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

До захисту допущено
Завідувач кафедри

_____ **Дмитро ЛАНДЕ**
(підпис)

«_____»_____2025 р.

Дипломна робота

на здобуття ступеня бакалавра

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

спеціальності 125 «Кібербезпека»

на тему: Автоматизація аудиту безпеки веб-застосунків у контексті кіберзахисту

Виконав (-ла): здобувач вищої освіти **IV** курсу, групи ФБ-11
(шифр групи)

Тимощук Ілья Олексійович
(прізвище, ім'я, по батькові) (підпис)

Керівник старший викладач кафедри ІБ, к.ф.-м.н., Рибак О. В.
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Рецензент доцент кафедри ММЗІ, к.т.н., Кучинська Н. В.
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище, ім'я, по батькові) (підпис)

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

Київ – 2025 року

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

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

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Дмитро ЛАНДЕ
(підпис)
«___»_____2025 р.

ЗАВДАННЯ
на дипломну роботу здобувачу вищої освіти

Тимощуку Ільї Олексійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи
Автоматизація аудиту безпеки веб-застосунків у контексті кіберзахисту
Переклад теми дипломної роботи (англ.):

Automation of Web Application Security Auditing in Cybersecurity Context

,
керівник роботи
Рибак Олександр Владиславович кандидат фізико-математичних наук, старший викладач кафедри інформаційної безпеки
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від « 26 » травня 2025 р. № 1761-с

2. Термін подання роботи здобувачем вищої освіти « 11 » червня 2025 р.

3. Вихідні дані до роботи
Літературні джерела з питань кіберзахисту, типів вразливостей веб-застосунків, методів сканування безпеки, інструменти аудиту безпеки (OWASP ZAP, Nmap з Vulscan, Nuclei, Wapiti, Nessus, Burp Suite Scanner), методи обробки результатів(нормалізація, фільтрація, сортування), системи виводу результатів (HTML, PDF, API), технології реалізації (Flask, Bootstrap, SQLite, Python Subprocess), тестове середовище (Metasploitable2)

4. Зміст роботи

Провести аналіз існуючих засобів аудиту безпеки веб-застосунків, розробити веб-застосунок для автоматичного аудиту безпеки веб-застосунків, провести тестування рішення на вразливому середовищі, порівняти отримані результати з комерційними продуктами (Nessus, Burp Suite).

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): презентація

6. Дата видачі завдання 3 жовтня 2024 року

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка
1	Визначення теми дипломної роботи та збір вихідних даних	03.10.2024 – 02.02.2025	Виконано
2	Аналіз загроз та типових вразливостей веб-застосунків	03.02.2025 – 09.02.2025	Виконано
3	Аналіз існуючих інструментів аудиту безпеки	10.02.2025 – 23.02.2025	Виконано
4	Проєктування архітектури веб-застосунку для аудиту безпеки	24.02.2025 – 02.03.2025	Виконано
5	Розробка бекенду застосунку	03.03.2025 – 06.04.2025	Виконано
6	Реалізація інтерфейсу користувача	07.04.2025 – 20.04.2025	Виконано
7	Нормалізація, фільтрація та сортування результатів сканування	21.04.2025 – 27.04.2025	Виконано
8	Проведення експериментального тестування з інтегрованими сканерами	28.04.2025 – 04.05.2025	Виконано
9	Порівняльний аналіз з комерційними інструментами	05.05.2025 – 18.05.2025	Виконано
10	Аналіз виявлених недоліків та оцінка придатності для кіберзахисту	19.05.2025 – 27.05.2025	Виконано
11	Оформлення дипломної роботи та створення презентації	28.05.2025 – 11.06.2025	Виконано
12	Передзахист	11.06.2025	Виконано
13	Захист	18.06.2025	Виконано

Здобувач вищої освіти _____ Ілья ТИМОЩУК
(підпис) (**Власне ім'я, ПРІЗВИЩЕ**)

Керівник роботи _____ Олександр РИБАК
(підпис) (**Власне ім'я, ПРІЗВИЩЕ**)

РЕФЕРАТ

Обсяг роботи: 112 сторінок, 24 ілюстрації, 7 таблиць, 7 додатків, 34 джерел літератури.

Об’єкт дослідження: процеси аудиту безпеки веб-застосунків. Предмет дослідження: засоби та методи автоматизації виявлення веб-вразливостей у контексті кіберзахисту.

Мета дослідження: покращення захисту веб-застосунків від критичних кіберзагроз, шляхом зменшення ризиків, пов’язаних із пізнім виявленням вразливостей, за допомогою автоматизованого аудиту безпеки на основі інтегрованого веб-інтерфейсу.

Методи дослідження: аналіз літературних джерел, тестування на вразливих середовищах (Metasploitable2), порівняльний аналіз результатів, систематизація отриманих даних.

Отримані результати: розроблено веб-застосунок, що автоматизує процес запуску відкритих сканерів безпеки (OWASP ZAP, Nuclei, Wapiti, Nmap+Vulscan) з єдиного інтерфейсу. Інструмент протестовано на контрольованому вразливому середовищі та порівняно з комерційними продуктами (Nessus, Burp Suite). Отримані результати показали співставну ефективність розробленого рішення при значно нижчих витратах. Під час тестування виявлено критичні та високоризикові вразливості, що підтверджує дієвість підходу. У застосунку наведено акти сканування та приклади виявлених вразливостей.

Рекомендації: використання створеного рішення може бути доцільним для малих і середніх компаній, які потребують доступного способу проведення первинного аудиту веб-безпеки. Розробка також може бути інтегрована у процес CI/CD.

Ключові слова: аудит безпеки, веб-застосунок, автоматизоване сканування, вразливість, кіберзахист.

ABSTRACT

This work contains 112 pages, 24 illustrations, 7 tables, 7 appendices, and 34 sources in the list of references.

The object of research is the process of web application security auditing. The subject of research is the tools and methods for automating the detection of web vulnerabilities in the context of cybersecurity.

The aim of the research is to develop a software solution for automated web application security auditing by integrating several open-source scanners into a unified web interface.

Research methods include the analysis of literature sources, testing in vulnerable environments (such as Metasploitable2), attack modeling, comparative analysis of results, and systematization of the obtained data.

As a result, a web application was developed to automate the launch of open-source security scanners (OWASP ZAP, Nuclei, Wapiti, Nmap+Vulscan, etc.) through a single interface. The tool was tested in a controlled vulnerable environment and compared with commercial products (Nessus, Burp Suite). The obtained results demonstrated comparable effectiveness of the developed solution at significantly lower cost. Critical and high-risk vulnerabilities were discovered during testing, confirming the effectiveness of the proposed approach. The developed application presents scan reports and examples of identified vulnerabilities.

Recommendations: the developed solution can be effectively used by small and medium-sized companies in need of an affordable method for conducting initial web security audits. The system can also be integrated into CI/CD pipelines.

Keywords: security audit, web application, automated scanning, vulnerability, cybersecurity.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

WAF (Web Application Firewall) — міжмережевий екран веб-застосунків, який фільтрує, контролює та блокує HTTP(S)-трафік до веб-застосунків з метою захисту від поширених атак, таких як SQL-ін’єкції, XSS, CSRF тощо.

OS (Operating System) — операційна система; базове програмне забезпечення, що забезпечує керування апаратними ресурсами комп’ютера та надає інтерфейс для взаємодії користувача з системою.

LDAP (Lightweight Directory Access Protocol) — протокол прикладного рівня, що використовується для доступу і управління розподіленими службами каталогів (наприклад, Active Directory) у мережах.

CI/CD (Continuous Integration / Continuous Deployment) — безперервна інтеграція та безперервне розгортання; методологія розробки програмного забезпечення, що забезпечує автоматичну перевірку коду та розгортання змін у середовище виконання.

API (Application Programming Interface) — інтерфейс прикладного програмування; набір визначень, протоколів і засобів для створення і взаємодії програмних компонентів.

REST (Representational State Transfer) — архітектурний стиль для побудови веб-сервісів, який базується на стандартних HTTP-методах (GET, POST, PUT, DELETE) і використовується для розробки API.

CLI (Command Line Interface) — інтерфейс командного рядка; спосіб взаємодії з операційною системою або програмою через введення текстових команд у терміналі.

UI (User Interface) — користувацький інтерфейс; елемент програмного забезпечення або пристрою, через який користувач взаємодіє з системою.

GUI (Graphical User Interface) — графічний інтерфейс користувача; тип інтерфейсу, що використовує графічні елементи (вікна, кнопки, іконки) для взаємодії користувача з програмами.

CRM (Customer Relationship Management) — система управління взаємовідносинами з клієнтами; програмне забезпечення, що використовується для організації, автоматизації та синхронізації процесів продажів, маркетингу та обслуговування клієнтів.

ВСТУП

У сучасному цифровому середовищі веб-застосунки стали основою для надання послуг, обробки даних та комунікації як у приватному, так і в державному секторі. Вони охоплюють фінансові платформи, медичні інформаційні системи, урядові портали, онлайн-магазини, соціальні мережі та інші критично важливі сервіси. Разом з розвитком веб-технологій зростає і рівень загроз — зловмисники постійно вдосконалюють методи атак, використовують автоматизовані сканери, експлойти, боти та соціальну інженерію для пошуку вразливостей.

Веб-застосунки часто стають основною ціллю кібератак, оскільки вони безпосередньо взаємодіють із зовнішнім середовищем, клієнтами та іншими сервісами. Недостатній рівень безпеки в їхній архітектурі або реалізації може призвести до серйозних наслідків — витоку персональних даних, фінансових втрат, компрометації системи або повної зупинки бізнес-процесів.

У зв’язку з цим особливої актуальності набуває завдання своєчасного виявлення вразливостей у веб-застосунках. Однак ручна перевірка на предмет безпеки є тривалою, неефективною і часто не дозволяє оперативно виявити повний спектр ризиків. Саме тому автоматизація аудиту безпеки стає необхідною умовою ефективного кіберзахисту веб-ресурсів.

Актуальність роботи

З огляду на стрімке зростання кількості атак на веб-застосунки, високий рівень ризиків і недостатню обізнаність розробників щодо безпеки, необхідність у створенні автоматизованих рішень для аудиту веб-застосунків є надзвичайно важливою. Існуючі інструменти або надто складні для інтеграції в розробку, або вимагають значних фінансових ресурсів, що обмежує їх використання в малому і середньому бізнесі. Тому розробка гнучкого, доступного й ефективного засобу

для виявлення вразливостей у веб-застосунках є актуальною науково-практичною задачею.

Мета і завдання дослідження

Покращення захисту веб-застосунків від критичних кіберзагроз, шляхом зменшення ризиків, пов’язаних із пізнім виявленням вразливостей, за допомогою автоматизованого аудиту безпеки на основі інтегрованого веб-інтерфейсу.

Завдання дослідження:

- Провести аналіз існуючих засобів аудиту безпеки веб-застосунків.
- Розробити веб-застосунок для автоматичного аудиту безпеки веб-застосунків.
- Провести тестування рішення на вразливому середовищі.
- Порівняти отримані результати з комерційними продуктами (Nessus, Burp Suite).

Об’єкт дослідження – процеси аудиту безпеки веб-застосунків.

Предмет дослідження – засоби та методи автоматизації виявлення веб-вразливостей у контексті кіберзахисту.

Методи дослідження – аналіз літературних джерел, практичне тестування, порівняльний аналіз результатів сканування, узагальнення та систематизація отриманих результатів.

1 АНАЛІЗ ПРОБЛЕМИ КІБЕРБЕЗПЕКИ ВЕБ-ЗАСТОСУНКІВ

1.1 Стан загроз у сфері веб-застосунків

Останні кілька років веб-застосунки перетворилися на справжній "ласий шматок" для хакерів. Це й не дивно — через них проходить величезний обсяг особистих, фінансових та корпоративних даних. І хоча про важливість кіберзахисту говорять вже давно, статистика показує, що більшість атак усе ще проходить саме через веб-застосунки [1].

У 2023 році ця проблема загострилася ще більше. Кількість атак не лише зросла, але й стали помітні нові схеми, складніші й технічно витонченіші. Одного лише антивіруса чи налаштування фаєрвола вже явно недостатньо. Захист має бути продуманий, гнучкий і системний — без цього жоден веб-застосунок не можна вважати безпечним [1].

Проєкт OWASP Top 10 є авторитетним джерелом, яке визначає найкритичніші ризики безпеки веб-застосунків [2]. У версії 2021 року до списку входять такі вразливості, як:

- A01:2021 – Broken Access Control: Неправильне управління доступом, що дозволяє несанкціонований доступ до ресурсів.
- A02:2021 – Cryptographic Failures: Недоліки в реалізації криптографії, що можуть призвести до компрометації даних.
- A03:2021 – Injection: Вразливості типу SQL, NoSQL, OS або LDAP ін'єкцій, які дозволяють виконання небажаних команд або доступ до конфіденційних даних.
- A04:2021 – Insecure Design: Недостатньо безпечне проєктування системи, що створює потенційні вразливості.
- A05:2021 – Security Misconfiguration: Неправильні налаштування безпеки, які можуть бути використані зловмисниками.

- A06:2021 – Vulnerable and Outdated Components: Використання застарілих або вразливих компонентів у затосунках.
- A07:2021 – Identification and Authentication Failures: Помилки в ідентифікації та автентифікації користувачів.
- A08:2021 – Software and Data Integrity Failures: Порушення цілісності програмного забезпечення та даних.
- A09:2021 – Security Logging and Monitoring Failures: Недостатнє ведення журналів безпеки та моніторинг.
- A10:2021 – Server-Side Request Forgery (SSRF): Атаки, що змушують сервер робити небажані запити до внутрішніх або зовнішніх ресурсів.

Ці вразливості залишаються актуальними і в 2025 році, оскільки багато організацій не встигають оновлювати свої системи безпеки відповідно до нових загроз [2].

У звіті Indusface за 2023 рік вказано, що їх WAF заблокував понад 6,8 мільярда атак [3]. Це не лише про масштаб, а й про інтенсивність — в середньому атаки на веб-застосунки ставалися частіше й ставали більш нав'язливими.

Найбільше атак зафіксували в Індії, що може бути пов’язано з кількістю онлайн-сервісів, які активно розвиваються в регіоні. Ще цікавий момент — у понад третині виявлених вразливостей компанії навіть через пів року не внесли жодних змін [3]. Тобто вразливість є, її виявили — але її просто ігнорують.

Cloudflare також наводить красномовну статистику. Щодня — понад 100 мільярдів заблокованих атак. І це лише через одну платформу. Але справжній рекорд стався у серпні: понад 201 мільйон запитів у секунду — масштабна DDoS-атака, яку вдалося зупинити [4]. Така цифра майже утричі більша за попередній рекорд і добре показує, наскільки потужні ресурси тепер мають зловмисники [5].

Головна думка, яка випливає з усього вищесказаного, доволі проста: безпека не може бути "другорядною задачею". Вона має бути вбудована в розробку з самого початку.

Якщо не оновлювати компоненти, не перевіряти логіку доступу, не логувати підозрілі дії — то рано чи пізно все це призведе до зламу. І ніякий хостинг чи фаєрвол не врятує, якщо всередині самого застосунку повно вразливостей.

Веб-безпека — це не про один інструмент. Це про підхід: аналіз ризиків, своєчасні оновлення, автоматичні перевірки, внутрішній контроль доступу й готовність до інцидентів.

Тому якщо веб-проєкт працює онлайн і має справу з користувачами — без належної уваги до кіберзахисту він просто ризикує все втратити. І станом на сьогодні ризики тільки зростають.

1.2 Типи вразливостей: SQLi, XSS, RCE, CSRF, SSRF, конфігураційні помилки

Сьогодні веб-застосунки — це одна з головних цілей для хакерів. Вони через них проникають у системи, крадуть дані і порушують роботу бізнесу. Тому треба розуміти, які бувають вразливості, щоб знати, як їх виявляти та боротися з ними. Розглянемо основні вразливості, які найчастіше використовують для атак.

1.2.1 SQL Injection (SQLi)

SQL-ін’єкція — це дуже поширена вразливість, коли хакер може вставити свій SQL-запит у форму на сайті, і база даних виконує цей шкідливий запит. Через це можна отримати доступ до чужих даних або навіть їх видалити. За статистикою, близько 21% сайтів мають таку проблему [6].

1.2.2 Cross-Site Scripting (XSS)

XSS — це атака, коли хакер впроваджує шкідливий код у сторінку, і він запускається у браузері користувача. Це дозволяє вкрасти сесії, перекинути на шахрайські сайти чи зробити ще щось погане. Є кілька типів XSS, наприклад відображений, збережений і DOM-базований. Приблизно третина сайтів уразлива до цього [7].

1.2.3 Remote Code Execution (RCE)

RCE — дуже небезпечна вразливість, бо через неї зломисник може запускати свій код на сервері і повністю контролювати систему. Він може викрасти дані, поставити бекдори або використати сервер для інших атак. Часто це через те, що дані неправильно обробляються або у зовнішніх бібліотеках є дірки.

1.2.4 Cross-Site Request Forgery (CSRF)

CSRF — це коли хакер змушує користувача робити щось на сайті, навіть якщо він не хоче і не знає про це. Він відправляє спеціальний запит, а браузер жертви виконує його автоматично, бо є сесійні куки. Через це можуть змінити пароль або зробити якусь операцію без згоди.

1.2.5 Server-Side Request Forgery (SSRF)

SSRF дозволяє хакеру змусити сервер робити запити куди не можна, наприклад у внутрішню мережу чи на інші сервіси. Це може привести до

викрадення даних чи сканування мережі. Зазвичай це трапляється, якщо сайт дозволяє вводити URL для завантаження чогось.

1.2.6 Конфігураційні помилки

І ще треба пам’ятати про конфігураційні помилки — це дуже часта причина вразливостей. Наприклад, якщо залишають стандартні паролі, відкриті порти, дають зайві права або не оновлюють систему [8]. OWASP каже, що це п’ята за важливістю загроза [9]. У 2024 році було знайдено більше 15 тисяч сайтів з такими проблемами в AWS [10].

Отже, розуміння цих вразливостей — дуже важливий крок для того, щоб правильно захищати веб-застосунки. Потрібно регулярно сканувати системи, використовувати сучасні інструменти, оновлювати програмне забезпечення і дотримуватися кращих практик. Безпека — це не раз і назавжди, це постійна робота. І чим краще ми знаємо про загрози, тим краще можемо їм протидіяти.

1.3 Сучасні інструменти аудиту безпеки (ZAP, Wapiti, Nmap+Vulscan, Nuclei, Nessus, Burp Suite)

Для захисту веб-застосунків зараз є багато різних інструментів аудиту безпеки. Вони допомагають автоматично шукати вразливості, робити аналіз і оцінювати ризики. Тут коротко описані деякі популярні і сучасні сканери, які часто використовують в кібербезпеці.

1.3.1 OWASP ZAP (Zed Attack Proxy)

ZAP — це дуже відомий і відкритий інструмент, який допомагає тестувати безпеку веб-застосунків. Він може працювати як автоматично, так і вручну, шукаючи типові вразливості — типу XSS, SQLi, CSRF і ще різні інші. Головна фішка ZAP у тому, що він гнучкий і підходить для різних випадків, наприклад, його можна інтегрувати в CI/CD, що зручно для розробників, які хочуть одразу перевіряти код на безпеку [11].

1.3.2 Wapiti

Wapiti — це безкоштовний веб-сканер, який працює за принципом “чорної скриньки”, тобто без знання, що всередині. Він шукає основні вразливості, такі як SQLi, XSS, LFI, RCE, SSRF. Wapiti робить детальні звіти і підтримує скрипти, щоб налаштовувати аналіз під свої задачі. Цей інструмент класно підходить для швидкої перевірки, бо не треба довго налаштовувати [12].

