. Додатки типу OWASP ZAP[3], які виявляють вразливості, повинні стати інструментом у щоденному арсеналі розробника. І хоча це не гарантує абсолютної безпеки, такий підхід значно знижує ризики, зберігаючи дані користувачів у безпеці.

Таким чином, конфіденційність у веб-додатках – це не просто набір правил чи стандартів, це безперервний процес адаптації, балансування між захистом і функціональністю та моральна відповідальність кожного, хто створює цифрові сервіси[4].

У веб-додатках цілісність відіграє одну з ключових ролей, забезпечуючи як надійність системи, так і довіру користувачів. У межах створення веб-рішень вона охоплює захист даних від будь-яких несанкціонованих змін і гарантує їхню правдивість[5]. Якщо цілісність буде порушена, наслідки можуть бути непередбачуваними: від збоїв у роботі додатка до серйозного удару по репутації.

У веб-додатках забезпечення цілісності інформації становить фундаментальний принцип, що безпосередньо впливає на надійність функціонування системи та рівень довіри користувачів. Забезпечення захисту даних від несанкціонованих змін досягається за рахунок застосування комплексного підходу, що включає використання криптографічних алгоритмів, протоколів шифрування, механізмів цифрового підпису та алгоритмів хешування. Реалізація зазначених технологій дозволяє не лише захищати інформацію під час її передачі, а й забезпечувати її автентичність при зберіганні, що є надзвичайно важливим в умовах зростаючих кіберзагроз. Застосування засобів журналювання та моніторингу подій у режимі реального часу сприяє оперативному виявленню потенційних спроб компрометації, а

також дозволяє проводити ретроспективний аналіз інцидентів з метою удосконалення системи безпеки.

Ретельно спроектована архітектура веб-додатка має враховувати принципи мінімізації привілеїв та розмежування доступу, що зменшує ризики несанкціонованого впливу на критичні компоненти системи. У цьому контексті впровадження механізмів аутентифікації та авторизації, що ґрунтуються на сучасних стандартах безпеки, є обов'язковим елементом, спрямованим на попередження несанкціонованих дій[6]. Поряд із цим, використання політик безпеки, що регламентують доступ до даних, дозволяє гарантувати їхню цілісність навіть у випадку спроб злому або соціальної інженерії. Науковий підхід до аналізу вразливостей системи сприяє виявленню потенційних загроз на ранніх стадіях розробки, що дозволяє впроваджувати превентивні заходи та мінімізувати можливі збитки.

Порушення цілісності даних може призводити до низки негативних наслідків, що охоплюють як технічні, так і організаційні аспекти роботи веб-додатків. Серед технічних наслідків виділяються можливість виникнення збоїв у роботі додатка, втрати даних або їх некоректної обробки, що, у свою чергу, може викликати порушення логіки бізнес-процесів та знизити ефективність взаємодії між компонентами системи[7]. Організаційні наслідки включають втрату довіри клієнтів, пошкодження репутації підприємства, а також можливі юридичні наслідки у вигляді штрафів чи компенсаційних виплат. В умовах сучасної інформаційної безпеки необхідність швидкого реагування на інциденти, аналізу причин їх виникнення та впровадження корегувальних заходів є важливими складовими стратегії управління ризиками.

Загалом, інтеграція методів забезпечення цілісності даних у веб-додатках сприяє підвищенню їх надійності та стійкості до зовнішніх загроз. Ретельне дослідження архітектури системи, впровадження сучасних криптографічних рішень, регулярне проведення аудитів безпеки та тестування вразливостей дозволяє створити надійний інформаційний простір, що здатний адаптуватися до змін у цифровому середовищі. Комплексний підхід до забезпечення

цілісності даних є невід'ємною умовою не тільки ефективної роботи веб-додатка, але й підтримання стабільності бізнес-процесів, що забезпечує довгострокову конкурентоспроможність та позитивний імідж організації на ринку цифрових технологій[8].

Важливо розуміти, що загрози для цілісності виникають не лише через хакерські атаки. Банальний баг у коді чи недбалість у валідації даних можуть створити ситуації, де зміна інформації відбудеться непомітно, без видимих слідів. Наприклад, якщо у формі введення користувача не враховується обробка спеціальних символів, це може дати можливість ввести SQL-ін'єкцію, що порушить як цілісність бази даних, так і загальну безпеку[9].

Методи забезпечення цілісності варіюються залежно від завдань і архітектури веб-додатку. Широко використовується хешування, яке дозволяє перевіряти, чи не були змінені дані під час передачі або зберігання. Але навіть така надійна, на перший погляд, практика потребує додаткового захисту. Отже, доцільно використовувати цифрові підписи або НМАС, які включають секретний ключ у процес перевірки.

Цікавим прикладом із практичного досвіду є використання блокчейну для гарантування незмінності даних. Ця технологія дозволяє створювати децентралізовані журнали подій, у яких кожна транзакція нерозривно пов'язана з попередньою, забезпечуючи високий рівень захисту та прозорості. На перший погляд, це виглядає як ідеальне рішення. Однак його впровадження нерідко супроводжується серйозними технічними та фінансовими труднощами, що може стати значною перешкодою для невеликих проєктів[10].

Іноді здається, що забезпечення цілісності — це проблема лише великих платформ. Але навіть у малих веб-додатках, наприклад, в інтернет-магазинах, де є корзина покупок, втрата цілісності може стати катастрофою. Наприклад, якщо ціни або кількість товарів зміняться через уразливість у JavaScript-коді, що працює на стороні клієнта[11].

Наостанок, слід зазначити, що забезпечення цілісності — це не лише технічне завдання, а й частина загальної культури розробки. Це про

відповідальність розробників, про бажання створювати не лише функціональний, а й надійний продукт. Бо тільки в поєднанні технологій, уважності до деталей і розуміння потенційних ризиків можна досягти того, щоб користувачі навіть не задумувалися про безпеку, просто довіряючи вашому додатку.

Доступність у веб-додатках є не просто додатковим аспектом розробки, а фундаментальною характеристикою сучасного програмного забезпечення[12]. Вона визначає можливість ефективного використання продукту людьми з різними фізичними, когнітивними або технічними обмеженнями. Водночас, цей параметр часто залишається недооціненим або неправильно реалізованим. У процесі розробки багато уваги зосереджено на функціоналі та візуальній привабливості, а доступність сприймається як вторинна вимога.

У сучасних веб-додатках доступність розглядається як інтегральна характеристика, що визначає можливості взаємодії користувачів із цифровим продуктом незалежно від їхніх фізичних, когнітивних чи технічних обмежень[13]. Забезпечення даної властивості вимагає системного підходу, що охоплює не лише поверхневий дизайн чи візуальну привабливість, а й детальне опрацювання семантичної структури документації, впровадження атрибутів доступності (ARIA) та оптимізацію для роботи з асистивними технологіями[14]. Методологія розробки повинна враховувати вимоги стандартів інклюзивного дизайну, таких як WCAG, що дозволяє забезпечити відповідність продукту найвищим вимогам якості та зручності використання[15]. Цей підхід сприяє створенню інтерфейсів, здатних адаптуватися до різноманітних умов експлуатації, що є особливо важливим у контексті зростаючої цифровізації суспільства[16].

Ретельне проектування інтерфейсів із врахуванням принципів доступності включає аналіз вимог користувачів з особливими потребами, що дозволяє розробляти функціонал, який є зрозумілим та легким у використанні для широкого кола аудиторії. Зокрема, застосування адаптивного дизайну, що враховує різноманітність пристроїв, роздільних здатностей екранів та умов

освітлення, дозволяє оптимізувати візуальне сприйняття інформації[17]. Одночасно, важливе місце займає інтеграція механізмів, що забезпечують можливість масштабування тексту, налаштування контрастності кольорової гами та застосування альтернативних текстових описів для зображень, що сприяє підвищенню зручності використання як для осіб з порушеннями зору, так і для користувачів, які стикаються з іншими формами обмежень[18].

Недооцінка аспектів доступності під час розробки веб-додатків може призводити до серйозних негативних наслідків як з технічної, так і з соціально-економічної точок зору. Неврахування вимог інклюзивності може створити бар'єри для доступу до інформації, що позначається на загальній ефективності взаємодії користувача з продуктом, а також може стати причиною юридичних претензій чи санкцій з боку регуляторних органів[19]. Крім того, недбале ставлення до інклюзивного дизайну негативно впливає на репутацію розробників і компаній, що, в свою чергу, обмежує можливості виходу на нові ринки та зменшує конкурентоспроможність програмного забезпечення у глобальному масштабі[20]. Врахування усіх зазначених аспектів дозволяє не лише уникнути потенційних ризиків, а й сприяє формуванню позитивного іміджу розробника як соціально відповідального суб'єкта у сфері інформаційних технологій.

Комплексний підхід до забезпечення доступності веб-додатків передбачає інтеграцію інструментів автоматизованого тестування, які здійснюють перевірку відповідності продукту стандартам інклюзивності, а також проведення регулярних аудитів та залучення експертів із спеціалізованих галузей[21]. Аналіз ефективності реалізованих рішень за допомогою статистичних даних, зворотного зв'язку від кінцевих користувачів та клінічних досліджень дозволяє вдосконалювати функціонал з урахуванням актуальних потреб аудиторії. Такий підхід сприяє формуванню високого рівня взаємодії з користувачем, забезпечує стабільну роботу додатку та створює умови для подальшої еволюції програмного продукту з урахуванням динаміки соціальних та технологічних змін[22].

Таким чином, інтеграція принципів доступності у процес розробки веб-додатків є невід'ємною умовою для створення сучасного, конкурентоспроможного та соціально відповідального програмного забезпечення. Забезпечення високих стандартів інклюзивності сприяє не лише технічному удосконаленню продукту, але й підвищенню його комерційної привабливості, розширенню аудиторії користувачів та зміцненню довіри як кінцевих споживачів, так і бізнес-партнерів[23].

Додавання базових елементів доступності, таких як текстові альтернативи для зображень, контрастні кольори чи підтримка клавіатурної навігації, не вимагає значних ресурсів[24]. Проте саме ці кроки можуть істотно підвищити якість взаємодії для значної кількості користувачів. Наприклад, інклюзивний дизайн не тільки допомагає людям із порушеннями зору, а й покращує досвід користування для тих, хто працює в складних умовах, як-от при яскравому сонячному світлі.

Нерідко розробники нехтують тестуванням продуктів із використанням допоміжних технологій, як-от програм екранного зчитування. Це відбувається не через недбалість, а через відсутність глибокого усвідомлення важливості таких технологій. Більшість веб-додатків навіть не проходять базового аудиту на відповідність стандартам WCAG[25], що визначає критерії доступності на різних рівнях. При цьому, реалізація цих стандартів забезпечує конкурентні переваги, про які часто не замислюються.

Примітно, що саме локальні ініціативи в Україні починають звертати увагу на доступність у цифровій сфері. Це може бути пов'язано з інтеграцією європейських цінностей та законодавства, а також зростанням попиту на міжнародному ринку, де доступність давно стала обов'язковим стандартом. Цікаво, що деякі українські розробники почали інтегрувати в свої додатки можливості персоналізації, як-от налаштування розміру шрифтів чи зміна кольорової схеми для людей із порушеннями зору.

Водночас, важливо підкреслити, що доступність — це не тільки про фізичні обмеження. Наприклад, користувачі з низькою швидкістю інтернет-

з'єднання також стикаються з труднощами через важкі сторінки або ресурси, що не оптимізовані. Використання технологій як-от прогресивні веб-додатки (PWA) значно покращує ситуацію[26], дозволяючи доступ навіть в офлайн-режимі. Таким чином, проблема доступності є багатовимірною та потребує уваги не лише з боку дизайну, але й із точки зору архітектури додатку.

Авторизація та аутентифікація у веб-додатках є фундаментальними складовими сучасних систем інформаційної безпеки. У своїй основі, аутентифікація виступає як процес перевірки, чи є користувач тим, за кого себе видає, тоді як авторизація визначає, які саме дії дозволено виконувати після успішного підтвердження особи. Ці поняття часто змішуються в повсякденному спілкуванні, однак їх розмежування критично важливе для розуміння архітектури захисту даних.

Важливість аутентифікації зростає в умовах дедалі складнішої мережевої взаємодії. Традиційні методи, такі як комбінації логіна та пароля, залишаються розповсюдженими, проте їхню безпеку значно підривають атаки типу "brute force" або фішингові кампанії. Альтернативні підходи, такі як біометричні дані або багатофакторна аутентифікація (MFA)[26], зменшують ризик несанкціонованого доступу, хоча й додають складності у впровадженні.

Авторизація, у свою чергу, базується на концепції ролей та прав доступу. Замість того щоб надавати всім користувачам однакові можливості, системи використовують принцип "найменших привілеїв". Наприклад, у системі управління контентом адміністратору надається доступ до всіх функцій, тоді як звичайний користувач може лише переглядати публікації. Це дозволяє зменшити обсяг шкоди у разі компрометації окремого облікового запису.

Сучасні веб-додатки дедалі частіше інтегруються з сервісами сторонніх розробників для реалізації аутентифікації. Протоколи на зразок OAuth 2.0 та OpenID Connect дозволяють користувачам входити до системи через свої акаунти у Google, Facebook або інших платформах[26]. Такий підхід значно полегшує життя кінцевого користувача, проте вимагає від розробника

глибокого розуміння принципів передачі токенів та забезпечення їхнього захисту.

Згадуючи про потенційні ризики, варто звернути увагу на проблеми витоку облікових даних або неправильного налаштування токенів доступу. Наприклад, якщо токен не обмежений у часі або дії, це може дозволити зловмиснику здійснити атаки "token replay". Одним із рішень цієї проблеми є регулярне оновлення токенів за допомогою механізмів оновлення сесій.

Розробка безпечної системи авторизації та аутентифікації вимагає врахування численних факторів. Навіть найкращі технології, такі як шифрування паролів за допомогою bcrypt або використання TLS для передачі даних, можуть втратити свою ефективність без правильної архітектури. У цьому контексті зручність для користувача часто вступає в конфлікт із рівнем безпеки, і завданням розробника стає пошук балансу між цими вимогами.

Таким чином, авторизація та аутентифікація у веб-додатках залишаються динамічною сферою, що потребує постійної адаптації до нових загроз. Розробники мають враховувати не лише сучасні технічні рішення, але й соціальні аспекти, такі як сприйняття користувачами додаткових рівнів захисту.

1.2 Аналіз аналогічних розробок

1.2.1 Nikto

Інтерфейс інструменту представлено на рисунку 1.1.

```

- Nikto v2.1.6
-----
+ Target IP:      87.229.144.34
+ Target Hostname: google.ru
+ Target Port:    80
+ Start Time:     2015-02-26 14:51:38 (GMT6)
-----
+ Server: gws
+ Uncommon header 'alternate-protocol' found, with contents: 80:quic,p=0.08
+ The X-Content-Type-Options header is not set. This could allow the user agent
to render the content of the site in a different fashion to the MIME type
+ Root page / redirects to: http://www.google.ru/
+ Server banner has changed from 'gws' to 'sfte' which may suggest a WAF, load b
alancer or proxy is in place
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ OSVDB-5737: WebLogic may reveal its internal IP or hostname in the Location he
ader. The value is "http://87.229.144.34/".

```

Рис.1.1 Інструмент Nikto

Nikto — це інструмент, створений для автоматизованого сканування вебсерверів з метою виявлення потенційних вразливостей, і його унікальність полягає в поєднанні простоти використання та гнучкості налаштувань[27]. Інструмент, будучи відкритим та безкоштовним, надає можливість аналізу широкого спектру серверів, перевіряючи їх на відомі проблеми безпеки, зокрема застарілі версії програмного забезпечення, небезпечні конфігурації та наявність стандартних файлів або директорій, які можуть стати об'єктом атак.

Інструмент Nikto є високоефективним засобом автоматизованого сканування вебсерверів, що спрямований на виявлення широкого спектру потенційних вразливостей у програмному забезпеченні та конфігураціях серверного середовища. Завдяки відкритій архітектурі та безкоштовному розповсюдженню, Nikto набуває популярності серед фахівців з кібербезпеки, оскільки дозволяє проводити швидкий та комплексний аудит систем безпеки шляхом надсилання специфічних HTTP-запитів до цільових ресурсів. Інструмент використовує детально структуровану базу даних відомих вразливостей, що регулярно оновлюється, що дозволяє ідентифікувати не лише загальновідомі недоліки, пов'язані із застарілими версіями програмного забезпечення, але й помітити небезпечні конфігурації, стандартні файли чи директорії, які можуть стати об'єктом експлуатації зловмисниками. Впровадження Nikto в процес тестування вебсерверів забезпечує можливість швидкого реагування на виявлені проблеми, сприяючи своєчасному

впровадженню корегувальних заходів та мінімізації ризиків компрометації інформаційного простору.