1.3.3 Nmap + Vulscan

Nmap — це відомий сканер мережі, він показує, які хости в мережі активні, які порти відкриті і які сервіси працюють. Якщо до нього додати Vulscan — це плагін, який автоматично перевіряє відомі вразливості (CVE) для знайдених сервісів. Так можна швидко оцінити, які ризики є в мережі і сервісах, що важливо для загального аудиту безпеки [13-14].

1.3.4 Nuclei

Nuclei — це сучасний і швидкий інструмент для пошуку вразливостей, який працює через шаблони. Він підтримує різні типи перевірок: HTTP, DNS, TCP, UDP, тому можна перевіряти багато протоколів. Він популярний через швидкість і можливість робити свої шаблони під конкретні задачі. Його часто використовують в автоматичних системах безпеки [15].

1.3.5 Nessus

Nessus — це комерційний, але дуже потужний сканер безпеки. Він робить глибокий аудит і має велику базу вразливостей, яку постійно оновлюють. Nessus підтримує перевірки з обліковими даними, що дає кращі результати. Його часто використовують у великих компаніях і навіть у державних структурах [16].

1.3.6 Burp Suite

Burp Suite — комплексний інструмент для тестування безпеки веб-застосунків, є безкоштовна і платна версія. Там є проксі, через який можна перехоплювати і змінювати трафік, інструменти для автоматичного і ручного пошуку вразливостей, а також можливість додавати розширення. Це фактично стандарт для професійних пентестерів [17].

Використання кількох таких інструментів разом дає змогу робити багаторівневий аудит безпеки. Наприклад, Nmap дає загальну картину мережі, Burp Suite — детально аналізує HTTP трафік, а Nuclei швидко шукає відомі проблеми по шаблонах. Якщо зібрати всі результати разом, можна отримати повну картину безпеки і краще захистити веб-застосунок.

1.4 Недоліки комерційних рішень, переваги відкритих альтернатив

Комерційні інструменти для перевірки безпеки веб-застосунків, такі як Nessus, Burp Suite Pro, Qualys або Rapid7 Nexpose, часто використовуються, бо в них багато різних функцій, постійно оновлюються бази вразливостей і ще є технічна підтримка. Це робить їх популярними. Але, як і у всього, у них є мінуси, через які чимало компаній починають дивитись в сторону безкоштовних, відкритих рішень.

1.4.1 Недоліки комерційних рішень

Ціна

Очевидно, що найбільший мінус — це скільки вони коштують. Наприклад, той же Burp Suite Pro — кілька сотень доларів щороку, а якщо говорити про більші системи типу Nessus або Qualys, то там ціна може бути взагалі десятки тисяч доларів, в залежності від того, скільки машин або юзерів буде підключено [18-20]. Такі витрати — це серйозна проблема для маленьких компаній або стартапів, які не мають зайвих грошей на кіберзахист.

Залежність від виробника

Більшість комерційних продуктів — це “закриті” системи, і користувачі залежать від того, як часто виробник оновлює програму і реагує на нові загрози. Якщо компанія припинить підтримку продукту або змінить умови ліцензії, користувачі можуть опинитись без потрібної підтримки, що породжує зайві ризики.

Обмежена кастомізація

Хоча деякі продукти дають змогу налаштовувати параметри, але більшість не дають доступу до внутрішньої логіки чи алгоритмів. Це ускладнює адаптацію під специфічні потреби бізнесу або під унікальні типи вразливостей.

Потрібні кваліфіковані спеціалісти

Щоб ефективно використовувати ці сканери, треба мати досвідчених фахівців, які розуміються на тонкощах інструменту. Без них можна отримати гору звітів з багатьма помилковими спрацьовуваннями, які потім важко аналізувати.

1.4.2 Переваги відкритих альтернатив

Безкоштовність і доступність

Відкриті інструменти, типу OWASP ZAP, Wapiti, Nmap, Nuclei, можна використовувати безкоштовно. Це дає можливість багатьом користувачам — студентам, дослідникам чи малому бізнесу — застосовувати їх без обмежень [21].

Гнучкість і відкритість коду

Відкритий код дозволяє кастомізувати інструменти, робити свої плагіни чи шаблони, інтегрувати їх з іншими системами. Це дуже корисно, якщо треба швидко підлаштуватися під нові загрози або конкретні задачі аудиту [21].

Активна спільнота

Багато відкритих проєктів підтримує жива спільнота розробників і користувачів, яка швидко реагує на нові вразливості, обмінюється досвідом і покращує інструменти. Це додає впевненості, що інструмент не закинуть.

Інтеграція з DevSecOps

Відкриті рішення легко вбудовуються у CI/CD процеси і підтримують різні формати виводу, API, що дає змогу їх автоматизувати і використовувати в сучасних workflow розробки.

Хоч комерційні рішення мають багатий функціонал і зручний інтерфейс, проблеми з високою ціною, тим що вони закриті й залежать від розробника, все ж створюють певні труднощі. Відкриті варіанти в цьому плані виграють: вони безкоштовні, їх можна налаштувати під себе, а ще за ними стоять активні спільноти, які швидко додають нові функції й виправлення. Через це багато хто віддає перевагу саме їм — особливо, якщо мова йде про невеликі компанії. У реальному житті найкращий варіант — комбінувати і ті, і ті інструменти. Так можна покрити максимум ризиків і нічого не пропустити.

Висновки до розділу 1

Сучасні веб-застосунки дуже часто стають мішенню для зловмисників, бо технології розвиваються швидко, а з ними й загрози. Вразливостей зараз багато — наприклад, ті ж SQL-ін’єкції, XSS, RCE, CSRF, SSRF та помилки в конфігурації. Для того щоб це все шукати, існує багато інструментів — і комерційних, і безкоштовних. Комерційні рішення, звісно, круті й функціональні, але коштують дорого й залежать від компаній, що їх зробили. Через це відкриті альтернативи виглядають краще для багатьох, особливо для тих, у кого немає великого бюджету. В ідеалі, треба комбінувати обидва підходи, щоб максимально добре захистити веб-застосунок.

2 ТЕОРЕТИЧНІ ОСНОВИ АВТОМАТИЗАЦІЇ АУДИТУ БЕЗПЕКИ

2.1 Автоматизований аудит і його значення

Автоматизований аудит безпеки — це процес використання спеціалізованих програмних інструментів для систематичного та регулярного аналізу інформаційних систем, мереж і застосунків на предмет виявлення вразливостей і загроз. Це набагато швидше і зручніше, ніж перевіряти все вручну. Основне завдання такого аудиту полягає у швидкому, ефективному та об’єктивному виявленні слабких місць у захисті без необхідності ручного втручання на кожному етапі.

Головне, що вони перевіряють одразу купу всього — від багів у коді веб-застосунків до проблем у налаштуваннях серверів, протоколів і мереж. Тобто, це не просто одноразова перевірка, а постійний контроль, що важливо в умовах, коли все швидко змінюється.

Ще одна зручна функція — звіти. Інструменти самі генерують звіти з описом знайдених проблем, пріоритетами і порадами, що з цим робити. Причому ці звіти можна адаптувати — наприклад, технічні для розробників і більш загальні для керівництва.

Плюс автоматизації в тому, що її легко масштабувати: можна перевіряти одразу багато систем або великі шматки коду. Це дуже зручно для великих компаній. І ще — автоматизація зменшує ризик помилок, які часто бувають при ручній перевірці, і дає змогу фахівцям зосередитися на аналізі результатів, а не на рутині.

Але, звісно, не все так ідеально. Автоматичні інструменти не завжди можуть виявити складні логічні баги чи нестандартні вразливості. Тому повністю покладатися тільки на автоматизацію — не варіант. Найефективніший варіант — поєднувати автоматзовану перевірку з ручним аудитом і досвідом експертів.

Підсумовуючи: автоматизований аудит — це потужний інструмент, який серйозно економить час і допомагає вчасно виявляти загрози. Особливо актуально це зараз, коли кіберризики ростуть із шаленою швидкістю.

2.2 Підходи до інтеграції сканерів в єдину систему

Щоб аудит безпеки працював ефективніше, часто треба поєднувати кілька сканерів в одну систему. У кожного інструменту свої особливості, сильні сторони і формат виводу, тому якщо об'єднати їх у спільне середовище, можна отримати більш повну та точну картину стосовно кіберзахисту продукта.

Один із найзручніших способів — зробити все модульно. Тобто кожен сканер підключається як окремий блок із чіткими правилами запуску та повернення результатів. Система просто запускає всі модулі (одночасно або по черзі), збирає звіти в єдиному форматі — і все працює разом. Додати новий інструмент у таку систему просто: наприклад, можна легко підключити Wapiti, OWASP ZAP, Nmap, Nuclei та інші.

Ще один популярний підхід — використовувати API або CLI (командний рядок). Наприклад, OWASP ZAP має повноцінний REST API, а Nuclei чи Wapiti можна запускати через команду в терміналі [22]. Це зручно, бо можна легко автоматизувати сканування за допомогою скриптів.

Також є варіант використання посередницького прошарку (middleware) — це як «перекладач» між головною системою та сканерами. Він може обробляти логіку запуску, форматувати звіти в єдиний вигляд, вести логування і слідкувати, щоб не було збоїв чи конфліктів. У Python для запуску зовнішніх сканерів часто використовують subprocess, який дозволяє запускати програми прямо з коду і одразу отримувати від них відповіді, ніби в терміналі.

Щоб не встановлювати кожний інструмент вручну, часто також все запускають через Docker. Це дозволяє створити окреме середовище для кожного

сканера з усіма залежностями, і відсутністю конфліктів. Плюс — це безпечніше й простіше масштабувати.

Якщо сканерів багато, і всі запускаються в різний час, зручно використовувати черги повідомлень, наприклад RabbitMQ. Так система сама вирішує, коли і що запускати, щоб усе працювало плавно, а не навалювалося одразу [23].

Ще один варіант — підхід ETL: спочатку витягуємо дані зі сканерів (Extract), потім приводимо все до одного формату, наприклад JSON (Transform), а далі зберігаємо в базу або аналізуємо (Load). Це допомагає не загубитись у звітах і легко працювати з результатами.

А щоб не розробляти свою систему з нуля, можна скористатися готовими платформами, як Vulners чи DefectDojo — вони вже вміють збирати результати з різних інструментів і красиво їх показувати [24-25].

Отже, щоб зробити круту і зручну систему аудиту безпеки, можна поєднувати модульну структуру, API або CLI-інтерфейси, контейнери (Docker), обробку результатів через ETL і, за потреби, додавати черги. Тоді все буде працювати гнучко, стабільно і масштабовано.

2.3 Централізований запуск і керування сканерами (через CLI/API/subprocess)

Щоб не запускати кожен сканер вручну і не морочитися з налаштуваннями, зручно зробити централізоване управління — коли всі інструменти запускаються з однієї точки, автоматично, за розкладом або при певній події. Це дозволяє уникнути плутанини з результатами, дублювання, чи просто людських помилок.

Найпоширеніші способи запуску сканерів — через CLI, API або subprocess у Python.

CLI (Command Line Interface) — це найпростіший і найзручніший спосіб запуску сканерів. Такі інструменти як Nmap, Nuclei чи Wapiti працюють прямо з

терміналу через команди. Їх легко вписати у скрипти, запускати за розкладом через cron, або використовувати разом із системами типу Celery, якщо треба керувати запуском автоматично [26-27]. Наприклад, щоб знайти вразливості через Nmap, достатньо простої команди як на рисунку 2.1:

```
nmap 192.168.56.101 -sS -sV -p- --script vulscan/vulscan.nse
```

Рисунок 2.1 – Запуск Nmap зі скриптом Vulscan

API — це ще потужніший варіант, особливо для інструментів із веб-інтерфейсом, таких як OWASP ZAP або Burp Suite Enterprise. Наприклад, ZAP має повноцінний REST API, через який можна запускати сканування, перевіряти їхній статус, витягувати знайдені проблеми, будувати звіти — і все це без участі людини [22]. Це зручно, якщо ви хочете зробити власну систему на Flask або Django, яка сама всім керує. Приклад запуску через API ZAP на рисунках 2.2 та рисунку 2.3:

```
# Spider the target
scan_id = zap.spider.scan(target_url)
```

Рисунок 2.2 – Запуск пасивного сканування ZAP

```
# Start scan with specific policy
ascan_id = zap.ascan.scan(
    target_url,
    scanpolicyname=policy_name,
    recurse=True,
    inscopeonly=False
)
```

Рисунок 2.3 – Запуск активного сканування ZAP

subprocess у Python — це універсальний спосіб запускати будь-які програми з коду [28]. Підходить, коли інструмент не має API, або просто зручно все запускати з Python. Наприклад запуск Nuclei на рисунку 2.4:

```
with tempfile.NamedTemporaryFile(mode='w+', delete=False) as tmp_output:
    output_path = tmp_output.name

templates_path = os.path.join("nuclei", "nuclei-templates")

cmd = [
    NUCLEI_EXECUTABLE_PATH,
    "-u", target_url,
    "-t", templates_path,
    "-json-export", output_path
]
subprocess.run(cmd, check=True)
```

Рисунок 2.4 – Запуск Nuclei через subprocess

Це гнучкий варіант: можна додати логіку перевірок, умови запуску, обробку помилок, таймаути, і навіть виводити результати в UI.

У більш складних системах ці підходи часто поєднують: де є API — використовують його (наприклад, ZAP, Nessus), де прості утиліти — використовують CLI (наприклад, Nuclei чи Wapiti), а subprocess — як універсальний варіант для всього іншого.

Щоб централізована система працювала масштабовано — особливо коли сканувань багато або користувачів кілька — використовують черги завдань, як-от Celery або Redis Queue [27, 29]. Вони дозволяють запускати сканери не всі разом, а по черзі або паралельно, в залежності від навантаження. Це допомагає уникнути надмірного навантаження і стабілізує роботу всієї системи.

Отже, ефективне управління сканерами базується на поєднанні різних технічних підходів — CLI, API та subprocess. Разом вони дають повний контроль, гнучкість і можливість масштабувати систему під реальні потреби безпеки.

2.4 Обробка результатів: нормалізація, сортування, фільтрація

Після завершення сканування ми зазвичай отримуємо необроблені дані — багато зайвого, різні формати, іноді дублі або просто важко читабельний

вивід. Щоб з цим можна було нормально працювати, потрібно пройти три важливі етапи: нормалізацію, сортування та фільтрацію. Це допомагає навести порядок у результатах і зробити їх придатними для аналізу, звітів або інтеграції в інші системи (типу SIEM чи SOC).

Нормалізація — це коли ми приводимо все до єдиного формату. Наприклад, Nmap, Nuclei, ZAP чи Nessus повертають результати по-різному: хтось у JSON, хтось в XML, інші просто як текст. Щоб потім все це об’єднати — треба зробити спільну структуру.

Що зазвичай включає нормалізація:

- однакові назви типів вразливостей (наприклад, SQL Injection, XSS, CSRF);
- уніфіковані рівні ризику (low, medium, high, critical);
- нормальний формат часу, наприклад, ISO 8601 (2025-05-30T14:12:00Z).

Завдяки цьому можна будувати універсальні звіти, порівнювати результати з різних сканерів і передавати їх у зовнішні системи. Добра практика — орієнтуватися на стандарти, як-от CVSS або CWE.

Сортування допомагає швидко знайти найважливіше. Наприклад, якщо ми просканували сотні IP-адрес, важливо одразу побачити, де знайдено критичні вразливості.

Основні варіанти сортування:

- за рівнем ризику (спочатку critical, потім high, і т.д.);
- за IP або доменом (щоб групувати по цілях);
- за типом сканера (щоб бачити, хто що знайшов);
- за датою (актуально, якщо сканування повторюється).

Це можна реалізувати прямо в базі даних (через ORDER BY) або в коді — наприклад, у Python з sorted() і кастомним ключем.

Фільтрація — ще один важливий крок, який дозволяє відсікти зайве:

- дублікати (одна і та ж вразливість знайдена кількома сканерами);
- false positives (помилкові спрацювання);
- низькоризикові речі, які поки не критичні.

У коді це виглядає як набір простих правил, типу:

- "виключити все з рівнем low",
- "не показувати повтори з однаковим CVE",
- "відкинути результати без чіткої експлуатації".

У складніших системах навіть можна підключити машинне навчання для покращення якості — наприклад, щоб автоматично виявляти аномальні або підозрілі результати. Або дати користувачу самостійно відфільтрувати все через UI.

Отже, без обробки результати сканування — це хаос. Нормалізація наводить порядок у форматах, сортування допомагає з пріоритетами, а фільтрація прибирає необов’язкове. Разом вони роблять систему зрозумілою, ефективною і здатною знаходити справді важливі проблеми.

2.5 Вивід результатів: HTML, PDF, API

Коли сканування завершено, далі постає питання: як зручно показати результати? В ідеалі — так, щоб це було не лише інформативно, але й зручно для перегляду, звітності чи автоматичної обробки. Найпоширеніші варіанти — HTML, PDF або API-вивід. У кожного — свої переваги, і вибір залежить від того, для кого й навіщо цей звіт.

2.5.1 HTML-вивід

HTML-звіт — зручний і гнучкий варіант для швидкого перегляду в браузері. Тут можна зрозуміло подати інформацію з кольоровими індикаторами ризиків (червоний — критично, жовтий — середній і т.д.), таблицями, фільтрами,

сортуванням і вкладками. Часто додають посилання прямо на сторінки CVE чи OWASP, щоб одразу перейти до деталей.

HTML-звіти легко вбудовуються у внутрішні панелі компанії або CI/CD пайплайни. Це особливо корисно для аналітиків, яким треба швидко переглянути результати без зайвих PDF-документів. Наприклад, Burp Suite, ZAP чи Nuclei вміють генерувати HTML звіти або напряму через команду в терміналі, або через GUI [30].

2.5.2 PDF-звіти

PDF-звіт — це вже класика для формального документування. Такий формат підходить для архівування, надсилання менеджерам, прикріплення до аудит-документації тощо.

Переваги PDF-виводу:

- фіксоване форматування — вміст не залежить від браузера чи ОС;
- можливість включення титульної сторінки, змісту, підписів і логотипів компанії;
- інтеграція з автоматизованим звітогенератором, наприклад, через бібліотеки Python (ReportLab, WeasyPrint) чи Node.js (puppeteer, pdfkit).

У багатьох системах, включаючи Nessus і Nikto, присутня підтримка експорту в PDF через GUI або API [31].

2.5.3 API-вивід

Для інтеграції з іншими системами безпеки, DevOps-пайплайнами або SIEM/EDR-рішеннями, результати сканування повинні бути доступними через RESTful API.

Типова структура відповіді API містить:

- метадані про ціль;
- список вразливостей з CVSS, описами, рівнем ризику;
- дату виявлення;
- рекомендації щодо усунення.

API-вивід дозволяє:

- автоматично оновлювати статус вразливостей у зовнішніх системах (наприклад, Jira, Splunk);
- інтегрувати аудиторський результат у CI/CD (GitLab, Jenkins);
- реалізувати інтерактивну панель керування (dashboard).

У підсумку: варіативність форматів — це добре. HTML — для швидкого перегляду, PDF — для офіційної звітності, API — для інтеграції та автоматизації. Система аудиту має давати вибір залежно від того, хто читає звіт — аналітик, менеджер чи скрипт.

2.6 Оцінка переваг автоматизованого підходу

З ускладненням архітектури веб-застосунків і зростанням числа атак автоматизованими ботнетами, ручні перевірки більше не відповідають вимогам часу. Вони не тільки повільні й обмежені масштабом, а й надто залежні від навичок конкретного фахівця. Автоматизований підхід усуває ці бар'єри: він забезпечує високу швидкість, стабільність та повторюваність, зменшуючи час реакції на загрози та дозволяючи командам концентруватися на пріоритетних інцидентах.

2.6.1 Швидкість і масштаб

Сучасні фреймворки, зокрема Nuclei, реалізують паралельне сканування тисяч цілей за секунди, використовуючи шаблонізовану логіку. Це забезпечує

гнучку перевірку типових вразливостей (XSS, SSRF, LFI тощо) та CVE-специфічних атак у масштабах, недосяжних для людини. Для організацій з мікросервісною архітектурою чи великою кількістю субдоменів це відкриває шлях до повного охоплення без шкоди продуктивності. Це критично для компаній, що мають десятки або сотні веб-сервісів, особливо у мікросервісних архітектурах. Там, де людині потрібні години — автоматизований сканер справляється за хвилини.

2.6.2 Повторюваність без помилок

Автоматичні сканери не втомлюються й не пропускають очевидне. Один і той самий набір перевірок виконується щоразу однаково, незалежно від рівня досвіду користувача. Інструменти на кшталт OWASP ZAP або Nessus забезпечують стабільну якість перевірок завдяки шаблонній логіці й регулярним оновленням бази вразливостей.

2.6.3 Інтеграція в CI/CD — DevSecOps на практиці

Найбільший плюс — автоматизований аудит легко інтегрується в CI/CD пайплайн. Це означає, що безпека перевіряється ще до виходу коду в продакшн. OWASP рекомендує такий підхід відомий як Security by Design — не «латати дірки», а вбудовувати безпеку в сам процес розробки [32].

2.6.4 Менше витрат — більше результату

Замість того щоб навантажувати дорогих спеціалістів рутинними сканами, команди можуть сфокусуватись на аналізі критичних випадків. Це дає змогу

зекономити ресурси, особливо для стартапів та малого бізнесу, де DevSecOps-команди часто відсутні.

2.6.5 Повноцінне логування і звітність

Більшість автоматизованих систем генерують структуровані логи, дозволяють API-доступ, зберігають історію перевірок. Це значно полегшує не лише створення аналітики, а й автоматизує побудову трендових графіків, кореляцію подій та формування звітів відповідно до вимог ISO/OWASP або корпоративних стандартів.

2.6.6 Адаптивність під специфіку компанії

Інструменти з відкритим кодом — зокрема Nuclei і Wapiti — підтримують глибоку кастомізацію: створення власних шаблонів, підключення скриптів на Python/Bash, інтеграцію з внутрішніми API. Це дозволяє враховувати галузеві особливості, внутрішні політики безпеки, а також нестандартні конфігурації інфраструктури.

У такому вигляді автоматизований аудит перетворюється на невід'ємну частину корпоративної ІБ-стратегії. Це не просто зручний інструмент, а основа для гнучкої, масштабованої та проактивної безпеки. Його впровадження дозволяє не лише економити ресурси, а й підвищити рівень готовності до інцидентів на рівні розробки, тестування й експлуатації. У цифрову епоху це вже не перевага — це умова виживання.

Висновки до розділу 2

Автоматизація аудиту безпеки вже давно перестала бути просто корисною — зараз це реально необхідна річ для захисту веб-застосунків. У цьому розділі була спроба розібратись, як саме працює цей підхід: які інструменти можна використовувати, як їх підключати до пайплайнів CI/CD, як керувати ними через CLI або API, ну і що потім робити з результатами перевірок.

Очевидна перевага — це швидкість і масштаб. Коли треба просканувати не один-два сайти, а сотні — автоматизація рятує. Дані можна отримати швидко, ще й у більш-менш зручному вигляді (іноді, щоправда, доводиться трохи поморочитись з обробкою). І так, стабільність є — якщо налаштовано все правильно. Однак бувають ситуації, коли щось падає або дає фальшиві спрацьовування, особливо при складних конфігураціях.

Ще один плюс — економія ресурсів. Автоматизовані інструменти справді знімають з фахівців рутину. Але варто розуміти, що ці інструменти самі по собі все не зроблять — іноді потрібне втручання, щоб щось підправити, налаштувати чи перевірити вручну. Для стартапів і невеликих компаній — це прямо must-have, бо не завжди є команда, яка буде займатись безпекою повний робочий день.

В цілому, автоматизація аудиту — це логічний крок уперед. Вона не ідеальна, потребує налаштування і розуміння, як все працює, але дозволяє реально зекономити час і бути більш впевненими у своїй безпеці. Як на мене, це одна з тих речей, які точно варто мати, якщо хочеш бути готовим до реальних загроз.

3 РОЗРОБКА ВЕБ-ЗАСТОСУНКУ ДЛЯ АВТОМАТИЗОВАНОГО АУДИТУ БЕЗПЕКИ

3.1 Архітектура рішення (Flask + Bootstrap + subprocess)

У цьому проєкті було зібрано просту, але гнучку систему для автоматизованого аудиту безпеки веб-застосунків. Основна ідея — об’єднати популярні інструменти безпеки в одному веб-застосунку зі зрозумілим інтерфейсом. Для цього використано Flask (бекенд), Bootstrap (фронтенд), і модуль subprocess, щоб запускати сканери з коду.

3.1.1 Бекенд на Flask

Flask — це легкий Python-фреймворк, який дозволяє швидко писати веб-застосунки [33]. Тут він відповідає за обробку запитів: запуск сканерів, збирання результатів, створення звітів і все інше, що відбувається на стороні сервера. Зручно те, що можна легко додавати нові маршрути — наприклад, окремі для кожного сканера.

Самі сканери (типу Wapiti або Nuclei) запускаються як окремі процеси через subprocess. Це дозволяє не зв’язуватись із їхнім API (яке іноді взагалі відсутнє), а просто викликати їх так, як у терміналі. Кожен сканер має свій обробник, який запускає його, чекає, поки той відпрацює, і парсить результати.

3.1.2 Фронтенд на Bootstrap

Для інтерфейсу використав Bootstrap 5. Нічого надто складного: є форма, в якій можна ввести ціль (наприклад, сайт), вибрати сканери, натиснути

"Сканувати" і побачити результат. Bootstrap допоміг зробити все адаптивним і зручним навіть без глибокого занурення в CSS або JavaScript. Є модальні вікна, таблиці, трохи динаміки — але без перевантаження.

3.1.3 Subprocess як міст між Flask і сканерами

Модуль subprocess тут — ключовий. Він дає змогу запускати сторонні CLI-інструменти, отримувати їхній вивід, оброблювати помилки, і передавати все це назад у веб-застосунок. Це рішення вийшло універсальним: не важливо, який сканер, головне, щоб він міг працювати через командний рядок. Flask запускає сканер, зчитує результат і віддає його у форматі JSON — далі фронтенд це вже гарно відображає.

3.1.4 Переваги архітектури

Ця архітектура вийшла простою, але реально робочою. Вона дозволяє легко додавати нові сканери, бо все працює через subprocess. А Flask і Bootstrap дають змогу створити нормальний веб-інтерфейс, не ускладнюючи проект. Все модульно — якщо треба, можна змінити одну частину, не чіпаючи інші.

Загалом, система вийшла зручною і для користувача, і для розробника. Вона дозволяє об’єднати кілька різних інструментів безпеки в одному місці й автоматизувати перевірки без зайвих дій.

3.2 Інтерфейс користувача: запуск, вибір модулів, перегляд результатів

Інтерфейс користувача в розробленому веб-застосунку є ключовим елементом, що забезпечує просту та ефективну взаємодію з системою без необхідності використовувати командний рядок чи конфігураційні файли. Основна мета UI (User Interface) — зробити процес аудиту безпеки інтуїтивно зрозумілим навіть для користувачів без глибокої технічної підготовки.

3.2.1 Головна сторінка і запуск сканування

На головній сторінці все по-простому: є поле, куди можна ввести адресу сайту або IP, і список чекбоксів — вибір інструментів, які будуть сканувати та їх модулі дял скануваннння конкретних вразливостей, як показано на рисунках 3.1-3.3.

Наприклад:

- Nmap + Vulscan
- OWASP ZAP
- Wapiti
- Nuclei
- Google Dorking

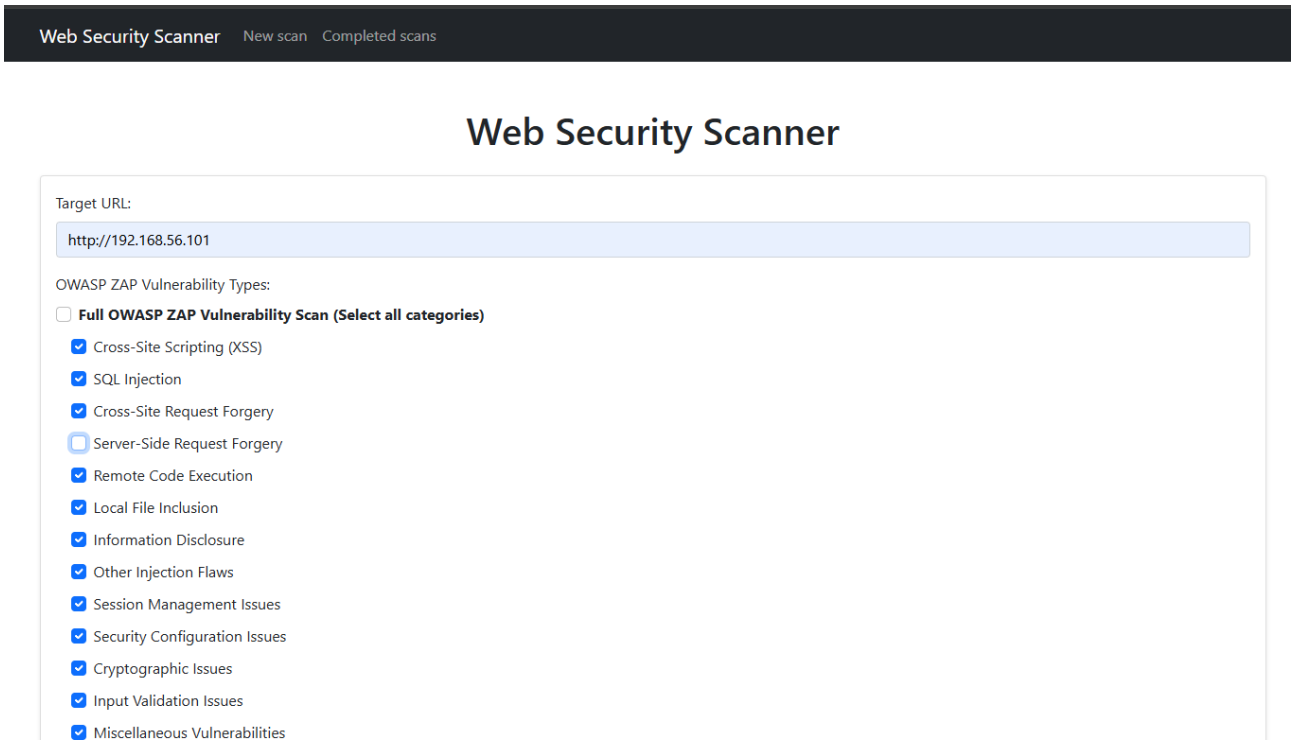


Рисунок 3.1 – Вибір сканування конкретних вразливостей для ZAP

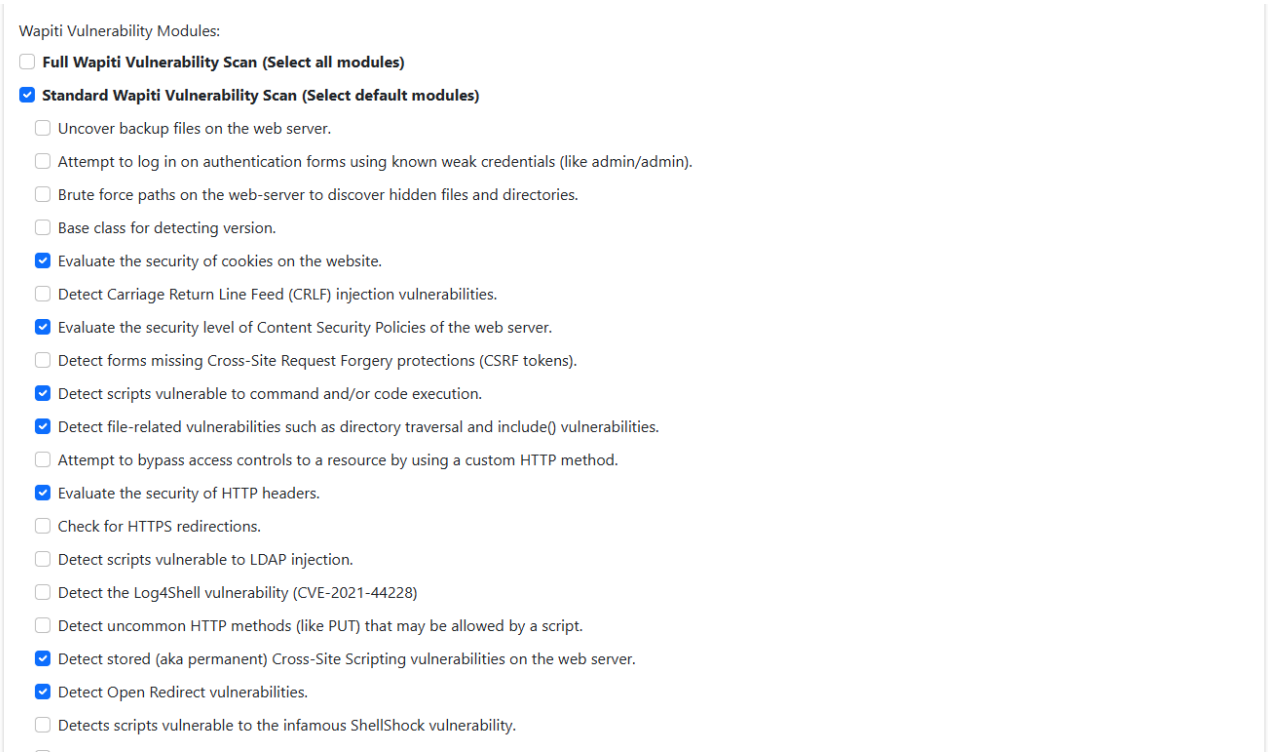


Рисунок 3.2 – Вибір сканування конкретних вразливостей для Wapiti

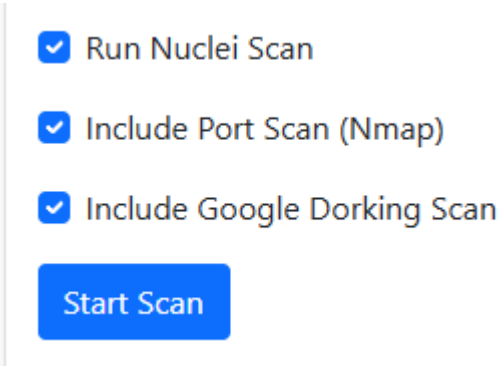


Рисунок 3.3 – Вибір сканування з Nuclei, Nmap та Google Dorking

Можна вибрати один, кілька або всі одразу. Це зручно, бо іноді треба швидко просканувати одним інструментом, а іноді — провести повну перевірку. Після натискання кнопки “Start Scan” Flask запускає фонові процеси через subprocess для обраних сканерів.

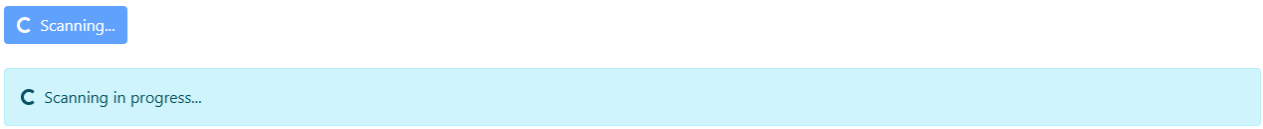


Рисунок 3.4 – Після натискання кнопки “Start Scan”

3.2.2 Перегляд результатів і історія

Коли сканування завершено, користувач бачить результати по кожному інструменту окремо. Для зручності вони виводяться в списках у вигляді карток та таблицях. Наприклад, Nmap показує відкриті порти і потенційні вразливості з Vulscan, ZAP — XSS, SQLi, CSRF і подібні речі.

Шляхи до звітів зберігаються в SQLite, тому є можливість подивитися історію сканувань — це теж корисно.

Що ще можна зробити:

- Завантажити звіт у PDF
- Відсортувати вразливості за рівнем критичності

- Подивитися технічні деталі кожної знайденої вразливості

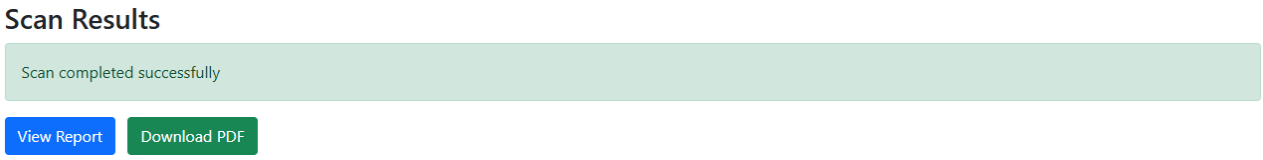


Рисунок 3.5 – Після завершення сканування

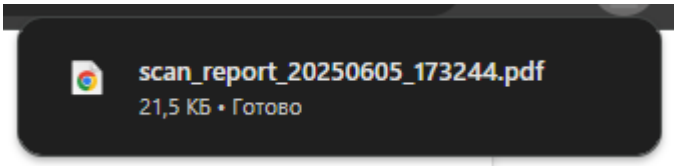


Рисунок 3.6 – Завантаження звіту у PDF

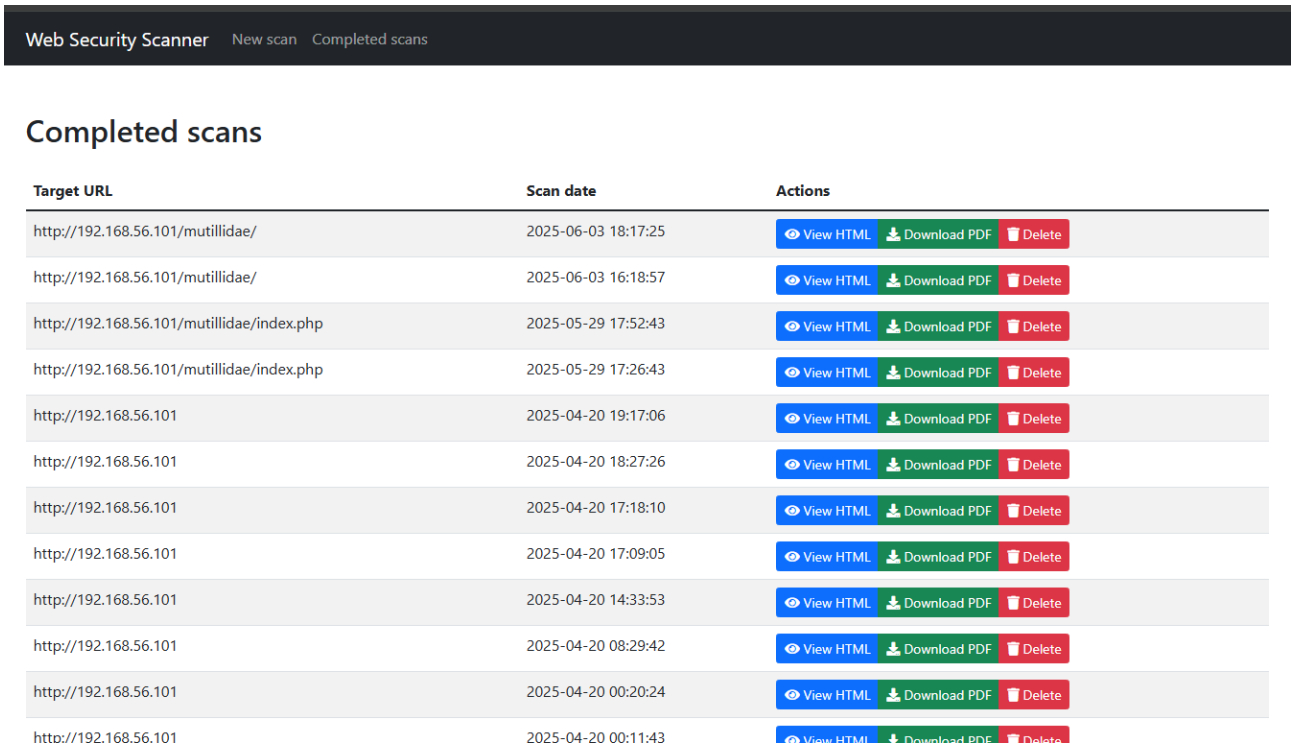


Рисунок 3.7 – Перегляд історії сканувань

Рисунок 3.8 – Повідомлення після видалення звіту

Загалом, інтерфейс зроблений з розрахунком на зручність та простоту. Налаштування майже не потрібні, все можна зробити з інтерфейсу без командного рядка. Це дозволяє навіть нетехнічним користувачам перевірити сайт на вразливості швидко й без технічних деталей.