Комплексність підходу, реалізованого у Nikto, забезпечується гнучкістю налаштувань, що дозволяє адаптувати параметри сканування до специфічних умов експлуатації цільових систем, а також інтегрувати його у більш широкі системи управління безпекою. За рахунок можливості проводити аналіз як окремих вебсерверів, так і комплексних інфраструктур, інструмент сприяє систематизації процесів аудиту та моніторингу безпеки, що є критично важливим в умовах зростаючих кіберзагроз. Деталізований звіт, що генерується після проведення сканування, містить опис виявлених недоліків, можливі варіанти їх експлуатації та рекомендації щодо впровадження заходів захисту, що дозволяє як оперативно усунути критичні вразливості, так і розробити стратегію довгострокового управління ризиками. Таким чином, Nikto виступає невід'ємною складовою сучасних систем забезпечення інформаційної безпеки, сприяючи підвищенню стійкості вебсерверів та зміцненню довіри користувачів до цифрової інфраструктури.

Важко заперечувати, що Nikto не можна назвати ідеальним. Наприклад, його відкритість призводить до того, що інструмент легко виявляється системами захисту, оскільки сканування відбувається без спроб замаскувати активність. Але цей недолік певною мірою компенсується швидкістю виконання завдань. Крім того, його базу вразливостей, що налічує десятки тисяч записів, можна оновлювати, забезпечуючи актуальність перевірок у світі, де загрози швидко еволюціонують.

Ще одним важливим аспектом є можливість створення власних плагінів. Наприклад, у проєктах, де вебсервер використовує рідкісне або самописне програмне забезпечення, стандартні сканери не завжди ефективні. Завдяки Nikto, можна швидко налаштувати перевірку під конкретний сценарій. Така функція може бути корисною, коли потрібно виявити проблеми не лише у загальновідомих компонентах, а й у тих, що є унікальними для певної компанії чи структури.

І хоча Nikto зазвичай використовують для пасивного дослідження інфраструктури, не варто забувати про його потенціал у навчальних цілях. Показуючи учням, як навіть базові інструменти здатні виявити серйозні недоліки, можна стимулювати глибше розуміння важливості регулярних перевірок безпеки. До прикладу, демонстрація того, як відкритий каталог адміністрування може бути знайдений лише за кілька секунд, допомагає сформувати уявлення про ризики.

Щодо інтерфейсу, Nikto лишається консольним інструментом, що може спершу відлякати новачків, проте досвідчені користувачі оцінять його мінімалізм та ефективність. Часом саме такі "спартанські" рішення стають незамінними, адже дозволяють зосередитися на результатах, а не на навчанні роботи з самим інструментом.

Таким чином, незважаючи на свою видиму простоту, Nikto залишається інструментом, здатним розкрити серйозні проблеми у захисті вебсервера. Його ефективність залежить не лише від бази вразливостей, але й від того, як користувач застосовує отримані дані. У кінцевому підсумку, Nikto — це не про інструмент як такий, а про усвідомлення того, наскільки легко може бути порушена безпека, якщо не приділяти їй належної уваги.

1.2.2 Wapiti

Інтерфейс інструменту представлено на рисунку 1.2.

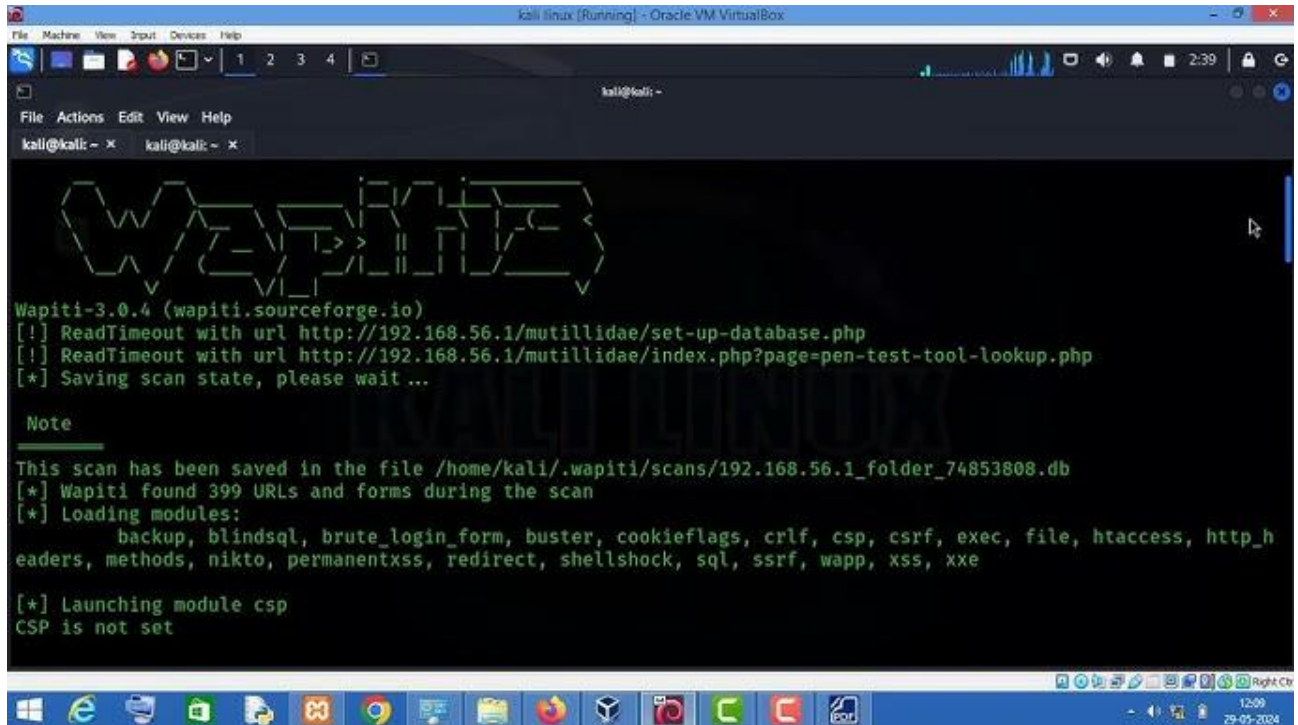


Рис.1.2 Інструмент Wapiti

Wapiti є потужним інструментом для аналізу веб-безпеки, що створений із акцентом на простоту використання і ефективність виявлення потенційних вразливостей. Цей сканер, який використовує підхід "чорної скриньки", спирається на моделювання взаємодії з веб-додатком, що дозволяє виявляти широкий спектр проблем, від SQL-ін'єкцій до XSS-атак. Унікальність Wapiti полягає у тому, що він аналізує веб-додаток не лише поверхово, але й занурюється у деталі, наприклад, обробку HTTP-запитів, що значно підвищує точність виявлення[28].

Wapiti представляє собою високоефективний інструмент аналізу веб-безпеки, розроблений із орієнтацією на спрощення процесу виявлення уразливостей без шкоди для якості проведеного дослідження. Використовуючи підхід «чорної скриньки», інструмент моделює поведінку звичайного користувача, не потребуючи доступу до вихідного коду додатку, що дозволяє проводити комплексний аналіз зовнішнього інтерфейсу веб-сервісу. У процесі сканування здійснюється автоматизований збір інформації про доступні URL, форми введення, параметри запитів та інші елементи, що беруть участь у комунікації між клієнтом і сервером. Такий метод дозволяє виявляти широкий спектр вразливостей, серед яких помітними є SQL-ін'єкції, XSS-атаки, а також

менш очевидні недоліки в обробці даних, що можуть бути пропущені при традиційному аналізі.

Значну увагу приділено аналізу структури HTTP-запитів, що є однією з унікальних особливостей Wariti. Інструмент не лише здійснює поверхневий моніторинг обміну даними, а й проводить детальний розбір запитів, виявляючи потенційні недоліки у їхній обробці, наприклад, неправильну санітизацію вхідних даних чи неналежну реалізацію механізмів аутентифікації та авторизації. Ретельна обробка запитів дозволяє ідентифікувати навіть ті уразливості, які виникають внаслідок специфічних налаштувань або нестандартних підходів у реалізації бізнес-логіки веб-додатку. Завдяки цьому забезпечується високий рівень точності сканування, що суттєво знижує ризики експлуатації виявлених вразливостей потенційними зловмисниками.

Важливим аспектом роботи Wariti є його модульна архітектура, що забезпечує гнучкість налаштування процесу сканування відповідно до специфіки тестованого веб-додатку. Система дозволяє адаптувати параметри аналізу, використовуючи різноманітні алгоритми парсингу та моделі поведінки, що сприяє виявленню навіть малопомітних аномалій. Застосування технологій автоматизації та інтегрованих механізмів журналювання дозволяє проводити моніторинг у режимі реального часу, що сприяє оперативному виявленню потенційних загроз і проведенню ретроспективного аналізу інцидентів. Такий підхід є надзвичайно актуальним в умовах постійного зростання кіберзагроз та необхідності забезпечення високих стандартів безпеки.

Інтеграція Wariti у процес розробки та тестування веб-додатків сприяє комплексному підходу до управління ризиками безпеки. Результати сканування надаються у вигляді докладних звітів, що містять інформацію про виявлені уразливості, їх потенційний вплив на роботу системи та рекомендації щодо їх усунення. Такий аналітичний підхід дозволяє не лише оперативно реагувати на конкретні загрози, але й планувати стратегічні заходи для посилення захисту інформаційних ресурсів у довгостроковій перспективі. Комплексне дослідження поведінки веб-додатків із застосуванням Wariti сприяє

формуванню стійкої системи безпеки, здатної адаптуватися до змін у сучасному цифровому середовищі, що є критично важливим для забезпечення довіри користувачів та підтримання репутації організації.

На відміну від багатьох інших інструментів, які часто є закритими і складними у налаштуванні, Wariti пропонує відкритий вихідний код. Це дозволяє легко інтегрувати його в існуючі процеси розробки чи тестування. Наприклад, його можна використовувати в поєднанні з CI/CD-конвеєрами для забезпечення безперервної перевірки безпеки на різних етапах життєвого циклу додатка.

Особливістю цього інструменту є його здатність працювати з HTTP-заголовками, куками та сесіями, що дозволяє точно моделювати поведінку реальних користувачів. Крім того, підтримка форматів звітів, таких як JSON або HTML, спрощує подальший аналіз знайдених проблем. Наприклад, якщо є проект, де необхідно оцінити рівень захищеності API, що обробляє фінансові транзакції; Wariti не лише виявить потенційні загрози, а й запропонує шляхи їх усунення.

Втім, є й обмеження. Наприклад, інструмент може пропустити вразливості, що виникають у специфічних сценаріях, пов'язаних із рідкісними конфігураціями серверів або нестандартними архітектурними рішеннями. Проте це не применшує його значущості, оскільки регулярне використання Wariti здатне суттєво покращити загальний стан безпеки системи.

Wariti є одним із небагатьох інструментів, який ефективно поєднує в собі простоту та багатий функціонал. Його гнучкість дозволяє налаштовувати параметри аналізу відповідно до потреб користувача, забезпечуючи високу адаптивність до різних умов. Усе це робить його незамінним засобом для сучасного програміста чи адміністратора, які прагнуть захистити свої проекти від можливих загроз.

1.2.3 Wfuzz

Інтерфейс інструменту представлено на рисунку 1.3.

```
*****
* Wfuzz 2.0 - The Web Bruteforcer *
* Coded by: *
* Christian Martorella (cmartorella@edge-security.com) *
* Xavier Mendez aka Javi (xmendez@edge-security.com) *
* Carlos del ojo (deepbit@gmail.com) *
*****

Target: http://localhost:8888/MAMP/FUZZ
Payload type: file,wordlist/general/common.txt

Total requests: 950

=====
ID      Response  Lines  Word    Chars    Server                                Redirect  Request
=====
00244:  C=301      9 L    30 W    330 Ch   Apache/2.0.63 (Un                    http://localhost:8888/MAMP/css/    " - css"
00324:  C=301      9 L    30 W    334 Ch   Apache/2.0.63 (Un                    http://localhost:8888/MAMP/english/  " - english"
00440:  C=200     10 L    37 W    586 Ch   Apache/2.0.63 (Un                    " - index"
00430:  C=301      9 L    30 W    333 Ch   Apache/2.0.63 (Un                    http://localhost:8888/MAMP/images/    " - images"
00470:  C=301      9 L    30 W    329 Ch   Apache/2.0.63 (Un                    http://localhost:8888/MAMP/js/        " - js"
00620:  C=301      9 L    30 W    330 Ch   Apache/2.0.63 (Un                    http://localhost:8888/MAMP/php/       " - php"
00628:  C=302      0 L     0 W     0 Ch     Apache/2.0.63 (Un                    /phpMyAdmin/                        " - phpmyadmin"
00783:  C=301      9 L    30 W    334 Ch   Apache/2.0.63 (Un                    http://localhost:8888/MAMP/spanish/    " - spanish"
```

Рис.1.3 Інструмент Wfuzz

Wfuzz є потужним інструментом, який використовується для фуззингу вебдодатків, і його ефективність вразила багатьох фахівців у сфері кібербезпеки. Ця програма з відкритим вихідним кодом забезпечує можливість виконання різноманітних тестів на проникнення, включаючи виявлення прихованих ресурсів чи перевірку наявності вразливостей у HTTP-запитах[29]. Вона не обмежує користувачів у сценаріях використання, що є суттєвим фактором для більш складних завдань.

Wfuzz є потужним інструментом для проведення фуззингу веб-додатків, що забезпечує можливість автоматизованого тестування на проникнення за рахунок надсилення великої кількості HTTP-запитів із різноманітною варіацією параметрів, шляхів, заголовків та значень, що суттєво розширює можливості виявлення прихованих ресурсів та невідомих уразливостей. Інструмент відзначається високою гнучкістю налаштувань, що дозволяє використовувати власні словники та конфігурації, адаптовані до специфічних умов цільових систем, таким чином сприяючи оптимізації процесу пошуку слабких місць у веб-інфраструктурі. Завдяки підтримці різноманітних режимів роботи, таких як фуззинг параметрів URL, модифікація даних у POST-запитах та маніпуляція HTTP-заголовками, забезпечується комплексний аналіз механізмів обробки запитів, що дозволяє виявити не лише очевидні, а й приховані вразливості, що можуть бути використані зловмисниками для несанкціонованого доступу чи компрометації даних.

Важливо відзначити, що Wfuzz, як інструмент з відкритим вихідним кодом, сприяє розвитку спільноти фахівців з кібербезпеки, що активно обмінюються методиками налаштування та інтеграції інструменту у ширший арсенал тестування на проникнення. Ретельно розроблена архітектура дозволяє використовувати багатопоточність та паралельне виконання тестових сценаріїв, що значно підвищує ефективність проведення масштабних аудитів безпеки, особливо при роботі з великими обсягами даних або в умовах високої частоти запитів. Інтеграція з іншими інструментами та системами аналізу вразливостей дозволяє створювати єдину інфраструктуру для комплексного дослідження стану безпеки веб-додатків, що включає як попередню ідентифікацію потенційних загроз, так і подальший аналіз отриманих результатів із застосуванням методів машинного навчання або інших сучасних підходів до обробки даних.

Комплексний підхід до використання Wfuzz у процесі тестування на проникнення дозволяє не лише виявляти стандартні вразливості, а й проводити глибокий аналіз реакції серверних компонентів на аномальні вхідні дані, що має вирішальне значення для забезпечення цілісності інформації в сучасних веб-додатках. Ретельна автоматизація процесів тестування сприяє своєчасному виявленню потенційних проблем, що може запобігти серйозним наслідкам у випадку компрометації системи, зокрема витоку конфіденційної інформації або порушення роботи критичних бізнес-процесів. Таким чином, застосування Wfuzz виступає невід'ємною складовою сучасного підходу до забезпечення кібербезпеки, що дозволяє формувати надійний захисний бар'єр перед зовнішніми загрозами та сприяти підвищенню рівня довіри користувачів до веб-додатків у цифровому середовищі.

Варто зауважити, що Wfuzz розрахований не лише на професіоналів, але й може бути освоєний новачками за наявності певного ентузіазму. Наприклад, інтеграція з Wordlists дозволяє легко налаштовувати тести, використовуючи заздалегідь підготовлені списки. Цей інструмент можна застосовувати і для задач, які на перший погляд здаються нетиповими, таких як перевірка API на

надійність. Саме така універсальність часто стає причиною його популярності серед тих, хто працює з мережею.

Незважаючи на простоту інтерфейсу, використання командного рядка може бути викликом для початківців. Проте це компенсується високим рівнем гнучкості та налаштовуваності. Наприклад, є можливість змінювати методи HTTP, вказувати специфічні заголовки або навіть працювати з проксі-серверами, що дозволяє ефективно адаптувати Wfuzz до будь-якого середовища тестування. Проте, слід бути обережним, щоб уникнути перевантаження серверів, адже інтенсивний фуззинг може призводити до небажаних наслідків, як-от блокування IP-адреси.