3.3 Інтеграція сканерів: Nmap + Vulscan, OWASP ZAP, Wapiti, Nuclei, Google Dorking API

Однією з ключових цілей створення веб-застосунку для автоматизованого аудиту безпеки було забезпечення одночасного запуску декількох сканерів із різними принципами роботи та форматами виводу. Це дозволяє користувачу отримувати повну й різнобічну інформацію про безпеку цільового веб-застосунку. У цьому підрозділі розглянуто, як саме відбувається інтеграція кожного інструмента в межах єдиної платформи.

3.3.1 Інтеграція Nmap + Vulscan

Nmap — це один із найпотужніших сканерів портів, який також дозволяє виявляти версії сервісів, операційні системи, конфігураційні помилки та інші параметри. В застосунку реалізовано запуск Nmap із додатковими ключами -sS (TCP SYN сканування) -sV (визначення версій сервісів) -p- (сканування всіх портів) та --script= vulscan/vulscan.nse, що дозволяє використовувати модуль Vulscan — базу знань вразливостей, яка корелює з даними з Exploit-DB, CVE та інших джерел.

Запуск реалізовано через Python-модуль subprocess, який дозволяє виконати команду як на рисунку 3.9.

```
ip_address = get_ip(target)
nm = nmap.PortScanner()
nm.scan(ip_address, arguments='-sS -sV -p- --script vulscan/vulscan.nse')
```

Рисунок 3.9 – Запуск Nmap з Vulscan через subprocess

Результати зчитуються у stdout, після чого парсяться для подальшої обробки.

```
ports = []
for host in nm.all_hosts():
    for proto in nm[host].all_protocols():
        for port in nm[host][proto]:
            port_info = nm[host][proto][port]
            ports.append({
                'port': port,
                'protocol': proto,
                'service': port_info.get('name', ''),
                'state': port_info.get('state', ''),
                'product': port_info.get('product', ''),
                'version': port_info.get('version', ''),
                'extrainfo': port_info.get('extrainfo', ''),
                'vulscan_results': parse_vulscan_output(port_info.get('script', {}).get('vulscan', ''))
            })
```

Рисунок 3.10 – Структура результатів Nmap з Vulscan

Система виділяє знайдені відкриті порти, сервіси та відповідні CVE, що додаються до загального звіту.

3.3.2 Інтеграція OWASP ZAP

OWASP ZAP (Zed Attack Proxy) — це один з найбільш популярних сканерів для динамічного аналізу веб-застосунків. Його інтеграція здійснюється через REST API, який ZAP надає після запуску у фоновому режимі.

```
zap = ZAPv2(apikey=[redacted], proxies={'http': 'http://127.0.0.1:8080', 'https': 'http://127.0.0.1:8080'})
```

Рисунок 3.11 – Інтеграція ZAP через REST API

Завдяки API-архітектурі ZAP, результат автоматично зберігається в структурованому JSON-форматі, що спрощує подальшу нормалізацію даних.

3.3.3 Інтеграція Wapiti

Wapiti — легкий сканер, написаний на Python, призначений для виявлення вразливостей у веб-застосунках за допомогою технік "чорної скриньки". Інтеграція реалізована запуск Wapiti як окремого процесу на рисунку 3.12.

```
report_path = f"static/reports/wapiti_{datetime.now().strftime('%Y%m%d_%H%M%S')}.json"
command = ["wapiti", "-u", target_url, "-f", "json", "-o", report_path]

if selected_modules:
    for module in selected_modules:
        command.extend(["-m", module])

subprocess.run(command, capture_output=True, text=True, check=True)
```

Рисунок 3.12 – Запуск Wapiti через subprocess

Результати зберігаються у файлі JSON, який завантажується, парситься, і на основі цього формується частина загального звіту. Wapiti дозволяє знаходити XSS, SQLi, CSRF та інші поширені вразливості.

3.3.4 Інтеграція Nuclei

Nuclei — сучасний високошвидкісний сканер шаблонів вразливостей від проєкту ProjectDiscovery. Його головна особливість — використання YAML-шаблонів для перевірки великого спектру CVE, конфігураційних проблем, сервісів тощо.

Запуск також виконується через subprocess показаний на рисунку 3.13.

```
with tempfile.NamedTemporaryFile(mode='w+', delete=False) as tmp_output:
    output_path = tmp_output.name

templates_path = os.path.join("nuclei", "nuclei-templates")

cmd = [
    NUCLEI_EXECUTABLE_PATH,
    "-u", target_url,
    "-t", templates_path,
    "-json-export", output_path
]
subprocess.run(cmd, check=True)
```

Рисунок 3.13 – Запуск Nuclei через subprocess

Вивід записується в реальному часі, парситься та інтегрується у фінальний формат. Особливістю є підтримка величезної бази шаблонів, що регулярно оновлюється спільнотою.

3.3.5 Інтеграція Google Dorking API

На відміну від традиційних сканерів, Google Dorking використовує можливості пошукових запитів Google для виявлення індексованих конфіденційних сторінок, файлів, відкритих адмін-панелей та інших вразливостей [34].

Інтеграція реалізована через кастомний Python-скрипт, що виконує серію пошукових запитів за шаблонами як на рисунку 3.14.

```
service = build("customsearch", "v1", developerKey=api_key)
dorks = [
    f'site:{target_url} ext:sql | ext:log | ext:bak | ext:old | ext:env',
    f'site:{target_url} intitle:"index of" "backup"',
    f'site:{target_url} inurl:admin',
    f'site:{target_url} inurl:login',
    f'site:{target_url} intitle:"Admin Login"',
    f'site:{target_url} inurl:phpmyadmin',
    f'site:{target_url} inurl:config',
    f'site:{target_url} intitle:"Index of /" +etc',
    f'site:{target_url} "Apache/2.4.1" inurl:status',
    f'site:{target_url} "Warning: mysql_" filetype:php',
    f'site:{target_url} intitle:"index of" "password"',
    f'site:{target_url} confidential',
    f'site:{target_url} "s3.amazonaws.com"',
]
```

Рисунок 3.14 – Шаблони для Google Dorking

Використовується офіційний Google Custom Search API, що дозволяє динамічно отримувати відповіді у форматі JSON. Отримані результати додаються до звіту в окремому блоці “Google Dorking Results”.

Усі інтегровані сканери реалізують різні аспекти безпеки:

- Nmap+Vulscan — мережевий рівень;
- ZAP, Wapiti, Nuclei — прикладний рівень;
- Google Dorking — інформаційний рівень.

Така багаторівнева структура дозволяє досягти максимальної глибини перевірки, а гнучкість налаштувань та об'єднання результатів забезпечують високу практичну цінність застосунку для IT-фахівців та пентестерів.

3.4 Генерація HTML/PDF звітів, збереження історії в SQLite

Щоб мати повну картину результатів сканування, дуже важливо вміти зберігати все, що було знайдено, у зручному вигляді. У створеному веб-

застосунку реалізовано можливість створення звітів у форматах HTML і PDF, а також збереження історії перевірок у базі даних SQLite.

3.4.1 HTML-звіт

HTML — це основний формат, який відображається прямо в браузері. Звіт автоматично заповнюються даними результатів сканування.

Він включає:

- заголовок із URL цілі та датою перевірки;
- таблиці знайдених вразливостей, підсвічені кольорами за рівнем ризику;
- секції з результатами кожного сканера;
- короткі поради, як усунути знайдені проблеми.

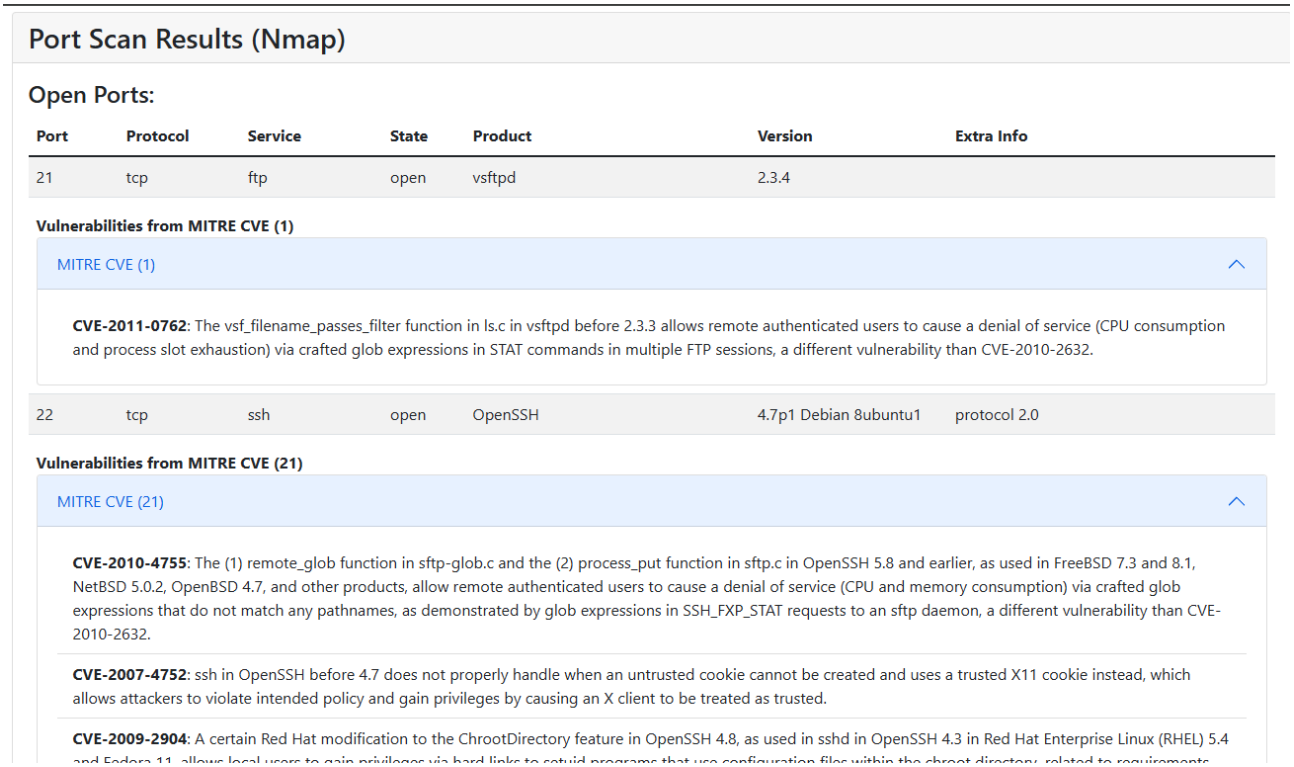


Рисунок 3.15 – Приклад результатів сканувааннння Nmap з Vulscan

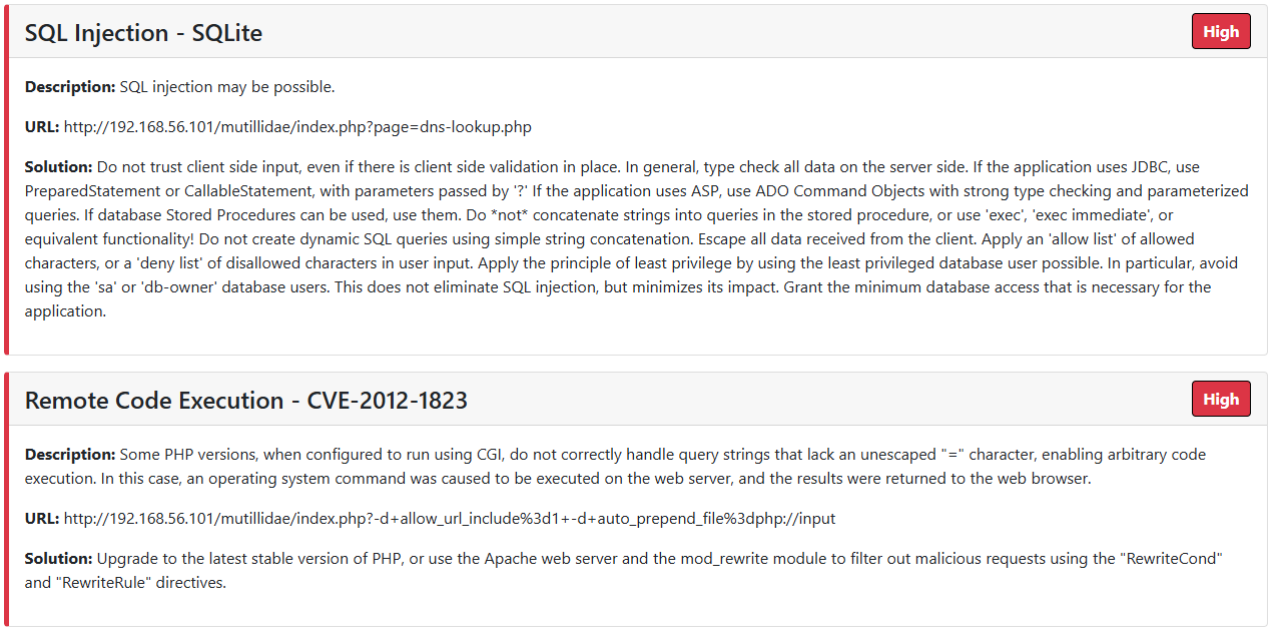


Рисунок 3.16 – Приклад результатів сканування ZAP

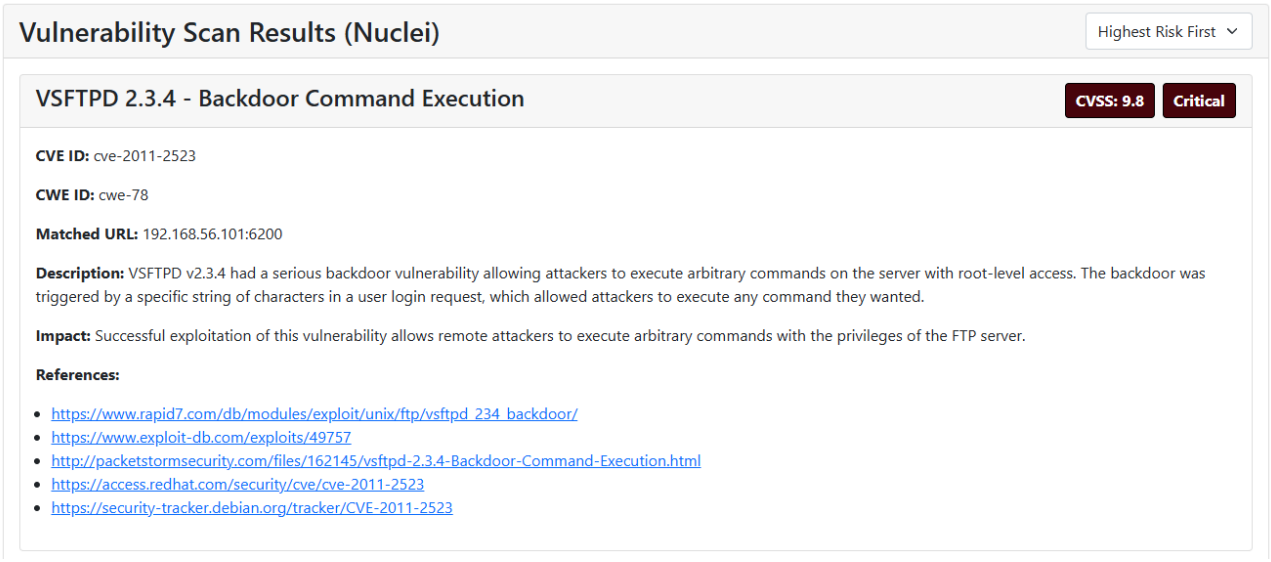
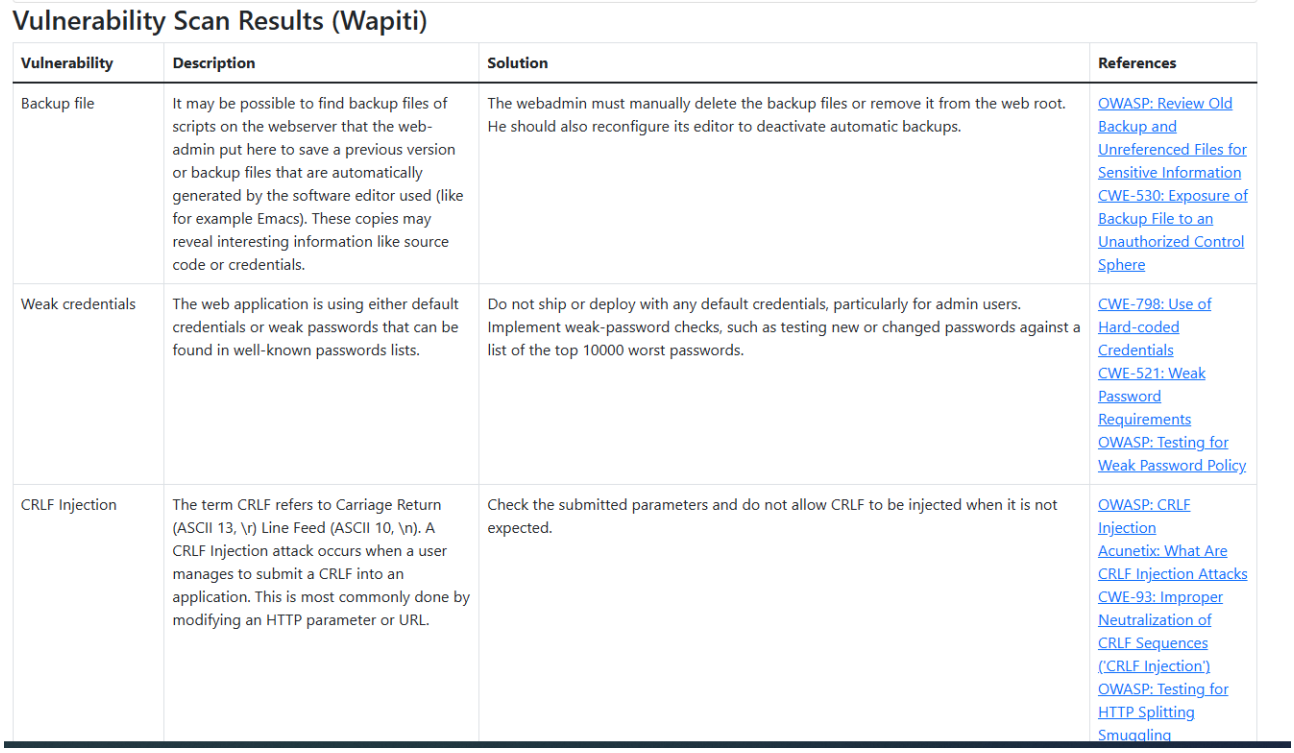


Рисунок 3.17 – Приклад результатів сканування Nuclei



Vulnerability Scan Results (Wapiti)

Vulnerability	Description	Solution	References
Backup file	It may be possible to find backup files of scripts on the webserver that the web-admin put here to save a previous version or backup files that are automatically generated by the software editor used (like for example Emacs). These copies may reveal interesting information like source code or credentials.	The webadmin must manually delete the backup files or remove it from the web root. He should also reconfigure its editor to deactivate automatic backups.	OWASP: Review Old Backup and Unreferenced Files for Sensitive Information CWE-530: Exposure of Backup File to an Unauthorized Control Sphere
Weak credentials	The web application is using either default credentials or weak passwords that can be found in well-known passwords lists.	Do not ship or deploy with any default credentials, particularly for admin users. Implement weak-password checks, such as testing new or changed passwords against a list of the top 10000 worst passwords.	CWE-798: Use of Hard-coded Credentials CWE-521: Weak Password Requirements OWASP: Testing for Weak Password Policy
CRLF Injection	The term CRLF refers to Carriage Return (ASCII 13, \r) Line Feed (ASCII 10, \n). A CRLF Injection attack occurs when a user manages to submit a CRLF into an application. This is most commonly done by modifying an HTTP parameter or URL.	Check the submitted parameters and do not allow CRLF to be injected when it is not expected.	OWASP: CRLF Injection Acunetix: What Are CRLF Injection Attacks CWE-93: Improper Neutralization of CRLF Sequences ('CRLF Injection') OWASP: Testing for HTTP Splitting Smuggling

Рисунок 3.18 – Приклад результатів сканування Wapiti

HTML створюється автоматично та його завжди можна подивитись у веб-інтерфейсі, завантажити відповідний йому PDF-звіт та видалити його.