Окремо заслуговує уваги можливість використання Wfuzz разом з іншими інструментами. Інтеграція з Burp Suite чи OWASP ZAP надає нові горизонти для створення більш складних сценаріїв тестування. Така взаємодія особливо цінна для великих команд, які займаються аудитом безпеки, де одночасне використання різних інструментів стає необхідністю.

Попри всі переваги, інструмент має певні обмеження. Наприклад, немає вбудованої графічної оболонки, що іноді ускладнює роботу для тих, хто звик до візуалізації результатів. Водночас, це не є критичним недоліком, адже текстові звіти Wfuzz містять вичерпну інформацію, яку легко можна обробити іншими засобами.

Таким чином, Wfuzz є прикладом ідеального балансу між функціональністю, гнучкістю і простотою, дозволяючи як професіоналам, так і новачкам у галузі кібербезпеки розширити свої можливості. Це справжній "швейцарський ніж" у світі веб-фуззінгу, який водночас викликає захоплення та надихає на нові відкриття.

1.3 Постановка задачі

Розробка додатку для оцінки безпеки веб-додатків є завданням, що водночас захоплює складністю та вимагає ґрунтовного підходу. У сучасному

інформаційному середовищі, де вразливості можуть мати катастрофічні наслідки, потреба у засобах, здатних швидко і точно оцінити рівень захищеності, є невідкладною. Ідея створення такого додатку ґрунтується не лише на теоретичних засадах, але й на практичних викликах, з якими щодня стикаються розробники та адміністратори.

Не можна ігнорувати той факт, що існують численні автоматизовані системи тестування, проте більшість із них не забезпечує адаптивності під специфіку кожного окремого проєкту. Постає завдання створити додаток, який буде здатний враховувати унікальні особливості кожного веб-додатку, що тестується.

Основний акцент у постановці задачі робиться на гнучкості, масштабованості та здатності інтеграції з уже існуючими інструментами. Зокрема, мова йде про необхідність реалізації методів статичного аналізу, що дозволяє виявляти потенційні загрози ще на стадії розробки коду, та динамічного аналізу, який забезпечує тестування системи під час її роботи. Система повинна підтримувати перевірку найпоширеніших вразливостей, таких як SQL-ін'єкції, XSS-атаки, вразливості авторизації та інші, з врахуванням OWASP Top 10.

Крім цього, особливу увагу слід приділити ергономіці користувацького інтерфейсу. Додаток повинен бути не лише функціональним, але й інтуїтивно зрозумілим для різних категорій користувачів – від новачків до досвідчених фахівців з кібербезпеки. Наприклад, візуалізація результатів у формі графіків і діаграм може значно полегшити сприйняття отриманих даних і сприяти швидшому ухваленню рішень.

У межах реалізації такого проєкту важливо також враховувати питання конфіденційності. Тестування веб-додатків часто передбачає роботу з чутливими даними, що ставить високі вимоги до захисту самих інструментів тестування. Реалізація відповідних заходів, таких як шифрування даних і обмеження доступу до результатів перевірок, є обов'язковою складовою проєкту.

Таким чином, завдання створення додатку для оцінки безпеки веб-додатків є багатоаспектним і потребує інтегрованого підходу. Це не лише технічне завдання, а й виклик у площині розуміння потреб користувачів і забезпечення максимальної ефективності в сучасному динамічному середовищі.

2 ПРОЕКТУВАННЯ ЗАСТОСУНКУ ДЛЯ ОЦІНКИ БЕЗПЕКИ ВЕБ-ДОДАТКІВ

2.1 Структура застосунку

Структура застосунку представлена на рисунку 2.1.

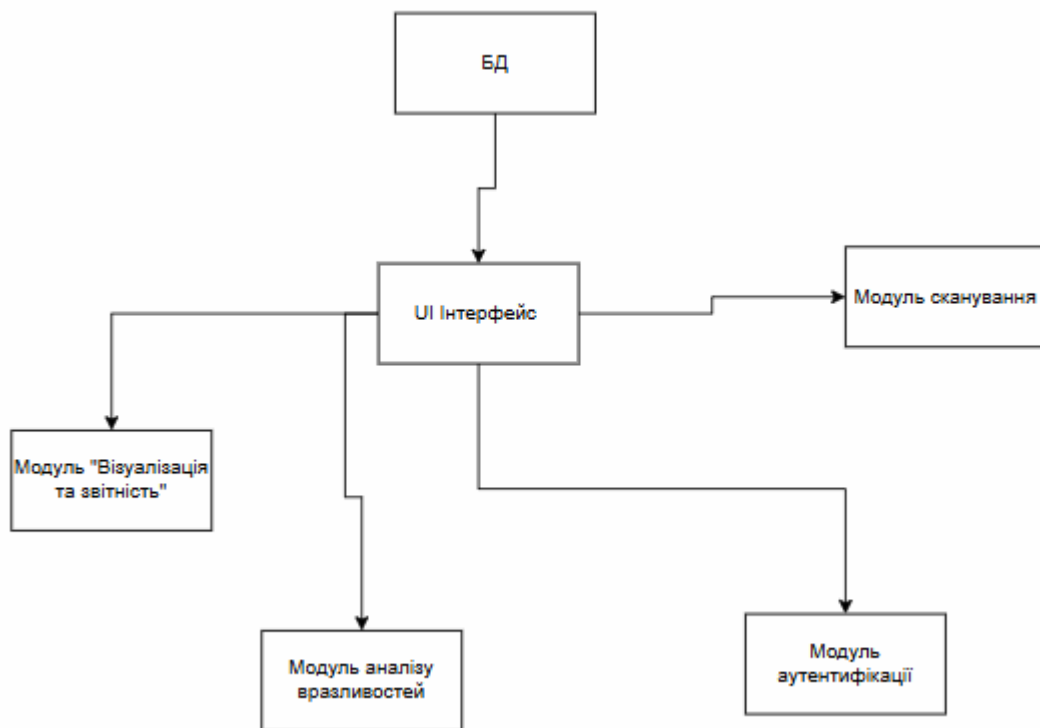


Рис.2.1 Структура застосунку для оцінки безпеки веб-додатків

UI інтерфейс у представленій схемі виконує провідну роль у забезпеченні доступу до всіх функціональних компонентів та формуванні комфортних умов взаємодії між користувачем і внутрішніми модулями. Він уможливорює відображення результатів, ініціювання процесів сканування, налаштування параметрів аутентифікації та належне керування обліковими записами фахівців, водночас слугуючи відправною точкою для запуску аналізу вразливостей і подальшої візуалізації та формування звітності. Через інтерфейс здійснюється введення даних, які надходять від модуля аутентифікації, що відповідає за верифікацію облікових записів, а також надає можливість редагування прав

доступу та персональних налаштувань, проте детальніше керування ролями відбувається на окремому рівні системи з огляду на безпекові вимоги.

Застосування інтерактивних елементів та чіткої ієрархії відображення даних дає змогу спростити навігацію та прискорити прийняття рішень, оскільки користувачі можуть оперативно переглядати підсумки сканувань та отримувати аналітику щодо оцінки ризиків. Важливо, що графічна оболонка безпосередньо пов'язана з базою даних, куди надходять сукупні результати та метадані. Завдяки цій взаємодії стає можливою миттєва актуалізація інформації та одночасна передача відповідних сигналів до модуля візуалізації й формування звітності. Структура розгортається таким чином, щоб інтерфейсна частина надсилала запити до модуля сканування в разі потреби ініціювати перевірку безпеки, а також отримувала проаналізовані дані з модуля аналізу вразливостей, формуючи єдиний робочий простір для відстеження загального стану захищеності відповідних веб-рішень. Такий підхід забезпечує логічну цілісність, полегшує масштабування та надає користувачам безперешкодний доступ до ключових компонентів системи без потреби обходити проміжні рівні, що позитивно впливає на ефективність процесів і загальну продуктивність інструменту.

База даних у цьому контексті посідає центральне місце, оскільки в ній зберігаються результати сканування, журнали діяльності користувачів та процесів, а також пов'язані з безпекою метадані. Водночас вона вміщує інформацію про налаштування застосунку, перелік облікових записів, зашифровані паролі та інші відомості, що потребують надійного зберігання й швидкого доступу. Раціональним вважається впровадження реляційної структури, що передбачає уніфіковані таблиці для розмежування різних типів даних, проте можливе застосування гібридного підходу, де окремі об'єкти або тимчасові результати аналізу можуть розміщуватися у нереляційних сховищах, аби спростити обробку великих або напівструктурованих обсягів. Фізична реалізація бази даних передбачає використання реплікації чи кластеризації задля підвищення відмовостійкості та унеможливлення збоїв унаслідок

збільшення обсягів записів або пікових навантажень. Під час проєктування особлива увага приділяється розмежуванню доступу, створенню плану бекапів і застосуванню криптографічного захисту для найважливіших полів, що дозволяє дотримуватися стандартів безпеки й зберігати цілісність даних навіть за умови виникнення непередбачуваних ситуацій. Крім того, доцільним є застосування аудитних механізмів, аби відстежувати всі операції зі створення, зміни або видалення записів, що дає можливість ефективно виявляти потенційні аномалії та оперативно реагувати на будь-які загрози або недотримання регламентованих процедур. Подібна архітектура бази даних надає підґрунтя для конструктивної взаємодії з іншими складниками системи, оскільки всі модулі, включно з аналітичним блоком і компонентою для візуалізації результатів, використовують однакові джерела інформації та уніфіковані методи доступу, що суттєво підвищує продуктивність і полегшує підтримку цілісного функціонування програмного комплексу.

Механізм перевірки достовірності користувачів у цій системі реалізується за допомогою спеціалізованої підсистеми, що відповідає за ідентифікацію та верифікацію облікових записів. Вона побудована з урахуванням суворих вимог до захисту інформації, зокрема застосуванням криптографічних алгоритмів для зберігання та передавання чутливих даних, а також гарантуванням конфіденційності персональних відомостей. Передбачена можливість реалізації як звичайного сценарію з логіном і паролем, так і багатофакторної перевірки, коли для доступу необхідно надати додаткові коди підтвердження, отримані за допомогою мобільних додатків або апаратних токенів. Такий підхід дає змогу істотно підвищити рівень надійності та захистити систему від несанкціонованого проникнення, водночас зберігаючи зручність користування та зрозумілі механізми відновлення доступу. Сама структурна організація процесів верифікації передбачає інтеграцію з централізованим реєстром облікових записів, розташованим у сховищі даних, де зберігаються всі ідентифікатори, криптографічні хеші паролів, ролі користувачів і допоміжна інформація. Після успішної ідентифікації користувача створюються сесійні

маркери, що передаються для перевірки кожного наступного запиту, забезпечуючи тим самим контроль прав доступу й унеможливаючи використання викрадених облікових відомостей без чинного авторизаційного токена. Важливим доповненням до описаних заходів є регулярний аудит логів, які відображають усі спроби входу або зміни параметрів під час роботи, а також застосування протоколів безпечного шифрування трафіку для передачі пов'язаних із підтвердженням особи повідомлень, що дає змогу мінімізувати ризики перехоплення або підроблення авторизаційних даних у загальному інформаційному контурі.

Модуль аналізу вразливостей покликаний глибоко досліджувати та оцінювати дані, зібрані внаслідок перевірок безпеки, а також визначати рівень критичності знайдених недоліків, які можуть негативно вплинути на стабільність або конфіденційність веб-застосунків. Він функціонує як центральна ланка між процесом сканування, що надає вихідні відомості про потенційні проблеми, та базою даних, куди повертає структуровану інформацію щодо типів і ступеня небезпеки виявлених загроз. У цьому підрозділі реалізуються різні підходи до систематизації даних, зокрема розрахунок показників ризику з використанням загальноприйнятих методологій і фреймворків, включно з CVSS чи власними розширеннями, які враховують особливості предметної галузі або специфіку компонентів, що перевіряються. Концепція програмного модуля полягає у проведенні комплексної обробки матеріалів, отриманих після попереднього відсіювання інформаційного шуму, з подальшою класифікацією загроз, визначенням першочергових заходів із усунення найнебезпечніших вразливостей, а також наданням прогнозів стосовно ймовірних наслідків, якщо відповідні проблеми не будуть виправлені вчасно. Усередині компонента часто застосовуються машинно-навчальні механізми для ранжування загроз, що уможливають гнучке реагування на зміну оточення та поява нових типів недоліків, оскільки системі надається можливість самостійно виявляти приховані зв'язки та закономірності, не завжди помітні при традиційних ручних підходах. Зрештою, результати аналізу

надходять у загальну структуру сховища та візуалізуються в призначеному для користувача інтерфейсі, що дозволяє задіяти центральну базу знань для подальшої оптимізації процесу виявлення і виправлення загроз, а також для кращого планування коригувальних робіт і цілеспрямованого розвитку системи безпеки.

Модуль візуалізації та звітності забезпечує зручне відображення результатів роботи всіх компонентів системи, надаючи можливість оперативно сприймати та інтерпретувати виявлені проблеми у сфері безпеки. Він узгоджується із центральною базою даних, звідки завантажує результати сканування, проаналізовані ризики, а також визначені пріоритети виправлення недоліків. За рахунок використання інтерактивних графіків і графічних елементів користувачі отримують узагальнену картину стану захищеності веб-додатків, що допомагає приймати виважені рішення щодо подальшої оптимізації процесів або зміни налаштувань інструментів оцінки. Завдяки цьому блоку в системі з'являються зрозумілі та структуровані підсумки, які можна виводити у формі аналітичних панелей або надавати у вигляді сформованих документів. Крім цього, функціональність модулю передбачає можливість збереження згенерованих підсумків у кількох поширених форматах, що полегшує поширення інформації серед членів команди безпеки та інших зацікавлених осіб. Розробники, відповідальні за обслуговування веб-рішень, отримують змогу своєчасно оцінити вплив потенційних ризиків, коригувати стратегії захисту та слідкувати за динамікою змін, не витрачаючи зайвих ресурсів на опрацювання первинних сканувальних звітів. Така інтеграція візуальних та звітних можливостей з іншими компонентами проєкту покращує координацію зусиль команди, допомагає у виявленні критичних точок, а також створює передумови для запровадження найкращих практик у сфері інформаційної безпеки.

2.2 Вибір та обґрунтування інструментальних засобів

Python посідає чільне місце у сфері розроблення програмного забезпечення, орієнтованого на ідентифікацію вразливостей та перевірку захищеності веб-додатків, перш за все завдяки своїй зручній синтаксичній структурі та розвинутому набору стандартних модулів. Використання цієї мови в контексті проектування системи дає змогу оперативно створювати прототипи окремих компонентів, а також легко переходити від експериментального рівня до промислового масштабування. Такий підхід зручний для розробників, які потребують швидкого внесення змін у логіку, додавання нових функціональних можливостей чи інтеграції зовнішніх сервісів, зокрема тих, що виконують автоматизоване сканування та аналіз отриманих результатів. Платформа Python вирізняється численними пакетами, здатними виконувати завдання різної складності, охоплюючи як звичайні запити до мережових інтерфейсів, так і вивантаження результатів у базу даних та паралельну обробку відомостей з декількох джерел[30]. Мова пропонує гнучку модель взаємодії між модулями, завдяки чому розробник може при потребі використовувати фреймворки для роботи з чергами повідомлень, управління асинхронними процесами чи підключення багатовимірних бібліотек для математичних обчислень. На практиці це означає, що під час розроблення компонентів, які відповідають за аналіз метрик ризику, сканування відкритих портів чи виявлення уразливостей типу SQL-ін'єкцій, не виникає жодних суттєвих перешкод у впровадженні необхідної функціональності. Окрім того, простий механізм управління залежностями через стандартні утиліти дає можливість керувати версіями зовнішніх пакетів та забезпечувати сумісність із різноманітними конфігураціями середовища. Усе це створює середовище, де можна безперервно впроваджувати нові ідеї та модернізувати вже готові модулі, залишаючись у рамках єдиної парадигми розробки та не витрачаючи надмірні ресурси на інтеграцію чи налаштування програмних компонентів.

Django розглядається як високорівневий веб-фреймворк, що суттєво спрощує розроблення програмних інструментів, орієнтованих на автоматизовані перевірки безпеки. Завдяки вбудованій структурі, яка охоплює

керування URL-шляхами, оброблення запитів, роботу з базами даних і механізми аутентифікації, даний підхід уможливорює швидке створення комплексних веб-застосунків без потреби писати надто багато супровідного коду. Це рішення має власну ORM-систему, що спрощує взаємодію з будь-якою популярною системою керування даними та уніфікує операції читання й запису інформації, включно з маніпулюванням файлами налаштувань і збереженням результатів сканування. Крім того, платформа пропонує модульну архітектуру, яка дає змогу відокремлювати функціональність, відповідальну за управління користувачами, контролі доступу та ініціювання процедур сканування. Завдяки вбудованим механізмам валідації вхідних даних та належному розмежуванню рівнів логіки і представлення створюється середовище, де легко відстежувати, аналізувати та локалізувати потенційні недоліки. Наявність адаптивної адмін-консолі забезпечує візуальне адміністрування ключових процесів, що особливо корисно в ситуаціях, коли треба оперативно оновлювати конфігурації без заглиблення в кодову базу. Підтримка масштабованості в комплексі з продуманою системою кешування дозволяє адаптувати інструмент до умов зростаючої кількості об'єктів для перевірки, а також попереджати надмірне навантаження при одночасному запуску кількох сканувань. Така сукупність можливостей гарантує, що процес розробки рішень для оцінки ступеня безпеки веб-додатків зосереджується передусім на оптимізації та аналізі виявлених ризиків, уникаючи надмірних трудовитрат на налаштування інфраструктури.

2.3 Вибір та обґрунтування бази даних

SQLite в контексті проектування програмного забезпечення для оцінки вразливостей веб-додатків вирізняється своєю здатністю функціонувати як повноцінна реляційна система управління даними без вимоги встановлення окремого серверного середовища, оскільки зберігає всю необхідну інформацію безпосередньо у файлі на локальному диску. Така особливість спрощує процес розгортання, адже відпадає потреба конфігурувати сервер і керувати

численними обліковими записами або правами доступу, що в свою чергу знижує обсяг додаткової інфраструктурної підтримки[31]. Крім того, цей підхід позитивно впливає на мобільність рішень, бо розробники можуть переносити базу даних разом із файлом програми й запускати її на різних платформах без ускладнень. Структура вбудованої бібліотеки підтримує транзакції з дотриманням принципів цілісності та атомарності, забезпечуючи правильність паралельного доступу в умовах, коли декілька компонентів системи можуть одночасно звертатися до одного джерела даних. Завдяки цьому вдається ефективно розмежовувати операції запису та читання, що є особливо важливим під час зберігання проміжних результатів аналізу вразливостей або формування підсумкових звітів у реальному часі. Інтеграція з мовою програмування Python відбувається за допомогою стандартних модулів, доступних безпосередньо “з коробки”, тому не виникає необхідності встановлювати додаткові драйвери чи залежності. Такий спосіб взаємодії дозволяє будувати оптимізовані запити до бази, виконувати сортування, фільтрацію та агрегацію великих масивів даних без суттєвих витрат ресурсів, а також гарантує постійну узгодженість і цілісність при додаванні, зміні чи видаленні записів. Подібне рішення чудово підходить як для побудови швидких прототипів з невеликими обсягами інформації, так і для розроблення більш комплексних програм, де робочі навантаження та масштаб можуть поступово зростати, проте потреба у встановленні потужнішого серверного середовища лишається необов’язковою, якщо технічні вимоги не перевищують можливостей вбудованої архітектури. Таким чином, застосування цього підходу пропонує вагому гнучкість у процесі реалізації інструменту для виявлення вразливостей, створює передумови для економії інфраструктурних витрат і спрощує підтримку розробки, оскільки системі не потрібно витрачати значні зусилля на адміністрування чи керування розподіленими вузлами.

2.4 Алгоритм роботи застосунку

Алгоритм попередньої обробки запиту представлено на рисунку 2.2.

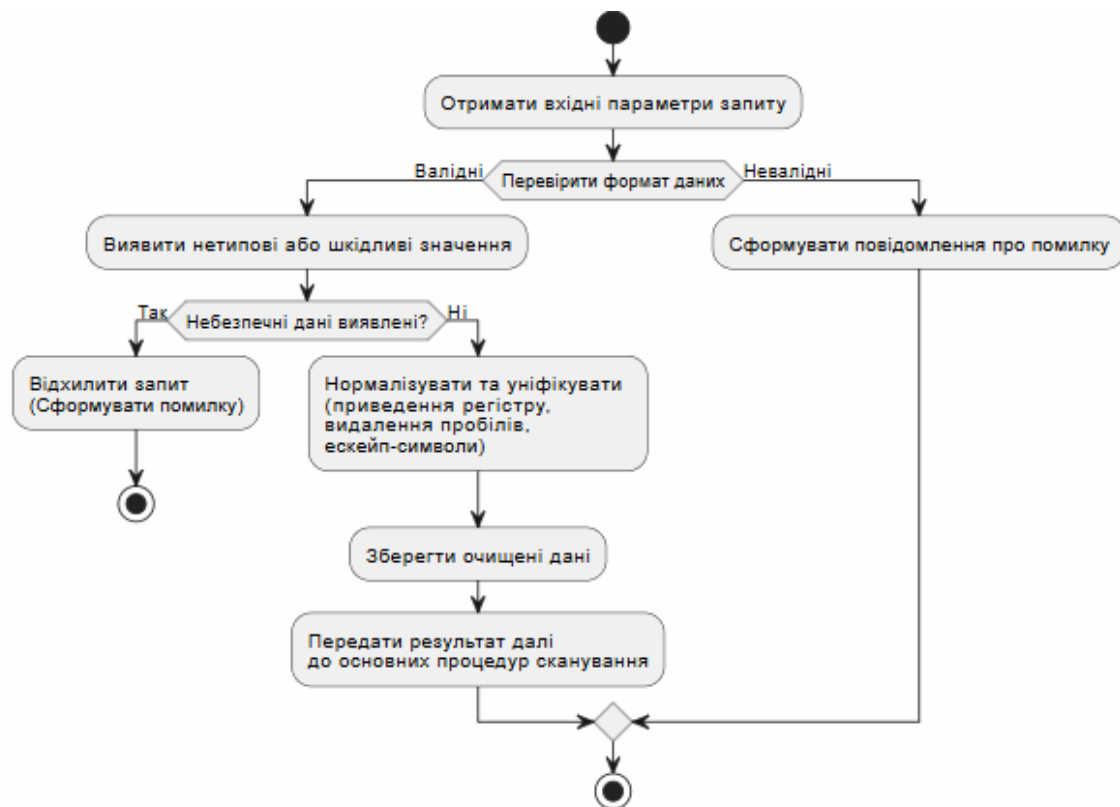


Рис.2.2 Алгоритм фільтрації запитів

Алгоритм передбачає послідовне читання і аналіз отриманих параметрів, метою чого є виявлення відхилень від прийнятої специфікації та формату, а також вчасне фільтрування шкідливих елементів. Спочатку система виконує попередню валідацію, визначаючи відповідність отриманих значень очікуваним критеріям, що дає змогу завчасно відсіяти неприпустимі або випадкові вхідні відомості. У межах цього етапу проводиться розпізнавання формату даних з урахуванням типу, довжини і передбаченої структури. Коли параметри не відповідають встановленим умовам, ініціюється процес відхилення запиту, а сам користувач отримує повідомлення про помилку. Якщо ж проблема не виявлена, виконується додаткове дослідження даних із метою виявлення потенційно небезпечних патернів, нетипових символів чи послідовностей, що можуть свідчити про намагання вчинити атаки на зразок ін'єкцій. Під час виявлення сумнівних елементів алгоритм припиняє подальшу обробку та повертає відповідну помилку захисту, і лише за відсутності підозрілих патернів

алгоритм переходить до наступного етапу, зосередженого на коригуванні та уніфікації даних.

На заключному кроці виконується нормалізація допустимих значень, коли непотрібні пробіли, зайві символи чи різниці у регістрі системно усуваються, а відомості, що мають бути відображені в узгодженому вигляді, зберігають узгодженість у контексті подальшої обробки. Такий підхід уможливорює подальшу передачу очищеної та відфільтрованої інформації до основного механізму сканування з мінімальним ризиком включення шкідливого вмісту. Зрештою, реалізована процедура ретельно перевіряє всі критичні аспекти, забезпечує сумісність отриманих відомостей із внутрішніми вимогами безпеки й готує підґрунтя для надійного аналізу наступними компонентами системи.

Алгоритм сканування безпеки представлено на рисунку 2.3.

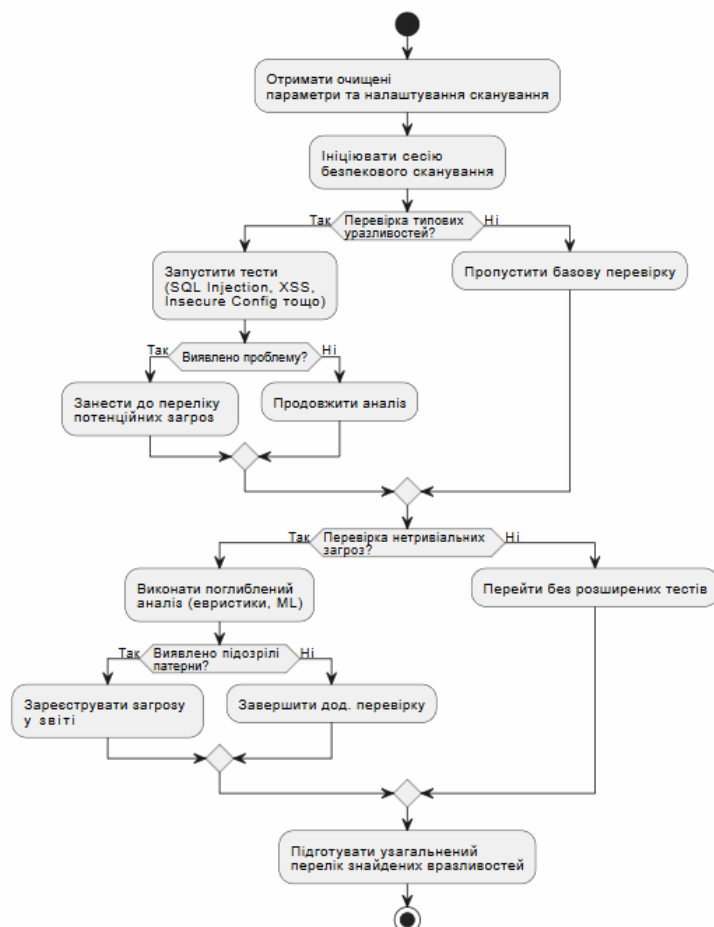


Рис.2.3 Алгоритм сканування

Алгоритм передбачає послідовні дії з дослідження цільового веб-додатка, що охоплюють як виявлення загальновідомих критичних точок, так і розпізнавання складніших аномалій. Спочатку відбувається підготовка робочого середовища, де визначаються вихідні налаштування, включаючи список адрес чи точок доступу та перелік базових тестів на поширені проблеми на кшталт SQL-ін'єкцій чи міжсайтового скриптингу. Після ініціалізації перевірки система послідовно генерує запити, змінюючи або модифікуючи вхідні дані з метою імітувати сценарії, які потенційно можуть бути використані зловмисником. На цьому етапі особливий наголос робиться на моніторинг відповідей сервера та аналіз їхньої структури, щоб виявити відхилення від норми, що зазвичай свідчить про існування вразливої ділянки або неправильної конфігурації. Зафіксовані ознаки слабких місць негайно заносяться до проміжної бази спостережень із зазначенням рівня пріоритетності та супутніх деталей, включно зі сценарієм, за якого було отримано підозрілу реакцію.

Після завершення первинних випробувань алгоритм переходить до детальнішого дослідження, зосередженого на нетривіальних загрозах, таких як логічні помилки в реалізації бізнес-процесів, уразливість до розподілених атак або прихованих механізмів обходу автентифікації. На цьому рівні доцільним стає залучення евристичних методів чи моделей машинного навчання, які дають змогу виявляти закономірності у трафіку й відповідях веб-додатка, що не належать до типових патернів уразливостей. Виявлені під час глибинного аналізу проблеми одразу додаються до зведеного переліку, де поєднуються з інформацією, зібраною раніше, утворюючи вичерпну базу знань про небезпечні елементи цільового ресурсу. Завершальним кроком є формування звіту, в якому кожен виявлений недолік супроводжується описом імовірних наслідків, орієнтовними рекомендаціями стосовно усунення та індикаторами пріоритету, що залежить від потенційного впливу на цілісність або доступність веб-системи. Такий узагальнений підхід дає змогу отримати комплексну картину про наявні недоліки безпеки та визначити подальші кроки щодо підвищення надійності програмного продукту.

Алгоритм оцінки й пріоритизації знайдених недоліків представлено на рисунку 2.4.

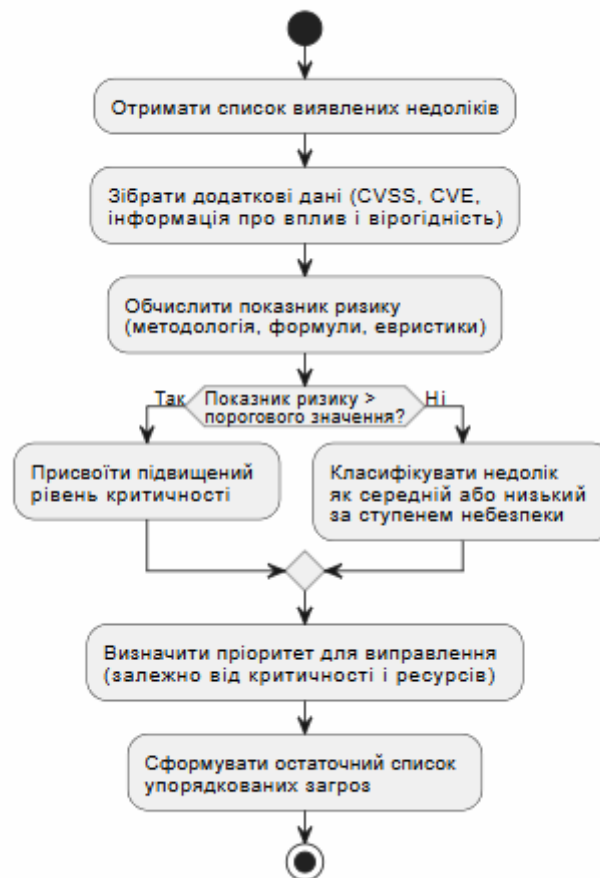


Рис.2.4 Алгоритм оцінки й пріоритизації знайдених недоліків

Алгоритм оцінки і пріоритизації знайдених недоліків, що базується на ризик-орієнтованих метриках і враховує критичність кожної виявленої проблеми, починається з формування первинного переліку вразливостей, які було ідентифіковано в результаті проведеного сканування. Спочатку до кожного з пунктів цього списку додаються супутні дані, де відображається інформація про потенційний вплив на систему, вірогідність експлуатації, відомі глобальні ідентифікатори на зразок CVE та поточна оцінка за обраною методологією, наприклад CVSS. Наступним кроком є обчислення інтегрального показника ризику за допомогою формул чи евристичних правил, що враховують бали загрозливості, можливі сценарії атак та показники поширеності. Якщо отримане значення перевищує визначений пороговий рівень, недолікові присвоюється статус підвищеної критичності, який

сигналізує про першочергову необхідність виправлення. Якщо ж показник лишається на помірному чи низькому рівні, вразливості приписується відносно нижчий ступінь небезпеки. Після цього система підбирає відповідні пріоритети з урахуванням ресурсів, що доступні для виправлення, а також враховує суміжні чинники, як-от час, необхідний для внесення змін, і можливі наслідки у разі ненегайної реакції. Результатом роботи алгоритму виступає впорядкований реєстр потенційних проблем, де найнебезпечніші випадки розміщено у верхній частині, що полегшує команді розроблення й безпеки зосередитися на найкритичніших загрозах, не втрачаючи з поля зору менш значущі недоліки.

Алгоритм кореляції та структурування результатів представлено на рисунку 2.5.