3.4.2 PDF-звіт

PDF — це той же HTML-звіт, але перетворений у формат, який зручно зберігати чи надсилати. В застосунку використана бібліотека WeasyPrint, яка бере HTML і стилі та робить з цього гарний PDF-документ. Виглядає все так само, але зручно для друку чи прикріплення до офіційних документів. Приклад частини PDF-звіту на рисунку 3.19.

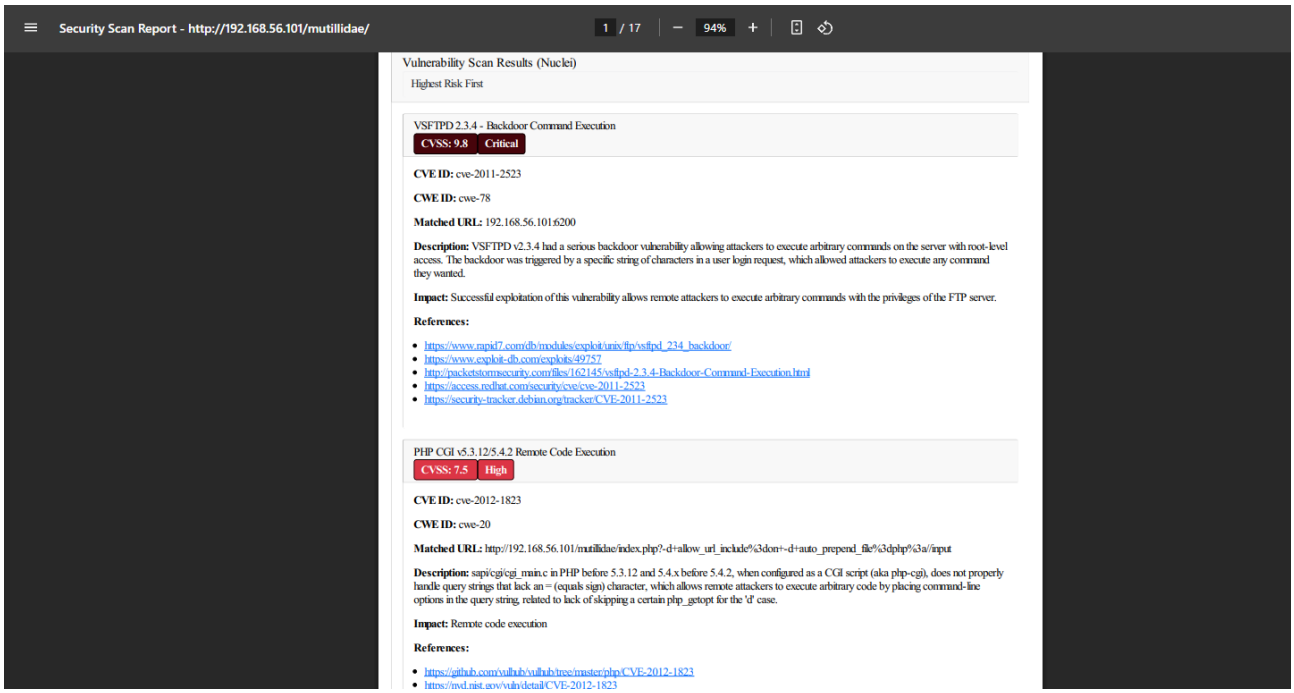


Рисунок 3.19 – Приклад частини PDF-звіту

Переваги PDF:

- легко пересилати на email або зберігати на диск;
- відкривається без браузера;
- підходить для звітності в компаніях.

3.4.3 Збереження історії (SQLite)

Щоб мати доступ до всіх попередніх перевірок, було вирішено використати SQLite. Це проста база даних, яка не потребує окремого сервера — все працює "в коробці". Створюється таблиця completed_scans, що показано на Рисунку 3.20.

```
conn = sqlite3.connect('scans.db')
c = conn.cursor()
c.execute('''CREATE TABLE IF NOT EXISTS completed_scans
             (id INTEGER PRIMARY KEY AUTOINCREMENT,
              target_url TEXT NOT NULL,
              scan_date TEXT NOT NULL,
              report_file TEXT NOT NULL,
              pdf_file TEXT NOT NULL)''')
conn.commit()
conn.close()
```

Рисунок 3.20 - Створення таблиці completed_scans

Після кожного сканування:

- Дані перетворюються у формат JSON;
- Генеруються HTML та PDF файли;
- Все записується в базу з посиланнями на файли.

Через інтерфейс можна подивитися попередні сканування, відкрити або завантажити звіти, а також порівняти результати для однієї й тієї ж цілі в різний час.

Переваги:

- Автономність — не треба налаштовувати окремий сервер баз даних.
- Прозорість — користувач бачить всю історію сканувань.
- Стандартизація — всі звіти мають один формат, з яким зручно працювати.

У результаті отримуємо не просто вивід сканера, а повноцінна система звітності, яку зручно використовувати і для себе, і для показу замовникам або командам безпеки.

Висновки до розділу 3

У цьому розділі було описано, як працює розроблений веб-застосунок для автоматизованого аудиту безпеки. Основна ідея — дати змогу запускати різні

сканери через веб-інтерфейс, не користуючись терміналом, що особливо зручно для менш досвідчених користувачів. Основу системи складають Flask (як backend), Bootstrap (для UI) і subprocess (щоб запускати сканери прямо з Python-коду).

Інтегровано декілька популярних інструментів — Nmap (з Vulscan), ZAP, Wapiti, Nuclei, а також Google Dorking API. Всі вони досить різні, тому важливо було зробити так, щоб результати виглядали більш-менш однаково.

Ще одна корисна функція — це історія сканувань. Реалізувана за допомогою SQLite. Вона зберігає ціль, результати, а також шляхи до HTML і PDF звітів. Потім все це можна подивитись через веб-інтерфейс — що було знайдено раніше, чи щось змінилось, і так далі. Дуже зручно, якщо треба подивитись динаміку чи готувати звіт.

Загалом, вийшов інструмент, який не претендує на професійний рівень або заміну великим системам, але цілком підходить для первинного аудиту. Особливо для тих, хто працює з сайтами малого або середнього бізнесу й не має можливості купити дорогі платні продукти. Так, система ще не ідеальна і є куди розвиватись, але основна функціональність уже працює стабільно й дає зрозумілий результат.

4 ЕКСПЕРИМЕНТАЛЬНЕ ТЕСТУВАННЯ ТА АНАЛІЗ ЕФЕКТИВНОСТІ

4.1 Опис тестового середовища (Metasploitable2)

Для перевірки працездатності, функціональності та ефективності створеного веб-застосунку було обрано повністю контрольоване тестове середовище на основі вразливої віртуальної машини Metasploitable2 — одного з найпопулярніших навчальних стендів для демонстрації вразливостей у веб-застосунках. Даний дистрибутив є навмисне вразливим сервером, побудованим на базі Ubuntu 18.04, і містить велику кількість типових уразливостей, включаючи SQL-ін'єкції, XSS, CSRF, RCE, відкриті порти, незахищені веб-інтерфейси тощо.

Тестування проводилося в середовищі VirtualBox на локальній машині з наступною конфігурацією:

- Хост система: Windows 10
- Тестова система: Metasploitable2
- Інші компоненти: встановлений та запущений OWASP ZAP у режимі API, Nmap, Wapiti, Nuclei та Google Dorking API.
- Мережеве з'єднання: режим "Host-Only Adapter", що забезпечує ізольованість та безпеку тестового середовища

Застосунок запускали у середовищі розробки (debug=True) з Flask, і він був доступний за локальним IP-адресом.

Кожен з інтегрованих у веб-застосунок сканерів виконувався окремо, з можливістю комбінованого запуску для повноцінного аналізу. Тестування проводилося у кілька етапів:

Первинне сканування портів та сервісів — здійснювалося за допомогою Nmap з підключеним скриптом vulscan. Цей етап дозволяв виявити базову інформацію про відкриті порти, версії служб, а також відомі CVE, пов’язані з кожною службою.

Сканування веб-інтерфейсів — виконувалося OWASP ZAP та Wapiti, які дозволяли виявляти класичні веб-вразливості: SQLi, XSS, CSRF, небезпечні форми, незахищені cookie тощо. ZAP був запущений через API в режимі активного сканування.

Цілеспрямований пошук шаблонних вразливостей — через Nuclei, який працював із повним набором шаблонів.

Google Dorking API — у рамках тестів була реалізована емуляція запитів за певними dork-фразами (наприклад, inurl:admin або intitle:"index of" "backup"), щоб показати можливість виявлення потенційно вразливих сторінок навіть до запуску безпосередніх сканерів.

Аналіз обробки результатів — після завершення кожного запуску результати зчитувалися, відображалися у веб-інтерфейсі та формувалися у вигляді HTML/PDF-звіту.

Щоб забезпечити об’єктивність експерименту, тестування проводилося з повторенням для кожного сканера, результати перевірялися на консистентність, а також фіксувалася продуктивність системи: час виконання аудиту, обсяг зібраних даних, кількість виявлених вразливостей та унікальність результатів між сканерами.

Таким чином, підготовлене тестове середовище забезпечило достатню кількість сценаріїв для повноцінної перевірки можливостей та обмежень створеного застосунку, а також дозволило отримати реальні результати для подальшого порівняльного аналізу в наступних підрозділах.

4.2 Результати запусків сканерів

Після проведення комплексного аудиту веб-застосунку, який був розгорнутий на вразливій віртуалці Metasploitable2, отримано результати з п’яти різних сканерів: це Nmap з Vulscan, OWASP ZAP, Nuclei, Wapiti і ще Google Dorking (він там теж був, хоч результатів не дав). Тестування відбувалося в

контрольованому середовищі, все пройшло нормально, і загалом сканування тривало десь 50 хвилин і ще трохи — 15 секунд. Кожен сканер знайшов різні штуки, тобто вразливості, і в результаті вийшло охопити і мережеву частину, і прикладні загрози — все як треба.

4.2.1 Nmap + Vulscan

Nmap разом з Vulscan добре просканував інфраструктуру, виявив 30 відкритих портів і 8745 потенційних вразливостей для 22 сервісів. Наприклад, там були:

- OpenSSH 4.7p1 — CVE-2008-1483
- Samba smbd 3.0.20 — CVE-2007-2446
- vsftpd 2.3.4 — CVE-2011-2523

Цей сканер допоміг побачити повну картину в низькорівневій інфраструктурі — тобто інфраструктурні баги, з чого, в принципі, і починається аудит.

4.2.2 OWASP ZAP

OWASP ZAP (Zed Attack Proxy) працював активно виявив 22 вразливості, які потім поділив на категорії за ризиком та 10 інформаційних повідомлень, детальніше у таблиці 4.1.

Таблиця 4.1 – Результат аудиту OWASP ZAP

Вразливість	**Рівень ризику**
Cross Site Scripting (Reflected)	High
SQL Injection - SQLite	High
Remote Code Execution - CVE-2012-1823	High
Source Code Disclosure - CVE-2012-1823	High
Content Security Policy (CSP) Header Not Set	Medium
Missing Anti-clickjacking Header	Medium
Application Error Disclosure	Medium
Vulnerable JS Library	Medium
Absence of Anti-CSRF Tokens	Medium
Directory Browsing	Medium
Hidden File Found	Medium
XSLT Injection	Medium
Parameter Tampering	Medium
Server Leaks Version Information via "Server" HTTP Response Header Field	Low
Cookie No HttpOnly Flag	Low
Cookie without SameSite Attribute	Low
Information Disclosure - Debug Error Messages	Low
Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)	Low
X-Content-Type-Options Header Missing	Low
Private IP Disclosure	Low
Big Redirect Detected (Potential Sensitive Information Leak)	Low
Timestamp Disclosure – Unix	Low
Session Management Response Identified	Info

Кінець таблиці 4.1

Information Disclosure - Suspicious Comments	Info
Modern Web Application	Info
User Controllable HTML Element Attribute (Potential XSS)	Info
Information Disclosure - Sensitive Information in URL	Info
Authentication Request Identified	Info
Cookie Poisoning	Info
User Controllable Charset	Info
GET for POST	Info
User Agent Fuzzer	Info

4.2.3 Wapiti

Цей сканер знайшов 32 вразливості, особливо в плані ін’єкцій:

- Backup file
- Weak credentials
- CRLF Injection
- Content Security Policy Configuration
- Cross Site Request Forgery
- Potentially dangerous file
- Command execution
- Path Traversal
- Htaccess Bypass
- HTML Injection
- Clickjacking Protection
- HTTP Strict Transport Security (HSTS)
- MIME Type Confusion
- HttpOnly Flag cookie

- Unencrypted Channels
- LDAP Injection
- Log4Shell
- Open Redirect
- Reflected Cross Site Scripting
- Secure Flag cookie
- Spring4Shell
- SQL Injection
- TLS/SSL misconfigurations
- Server Side Request Forgery
- Stored HTML Injection
- Stored Cross Site Scripting
- Subdomain takeover
- Blind SQL Injection
- Unrestricted File Upload
- Vulnerable software
- Internal Server Error
- Resource consumption

Та 5 інформаційних повідомлень:

- Fingerprint web application framework
- Fingerprint web server
- Review Webserver Metafiles for Information Leakage
- Fingerprint web technology
- HTTP Methods

Wapiti знайшов деякі речі, які ZAP пропустив. Особливо логічні проблеми і такі, що виникають через неправильні конфігурації.

4.2.4 Nuclei

Nuclei працював по шаблонах і виявив 9 вразливостей та 14 інформаційних повідомлень, детальніше у таблиці 4.2.

Таблиця 4.2 – Результат аудиту Nuclei

Вразливість	**Рівень ризику**
VSFTPD 2.3.4 - Backdoor Command Execution	Critical
PHP CGI v5.3.12/5.4.2 Remote Code Execution	High
Distccd v1 - Remote Code Execution	High
SSH Weak Algorithms Supported	Medium
FTP Anonymous Login	Medium
SSH Diffie-Hellman Modulus <= 1024 Bits	Low
SSH Server CBC Mode Ciphers Enabled	Low
SSH Weak Key Exchange Algorithms Enabled	Low
SSH Weak MAC Algorithms Enabled	Low
SMB v1 Supported - Detection	Info
SMB Version - Detection	Info
SSH SHA-1 HMAC Algorithms Enabled	Info
SSH Server Software Enumeration	Info
SSH Password-based Authentication	Info
OpenSSH Service - Detect	Info
VNC Service Detection	Info
SMTP Service Detection	Info
ESMTP - Detect	Info
SMTP Commands Enumeration	Info
PHP errors	Info
Apache Detection	Info
PHP Detect	Info
Wappalyzer Technology Detection	Info

Особливість Nuclei в тому, що він підтягує актуальні шаблони CVE, тому може знайти щось нове, чого інші ще не помітили.

4.2.5 Google Dorking

Google Dorking нічого не знайшов, бо застосунок був розгорнутий тільки локально і гугл його не індексував. Це очікувано, але все одно тест був проведений.

Таблиця 4.3 – Загальні результати

Сканер	**Кількість вразливостей**	**Основні типи вразливостей**
Nmap + Vulscan	8745 (потенційних)	CVE по версіях, відкриті порти
OWASP ZAP	22	XSS, SQLi, RCE, CSP, CSRF, інформаційні витоки
Wapiti	32	XSS, ін’єкції, Open Redirect, TLS проблеми
Nuclei	9	CVE, RCE, слабкі налаштування SSH/FTP
Google Dorking	0	— (бо не індексується)

Результати зазначені в таблиці 4.3 свідчать про те, що комбіноване використання сканерів дозволяє значно підвищити глибину і точність аудиту безпеки. Жоден з інструментів не забезпечив повного охоплення вразливостей самостійно — саме тому об’єднання результатів і їх аналіз у єдиному веб-застосунку демонструє свою ефективність.

4.3 Порівняння із комерційним сканером Nessus

Щоб об’єктивно оцінити ефективність розробленого веб-застосунку для автоматизованого аудиту безпеки, було проведено порівняльне тестування з одним із найпопулярніших комерційних сканерів — Tenable Nessus. Nessus є стандартом у галузі вразливостей, підтримується великою базою CVE, має потужний механізм виявлення конфігураційних помилок, уразливостей програмного забезпечення та слабких місць мережевих служб.

Для забезпечення коректності експерименту використовувалося однакове тестове середовище — Metasploitable2, яке містить значну кількість навмисно вразливих сервісів, таких як MySQL, Apache, VSFTPD, Samba тощо. Запуски Nessus та розробленого застосунку виконувалися в один і той самий проміжок часу, результати зберігалися в окремі звіти, після чого виконувалося їх порівняння.

Тривалість аудиту Nessus в режимі Web Application Test становила 18 хвилин і було виявлено 23 вразливості та 40 інформаційних повідомлень, детальніше у таблиці 4.4.

Таблиця 4.4 – Результат аудиту Nessus

Вразливість	**Рівень ризику**
Canonical Ubuntu Linux SEoL	Critical
VNC Server 'password' Password	Critical
SSL Version 2 and 3 Protocol Detection	Critical
Bind Shell Backdoor Detection	Critical
Apache Tomcat Multiple Issues	Critical, Medium, Info
Debian OpenSSH/OpenSSL Weak Random Generator	Critical
NFS Shares World Readable	High
rlogin Service Detection	High
Samba Badlock Vulnerability	High
SSL Certificate and Cipher Issues	High, Medium, Low, Info
ISC BIND Vulnerabilities	High, Medium, Info
TLS Version 1.0 Protocol Detection	Medium
Unencrypted Telnet Server	Medium

Продовження таблиці 4.4

SSL Anonymous Cipher Suites Supported	Medium
SSL DROWN Attack Vulnerability	Medium
SSH Weak Configuration	Medium, Low, Info
HTTP TRACE / TRACK Methods Allowed & HTTP Information	Medium, Info
SMB Signing Not Required	Medium, Info
SSL EXPORT Cipher Suites Vulnerabilities	Medium, Low
SMTP STARTTLS Command Injection	Medium, Info
Diffie-Hellman Key Size Vulnerability	Low
X Server Detection	Low
Microsoft Windows SMB Information Disclosure	Info
SSL/TLS Version and Cipher Suite Recommendations	Info
DNS Server Detection	Info
VNC Server Detection	Info
Apache HTTP Server Information	Info
PHP Version and Backported Patch Detection	Info
RPC Portmapper Detection	Info
SSH Backported Patch & Protocol Support Detection	Info
SSH Password and Server Info	Info
Web Server Favicon and Default Page	Info
Nessus SYN Scanner	Info
RPC Services Enumeration	Info
Service Detection	Info
Unknown Service Banner Detection	Info
OpenSSL Detection	Info
RMI Registry Detection	Info
HELP Banner Detection	Info
AJP Connector Detection	Info
WWW Backported Security Patch Detection	Info
Common Platform Enumeration (CPE)	Info
Device Type Detection	Info
FTP Server Detection	Info
MySQL Server Detection	Info
Nessus Scan Metadata	Info
NFS Share Export List	Info
OpenSSH Detection	Info
OS Fingerprinting	Info
OS Identification	Info
OS Patch Assessment Unavailable	Info
Patch Report	Info
PostgreSQL Server Detection	Info

Кінець таблиці 4.4

PostgreSQL STARTTLS Support	Info
Samba Server Detection	Info
Samba Version Detection	Info
SMTP Server Detection	Info
No Credentials Provided (Authentication Status)	Info
Telnet Server Detection	Info
Traceroute Information	Info
vsftpd Detection	Info
WebDAV Detection	Info
WMI Not Available	Info

Порівняння знайдених вразливостей розробленим веб-застосунком та Nessus у таблиці 4.5.