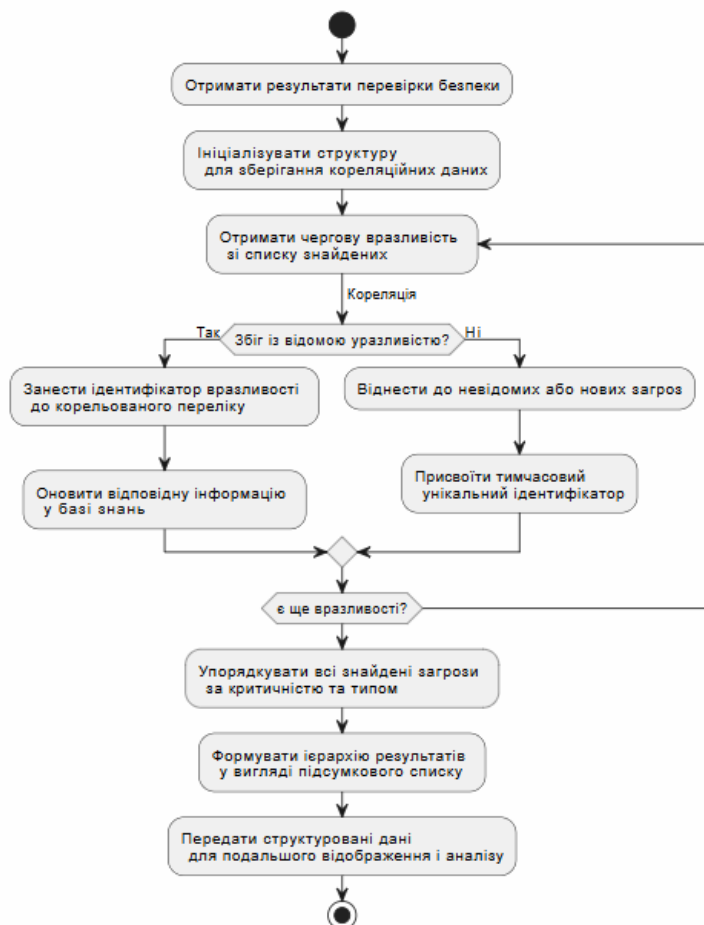


Рис.2.5 Алгоритм кореляції та структурування

Алгоритм зосереджується на порівнянні знайдених загроз із наявними шаблонами чи довідковою інформацією, що зберігається у спеціалізованому сховищі, аби виявити їхню подібність до відомих типових прикладів, а також

однозначно визначити ступінь серйозності. Спочатку із загального переліку зібраних під час перевірки даних послідовно вилучаються записані випадки вразливостей. Для кожного знайденого елемента відбувається спроба знайти його відповідник у базі знань, що містить ідентифікатори відомих недоліків та їхні характерні риси. Якщо виявляється збіг з існуючим описом, алгоритм відзначає його як повторний і доповнює інформацію супутніми деталями, наприклад індикаторами критичності та можливими методами усунення проблеми. У протилежному разі, коли збіг відсутній, нова потенційна небезпека позначається унікальним ідентифікатором і додається до списку непевних або поки що не класифікованих випадків для подальшого поглибленого аналізу. Таким чином відбувається формування структурованого набору знайдених загроз, поділених на відомі та унікальні. На завершальному кроці алгоритм виконує впорядкування усіх виявлених проблем за рівнем ризику, типом чи іншим визначеним параметром, складаючи єдиний підсумок у вигляді зручного й узгодженого переліку з даними про джерело, часові позначки, статус та рекомендації. Такий підхід надає можливість не лише швидко зрозуміти пріоритети усунення, а й побачити загальну структуру проблем безпеки, що полегшує формування подальших кроків з нейтралізації або профілактики повторних випадків.

Алгоритм формування підсумкової звітності та візуалізації представлено на рисунку 2.6.

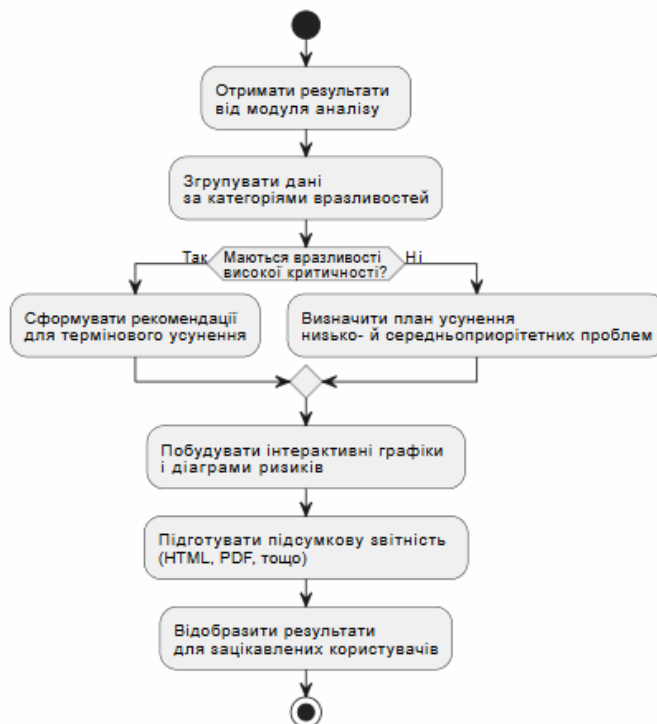


Рис.2.6 Алгоритм формування звітності та візуалізації

Алгоритм передбачає послідовне надходження результатів від модуля аналізу, що містять систематизовані відомості про виявлені вразливості різного рівня критичності. Спочатку виконується збирання та структурування цих даних у зручному для обробки форматі, який відображає кожен недолік у контексті потенційного впливу на загальний стан безпеки. У наступному кроці інформація групується за ключовими ознаками, такими як тип загрози, сфера впливу чи визначена пріоритизація, аби полегшити подальшу обробку та виявити закономірності у розподілі ризиків. Якщо серед отриманих вразливостей виокремлюються такі, що несуть високий рівень критичності, алгоритм ініціює формування невідкладних рекомендацій для усунення цих проблем, указуючи конкретні заходи, які дають змогу мінімізувати наявні загрози. У ситуаціях, коли в системі переважають недоліки середнього чи низького рівня, формується план дій, зорієнтований на поступове розв'язання відповідних уразливостей.

Після завершення етапу розподілу й уточнення першочергових кроків відбувається візуалізація отриманої статистики через інтерактивні графіки, що

відображають динаміку та співвідношення різних видів уразливостей. На цьому етапі важливо використовувати інформаційно насичені, проте зручні для швидкого сприйняття схеми, які допомагають визначати тенденції та осмислювати взаємозв'язок між виявленими вадами. Згодом формується єдиний звіт, що містить узагальнену аналітику, кількісні показники щодо стану безпеки та надає рекомендації стосовно пріоритетних та другорядних заходів. Доцільним є використання універсальних форматів файлів, аби полегшити розповсюдження цих висновків серед залучених фахівців та зберегти однаковий вигляд звіту на різних пристроях. На завершальній стадії результати надсилаються відповідальним особам або виводяться у веб-інтерфейсі, де зацікавлені користувачі отримують змогу проаналізувати повну картину, обговорити нагальні завдання й узгодити подальшу стратегію вдосконалення безпекової політики.

3 РОЗРОБКА ТА ТЕСТУВАННЯ ЗАСТОСУНКУ ДЛЯ ОЦІНКИ БЕЗПЕКИ ВЕБ-ДОДАТКІВ

3.1 Розробка бази даних

Структура бази даних представлена на рисунку 3.1.

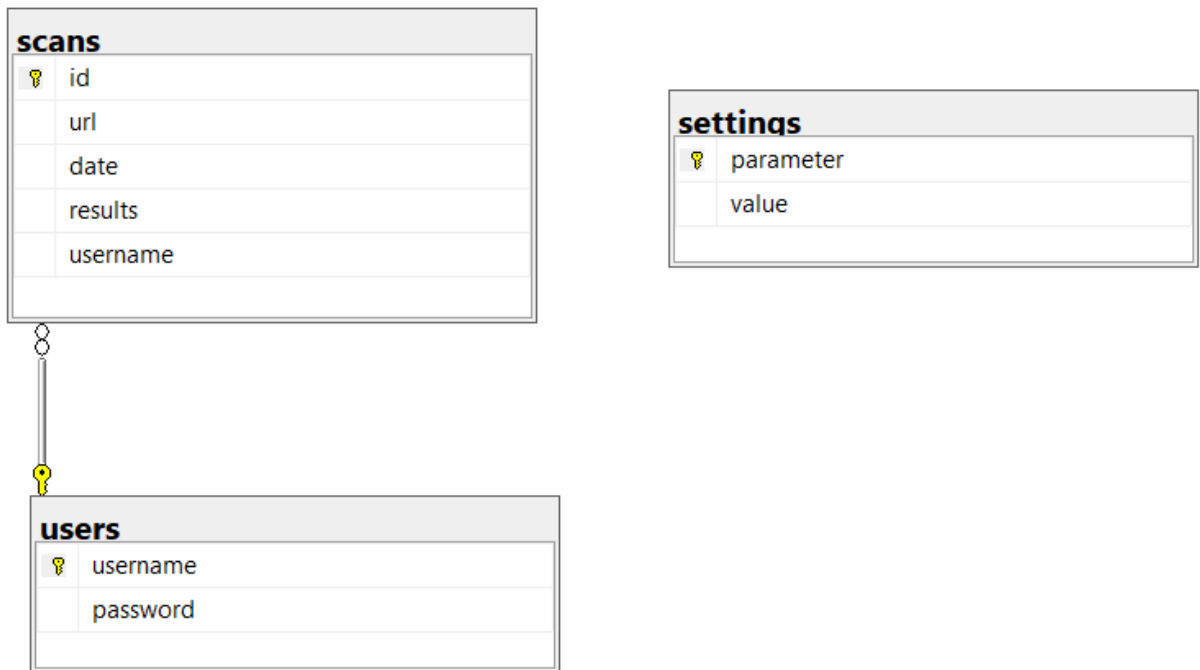


Рис.3.1 Структура бази даних застосунку

База даних, розроблена для застосування в системі оцінки безпеки веб-додатків, являє собою інтегроване сховище, яке організовано таким чином, що забезпечує збереження та обробку критично важливої інформації для роботи програми. У ній використовується підхід, що базується на розмежуванні функціональності шляхом створення кількох таблиць, кожна з яких має свою спеціалізовану роль у забезпеченні цілісності та зручності доступу до даних. Зокрема, таблиця, що містить інформацію про користувачів, розрахована на зберігання унікальних ідентифікаторів, які виконують роль первинного ключа, а також хешованих значень паролів, що гарантують захищеність облікових записів і запобігають несанкціонованому доступу. Наприклад, у визначенні стовпця для ідентифікації користувача використовується конструкція на зразок «username NVARCHAR(50) NOT NULL PRIMARY KEY», що забезпечує

обов'язковість заповнення та унікальність кожного запису. Друга важлива складова бази даних реалізована у вигляді таблиці, яка призначена для збереження результатів проведених сканувань; вона містить такі атрибути, як URL-адреса об'єкта перевірки, дата виконання операції, детальний звіт із виявленими вразливостями, а також поле, яке служить для встановлення зв'язку із таблицею користувачів, забезпечуючи тим самим відповідність кожного запису користувача, що ініціював сканування. Цей зв'язок встановлюється через зовнішній ключ, що гарантує референтну цілісність даних, адже значення, яке міститься у стовпці, напрямку посилається на відповідний запис у таблиці користувачів. Третя складова бази даних полягає у створенні таблиці налаштувань, де зберігаються параметри конфігурації застосунку, що впливають на поведінку мережових запитів і загальну продуктивність системи. У цій таблиці кожен параметр, наприклад, значення таймауту або специфікація користувацького агента, визначається як первинний ключ, що дозволяє оперативно отримувати відповідну конфігураційну інформацію. Таким чином, структура бази даних побудована таким чином, що забезпечує не лише ефективне зберігання і впорядкування даних, але й підтримує високий рівень безпеки та зручність управління інформацією, що є ключовим для належного функціонування системи оцінки безпеки веб-додатків.

3.2 Розробка інтерфейсу користувача

У розробці інтерфейсу користувача даного застосунку основну увагу приділено забезпеченню інтуїтивного та функціонально насиченого середовища, яке сприяє ефективній взаємодії користувача із системою аналізу безпеки веб-додатків. Концептуально, структура інтерфейсу формується із застосуванням декількох взаємопов'язаних вікон, де кожне з них відповідає окремому аспекту роботи програми, забезпечуючи логічний перехід від авторизації до проведення тестування, перегляду історії виконаних операцій, налаштувань параметрів системи та генерації звітів.

При запуску застосунку формується вікно авторизації, яке виступає в ролі першого бар'єру доступу. Це вікно містить елементи введення для логіна та паролю, що супроводжуються відповідними текстовими підказками. Синтаксично ці елементи реалізовано через виджети, схожі на звичайні текстові поля, які забезпечують зручність введення даних, що в подальшому використовуються для автентифікації. Наприклад, рядок коду, що ініціалізує поле для введення паролю, виглядає наступним чином:

```
self.ent_pass = ttk.Entry(frm, show="*")[32]
```

Цей підхід дозволяє забезпечити конфіденційність введених даних, що є важливим аспектом безпеки.

Після успішної авторизації користувачу надається доступ до головного вікна застосунку, яке структуровано у вигляді вкладок, що символізують різні модулі програми. Кожна вкладка органічно інтегрована у загальний інтерфейс, завдяки чому забезпечується єдність функціоналу та зручність перемикання між режимами роботи. Так, вкладка для сканування містить поле введення URL, кнопку для запуску процесу аналізу, індикатор виконання у вигляді прогрес-бару, а також текстову область для відображення послідовності результатів тестування[33]. У цьому контексті інтерфейс розроблено з використанням стилістичних засобів, що гарантують естетичну привабливість та зрозумілість. Кнопка для запуску тестування, наприклад, створена таким чином:

```
self.scan_btn = ttk.Button(frm, text="Запустити сканування",  
command=self.start_scan)
```

Цей елемент не тільки сигналізує про можливість ініціювання процесу, а й виконує функцію зворотного зв'язку, що відображається у відповідному індикаторі.

Особливу увагу приділено модулю перегляду історії виконаних сканувань, який реалізовано через спеціалізований компонент, що відображає дані у вигляді таблиці. Такий підхід дозволяє користувачу швидко орієнтуватися у результатах попередніх операцій, аналізувати збережені дані та

детально переглядати інформацію про кожен виконаний тест за допомогою розгорнутого вікна деталей. Завдяки інтеграції механізму подвійного клацання, користувач може за потребою відкрити додаткове вікно з розширеним описом результатів, що містить інформацію про дату сканування, URL-адресу та повний текст звіту. Цей функціонал реалізовано за допомогою наступного фрагмента коду:

```
self.history_tree.bind("<Double-1>", self.view_scan_detail)
```

Таким чином, взаємодія з історичними даними є як інтуїтивно зрозумілою, так і функціонально розширеною.

Інтерфейс також містить вкладку з налаштуваннями, яка надає можливість користувачу конфігурувати основні параметри застосунку, такі як значення таймауту та параметри користувацького агента, що безпосередньо впливають на поведінку мережевих запитів. Цей модуль реалізовано за допомогою комбінованого використання полів введення та кнопок для збереження змін, що гарантує збереження актуальних налаштувань у базі даних. Завдяки цьому користувач отримує можливість адаптувати систему до специфічних умов роботи, що значно підвищує ефективність використання застосунку.

Узагальнюючи, розробка інтерфейсу даного застосунку побудована на принципах інтуїтивності, функціональності та естетичної привабливості, що забезпечує зручність використання навіть для користувачів з обмеженим досвідом роботи з подібними системами. Використання сучасних бібліотек та гнучких засобів для побудови графічного інтерфейсу дозволяє реалізувати високорівневу взаємодію з програмою, забезпечуючи як оперативність роботи, так і глибокий аналіз результатів безпеки веб-додатків.

3.3 Тестування функціоналу застосунку

При запуску застосунку користувач отримає наступне вікно (рис.3.2).

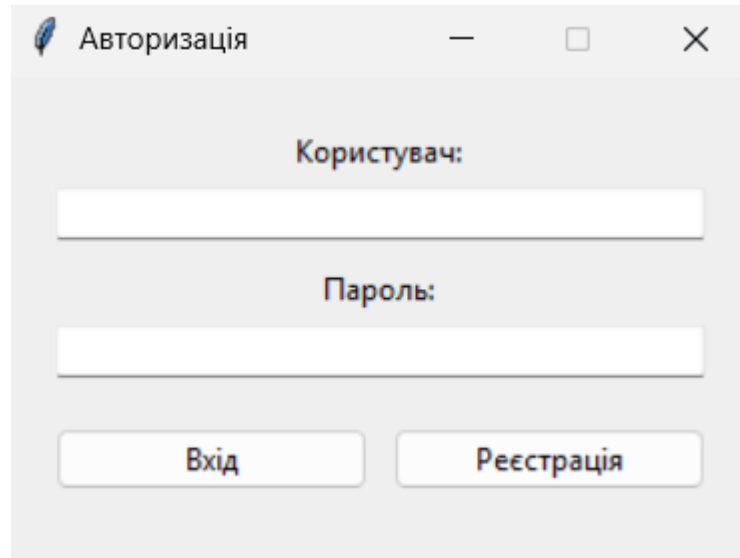


Рис.3.2 Авторизація користувача

При запуску застосунку користувачу пропонується авторизуватись в ньому. Якщо в користувача ще не має акаунту він може зареєструватись у ньому (рис.3.3).

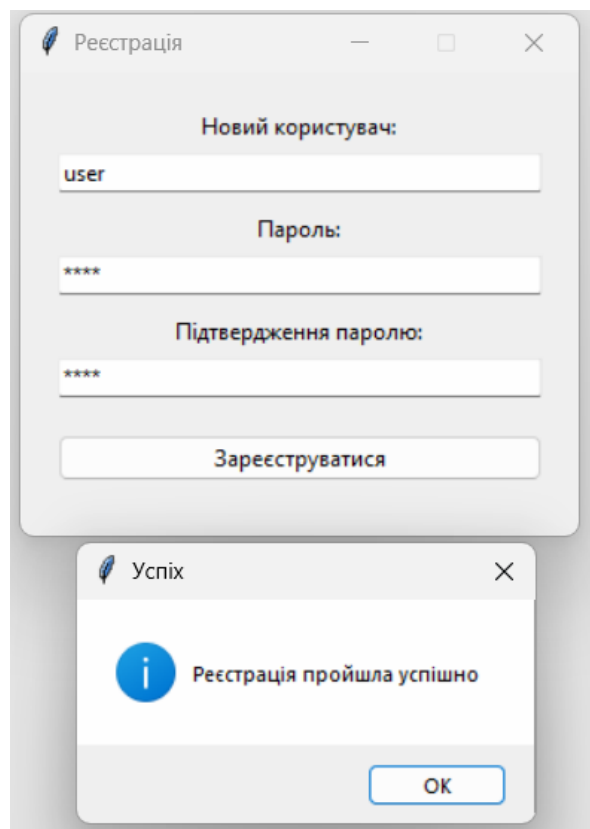


Рис.3.3 Реєстрація користувача

Реєстрація включає в себе створення імені користувача, паролю та підтвердження паролю. Після чого користувачу буде запропоновано відповідне повідомлення про успішну реєстрацію, яке підтверджує, що користувач

zareestrovaniy. Після чого його дані авторизації записуються в базу даних. Далі він зможе авторизуватись в застосунку з цими даними (рис.3.4).

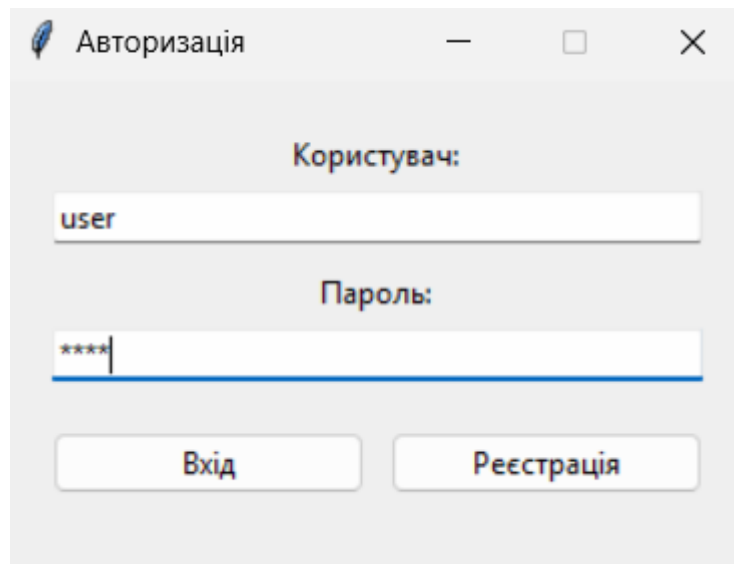


Рис.3.4 Авторизація користувача

Тестування авторизації продемонструвало, що вона працює коректно. Оскільки користувач зміг увійти у систему (рис.3.5).

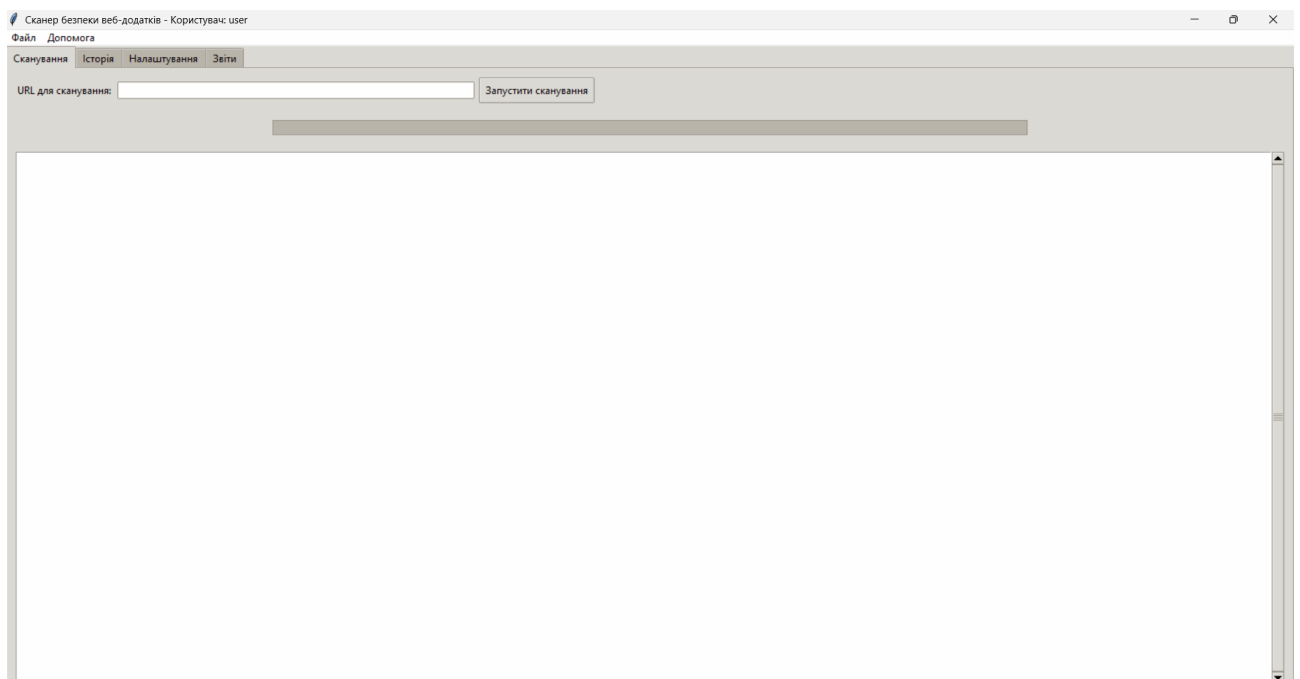


Рис.3.5 Інтерфейс застосунку

Після цього користувач зможе протестувати функціонал застосунку. Для початку тестується процес сканування веб-адреси (рис.3.6).

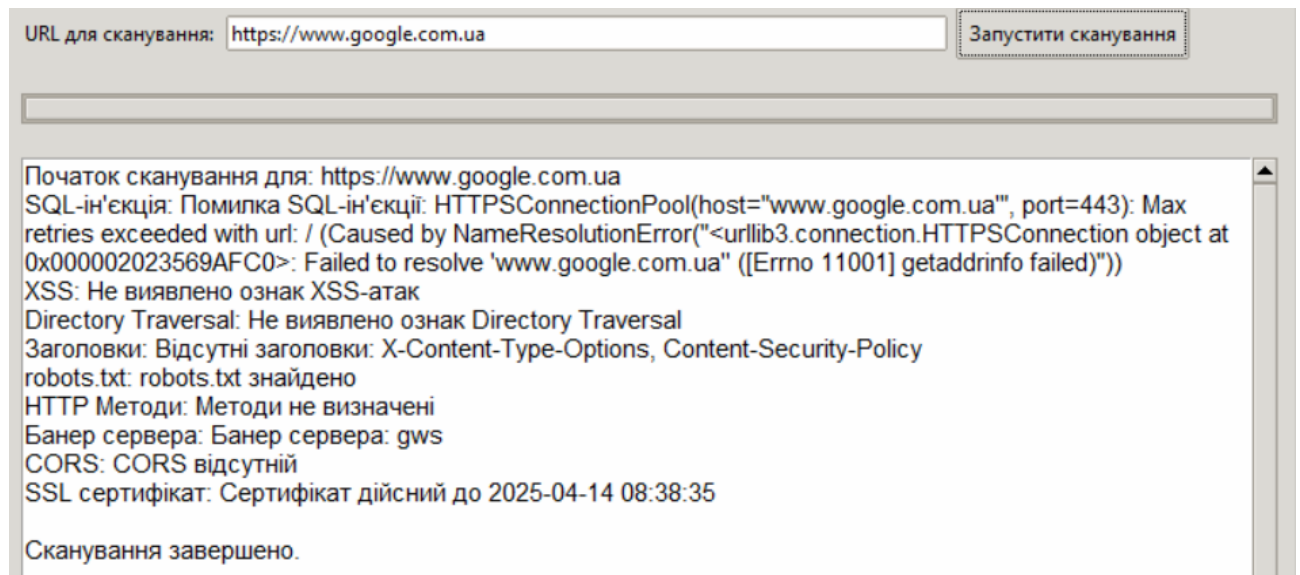


Рис.3.6 Процес сканування веб-адреси

Під час сканування вводиться вказана веб-адреса `https://www.google.com.ua`, після чого виконується послідовне встановлення з'єднання з ресурсом та аналіз основних вразливостей. На цьому етапі система здійснює спробу ініціювати запит через HTTPS з використанням бібліотек Python (згадується `urllib3`), проте з'являється повідомлення про неможливість розв'язати доменне ім'я `www.google.com.ua`, що свідчить про помилку `NameResolutionError` та, як наслідок, унеможливорює повну перевірку на ін'єкції SQL. Незважаючи на виявлену помилку під час з'єднання, сканер переходить до визначення потенційних XSS-вразливостей і встановлює, що ознак міжсайтового сценаріювання немає. Аналогічно перевіряється наявність загроз, пов'язаних з `Directory Traversal`, і відзначається відсутність відповідних небезпек. Потім здійснюється перевірка заголовків, де програма фіксує, що серед них відсутні `X-Content-Type-Options` та `Content-Security-Policy`, що може бути вказівкою на потребу додаткового налаштування безпеки. У цьому ж процесі шукається файл `robots.txt`, виявляється його наявність, але не відзначаються додаткові дані щодо його вмісту, а також робиться спроба ідентифікувати підтримувані HTTP-методи, однак методи не визначаються. Разом із тим відбувається сканування банера веб-сервера, який ідентифікується як `gws`, та перевірка CORS, котра фіксує відсутність політики `Cross-Origin Resource Sharing`. Наступним кроком проводиться оцінка SSL-сертифіката, що

демонструє його чинність до 2025-04-14 08:38:35. Завершується весь процес сканування повідомленням про успішне завершення, попри часткові ускладнення на початковому етапі через помилку DNS-розв’язання, що не дало змоги повністю перевірити ресурс на SQL-ін’єкції, проте дозволило отримати решту зазначених результатів у контексті безпеки та належних конфігурацій.

Таким чином, сканування веб-адреси пройшло успішно. Після успішного сканування адреси зберігаються в історію (рис.3.7 – 3.8).

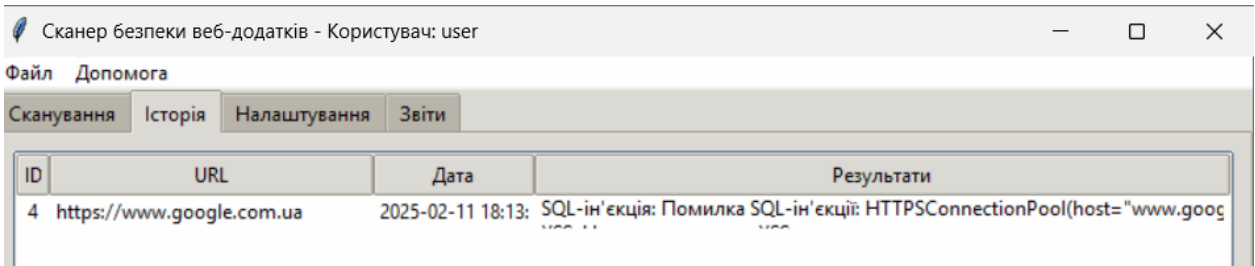


Рис.3.7 Історія сканування адрес

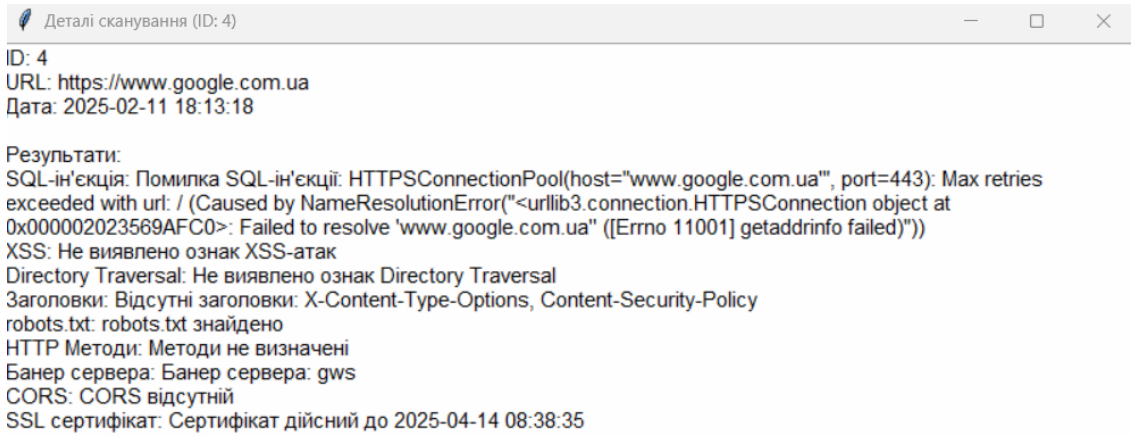


Рис.3.8 Деталі сканування

Історія сканувань адрес зберігається коректно. Історія сканувань буде різною для кожного акаунту (рис.3.9).

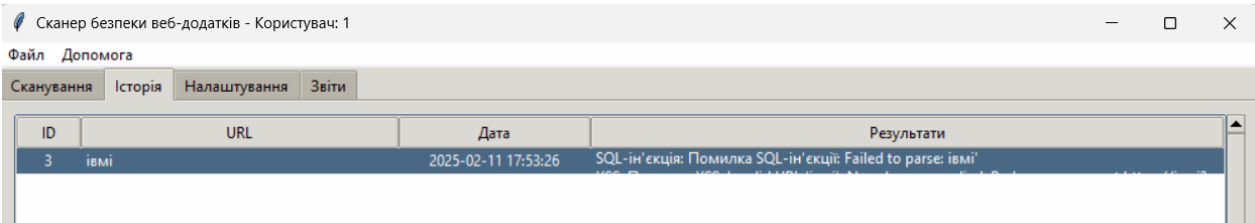


Рис.3.9 Історія для іншого акаунту

Тестування пройшло успішно, історія сканувань різна для кожного акаунту.

На рисунку 3.10 представлено результат експорту історії в файл.

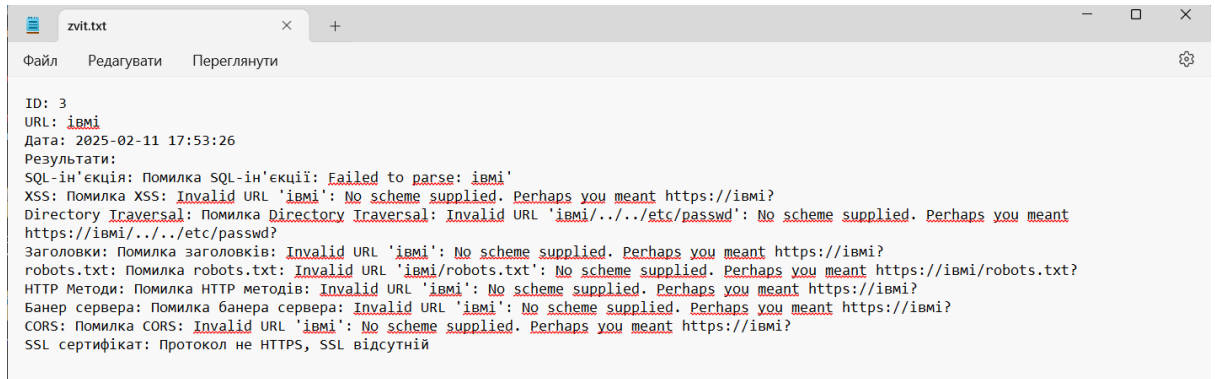


Рис.3.10 Звіт сканування адрес

Історія сканування успішно зберігається, тестування пройшло успішно. Також користувач може налаштувати параметри сканування (рис.3.11).