Таблиця 4.5 – Порівняння знайдених вразливостей з Nessus

Вразливість	**ZAP / Wapiti / Nuclei**	**Nessus**
Remote Code Execution	✅ ZAP, Nuclei	✅ Apache Tomcat Ghostcat (CVE-2020-1938), distccd RCE, PHP CGI RCE
Backdoor in VSFTPD 2.3.4	✅ Nuclei	❌
PHP CGI Remote Code Execution	✅ Nuclei	✅
distccd Remote Code Execution	✅ Nuclei	✅
SQL Injection	✅ ZAP, Wapiti	❌
Blind SQL Injection	✅ Wapiti	❌
LDAP Injection	✅ Wapiti	❌
CRLF Injection	✅ Wapiti	❌
Reflected XSS	✅ ZAP, Wapiti	❌
Stored XSS	✅ Wapiti	❌
HTML Injection	✅ Wapiti	❌
Stored HTML Injection	✅ Wapiti	❌
Cross-Site Request Forgery (CSRF)	✅ ZAP, Wapiti	❌

Продовження таблиці 4.5

SSRF (Server-Side Request Forgery)	✅ Wapiti	❌
Clickjacking	✅ ZAP, Wapiti	❌
Open Redirect	✅ Wapiti	❌
Path Traversal	✅ Wapiti	❌
Unrestricted File Upload	✅ Wapiti	❌
Hidden / Backup File	✅ ZAP, Wapiti	❌
Htaccess Bypass	✅ Wapiti	❌
Subdomain Takeover	✅ Wapiti	❌
Directory Browsing	✅ ZAP	❌
Parameter Tampering	✅ ZAP	❌
Cookie HttpOnly Missing	✅ ZAP, Wapiti	❌
Cookie Secure Flag Missing	✅ Wapiti	❌
Cookie SameSite Missing	✅ ZAP	❌
Cookie Poisoning	✅ ZAP	❌
CSP Header Missing	✅ ZAP, Wapiti	❌
HSTS Header Missing	✅ Wapiti	❌
TLS/SSL Misconfiguration	✅ Wapiti	✅ SSLv2/v3, SWEET32, FREAK, DROWN, POODLE, Logjam, RC4
SSH Weak Algorithms (CBC, MAC, etc.)	✅ Nuclei	✅ Weak SSH algorithms, CBC mode, weak KEX, weak MACs
FTP Anonymous Login	✅ Nuclei	❌
Weak Credentials	✅ Wapiti	✅ VNC 'password' password
Private IP Disclosure	✅ ZAP	❌
Server Header Disclosure	✅ ZAP	❌
X-Powered-By Header Disclosure	✅ ZAP	❌
X-Content-Type-Options Missing	✅ ZAP	❌
Application Error Disclosure	✅ ZAP	❌
Debug/Error Message Disclosure	✅ ZAP	❌
Suspicious Comments	✅ ZAP	❌
Sensitive Info in URL	✅ ZAP	❌

Кінець таблиці 4.5

Internal Server Error (500)	✅ Wapiti	❌
Resource Consumption Issue	✅ Wapiti	❌
Fingerprint Web App / Server / Tech	✅ Wapiti, Nuclei	✅ Apache, PHP, SSH, MySQL, PostgreSQL, SMB, etc.
Bind Shell Backdoor Detection	❌	✅
Debian OpenSSH/OpenSSL RNG Weakness	❌	✅
NFS Shares World Readable	❌	✅
rlogin Service Detection	❌	✅
Samba Badlock Vulnerability	❌	✅
HTTP TRACE / TRACK Methods Allowed	❌	✅
SMB Signing not required	❌	✅
SMTP STARTTLS Plaintext Command Injection	❌	✅
X Server Detection	❌	✅

Загальна кількість знайдених вразливостей була дещо вищою у розробленому застосунку, що пов’язано з використанням кількох сканерів одночасно: ZAP, Wapiti, Nmap, Nuclei тощо. Nessus, хоч і має глибокі алгоритми, базується переважно на CVE-ідентифікаторах і конфігураційних шаблонах. Він не завжди ефективно виконує веб-сканування, особливо при перевірці XSS або логічних помилок у формі.

Розроблений застосунок помітно краще впорався з веб-специфічними уразливостями — типу XSS, SQL ін’єкцій, HTML ін’єкцій, CSRF, SSRF і відкритих редиректів. Nessus цього всього майже не побачив. Це логічно, бо він більше про мережу, сервіси і системні баги.

Але при цьому Nessus показав, що він кращий у класичній мережевій безпеці. Наприклад, він знайшов уразливості, пов’язані з Samba, NFS, OpenSSL RNG у Debian, X server, підтримкою HTTP TRACE тощо. Усі ці речі створений застосунок не покриває.

З іншого боку, Nessus показав вищу якість деталізації результатів: кожна вразливість містить опис, ризик, шлях до виправлення, посилання на офіційну документацію CVE. У запропонованому веб-застосунку також є базова документація, однак глибина автоматичних описів поступається комерційному продукту.

Порівняння показало, що обидва підходи мають свої сильні сторони. Розроблений веб-застосунок завдяки мультисканерному підходу ефективно виявляє веб-вразливості та забезпечує швидке отримання звіту. З іншого боку, Nessus є більш повним у сфері мережевої безпеки та надає багатий аналітичний супровід виявлених проблем. У контексті веб-захисту, особливо для внутрішнього використання в організаціях малого і середнього бізнесу, створений інструмент демонструє конкурентну ефективність, гнучкість та доступність.

4.4 Порівняння із комерційним сканером Burp Scanner

Щоб краще зрозуміти, наскільки добре працює створений веб-застосунок для автоматичного пошуку вразливостей, було вирішено провести ще одне порівняння — цього разу з Burp Suite Scanner. Це досить популярний інструмент, який використовують у пентестах, особливо в комерційних і професійних середовищах. Burp має багато модулів, дозволяє налаштовувати перевірку під себе і може як автоматично сканувати, так і давати змогу втручатися вручну.

Знову ж таки для тесту використовувалося тестову середовище Metasploitable2. Для Burp було обрано режим "Lightweight", бо в іншому випадку він сканує дуже довго, і щоб час сканування був якнайбільше схожим з розробленим застосунком. Сканування тривало 1 годину, 8 хвилин і 56 секунд, та було знайдено 13 вразливостей та 15 інформаційних повідомлень, детальніше у таблиці 4.6.

Таблиця 4.6 – Результати аудиту Burp

Вразливість	**Рівень ризику**
SQL Injection	High
File Path Traversal	High
File Path Manipulation	High
Cross-Site Scripting (Stored)	High
Cross-Site Scripting (Reflected)	High
Cross-Site Scripting (DOM-based)	High
Cleartext Submission of Password	High
Password Submitted Using GET Method	Low
Open Redirection (Reflected)	Low
Cookie Without HttpOnly Flag Set	Low
Password Field With Autocomplete Enabled	Low
Client-Side HTTP Parameter Pollution (Reflected)	Low
Unencrypted Communications	Low
Path-Relative Style Sheet Import	Info
Cross-Site Request Forgery	Info
Referer-Dependent Response	Info
User Agent-Dependent Response	Info
Input Returned in Response (Stored)	Info
Input Returned in Response (Reflected)	Info
Suspicious Input Transformation (Reflected)	Info
Suspicious Input Transformation (Stored)	Info
Cross-Domain Referer Leakage	Info
Frameable Response (Potential Clickjacking)	Info
HTTP TRACE Method is Enabled	Info
DOM Data Manipulation (DOM-based)	Info
Email Addresses Disclosed	Info
Private IP Addresses Disclosed	Info
HTML Does Not Specify Charset	Info

Результати порівняння зібрані в таблиці 4.7

Таблиця 4.7 – Порівняння знайдених вразливостей з Burp

Вразливість	**ZAP / Wapiti / Nuclei**	**Burp**
Remote Code Execution	✅ ZAP, Nuclei	❌
Backdoor in VSFTPD 2.3.4	✅ Nuclei	❌
PHP CGI Remote Code Execution	✅ Nuclei	❌

Продовження таблиці 4.7

Distccd Remote Code Execution	✅ Nuclei	❌
SQL Injection	✅ ZAP, Wapiti	✅
Blind SQL Injection	✅ Wapiti	✅ (розглядається як частина SQLi)
LDAP Injection	✅ Wapiti	❌
CRLF Injection	✅ Wapiti	❌
Reflected XSS	✅ ZAP, Wapiti	✅
Stored XSS	✅ Wapiti	✅
DOM-based XSS	❌	✅
HTML Injection	✅ Wapiti	❌
Stored HTML Injection	✅ Wapiti	✅ (stored input)
Cross-Site Request Forgery (CSRF)	✅ ZAP, Wapiti	✅
SSRF (Server-Side Request Forgery)	✅ Wapiti	❌
Clickjacking	✅ ZAP, Wapiti	✅
Open Redirect	✅ Wapiti	✅
Path Traversal	✅ Wapiti	✅
File Path Manipulation	❌	✅
Unrestricted File Upload	✅ Wapiti	❌
Hidden / Backup File	✅ ZAP, Wapiti	❌
Htaccess Bypass	✅ Wapiti	❌
Subdomain Takeover	✅ Wapiti	❌
Directory Browsing	✅ ZAP	❌
Parameter Tampering	✅ ZAP	❌
Cookie HttpOnly Missing	✅ ZAP, Wapiti	✅
Cookie Secure Flag Missing	✅ Wapiti	❌
Cookie SameSite Missing	✅ ZAP	❌
Cookie Poisoning	✅ ZAP	❌
CSP Header Missing	✅ ZAP, Wapiti	❌
HSTS Header Missing	✅ Wapiti	❌

Кінець таблиці 4.7

TLS/SSL Misconfiguration	✅ Wapiti	✅ (unencrypted communications)
SSH Weak Algorithms (CBC, MAC, etc.)	✅ Nuclei	❌
FTP Anonymous Login	✅ Nuclei	❌
Weak Credentials	✅ Wapiti	❌
Cleartext Password Submission	❌	✅
Password in URL (GET Method)	❌	✅
Password Field Autocomplete	❌	✅
Client-side HTTP Parameter Pollution	❌	✅
Path-relative Stylesheet	❌	✅
Referer-dependent Response	❌	✅
User-Agent-dependent Response	❌	✅
Input Reflected in Response	❌	✅
Input Stored in Response	❌	✅
Suspicious Input Transformation	❌	✅
Cross-Domain Referer Leakage	❌	✅
Frameable Response	❌	✅
HTTP TRACE Enabled	❌	✅
Email Address Disclosure	❌	✅
Private IP Disclosure	✅ ZAP	✅
Server Header Disclosure	✅ ZAP	❌
X-Powered-By Header Disclosure	✅ ZAP	❌
X-Content-Type-Options Missing	✅ ZAP	❌
Application Error Disclosure	✅ ZAP	❌
Debug/Error Message Disclosure	✅ ZAP	❌
Suspicious Comments	✅ ZAP	❌
Sensitive Info in URL	✅ ZAP	❌
Internal Server Error (500)	✅ Wapiti	❌
Resource Consumption Issue	✅ Wapiti	❌
Fingerprint Web App / Server / Tech	✅ Wapiti, Nuclei	❌

Burp добре виявляє набір веб-вразливостей — тобто XSS, SQLi, CSRF, редиректи, і все, що входить до класичних OWASP Top 10. Варто зазначити, що він знайшов навіть DOM-based XSS, а також витягнув деякі потенційні витоки

— типу IP-адреси, email-и, заголовки сервера та інше, що можна десь забути прибрати.

Але при цьому є нюанс — Burp практично нічого не знайшов з боку системи. Наприклад, не було виявлено жодного RCE (віддалене виконання коду), і взагалі нічого по бекдорах або CVE, пов’язаних із демонами типу vsftpd, distccd чи PHP CGI. Це тому, що Burp переважно працює з HTTP/HTTPS і не сильно заглядає вглиб системного рівня. CVE він теж повноцінно не перевіряє.

Розроблений веб-застосунок у цьому плані показав себе краще, бо, наприклад, Nuclei дозволив знаходити RCE, недоліки в SSL/TLS, конфігураційні дірки в SSH та FTP, і навіть деякі бекдори. Також були виявлені проблеми з авторизацією та автентифікацією.

Burp, в свою чергу, краще працює з фронтендом. Наприклад, він вміє аналізувати, як рендериться контент, чи є аномальна поведінка при натисканні кнопок, або чи коректно обробляються посилання. Також він аналізує iframe’и, політики Referer і клієнтську поведінку — тобто все те, що часто упускається іншими сканерами.

Створений сканер тут поки що трохи відстає — він більше про покриття CVE та серверні загрози, ніж про глибину контенту. Але це теж важливо, бо на практиці потрібно знати не лише як себе поводить фронт, а й що відбувається на стороні сервера

Загалом, обидва інструменти доповнюють один одного. Burp — це найкращий вибір для класичних XSS/CSRF і тестування з точки зору користувача. А розроблений застосунок більше заточений під пошук системних і мережевих вразливостей, з ширшим покриттям по CVE і глибшими перевірками, які Burp не робить.

Ідеальний варіант — це коли використовуються обидва, тоді можна максимально все перекрити. Але якщо немає бюджету чи часу, то розроблений застосунок реально може бути хорошим вибором, особливо для внутрішнього використання або для невеликих компаній. Він автоматизований, використовує купу різних інструментів під капотом і взагалі — не такий уже й простий, як здається.

4.5 Виявлені недоліки: повтори, формати

У ході тестування розробленого веб-застосунку для автоматизованого аудиту безпеки виявилось, що, хоч він і працює, але деякі моменти ще треба допрацювати. Вони не критичні, тобто нічого не ламається, але все ж заважають зробити все ідеально.

Один із найбільш помітних мінусів — дублювання одних і тих самих вразливостей, якщо їх знайшли кілька сканерів. Наприклад, ZAP і Wapiti можуть знайти одну і ту ж XSS. Через це в звітах виглядає, ніби проблем більше, ніж є насправді, що трохи плутає. Проблема в тому, що різні інструменти по-своєму описують вразливості — назви різні, формат інший, ще й параметри трохи по-різному вказані.

Ще один момент — всі сканери виводять результати хто як. Наприклад, Nmap з Vulscan — просто текст, Nuclei та Wapity — JSON, ZAP — API. Тому довелось для кожного писати свій парсер.

Ці недоліки — не щось унікальне, а скоріше типове для систем, де багато інструментів намагаються працювати разом. Але вони трохи ускладнюють реальне використання. Тим не менш, розробка все одно корисна, і ці моменти просто показують, де можна зробити ще краще.

4.6 Практична оцінка застосунку у сфері кіберзахисту малого та середнього бізнесу

Розроблений під час цього дослідження веб-застосунок, на мій погляд, може бути реально корисним саме для малого і середнього бізнесу. Особливо для тих, у кого нема окремого відділу з кібербезпеки або ж просто не вистачає ресурсів на щось дороге й складне. Тут розглянуто, як ця система показала себе в умовному реальному сценарії.

На відміну від великих компаній, які можуть дозволити собі круті штуки типу Nessus Pro чи Burp Suite Enterprise, у малого бізнесу такої розкоші часто нема. Але сайти, CRM-ки, кабінети клієнтів — все це так само вразливе до атак. І ще гірше, що більшість таких компаній не проводять регулярний аудит взагалі.

Простий інтерфейс — запускати сканування можуть навіть ті, хто не дуже в темі. Є шаблони, які вже налаштовані, треба тільки вибрати й натиснути кнопку.

Швидкі звіти — все зібране подається в HTML або PDF. Їх можна одразу надіслати розробникам, щоб ті виправляли.

Історія перевірок — звіти зберігаються в SQLite, можна бачити, як змінювався стан безпеки з часом.

В цілому, як показало тестування, застосунок може стати корисним інструментом для компаній, які:

- хочуть самі проводити хоча б базові перевірки безпеки,
- не мають коштів на дорогі рішення або послуги,
- прагнуть хоч якось підняти рівень безпеки,
- хочуть швидко реагувати на проблеми, коли вони ще не стали критичними.

Застосунок вийшов досить дієвим для своєї цілі — допомагати МСБ без великих затрат на кіберзахист. Так, це не ідеальний інструмент: інтерфейс ще можна поліпшити, автоматизації не всюди вистачає, є нюанси з оновленням шаблонів, але як стартова точка — цілком придатний. Для компаній, які розуміють, що захищати свій веб-ресурс треба, але не знають з чого почати — це непоганий варіант. Особливо враховуючи, що він відкритий і його можна дороблювати під себе.

Висновки до розділу 4

У цьому розділі проведене повне тестування створеного веб-застосунку для автоматичного сканування безпеки, а також подивився, наскільки він реально

корисний, особливо для малого та середнього бізнесу. З результатів видно, що інструмент справді працює і може бути корисним на практиці. Хоча він ще не ідеальний, але вже зараз може покривати доволі широкий спектр загроз, використовуючи при цьому лише безкоштовні інструменти.

По-перше, вдалося об'єднати кілька різних сканерів (типу Nmap з Vulscan, OWASP ZAP, Nuclei, Wapiti, а також Google Dorking) в одну систему. Це дозволило знайти багато різних вразливостей, включно з тими, які не знаходив Nessus. Виходить, що навіть безкоштовні інструменти можуть бути досить ефективними, якщо правильно їх зібрати докупи.

По-друге, розроблена система показала, що нею досить зручно користуватися. Усе запускається з одного місця, а звіти одразу формуються у зрозумілому вигляді — HTML або PDF. Плюс, є історія аудитів у SQLite, що дозволяє відстежувати, що й коли було знайдено. Це все спрощує життя тим, хто не дуже глибоко розуміється у кібербезпеці, але хоче мати хоча б базовий захист.

По-третє, хоч є й деякі мінуси — наприклад, дублювання знайдених вразливостей або різні формати вихідних даних — усе це не робить систему непрацюючою. Просто треба буде з часом її трохи доопрацювати й оптимізувати.

І наостанок, якщо говорити конкретно про МСБ, то саме тут цей застосунок виглядає найбільш доречно. Не треба витрачати багато грошей, не обов’язково мати команду спеціалістів — і вже можна проводити перевірки. Це важливо, бо багато малих компаній взагалі не роблять нічого в плані кібербезпеки, і такий інструмент може стати для них стартовим рішенням.

Отже, навіть попри деякі недоліки, веб-застосунок довів свою корисність і потенціал для розвитку. Його вже можна реально використовувати, а з часом, допрацювавши, він може стати ще кращим.