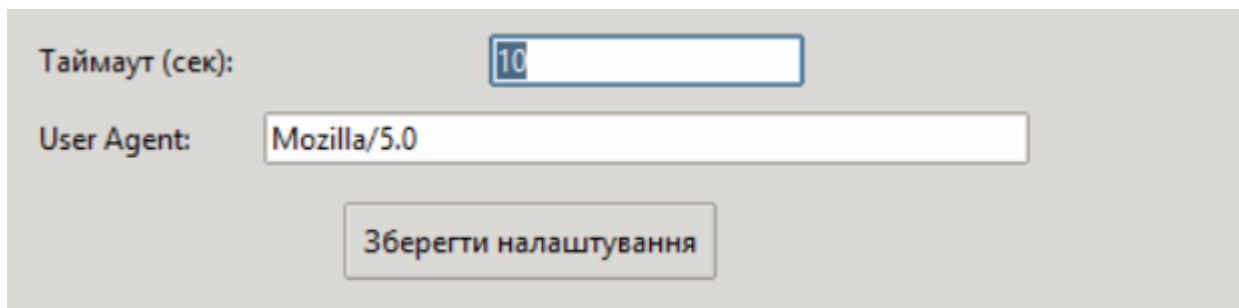


Рис.3.11 Налаштування сканування

У даному застосунку налаштування виконують ключову роль у регулюванні поведінки мережевих запитів, що спрямовані на оцінку безпеки веб-додатків, адже вони дозволяють адаптувати роботу програми до конкретних умов мережевої інфраструктури та забезпечують оптимальне використання ресурсів системи. Значення параметра таймауту, яке, зокрема, встановлено на 10 секунд, регламентує максимальний проміжок часу, протягом якого очікується відповідь від серверів під час здійснення HTTP-запитів. Такий підхід дозволяє уникнути зависання процесу аналізу у випадку затримок у відповіді або при відсутності зв'язку із сервером, що є надзвичайно важливим з точки зору забезпечення стабільності та надійності роботи застосунку.

Цей механізм гарантує, що система не буде чекати надто довго на відповідь, що сприяє підвищенню ефективності сканування.

Інший аспект налаштувань полягає у конфігурації користувацького агента, що представлено рядком, подібним до "Mozilla/5.0 (Windows NT 10.0; Win64; x64)". Даний параметр виконує функцію емуляції поведінки сучасних

браузерів під час відправки запитів, що дозволяє застосунку не тільки збільшити сумісність із різними серверними системами, але й потенційно обійти деякі обмеження, накладені на автоматизовані запити. Використання такої емульованої ідентифікації забезпечує імітацію справжнього трафіку, що є важливим для коректного аналізу веб-ресурсів.

Таким чином, інтерфейс налаштувань є невід'ємною складовою системи, оскільки він дозволяє не лише налаштовувати параметри, що безпосередньо впливають на якість взаємодії з мережевими ресурсами, але й забезпечує адаптивність застосунку до змін у мережевому середовищі, що сприяє підвищенню якості проведення тестування безпеки веб-додатків. Кожен елемент конфігурації має своє значення, яке впливає на загальну продуктивність системи, адже правильне встановлення таймауту забезпечує своєчасне завершення операцій, а використання актуального користувацького агента дозволяє емулювати реальні умови роботи браузера, що в сукупності сприяє досягненню високого рівня надійності та ефективності застосунку.

ВИСНОВКИ

Розроблений застосунок для оцінки безпеки веб-додатків продемонстрував свою ефективність у виявленні вразливостей, що можуть становити загрозу для інформаційних систем. Впродовж роботи було реалізовано систему, яка дозволяє здійснювати комплексний аналіз рівня безпеки за допомогою інтегрованих механізмів сканування, класифікації загроз та генерації звітності. Архітектурна побудова застосунку забезпечила узгоджене функціонування всіх його складових, дозволяючи користувачам взаємодіяти з інструментами аналізу через зручний графічний інтерфейс.

Інтерфейс користувача, розроблений у межах цього проєкту, відіграв ключову роль у забезпеченні доступу до основних функцій системи. Його структура передбачала панель управління, де користувачі могли вводити вихідні дані для перевірки безпеки веб-додатків, ініціювати процедури сканування, переглядати отримані результати та отримувати детальні рекомендації щодо усунення виявлених загроз. Використання інтерактивних елементів та адаптивної візуалізації сприяло підвищенню зручності роботи із системою та забезпечувало швидкий доступ до необхідної інформації. Крім того, графічний інтерфейс був розроблений таким чином, щоб спростити аналіз отриманих результатів завдяки інтегрованим графікам, таблицям і аналітичним індикаторам, що дозволяли оцінювати рівень ризику в реальному часі.

Центральним компонентом застосунку став модуль сканування, який забезпечував автоматичне виявлення вразливостей на основі заданих критеріїв. Він використовував алгоритми перевірки на поширені типи атак, зокрема SQL-ін'єкції, XSS-атаки, уразливості у механізмах аутентифікації та некоректне конфігурування серверних середовищ. Робота цього модуля ґрунтувалася на принципах послідовного аналізу HTTP-запитів і відповідей, що дозволяло виявляти як відомі загрози, так і потенційні аномалії, які могли свідчити про наявність невідомих вразливостей. У процесі тестування застосунку було підтверджено його здатність адаптивно реагувати на особливості

перевірюваних веб-додатків, що сприяло підвищенню ефективності аналізу безпеки.

Важливим етапом розробки стало впровадження бази даних, яка відповідала за збереження та управління накопиченими результатами аналізу. Обрана структура бази даних забезпечила високу швидкість доступу до інформації, що дозволило миттєво оновлювати дані про стан безпеки та здійснювати моніторинг змін у реальному часі. Оптимізація запитів до бази даних дозволила значно зменшити затримки у роботі системи, а механізми захисту, зокрема шифрування чутливих даних, сприяли підвищенню рівня безпеки самого застосунку.

У процесі тестування системи було виявлено її здатність адаптуватися до змін у середовищі функціонування та коректно працювати з різними веб-платформами. Аналіз продуктивності показав, що застосунок має високу швидкість обробки даних, а використання багаторівневого підходу до аналізу загроз дозволяє отримувати точні результати навіть у складних випадках. У ході дослідження також було визначено потенційні напрямки вдосконалення системи, серед яких подальша автоматизація аналізу результатів, інтеграція з іншими засобами оцінки безпеки та розширення функціональних можливостей.

Таким чином, реалізований застосунок відповідає поставленим завданням і може використовуватися для всебічної оцінки рівня безпеки веб-додатків. Його розробка продемонструвала ефективність інтеграції методів автоматизованого аналізу вразливостей із засобами візуалізації та звітності, що дозволило створити комплексний інструмент для дослідження стану інформаційної безпеки. Отримані результати підтверджують доцільність подальшого розвитку цього напрямку, що дозволить удосконалювати методи кіберзахисту та підвищувати надійність сучасних веб-систем.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Боскін, О. О., Корніловська, Н. В., Поліщук, В. М., & Сарафаннікова, Н. В. (2023). БЕЗПЕКА ВЕБ-ДОДАТКІВ ТА ХАКЕРСЬКІ АТАКИ. *Вестник Херсонского национального технического университета*, (3 (86)), 83-92.
2. Панасенко, Д. О. (2025). Дослідження методів тестування безпеки веб-додатків та впровадження інструментів для виявлення вразливостей.
3. Євгенєв, А. М., Бичковський, І. Ю., & Уманець, М. С. (2023). *Аналіз вразливостей веб-додатків з використанням мови програмування Python* (Doctoral dissertation).
4. Степанов, А. (2021). *Удосконалення стандартних методів захисту веб-додатків* (Master's thesis, ТНТУ).
5. Ivanenko, D., & Pryshchepa, O. (2021). Evaluation of the efficiency of web-application safety scanners. *Computer Science and Cybersecurity*, (2), 4-14.
6. Чех, Т. П. (2022). *Розробка інформаційної системи для виявлення вразливостей веб-сайтів* (Bachelor's thesis, ТНТУ).
7. Якубич, К. О., & Потапова, Н. А. (2021). Безпека у веб-сервісах. *Прикладні аспекти сучасних міждисциплінарних досліджень*, 127-129.
8. Єпик, О. О. (2024). Удосконалення засобів захисту веб-застосунків від несанкціонованого впливу.
9. Кільдішев, В. Й., & Тюпо, І. С. (2022). ЗАХИСТ ВЕБ-ДОДАТКІВ: АКТУАЛЬНІ ТЕНДЕНЦІЇ. *Молодий вчений*.
10. Кончак, М. (2024). Аналіз методів захисту веб додатків від кібератак. *Матеріали науково-технічної конференції „Інформаційні моделі, системи та технології “*, 41-41.
11. Циганенко, Д. П. (2024). Методи та технології захисту цифрових активів веб-додатків в умовах DDoS-атак.
12. Рудич, Є. О., & Довгалець, С. М. (2021). *Розробка безпечного веб-додатку з клієнт-серверною архітектурою* (Doctoral dissertation, ВНТУ).

13. Костюк, Ю., & Гейдаров, Г. (2024). РОЛЬ ТОКЕНІВ ТА СЕСІЙНОГО КЕРУВАННЯ В СИСТЕМАХ ІНФОРМАЦІЙНОЇ БЕЗПЕКИ ДЛЯ ПРОТИДІЇ МІЖСАЙТОВИМ АТАКАМ. *Наука і техніка сьогодні*, (5 (33)).

14. Renaud, K., & Coles-Kemp, L. (2022). Accessible and inclusive cyber security: a nuanced and complex challenge. *SN Computer Science*, 3(5), 346.

15. Hitch, N., & Simon Bell, I. S. G. (2022). Measuring Adoption of Security Mechanisms in the HTTPS Ecosystem. *ISG MSc Information Security*.

16. Rutter R et al. Web Accessibility: Web Standards and Regulatory Compliance. [A Book Apart, 2006. 744 p.

17. Shawn H. Just Ask: Integrating Accessibility Throughout Design. [Lulu.com, 2007. 200 p.

18. Pickering H. Inclusive Design Patterns. Smashing Magazine GmbH, 2016. 291 p.

19. Kalbag L. Accessibility for Everyone. [A Book Apart, 2017. 166 p.

20. Horton S. , Quesenbery W. A Web for Everyone: Designing Accessible User Experiences. Rosenfeld Media, 2014. 288 p.

21. Clark J. Building Accessible Websites. New Riders Pub, 2002. 415 p.

22. Chisholm W. , May M. Universal Design for Web Applications: Web Applications That Reach Everyone. O'Reilly Media, 2008. 196 p.

23. Krug S. Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability. New Riders, 2013. 216 p.

24. Firth A. , and Sutton M. Practical Web Inclusion and Accessibility: A Comprehensive Guide to Access Needs, 2019, 498 p.

25. Cunningham K. Accessibility Handbook. O'Reilly Media, 2012. 96p.

26. Matthes.E. Python Crash Course, 2nd Edition: A Hands-On, Project-Based Introduction to Programming. No Starch Press, 2019. 544p.

27. Nikto. URL: <https://www.cirt.net/Nikto2> (дата звернення: 20.01.2025).

28. Wapiti. URL: <https://wapiti-scanner.github.io/> (дата звернення: 20.01.2025).

29. Wfuzz. URL: <https://www.kali.org/tools/wfuzz/> (дата звернення: 20.01.2025).

30. Ramalho L. Fluent Python: Clear, Concise, and Effective Programming. O'Reilly Media, 2022. 1012 p.

31. Введення в базу даних Sqlite. URL: <http://yoip.com.ua/vvedennya-v-bazu-danih-sqlite/> (дата звернення: 02.02.2025).

32. Python. Програми з графічним інтерфейсом. URL: <https://www.miyklas.com.ua/p/informatica/8-klas/algoritmi-ta-programi-394917/programi-z-grafichnim-interfeisom-395197/re-6038e80b-3201-49cd-a207-7518a6aab2c0> (дата звернення: 03.02.2025).

33. Datetime — Основні типи дати та часу. URL: <https://www.miyklas.com.ua/System/WrongBrowser?originalUrl=https%3A%2F%2Fwww.miyklas.com.ua%2Fp%2Finformatica%2F8-klas%2Falgoritmi-ta-programi-394917%2Fprogrami-z-grafichnim-interfeisom-395197%2Fre-6038e80b-3201-49cd-a207-7518a6aab2c0> (дата звернення: 03.02.2025).

ДОДАТКИ

Додаток А «Лістинг програмного коду»

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import tkinter as tk
from tkinter import ttk, messagebox, filedialog
import sqlite3
import requests
import threading
import queue
import datetime
import re
import socket, ssl
from urllib.parse import urlparse
import hashlib

def hash_password(password):
    return hashlib.sha256(password.encode('utf-8')).hexdigest()

def scan_sql_injection(url, timeout):
    payload = ""
    try:
        response = requests.get(url + payload, timeout=timeout)
        if re.search(r"(SQL|sql|syntax)", response.text):
            return "Можлива вразливість до SQL-ін'єкцій"
```

```

        return "Не виявлено ознак SQL-ін'єкції"
except Exception as e:
    return "Помилка SQL-ін'єкції: " + str(e)

def scan_xss(url, timeout):
    payload = "<script>alert('XSS')</script>"
    try:
        response = requests.get(url, params={'q': payload}, timeout=timeout)
        if payload in response.text:
            return "Можлива вразливість до XSS-атак"
        return "Не виявлено ознак XSS-атак"
    except Exception as e:
        return "Помилка XSS: " + str(e)

def scan_directory_traversal(url, timeout):
    payload = "../etc/passwd"
    try:
        response = requests.get(url.rstrip("/") + "/" + payload, timeout=timeout)
        if "root:" in response.text:
            return "Можлива вразливість до Directory Traversal"
        return "Не виявлено ознак Directory Traversal"
    except Exception as e:
        return "Помилка Directory Traversal: " + str(e)

def scan_header_info(url, timeout):
    try:
        response = requests.head(url, timeout=timeout)
        headers = response.headers
        missing = []

```

```

if "X-Content-Type-Options" not in headers:
    missing.append("X-Content-Type-Options")
if "X-Frame-Options" not in headers:
    missing.append("X-Frame-Options")
if "Content-Security-Policy" not in headers:
    missing.append("Content-Security-Policy")
if missing:
    return "Відсутні заголовки: " + ", ".join(missing)
return "Заголовки налаштовані"
except Exception as e:
    return "Помилка заголовків: " + str(e)

def scan_robots_txt(url, timeout):
    try:
        robots_url = url.rstrip("/") + "/robots.txt"
        response = requests.get(robots_url, timeout=timeout)
        if response.status_code == 200:
            content = response.text.strip()
            if content:
                return "robots.txt знайдено"
            return "robots.txt порожній"
        return "robots.txt відсутній"
    except Exception as e:
        return "Помилка robots.txt: " + str(e)

def scan_http_methods(url, timeout):
    try:
        response = requests.options(url, timeout=timeout)
        allowed = response.headers.get("Allow", "")

```

```

    if allowed:
        return "Дозволені методи: " + allowed
    return "Методи не визначені"
except Exception as e:
    return "Помилка HTTP методів: " + str(e)

def scan_server_banner(url, timeout):
    try:
        response = requests.head(url, timeout=timeout)
        server = response.headers.get("Server", "Не визначено")
        return "Банер сервера: " + server
    except Exception as e:
        return "Помилка банера сервера: " + str(e)

def scan_cors_policy(url, timeout):
    try:
        response = requests.get(url, timeout=timeout)
        cors = response.headers.get("Access-Control-Allow-Origin", "")
        if cors:
            return "CORS налаштований: " + cors
        return "CORS відсутній"
    except Exception as e:
        return "Помилка CORS: " + str(e)

def scan_ssl_certificate(url, timeout):
    parsed = urlparse(url)
    if parsed.scheme != "https":
        return "Протокол не HTTPS, SSL відсутній"
    host = parsed.hostname

```

```

port = parsed.port if parsed.port else 443
try:
    context = ssl.create_default_context()
    conn = socket.create_connection((host, port), timeout=timeout)
    sock = context.wrap_socket(conn, server_hostname=host)
    cert = sock.getpeercert()
    sock.close()
    notAfter = cert.get('notAfter')
    try:
        expiry_date = datetime.datetime.strptime(notAfter, "%b %d %H:%M:%S %Y
%Z")
    except:
        expiry_date = datetime.datetime.strptime(notAfter, "%b %d %H:%M:%S
%Y")
    now = datetime.datetime.utcnow()
    if now > expiry_date:
        return "Сертифікат прострочено (до {})".format(expiry_date)
    return "Сертифікат дійсний до {}".format(expiry_date)
except Exception as e:
    return "Помилка SSL: " + str(e)

class ScannerThread(threading.Thread):
    def __init__(self, url, timeout, out_queue):
        super().__init__()
        self.url = url
        self.timeout = timeout
        self.out_queue = out_queue
    def run(self):
        results = []

```

```

tests = [
    ("SQL-ін'єкція", scan_sql_injection),
    ("XSS", scan_xss),
    ("Directory Traversal", scan_directory_traversal),
    ("Заголовки", scan_header_info),
    ("robots.txt", scan_robots_txt),
    ("HTTP Методи", scan_http_methods),
    ("Банер сервера", scan_server_banner),
    ("CORS", scan_cors_policy),
    ("SSL сертифікат", scan_ssl_certificate)
]
total = len(tests)
self.out_queue.put(('result', "Початок сканування для: " + self.url + "\n"))
for idx, (name, func) in enumerate(tests, start=1):
    try:
        res = func(self.url, self.timeout)
    except Exception as e:
        res = "Помилка: " + str(e)
    results.append(f'{name}: {res}')
    progress = int((idx / total) * 100)
    self.out_queue.put(('progress', progress))
    self.out_queue.put(('result', f'{name}: {res}\n'))
final_result = "\n".join(results)
self.out_queue.put(('result', "\nСканування завершено.\n"))
self.out_queue.put(('complete', final_result))

```

```

class LoginWindow(tk.Toplevel):
    def __init__(self, parent, db_conn):
        super().__init__(parent)

```

```

self.parent = parent
self.db_conn = db_conn
self.title("Авторизація")
self.geometry("300x200")
self.resizable(False, False)
self.grab_set()
self.init_ui()
def init_ui(self):
    frm = ttk.Frame(self)
    frm.pack(padx=20, pady=20, fill="both", expand=True)
    lbl_user = ttk.Label(frm, text="Користувач:")
    lbl_user.pack(pady=(0,5))
    self.ent_user = ttk.Entry(frm)
    self.ent_user.pack(pady=(0,10), fill="x")
    lbl_pass = ttk.Label(frm, text="Пароль:")
    lbl_pass.pack(pady=(0,5))
    self.ent_pass = ttk.Entry(frm, show="*")
    self.ent_pass.pack(pady=(0,10), fill="x")
    btn_frame = ttk.Frame(frm)
    btn_frame.pack(pady=10, fill="x")
    btn_login = ttk.Button(btn_frame, text="Вхід", command=self.attempt_login)
    btn_login.pack(side="left", expand=True, fill="x", padx=(0,5))
    btn_register = ttk.Button(btn_frame, text="Реєстрація",
command=self.open_register)
    btn_register.pack(side="left", expand=True, fill="x", padx=(5,0))
def attempt_login(self):
    username = self.ent_user.get().strip()
    password = self.ent_pass.get().strip()
    if not username or not password:

```

```

        messagebox.showwarning("Помилка", "Введіть користувача та пароль")
        return
    cur = self.db_conn.cursor()
        cur.execute("SELECT password FROM users WHERE username=?",
(username,))
    row = cur.fetchone()
    if row and row[0] == hash_password(password):
        self.parent.logged_in_user = username
        self.destroy()
    else:
        messagebox.showerror("Помилка", "Невірний логін або пароль")
def open_register(self):
    RegistrationWindow(self, self.db_conn)

class RegistrationWindow(tk.Toplevel):
    def __init__(self, parent, db_conn):
        super().__init__(parent)
        self.db_conn = db_conn
        self.title("Реєстрація")
        self.geometry("300x250")
        self.resizable(False, False)
        self.grab_set()
        self.init_ui()
    def init_ui(self):
        frm = ttk.Frame(self)
        frm.pack(padx=20, pady=20, fill="both", expand=True)
        lbl_user = ttk.Label(frm, text="Новий користувач:")
        lbl_user.pack(pady=(0,5))
        self.ent_user = ttk.Entry(frm)

```

```

self.ent_user.pack(pady=(0,10), fill="x")
lbl_pass = ttk.Label(frm, text="Пароль:")
lbl_pass.pack(pady=(0,5))
self.ent_pass = ttk.Entry(frm, show="*")
self.ent_pass.pack(pady=(0,10), fill="x")
lbl_pass2 = ttk.Label(frm, text="Підтвердження паролю:")
lbl_pass2.pack(pady=(0,5))
self.ent_pass2 = ttk.Entry(frm, show="*")
self.ent_pass2.pack(pady=(0,10), fill="x")
btn_reg = ttk.Button(frm, text="Зареєструватися", command=self.register)
btn_reg.pack(pady=10, fill="x")
def register(self):
    username = self.ent_user.get().strip()
    password = self.ent_pass.get().strip()
    password2 = self.ent_pass2.get().strip()
    if not username or not password or not password2:
        messagebox.showwarning("Помилка", "Заповніть усі поля")
        return
    if password != password2:
        messagebox.showwarning("Помилка", "Паролі не співпадають")
        return
    cur = self.db_conn.cursor()
    cur.execute("SELECT * FROM users WHERE username=?", (username,))
    if cur.fetchone():
        messagebox.showwarning("Помилка", "Користувач вже існує")
        return
    cur.execute("INSERT INTO users (username, password) VALUES (?, ?)",
    (username, hash_password(password)))
    self.db_conn.commit()

```

```

messagebox.showinfo("Успіх", "Реєстрація пройшла успішно")
self.destroy()

```

```

class MainApp(tk.Tk):

```

```

    def __init__(self, db_conn, logged_in_user):

```

```

        super().__init__()

```

```

        self.db_conn = db_conn

```

```

        self.logged_in_user = logged_in_user

```

```

        self.title("Сканер безпеки веб-додатків - Користувач: " + self.logged_in_user)

```

```

        self.geometry("1000x750")

```

```

        self.style = ttk.Style(self)

```

```

        self.style.theme_use("clam")

```

```

        self.queue = queue.Queue()

```

```

        # Спочатку ініціалізуємо базу даних, щоб створити таблиці з необхідними
        колонками

```

```

        self.init_db_tables()

```

```

        # Після цього ініціалізуємо інтерфейс користувача

```

```

        self.init_ui()

```

```

        self.after(100, self.process_queue)

```

```

    def init_db_tables(self):

```

```

        cur = self.db_conn.cursor()

```

```

        # Створення таблиці scans, якщо не існує

```

```

        cur.execute("""CREATE TABLE IF NOT EXISTS scans (
                        id INTEGER PRIMARY KEY AUTOINCREMENT,
                        url TEXT,
                        date TEXT,
                        results TEXT,
                        username TEXT
                    )""")

```

Якщо таблиця вже існує, перевіряємо наявність колонки username та додаємо її, якщо відсутня

```
cur.execute("PRAGMA table_info(scans)")
```

```
columns = [col[1] for col in cur.fetchall()]
```

```
if "username" not in columns:
```

```
    cur.execute("ALTER TABLE scans ADD COLUMN username TEXT")
```

Створення таблиці settings, якщо не існує

```
cur.execute("CREATE TABLE IF NOT EXISTS settings (parameter TEXT  
PRIMARY KEY, value TEXT)")
```

```
self.db_conn.commit()
```

```
cur.execute("SELECT COUNT(*) FROM settings")
```

```
if cur.fetchone()[0] == 0:
```

```
    cur.execute("INSERT INTO settings (parameter, value) VALUES (?, ?)",  
("timeout", "10"))
```

```
    cur.execute("INSERT INTO settings (parameter, value) VALUES (?, ?)",  
("user_agent", "Mozilla/5.0"))
```

```
    self.db_conn.commit()
```

```
def init_ui(self):
```

```
    self.notebook = ttk.Notebook(self)
```

```
    self.notebook.pack(fill="both", expand=True)
```

```
    self.scan_tab = ttk.Frame(self.notebook)
```

```
    self.history_tab = ttk.Frame(self.notebook)
```

```
    self.settings_tab = ttk.Frame(self.notebook)
```

```
    self.reports_tab = ttk.Frame(self.notebook)
```

```
    self.notebook.add(self.scan_tab, text="Сканування")
```

```
    self.notebook.add(self.history_tab, text="Історія")
```

```
    self.notebook.add(self.settings_tab, text="Налаштування")
```

```
    self.notebook.add(self.reports_tab, text="Звіти")
```

```
    self.init_scan_tab()
```

```
    self.init_history_tab()
```

```

self.init_settings_tab()
self.init_reports_tab()
menubar = tk.Menu(self)
filemenu = tk.Menu(menubar, tearoff=0)
filemenu.add_command(label="Вихід", command=self.on_close)
menubar.add_cascade(label="Файл", menu=filemenu)
helpmenu = tk.Menu(menubar, tearoff=0)
helpmenu.add_command(label="Про програму", command=self.show_about)
menubar.add_cascade(label="Допомога", menu=helpmenu)
self.config(menu=menubar)

def init_scan_tab(self):
    frm = ttk.Frame(self.scan_tab)
    frm.pack(padx=10, pady=10, fill="x")
    lbl = ttk.Label(frm, text="URL для сканування:")
    lbl.pack(side="left", padx=(0,5))
    self.url_entry = ttk.Entry(frm, width=70)
    self.url_entry.pack(side="left", padx=(0,5))
    self.scan_btn = ttk.Button(frm, text="Запустити сканування",
command=self.start_scan)
    self.scan_btn.pack(side="left", padx=(0,5))
    self.progress = ttk.Progressbar(self.scan_tab, orient="horizontal", length=900,
mode="determinate")
    self.progress.pack(padx=10, pady=10)
    txt_frame = ttk.Frame(self.scan_tab)
    txt_frame.pack(padx=10, pady=10, fill="both", expand=True)
    self.scan_text = tk.Text(txt_frame, wrap="word", font=("Helvetica", 11))
    self.scan_text.pack(side="left", fill="both", expand=True)
    scroll = ttk.Scrollbar(txt_frame, command=self.scan_text.yview)
    scroll.pack(side="right", fill="y")

```

```

self.scan_text.config(yscrollcommand=scroll.set)

def init_history_tab(self):
    frm = ttk.Frame(self.history_tab)
    frm.pack(padx=10, pady=10, fill="both", expand=True)

    self.history_tree = ttk.Treeview(frm, columns=("ID", "URL", "Дата",
"Результати"), show="headings")
    self.history_tree.heading("ID", text="ID")
    self.history_tree.heading("URL", text="URL")
    self.history_tree.heading("Дата", text="Дата")
    self.history_tree.heading("Результати", text="Результати")
    self.history_tree.column("ID", width=50, anchor="center")
    self.history_tree.column("URL", width=250)
    self.history_tree.column("Дата", width=150, anchor="center")
    self.history_tree.column("Результати", width=500)
    self.history_tree.pack(side="left", fill="both", expand=True)
    scroll = ttk.Scrollbar(frm, command=self.history_tree.yview)
    scroll.pack(side="right", fill="y")
    self.history_tree.config(yscrollcommand=scroll.set)
    self.history_tree.bind("<Double-1>", self.view_scan_detail)

    btn = ttk.Button(self.history_tab, text="Оновити історію",
command=self.load_history)
    btn.pack(pady=5)
    self.load_history()

def view_scan_detail(self, event):
    selected = self.history_tree.selection()
    if not selected:
        return
    item = self.history_tree.item(selected[0])
    values = item["values"]

```

```

detail_win = tk.Toplevel(self)
detail_win.title("Деталі сканування (ID: {})".format(values[0]))
detail_win.geometry("800x600")
txt = tk.Text(detail_win, wrap="word", font=("Helvetica", 11))
txt.pack(fill="both", expand=True)

txt.insert(tk.END, "ID:  {}\nURL:  {}\nДата:
{}\n\nРезультати:\n{}".format(values[0], values[1], values[2], values[3]))

btn_close = ttk.Button(detail_win, text="Закрити",
command=detail_win.destroy)
btn_close.pack(pady=5)

def init_settings_tab(self):
    frm = ttk.Frame(self.settings_tab)
    frm.pack(padx=10, pady=10, fill="x")
    lbl_timeout = ttk.Label(frm, text="Таймаут (сек):")
    lbl_timeout.grid(row=0, column=0, sticky="w", padx=5, pady=5)
    self.timeout_entry = ttk.Entry(frm, width=20)
    self.timeout_entry.grid(row=0, column=1, padx=5, pady=5)
    lbl_ua = ttk.Label(frm, text="User Agent:")
    lbl_ua.grid(row=1, column=0, sticky="w", padx=5, pady=5)
    self.ua_entry = ttk.Entry(frm, width=50)
    self.ua_entry.grid(row=1, column=1, padx=5, pady=5)
    btn_save = ttk.Button(frm, text="Зберегти налаштування",
command=self.save_settings)
    btn_save.grid(row=2, column=0, columnspan=2, pady=10)
    cur = self.db_conn.cursor()
    cur.execute("SELECT value FROM settings WHERE parameter='timeout'")
    timeout_val = cur.fetchone()[0]
    self.timeout_entry.delete(0, tk.END)
    self.timeout_entry.insert(0, timeout_val)
    cur.execute("SELECT value FROM settings WHERE parameter='user_agent'")

```

```

ua_val = cur.fetchone()[0]
self.ua_entry.delete(0, tk.END)
self.ua_entry.insert(0, ua_val)
def init_reports_tab(self):
    frm = ttk.Frame(self.reports_tab)
    frm.pack(padx=10, pady=10, fill="both", expand=True)
    btn_export = ttk.Button(frm, text="Експортувати історію в файл",
command=self.export_history)
    btn_export.pack(pady=5)
    txt_frame = ttk.Frame(frm)
    txt_frame.pack(fill="both", expand=True)
    self.report_text = tk.Text(txt_frame, wrap="word", font=("Helvetica", 11))
    self.report_text.pack(side="left", fill="both", expand=True)
    scroll = ttk.Scrollbar(txt_frame, command=self.report_text.yview)
    scroll.pack(side="right", fill="y")
    self.report_text.config(yscrollcommand=scroll.set)
def start_scan(self):
    url = self.url_entry.get().strip()
    if not url:
        messagebox.showwarning("Помилка", "Не вказано URL для сканування")
        return
    try:
        timeout_val = float(self.timeout_entry.get().strip())
    except:
        timeout_val = 10
    self.scan_btn.config(state="disabled")
    self.scan_text.delete("1.0", tk.END)
    self.progress["value"] = 0
    self.scan_thread = ScannerThread(url, timeout_val, self.queue)

```

```

        self.scan_thread.start()
def process_queue(self):
    try:
        while True:
            item = self.queue.get_nowait()
            if item[0] == 'progress':
                self.progress["value"] = item[1]
            elif item[0] == 'result':
                self.scan_text.insert(tk.END, item[1])
                self.scan_text.see(tk.END)
            elif item[0] == 'complete':
                final_result = item[1]
                current_date = datetime.datetime.now().strftime("%Y-%m-%d
%H:%M:%S")
                url = self.url_entry.get().strip()
                cur = self.db_conn.cursor()
                cur.execute("INSERT INTO scans (username, url, date, results)
VALUES (?, ?, ?, ?)", (self.logged_in_user, url, current_date, final_result))
                self.db_conn.commit()
                self.load_history()
                self.scan_btn.config(state="normal")
        except queue.Empty:
            pass
        self.after(100, self.process_queue)
def load_history(self):
    for row in self.history_tree.get_children():
        self.history_tree.delete(row)
    cur = self.db_conn.cursor()
    cur.execute("SELECT id, url, date, results FROM scans WHERE username=?
ORDER BY id DESC", (self.logged_in_user,))

```

```

rows = cur.fetchall()
for row in rows:
    self.history_tree.insert("", tk.END, values=row)
def save_settings(self):
    timeout_val = self.timeout_entry.get().strip()
    ua_val = self.ua_entry.get().strip()
    cur = self.db_conn.cursor()
    cur.execute("UPDATE settings SET value=? WHERE parameter='timeout'",
(timeout_val,))
    cur.execute("UPDATE settings SET value=? WHERE parameter='user_agent'",
(ua_val,))
    self.db_conn.commit()
    messagebox.showinfo("Налаштування", "Налаштування збережено")
def export_history(self):
    cur = self.db_conn.cursor()
    cur.execute("SELECT id, url, date, results FROM scans WHERE username=?
ORDER BY id DESC", (self.logged_in_user,))
    rows = cur.fetchall()
    export_text = ""
    for row in rows:
        export_text += "ID: {} \nURL: {} \nДата: {} \nРезультати: \n{} \n\n".format(row[0], row[1], row[2], row[3])
    file_name = filedialog.asksaveasfilename(title="Зберегти звіт",
defaulttextension=".txt", filetypes=[("Text Files", "*.txt")])
    if file_name:
        with open(file_name, "w", encoding="utf-8") as f:
            f.write(export_text)
        self.report_text.delete("1.0", tk.END)
        self.report_text.insert(tk.END, "Звіт збережено в: " + file_name)
def show_about(self):

```

messagebox.showinfo("Про програму", "Додаток для оцінки безпеки веб-додатків\nРозроблено на Python з використанням Tkinter\nАвторизація з роздільною історією для кожного акаунту")

```

def on_close(self):
    self.db_conn.close()
    self.destroy()

def init_database():
    conn = sqlite3.connect("scanner_history.db")
    cur = conn.cursor()

    cur.execute("CREATE TABLE IF NOT EXISTS users (username TEXT
PRIMARY KEY, password TEXT)")

    conn.commit()

    cur.execute("SELECT COUNT(*) FROM users")

    if cur.fetchone()[0] == 0:
        cur.execute("INSERT INTO users (username, password) VALUES (?, ?)",
("admin", hash_password("admin")))

        conn.commit()

    return conn

def main():
    db_conn = init_database()
    root = tk.Tk()
    root.withdraw()
    login = LoginWindow(root, db_conn)
    root.wait_window(login)
    if not hasattr(root, "logged_in_user") or not root.logged_in_user:
        db_conn.close()
        return

    app = MainApp(db_conn, root.logged_in_user)
    app.mainloop()

if __name__ == "__main__"

```