ВИСНОВКИ

Метою цієї дипломної роботи було створення програмного рішення для автоматизованого аудиту безпеки веб-застосунків. Ідея полягала в тому, щоб об’єднати кілька відомих безкоштовних сканерів в одну систему, яку можна буде запускати через простий інтерфейс, переглядати результати та формувати звіти. Основні задачі вдалося виконати: було проведено огляд сучасних кіберзагроз, розглянуто доступні інструменти безпеки, продумано архітектуру застосунку на Flask і Bootstrap, а також підключено сканери Nmap з Vulscan, OWASP ZAP, Wapiti, Nuclei і Google Dorking API. Реалізовано єдиний механізм запуску сканувань і обробки результатів, які потім зберігаються у форматі, схожому на JSON. Також зроблено формування звітів у HTML та PDF, з історією сканувань у SQLite. Все це тестувалось на Metasploitable2 і показало, що застосунок працює.

Наукова новизна, полягає в тому, що в цій роботі вдалося зібрати різні сканери в одну систему, і при цьому використовувались тільки відкриті технології. З практичного боку, зроблено зручний веб-інтерфейс, який дозволяє запускати сканування навіть людям, які не сильно розбираються в пентестінгу. Плюс є автоматичне збереження, генерування звітів і базовий захист від деяких атак (наприклад, RCE). Це робить систему більш реальною для використання в МСБ, де зазвичай немає грошей на дорогі продукти.

При порівнянні застосунока з комерційними продуктами типу Nessus або Burp Suite, звісно, видно, що він програє в деяких моментах. Але при цьому має свої плюси — він безкоштовний, простіший у налаштуванні та дозволяє автоматично запускати скани. Nessus дуже потужний, але платний і складний, а Burp більше підходить для ручної перевірки (і то треба мати певні знання, щоб ним ефективно користуватись). У даному випадку все заточено на автоматизацію та доступність. Окремо хочу відзначити, що інтеграція Google Dorking API додає трохи нестандартності — не всі навіть згадані сканери мають щось подібне. Але, звісно, розроблений застосунок ще не ідеальний — є питання

з масштабованістю, іноді результати дублюються, треба ще попрацювати над цим.

Є ще багато ідей, як покращити систему. Наприклад, додати розклад (scheduler), щоб можна було планувати перевірки на ніч або в певний час. Це зручно для великих сайтів, бо не навантажує систему у робочі години. І в майбутньому планується підключити ще інші сканери чи навіть інтегрувати з системами типу vulnerability management.

Цей застосунок може бути корисним не тільки для бізнесу, а й у навчанні. Наприклад, у лабораторіях він міг би допомогти студентам розібратись, як працюють сканери безпеки, не копаючись довго в їхньому встановленні. Інтерфейс простий, тож можна зосередитися саме на аналізі. У пентест-командах він може використовуватись як перший крок перед ручним аудитом — для швидкого сканування. У CTF-сценаріях — для перевірки завдань на наявність зайвих вразливостей, щоб учасники не знайшли те, що не планувалося.

Цей застосунок вийшов доволі універсальним: він поєднує в собі і теорію, і практику, і сучасні інструменти. Його можна використовувати в багатьох сферах, а далі — вдосконалювати під конкретні задачі. Так, він не без недоліків, але вже зараз його можна застосовувати у реальних умовах, що вже є непоганим результатом.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Кібербезпека в інформаційному суспільстві: Інформаційно-аналітичний дайджест / відп. ред. О. Довгань; упоряд. О. Довгань, Л. Литвинова, С. Дорогих; Державна наукова установа «Інститут інформації, безпеки і права НАПрН України»; Національна бібліотека України ім. В.І. Вернадського. – К., 2023. – №7 (липень). – 270 с. [Електронний ресурс] – Режим доступу до ресурсу: https://ippi.org.ua/sites/default/files/2023-7.pdf
2. OWASP Top Ten [Електронний ресурс] // OWASP. – Режим доступу до ресурсу: https://owasp.org/www-project-top-ten/
3. State of Application Security 2023 Annual Report [Електронний ресурс] // Indusface. – 2023. – Режим доступу до ресурсу: https://www.indusface.com/resources/research-reports/state-of-application-security-2023-annual-report/
4. DDoS Threat Report 2023 Q3 [Електронний ресурс] // Cloudflare Blog. – 2023. – Режим доступу до ресурсу: https://blog.cloudflare.com/ddos-threat-report-2023-q3/
5. Cloudflare mitigates record-breaking 71 million request per second DDoS attack [Електронний ресурс] // Cloudflare Blog. – Режим доступу до ресурсу: https://blog.cloudflare.com/cloudflare-mitigates-record-breaking-71-million-request-per-second-ddos-attack/
6. The Top 7 Most Common Web Vulnerabilities [Електронний ресурс] // Security Boulevard. – 2022. – Режим доступу до ресурсу: https://securityboulevard.com/2022/04/the-top-7-most-common-web-vulnerabilities/
7. Cross-Site Scripting (XSS) [Електронний ресурс] // Invicti. – Режим доступу до ресурсу: https://www.invicti.com/learn/cross-site-scripting-xss/
8. Security Misconfiguration [Електронний ресурс] // SentinelOne. – Режим доступу до ресурсу: https://www.sentinelone.com/cybersecurity-101/cybersecurity/security-misconfiguration/

9. Security Misconfiguration [Електронний ресурс] // OWASP Top 10. – 2021. – Режим доступу до ресурсу: https://owasp.org/Top10/A05_2021-Security_Misconfiguration/
10. AWS Application Load Balancer Implementation Compromise [Електронний ресурс] // Wired. – Режим доступу до ресурсу: https://www.wired.com/story/aws-application-load-balancer-implementation-compromise/
11. Zapping the Top 10: 2021 [Електронний ресурс] // ZAPROXY. – Режим доступу до ресурсу: https://www.zaproxy.org/docs/guides/zapping-the-top-10-2021/
12. Wapiti Scanner [Електронний ресурс] – Режим доступу до ресурсу: https://wapiti-scanner.github.io/
13. Nmap Reference Guide [Електронний ресурс] // Nmap. – Режим доступу до ресурсу: https://nmap.org/book/man.html
14. Vulscan [Електронний ресурс] // Github – Режим доступу до ресурсу: https://github.com/scipag/vulscan
15. Nuclei [Електронний ресурс] // Github – Режим доступу до ресурсу: https://github.com/projectdiscovery/nuclei
16. Web App Scanning [Електронний ресурс] // Tenable Docs – Режим доступу до ресурсу: https://docs.tenable.com/web-app-scanning.htm
17. Burp Suite Documentation [Електронний ресурс] // PortSwigger – Режим доступу до ресурсу: https://portswigger.net/burp/documentation/contents
18. Burp Suite Professional [Електронний ресурс] // PortSwigger – Режим доступу до ресурсу: https://portswigger.net/buy/pro
19. Tenable Buy [Електронний ресурс] // Tenable – Режим доступу до ресурсу: https://www.tenable.com/buy
20. Qualys Pricing: Ultimate Guide for Security Products [Електронний ресурс] // UnderDefense – Режим доступу до ресурсу: https://underdefense.com/industry-pricings/qualys-pricing-ultimate-guide-for-security-products/
21. The Role of Open Source Software in Cybersecurity: Benefits and Challenges [Електронний ресурс] // LinkedIn – Режим доступу до ресурсу:

https://www.linkedin.com/pulse/role-open-source-software-cybersecurity-benefits-challenges-ahcfc

22. ZAP API Documentation [Електронний ресурс] // ZAPROXY – Режим доступу до ресурсу: https://www.zaproxy.org/docs/api/
23. RabbitMQ [Електронний ресурс] – Режим доступу до ресурсу: https://www.rabbitmq.com/
24. Vulners Search [Електронний ресурс] – Режим доступу до ресурсу: https://vulners.com/search
25. DefectDojo Integrations [Електронний ресурс] – Режим доступу до ресурсу: https://defectdojo.com/integrations
26. cron(8) — Linux man page [Електронний ресурс] – Режим доступу до ресурсу: https://man7.org/linux/man-pages/man8/cron.8.html
27. Celery Periodic Tasks [Електронний ресурс] // Celery Docs – Режим доступу до ресурсу: https://docs.celeryq.dev/en/stable/userguide/periodic-tasks.html
28. subprocess — Subprocess management [Електронний ресурс] // Python Docs – Режим доступу до ресурсу: https://docs.python.org/3/library/subprocess.html
29. Redis Queue [Електронний ресурс] // Redis – Режим доступу до ресурсу: https://redis.io/glossary/redis-queue/
30. Report Generation Add-on [Електронний ресурс] // ZAPROXY – Режим доступу до ресурсу: https://www.zaproxy.org/docs/desktop/addons/report-generation/
31. Report Templates [Електронний ресурс] // Tenable Security Center Docs – Режим доступу до ресурсу: https://docs.tenable.com/security-center/Content/Reports/ReportTemplates.htm
32. OWASP in SDLC [Електронний ресурс] // OWASP Integration Standards – Режим доступу до ресурсу: https://owasp.org/www-project-integration-standards/writeups/owasp_in_sdlc/
33. Flask Documentation [Електронний ресурс] // Flask – Режим доступу до ресурсу: https://flask.palletsprojects.com/en/latest/

34. Custom Search JSON API Overview [Електронний ресурс] // Google Developers – Режим доступу до ресурсу: https://developers.google.com/custom-search/v1/overview

ДОДАТОК А ПРОГРАМНИЙ КОД РОЗРОБЛЕНОГО ВЕБ-ЗАСТОСУНКУ

Лістинг А.1 – Головний файл застосунку (app.py)

```
from flask import Flask, render_template, request, send_file, jsonify, url_for,
redirect
from zapv2 import ZAPv2
import nmap
import json
import os
from datetime import datetime
import pdfkit
import threading
import time
import socket
from urllib.parse import urlparse
from tabulate import tabulate
import sqlite3
from googleapiclient.discovery import build
import requests
import subprocess
import tempfile
import re

app = Flask(__name__)
app.config['SECRET_KEY'] = 'your-secret-key-here'

# Initialize ZAP API
zap = ZAPv2(apikey='your-api', proxies={'http': 'http://127.0.0.1:8080', 'https':
'http://127.0.0.1:8080'})

# Available vulnerability types for scanning
VULNERABILITY_TYPES = {
    'xss': 'Cross-Site Scripting (XSS)',
    'sqli': 'SQL Injection',
    'csrf': 'Cross-Site Request Forgery',
    'ssrf': 'Server-Side Request Forgery',
    'rce': 'Remote Code Execution',
    'lfi': 'Local File Inclusion',
    'info': 'Information Disclosure',
    'injection': 'Other Injection Flaws',
    'session': 'Session Management Issues',
    'config': 'Security Configuration Issues',
    'crypto': 'Cryptographic Issues',
    'input': 'Input Validation Issues',
    'misc': 'Miscellaneous Vulnerabilities',
    'ssti': 'Server-Side Template Injection'
}

WAPITI_MODULES = {
```

```
    'backup': 'Uncover backup files on the web server.',
    'brute_login_form': 'Attempt to log in on authentication forms using known weak
credentials (like admin/admin).',
    'buster': 'Brute force paths on the web-server to discover hidden files and
directories.',
    'cms': 'Base class for detecting version.',
    'cookieflags': 'Evaluate the security of cookies on the website.',
    'crlf': 'Detect Carriage Return Line Feed (CRLF) injection vulnerabilities.',
    'csp': 'Evaluate the security level of Content Security Policies of the web
server.',
    'csrf': 'Detect forms missing Cross-Site Request Forgery protections (CSRF
tokens).',
    'exec': 'Detect scripts vulnerable to command and/or code execution.',
    'file': 'Detect file-related vulnerabilities such as directory traversal and
include() vulnerabilities.',
    'htaccess': 'Attempt to bypass access controls to a resource by using a custom
HTTP method.',
    'http_headers': 'Evaluate the security of HTTP headers.',
    'https_redirect': 'Check for HTTPS redirections.',
    'ldap': 'Detect scripts vulnerable to LDAP injection.',
    'log4shell': 'Detect the Log4Shell vulnerability (CVE-2021-44228)',
    'methods': 'Detect uncommon HTTP methods (like PUT) that may be allowed by a
script.',
    'permanentxss': 'Detect stored (aka permanent) Cross-Site Scripting
vulnerabilities on the web server.',
    'redirect': 'Detect Open Redirect vulnerabilities.',
    'shellshock': 'Detects scripts vulnerable to the infamous ShellShock
vulnerability.',
    'spring4shell': 'Detect the Spring4Shell vulnerability',
    'sql': 'Detect SQL (also XPath) injection vulnerabilities using error-based or
boolean-based (blind) techniques.',
    'ssl': 'Evaluate the security of SSL/TLS certificate configuration.',
    'ssrf': 'Detect Server-Side Request Forgery vulnerabilities.',
    'takeover': 'Detect subdomains vulnerable to takeover (CNAME records pointing
to non-existent and/or available domains)',
    'timesql': 'Detect SQL injection vulnerabilities using blind time-based
technique.',
    'upload': 'Detect unrestricted file upload vulnerabilities.',
    'wapp': 'Identify web technologies used by the web server using Wappalyzer
database.',
    'wp_enum': 'Detect WordPress Plugins with version.',
    'xss': 'Detects stored (aka permanent) Cross-Site Scripting vulnerabilities on
the web server.',
    'xxe': 'Detect scripts vulnerable to XML external entity injection (also known
as XXE).'
}

# Define scan policy rules for each vulnerability type
SCAN_POLICIES = {
    'xss': {
        'name': 'Cross-Site Scripting (XSS)',
        'rules': [
```

```
            40012,  # Cross Site Scripting (Reflected)
            40014,  # Cross Site Scripting (Persistent)
            40016,  # Cross Site Scripting (Persistent) - Prime
            40017,  # Cross Site Scripting (Persistent) - Spider
            40026   # Cross Site Scripting (DOM Based)
        ]
    },
    'sqli': {
        'name': 'SQL Injection',
        'rules': [
            40018,  # SQL Injection
            40019,  # SQL Injection - MySQL
            40020,  # SQL Injection - Hypersonic SQL
            40021,  # SQL Injection - Oracle
            40022,  # SQL Injection - PostgreSQL
            40024,  # SQL Injection - SQLite
            40027   # SQL Injection - MsSQL
        ]
    },
    'csrf': {
        'name': 'Cross-Site Request Forgery',
        'rules': [
            90022  # CSRF
        ]
    },
    'rce': {
        'name': 'Remote Code Execution',
        'rules': [
            20018,  # Remote Code Execution
            90019,  # Server Side Code Injection
            90020   # Remote OS Command Injection
        ]
    },
    'lfi': {
        'name': 'Local File Inclusion',
        'rules': [
            6,  # Path Traversal
            7,  # Remote File Inclusion
            40009  # Server Side Include
        ]
    },
    'info': {
        'name': 'Information Disclosure',
        'rules': [
            10045,  # Source Code Disclosure - /WEB-INF Folder
            20017,  # Source Code Disclosure
            40028,  # ELMAH Information Leak
            40029,  # Trace.axd Information Leak
            40032,  # .htaccess Information Leak
            40034,  # .env Information Leak
            40035,  # Hidden File Finder
            40042   # Spring Actuator Information Leak
```

```
        ]
    },
    'crypto': {
        'name': 'Cryptographic Issues',
        'rules': [
            20015,  # Heartbleed OpenSSL Vulnerability
            90024   # Generic Padding Oracle
        ]
    },
    'config': {
        'name': 'Security Configuration Issues',
        'rules': [
            40043,  # Log4Shell
            40045   # Spring4Shell
        ]
    },
    'injection': {
        'name': 'Other Injection Flaws',
        'rules': [
            90021,  # XPath Injection
            90023,  # XML External Entity Attack
            90017,  # XSLT Injection
            90026,  # SOAP Action Spoofing
            90029   # SOAP XML Injection
        ]
    },
    'input': {
        'name': 'Input Validation Issues',
        'rules': [
            30001,  # Buffer Overflow
            30002,  # Format String Error
            40003,  # CRLF Injection
            40008   # Parameter Tampering
        ]
    },
    'session': {
        'name': 'Session Management Issues',
        'rules': [
            10058  # GET for POST
        ]
    },
    'ssrf': {
        'name': 'Server-Side Request Forgery',
        'rules': [
            90034  # Cloud Metadata Potentially Exposed
        ]
    },
    'misc': {
        'name': 'Miscellaneous Vulnerabilities',
        'rules': [
            0,      # Directory Browsing
            20019,  # External Redirect
```

```
            50000,  # Script Active Scan Rules
            10104   # User Agent Fuzzer
        ]
    },
    'ssti': {
        'name': 'Server-Side Template Injection',
        'rules': [
            90035,  # Server Side Template Injection
            90036   # Server Side Template Injection (Blind)
        ]
    }
}

NUCLEI_EXECUTABLE_PATH = os.path.join("nuclei", "nuclei.exe")

def get_ip(target):
    """Resolve domain to IP address if necessary"""
    try:
        # Extract netloc (domain) from URL if present
        parsed_url = urlparse(target)
        domain = parsed_url.netloc if parsed_url.netloc else target.split('/')[0]
        return socket.gethostbyname(domain)
    except socket.gaierror:
        return target  # Return as is if resolution fails

def setup_scan_policies(zap):
    """
    Create and configure scan policies for different vulnerability types
    """
    try:
        # First, remove any existing policies with the same names
        existing_policies = zap.ascan.scan_policy_names
        for policy in SCAN_POLICIES.values():
            if policy['name'] in existing_policies:
                zap.ascan.remove_scan_policy(policy['name'])

        # Create new policies for each vulnerability type
        for vuln_type, policy_config in SCAN_POLICIES.items():
            # Create new policy
            zap.ascan.add_scan_policy(policy_config['name'])

            # Disable all scan rules first
            zap.ascan.disable_all_scanners(policy_config['name'])

            # Enable only specific rules for this policy
            for rule_id in policy_config['rules']:
                zap.ascan.enable_scanners([rule_id], policy_config['name'])

        return True

    except Exception as e:
        print(f"Error setting up scan policies: {str(e)}")
```

```
        return False

def perform_zap_scan(target_url, selected_vulns):
    """Perform ZAP scan with selected vulnerability types"""
    try:
        # Start new session
        zap.core.new_session()

        # Set up scan policies
        if not setup_scan_policies(zap):
            raise Exception("Failed to set up scan policies")

        # Spider the target
        scan_id = zap.spider.scan(target_url)

        # Wait for spider to complete
        while int(zap.spider.status(scan_id)) < 100:
            time.sleep(2)

        # Perform active scan for each selected vulnerability type
        scan_results = []
        seen_alerts = set()  # Set for storing unique alert keys

        nodes = zap.core.sites
        print(nodes)

        for vuln in selected_vulns:
            if vuln in SCAN_POLICIES:
                policy_name = SCAN_POLICIES[vuln]['name']
                print(f"Starting scan with policy: {policy_name}")

                # Start scan with specific policy
                ascan_id = zap.ascan.scan(
                    target_url,
                    scanpolicyname=policy_name,
                    recurse=True,
                    inscopeonly=False
                )

                # Wait for scan to complete
                while int(zap.ascan.status(ascan_id)) < 100:
                    time.sleep(2)

                # Get results for this policy
                alerts = zap.core.alerts()

                for alert in alerts:
                    alert_key = (alert['name'], alert['risk'],
alert['description'], alert['solution'])  # Уникальный ключ
                    if alert_key not in seen_alerts:
                        seen_alerts.add(alert_key)
                        scan_results.append(alert)
```

```
        # Generate report
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        report_filename = f'scan_report_{timestamp}.html'
        pdf_filename = f'scan_report_{timestamp}.pdf'

        # Save scan details to database
        save_scan_to_db(target_url, report_filename, pdf_filename)

        with open(f"static/reports/{report_filename}", 'w') as f:
            html_content = render_template(
                'report_template.html',
                scan_results=scan_results,
                target_url=target_url,
                timestamp=timestamp
            )
            f.write(html_content)

        # Convert HTML to PDF
        convert_to_pdf(f"static/reports/{report_filename}",
f"static/reports/{pdf_filename}")

        return scan_results

    except Exception as e:
        print(f"Error during ZAP scan: {str(e)}")
        return str(e)

def run_wapiti_scan(target_url, selected_modules):
    try:
        report_path =
f"static/reports/wapiti_{datetime.now().strftime('%Y%m%d_%H%M%S')}.json"
        command = ["wapiti", "-u", target_url, "-f", "json", "-o", report_path]

        if selected_modules:
            for module in selected_modules:
                command.extend(["-m", module])

        subprocess.run(command, capture_output=True, text=True, check=True)

        with open(report_path, "r", encoding="utf-8") as f:
            results = json.load(f)
        return results
    except Exception as e:
        print(f"Error running Wapiti scan: {str(e)}")
        return {}

def run_nuclei_scan(target_url):
    """Run Nuclei scan on a target URL"""
    try:
        if not os.path.exists(NUCLEI_EXECUTABLE_PATH):
```

```
            raise FileNotFoundError(f"nuclei.exe не знайдено за шляхом
{NUCLEI_EXECUTABLE_PATH}")

        with tempfile.NamedTemporaryFile(mode='w+', delete=False) as tmp_output:
            output_path = tmp_output.name

        templates_path = os.path.join("nuclei", "nuclei-templates")

        cmd = [
            NUCLEI_EXECUTABLE_PATH,
            "-u", target_url,
            "-t", templates_path,
            "-json-export", output_path
        ]
        subprocess.run(cmd, check=True)

        results = []
        seen_entries = set()  # to avoid duplicates

        with open(output_path, 'r', encoding='utf-8') as f:
            for line in f:
                try:
                    parsed = json.loads(line.strip())

                    parsed_items = parsed if isinstance(parsed, list) else [parsed]

                    for item in parsed_items:
                        info = item.get("info", {})
                        classification = info.get("classification", {})

                        name = info.get("name", "N/A")
                        matched = item.get("matched-at", "")

                        # Key to uniqueness
                        entry_key = (name, matched)
                        if entry_key in seen_entries:
                            continue  # skip duplicates
                        seen_entries.add(entry_key)

                        result_entry = {
                            "template_id": item.get("template-id", "N/A"),
                            "name": name,
                            "description": info.get("description", "N/A"),
                            "impact": info.get("impact", "N/A"),
                            "severity": info.get("severity", "info").capitalize(),
                            "cve_id": classification.get("cve-id", []),
                            "cwe_id": classification.get("cwe-id", []),
                            "cvss_score": classification.get("cvss-score", "N/A"),
                            "matched": matched,
                            "references": info.get("reference", [])
                        }
```

```
                        results.append(result_entry)

                except json.JSONDecodeError:
                    continue

        os.remove(output_path)
        return results

    except subprocess.CalledProcessError as e:
        print(f"[!] Nuclei scan failed: {e}")
        return []
    except FileNotFoundError as e:
        print(f"[!] Error: {e}")
        return []
    except Exception as e:
        print(f"[!] Error running Nuclei: {e}")
        return []

def parse_vulscan_output(script_output: str):
    """ Parses the result of the vulscan.nse script into a convenient structure"""
    parsed_results = []
    current_section = None
    current_vulns = []

    lines = script_output.strip().splitlines()
    for line in lines:
        line = line.strip()
        if not line:
            continue

        # Verification: this is the source title (e.g., MITRE CVE - https://...)
        header_match = re.match(r'^(.+?)\s+-\s+(https?://\S+):$', line)
        if header_match:
            # Save the previous block
            if current_section and current_vulns:
                parsed_results.append({
                    'source': current_section,
                    'vulnerabilities': current_vulns
                })
                current_vulns = []
            # Creating a new one
            current_section = header_match.group(1)
            continue

        # If it is a vulnerability
        vuln_match = re.match(r'^\[(.*?)\]\s*(.+)', line)
        if vuln_match:
            vuln_id = vuln_match.group(1)
            vuln_desc = vuln_match.group(2)
            current_vulns.append({
                'id': vuln_id,
                'description': vuln_desc
```

```
            })

    # Add the last block
    if current_section and current_vulns:
        parsed_results.append({
            'source': current_section,
            'vulnerabilities': current_vulns
        })

    return parsed_results

def perform_nmap_scan(target):
    """Perform Nmap port scan"""
    try:
        ip_address = get_ip(target)
        nm = nmap.PortScanner()
        nm.scan(ip_address, arguments='-sS -sV -p- --script vulscan/vulscan.nse')

        # Format results for template
        ports = []
        for host in nm.all_hosts():
            for proto in nm[host].all_protocols():
                for port in nm[host][proto]:
                    port_info = nm[host][proto][port]
                    ports.append({
                        'port': port,
                        'protocol': proto,
                        'service': port_info.get('name', ''),
                        'state': port_info.get('state', ''),
                        'product': port_info.get('product', ''),
                        'version': port_info.get('version', ''),
                        'extrainfo': port_info.get('extrainfo', ''),
                        'vulscan_results':
parse_vulscan_output(port_info.get('script', {}).get('vulscan', ''))
                    })

        return {
            'ports': ports,
            'error': None
        }
    except Exception as e:
        return {
            'ports': [],
            'error': str(e)
        }

def perform_google_dorking(target_url):
    """Perform Google Dorking"""
    try:
        api_key = "your-api-key"
        cse_id = "your-cse-id"
        service = build("customsearch", "v1", developerKey=api_key)
```

```
        dorks = [
            f'site:{target_url} ext:sql | ext:log | ext:bak | ext:old | ext:env',
            f'site:{target_url} intitle:"index of" "backup"',
            f'site:{target_url} inurl:admin',
            f'site:{target_url} inurl:login',
            f'site:{target_url} intitle:"Admin Login"',
            f'site:{target_url} inurl:phpmyadmin',
            f'site:{target_url} inurl:config',
            f'site:{target_url} intitle:"Index of /" +etc',
            f'site:{target_url} "Apache/2.4.1" inurl:status',
            f'site:{target_url} "Warning: mysql_" filetype:php',
            f'site:{target_url} intitle:"index of" "password"',
            f'site:{target_url} confidential',
            f'site:{target_url} "s3.amazonaws.com"',
        ]

        all_results = []

        for query in dorks:
            try:
                res = service.cse().list(q=query, cx=cse_id).execute()
                if 'items' in res:
                    all_results.extend(res['items'])
            except Exception as inner_e:
                print(f"[!] Error for dork '{query}': {inner_e}")
                continue

        print(f"[+] Google Dorking completed. Total results: {len(all_results)}")
        return all_results

    except Exception as e:
        print(f"Error during Google Dorking: {str(e)}")
        return []

def generate_html_report(scan_results, wapiti_results, nuclei_results,
port_scan_results, google_dorking_results, target_url, scan_start_str,
scan_end_str, scan_duration_str):
    """Generate HTML report from scan results"""
    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    report_file = f"report_{timestamp}.html"

    # Sort results by risk level (high to low)
    risk_order = {'High': 4, 'Medium': 3, 'Low': 2, 'Informational': 1}
    sorted_results = sorted(scan_results, key=lambda x: risk_order.get(x['risk'],
0), reverse=True)

    # Sort nuclei results by severity (from highest to lowest)
    severity_order = {'Critical': 5, 'High': 4, 'Medium': 3, 'Low': 2,
'Informational': 1, 'Info': 1, 'N/A': 0}
    nuclei_results = sorted(nuclei_results, key=lambda x:
severity_order.get(x['severity'], 0), reverse=True)
```

```
    with open(f"static/reports/{report_file}", 'w', encoding='utf-8',
errors="replace") as f:
        html_content = render_template(
            'report_template.html',
            scan_results=sorted_results,
            wapiti_results=wapiti_results,
            nuclei_results=nuclei_results,
            port_scan_results=port_scan_results,
            google_dorking_results=google_dorking_results,
            target_url=target_url,
            timestamp=timestamp,
            scan_start=scan_start_str,
            scan_end=scan_end_str,
            scan_duration=scan_duration_str,
            current_sort='desc'
        )
        f.write(html_content)

    return report_file

def convert_to_pdf(html_path, pdf_path):
    """Convert HTML report to PDF"""
    try:
        config = pdfkit.configuration(wkhtmltopdf=r'C:\\Program
Files\\wkhtmltopdf\\bin\\wkhtmltopdf.exe')
        options = {
            'page-size': 'A4',
            'margin-top': '0.00in',
            'margin-right': '0.00in',
            'margin-bottom': '0.00in',
            'margin-left': '0.00in',
            'encoding': "UTF-8"
        }
        pdfkit.from_file(html_path, pdf_path, options=options,
configuration=config)
        return True
    except Exception as e:
        print(f"Error converting HTML to PDF: {str(e)}")
        return False

def init_db():
    conn = sqlite3.connect('scans.db')
    c = conn.cursor()
    c.execute('''CREATE TABLE IF NOT EXISTS completed_scans
                 (id INTEGER PRIMARY KEY AUTOINCREMENT,
                  target_url TEXT NOT NULL,
                  scan_date TEXT NOT NULL,
                  report_file TEXT NOT NULL,
                  pdf_file TEXT NOT NULL)''')
    conn.commit()
    conn.close()
```

```
def save_scan_to_db(target_url, report_file, pdf_file):
    conn = sqlite3.connect('scans.db')
    c = conn.cursor()
    c.execute('INSERT INTO completed_scans (target_url, scan_date, report_file,
pdf_file) VALUES (?, ?, ?, ?)',
              (target_url, datetime.now().strftime('%Y-%m-%d %H:%M:%S'),
report_file, pdf_file))
    conn.commit()
    conn.close()

def delete_scan_from_db(scan_id):
    conn = sqlite3.connect('scans.db')
    c = conn.cursor()

    # Get file names before deletion
    c.execute('SELECT report_file, pdf_file FROM completed_scans WHERE id = ?',
(scan_id,))
    files = c.fetchone()

    if files:
        # Delete from database
        c.execute('DELETE FROM completed_scans WHERE id = ?', (scan_id,))
        conn.commit()
        conn.close()

        # Delete files
        report_path = os.path.join('static/reports', files[0])
        pdf_path = os.path.join('static/reports', files[1])

        try:
            if os.path.exists(report_path):
                os.remove(report_path)
            if os.path.exists(pdf_path):
                os.remove(pdf_path)
        except Exception as e:
            print(f"Error deleting files: {str(e)}")

        return True

    conn.close()
    return False

@app.route('/')
def index():
    return render_template('index.html', vuln_types=VULNERABILITY_TYPES,
wapiti_modules=WAPITI_MODULES)

@app.route('/scan', methods=['POST'])
def scan():
    try:
        data = request.get_json()
        target_url = data.get('target_url')
```

```
        selected_vulns = data.get('vuln_types', [])
        wapiti_modules = data.get('wapiti_modules', [])
        include_nuclei = data.get('include_nuclei', False)
        include_port_scan = data.get('include_port_scan', False)
        include_google_dork = data.get('include_google_dork', False)

        if not target_url:
            return jsonify({'error': 'Target URL is required'}), 400

        scan_start_time = datetime.now()

        zap_results = None
        if selected_vulns:
            # Start ZAP scan only if vulnerabilities are selected
            zap_results = perform_zap_scan(target_url, selected_vulns)

        wapiti_results  = None
        if wapiti_modules:
            wapiti_results  = run_wapiti_scan(target_url, wapiti_modules)

        nuclei_results = None
        if include_nuclei:
            nuclei_results = run_nuclei_scan(target_url)

        # Perform port scan if requested
        port_scan_results = None
        if include_port_scan:
            port_scan_results = perform_nmap_scan(target_url)

        # Perform Google Dorking
        google_dorking_results = None
        if include_google_dork:
            google_dorking_results = perform_google_dorking(target_url)

        # Generate timestamp for report files
        scan_end_time = datetime.now()
        scan_duration = scan_end_time - scan_start_time
        scan_start_str = scan_start_time.strftime('%Y-%m-%d %H:%M:%S')
        scan_end_str = scan_end_time.strftime('%Y-%m-%d %H:%M:%S')

        duration_seconds = int(scan_duration.total_seconds())
        hours, remainder = divmod(duration_seconds, 3600)
        minutes, seconds = divmod(remainder, 60)

        parts = []
        if hours:
            parts.append(f"{hours} hr")
        if minutes:
            parts.append(f"{minutes} min")
        if seconds:
            parts.append(f"{seconds} sec")
```

```
        scan_duration_str = ' '.join(parts)

        timestamp = scan_start_time.strftime("%Y%m%d_%H%M%S")
        report_filename = f'scan_report_{timestamp}.html'
        pdf_filename = f'scan_report_{timestamp}.pdf'

        # Generate HTML report including Google Dorking results
        zap_results = zap_results or {}
        wapiti_results  = wapiti_results  or {}
        nuclei_results = nuclei_results or {}
        port_scan_results = port_scan_results or {}
        google_dorking_results = google_dorking_results or {}
        report_filename = generate_html_report(zap_results, wapiti_results,
nuclei_results, port_scan_results, google_dorking_results, target_url,
scan_start_str, scan_end_str, scan_duration_str)

        # Convert to PDF
        convert_to_pdf(f"static/reports/{report_filename}",
f"static/reports/{pdf_filename}")

        # Save scan to database
        save_scan_to_db(target_url, report_filename, pdf_filename)

        return jsonify({
            'status': 'success',
            'message': 'Scan completed successfully',
            'report_url': f'/reports/{report_filename}'
        })

    except Exception as e:
        return jsonify({'error': str(e)}), 500

@app.route('/completed_scans')
def completed_scans():
    conn = sqlite3.connect('scans.db')
    c = conn.cursor()
    c.execute('SELECT * FROM completed_scans ORDER BY scan_date DESC')
    scans = c.fetchall()
    conn.close()
    return render_template('completed_scans.html', scans=scans)

@app.route('/reports/<filename>')
def get_report(filename):
    return send_file(f'static/reports/{filename}')

@app.route('/download_pdf/<filename>')
def download_pdf(filename):
    try:
        # Get the HTML file path
        html_path = os.path.join('static', 'reports', filename)

        # Get PDF filename from the database to maintain the original name
```

```
        conn = sqlite3.connect('scans.db')
        c = conn.cursor()
        c.execute('SELECT pdf_file FROM completed_scans WHERE report_file = ?',
(filename,))
        result = c.fetchone()
        conn.close()

        if result:
            # Use existing PDF filename
            pdf_filename = result[0]
        else:
            # Generate new PDF filename if not found in database
            pdf_filename = filename.rsplit('.', 1)[0] + '.pdf'

        pdf_path = os.path.join('static', 'reports', pdf_filename)

        # Remove existing PDF if it exists
        if os.path.exists(pdf_path):
            os.remove(pdf_path)

        # Convert HTML to PDF
        if convert_to_pdf(html_path, pdf_path):
            return send_file(pdf_path, as_attachment=True,
download_name=pdf_filename)
        else:
            return jsonify({'error': 'Failed to convert to PDF'}), 500

    except Exception as e:
        return jsonify({'error': str(e)}), 500

@app.route('/delete_scan/<int:scan_id>', methods=['POST'])
def delete_scan(scan_id):
    try:
        if delete_scan_from_db(scan_id):
            return jsonify({'status': 'success', 'message': 'Scan deleted
successfully'})
        else:
            return jsonify({'error': 'Scan not found'}), 404
    except Exception as e:
        return jsonify({'error': str(e)}), 500

if __name__ == '__main__':
    # Create reports directory if it doesn't exist
    os.makedirs('static/reports', exist_ok=True)
    init_db()
    app.run(debug=True)
```

Лістинг А.2 – Головна HTML-сторінка інтерфейсу (templates/index.html)

```
{% extends "base.html" %}

{% block content %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Web Security Scanner</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
rel="stylesheet">
    <link href="{{ url_for('static', filename='css/style.css') }}"
rel="stylesheet">
</head>
<body>
    <div class="container mt-5">
        <h1 class="text-center mb-4">Web Security Scanner</h1>

        <div class="card">
            <div class="card-body">
                <form id="scanForm">
                    <div class="mb-3">
                        <label for="target_url" class="form-label">Target
URL:</label>
                        <input type="url" class="form-control" id="target_url"
name="target_url" required>
                    </div>

                    <div class="mb-3">
                        <label class="form-label">OWASP ZAP Vulnerability
Types:</label>

                        <div class="form-check">
                            <input class="form-check-input" type="checkbox"
id="selectAllZapModules">
                            <label class="form-check-label fw-bold"
for="selectAllZapModules">
                                Full OWASP ZAP Vulnerability Scan (Select all
categories)
                            </label>
                        </div>

                        {% for key, value in vuln_types.items() %}
                        <div class="form-check ms-3">
                            <input class="form-check-input zap-module"
type="checkbox" name="vuln_types" value="{{ key }}" id="zap_{{ key }}">
                            <label class="form-check-label" for="zap_{{ key }}">
                                {{ value }}
                            </label>
```

```
                        </div>
                        {% endfor %}
                    </div>

                    <div class="mb-3">
                        <label class="form-label">Wapiti Vulnerability
Modules:</label>

                        <div class="form-check">
                            <input class="form-check-input" type="checkbox"
id="selectAllWapitiModules">
                            <label class="form-check-label fw-bold"
for="selectAllWapitiModules">
                                Full Wapiti Vulnerability Scan (Select all modules)
                            </label>
                        </div>

                        <div class="form-check">
                            <input class="form-check-input" type="checkbox"
id="selectDefaultWapitiModules">
                            <label class="form-check-label fw-bold"
for="selectDefaultWapitiModules">
                                Standard Wapiti Vulnerability Scan (Select default
modules)
                            </label>
                        </div>

                        {% for key, value in wapiti_modules.items() %}
                        <div class="form-check ms-3">
                            <input class="form-check-input wapiti-module"
type="checkbox" name="wapiti_modules" value="{{ key }}" id="{{ key }}">
                            <label class="form-check-label" for="{{ key }}">
                                {{ value }}
                            </label>
                        </div>
                        {% endfor %}
                    </div>

                    <div class="mb-3">
                        <div class="form-check">
                            <input class="form-check-input" type="checkbox"
id="include_nuclei">
                            <label class="form-check-label" for="include_nuclei">
                                Run Nuclei Scan
                            </label>
                        </div>
                    </div>

                    <div class="mb-3">
                        <div class="form-check">
                            <input class="form-check-input" type="checkbox"
name="include_port_scan" id="include_port_scan" value="true">
```

```
                            <label class="form-check-label"
for="include_port_scan">
                                Include Port Scan (Nmap)
                            </label>
                        </div>
                    </div>

                    <div class="mb-3">
                        <div class="form-check">
                            <input class="form-check-input" type="checkbox"
name="include_google_dork" id="include_google_dork" value="true">
                            <label class="form-check-label"
for="include_google_dork">
                                Include Google Dorking Scan
                            </label>
                        </div>
                    </div>

                    <button type="submit" class="btn btn-primary"
id="startScan">Start Scan</button>
                </form>

                <div id="scanStatus" class="mt-4" style="display: none;">
                    <div class="alert alert-info">
                        <div class="d-flex align-items-center">
                            <div class="spinner-border spinner-border-sm me-2"
role="status">
                                <span class="visually-hidden">Loading...</span>
                            </div>
                            <span>Scanning in progress...</span>
                        </div>
                    </div>
                </div>

                <div id="scanResults" class="mt-4" style="display: none;">
                    <h3>Scan Results</h3>
                    <div id="resultsContent"></div>
                    <div class="mt-3">
                        <a href="#" class="btn btn-primary me-2"
id="viewReportBtn">View Report</a>
                        <a href="#" class="btn btn-success"
id="downloadPdfBtn">Download PDF</a>
                    </div>
                </div>
            </div>
        </div>
    </div>

    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
    <script src="{{ url_for('static', filename='js/main.js') }}"></script>
{% endblock %}
</body>
```

```
</html>
```

Лістинг А.3 – Шаблон HTML-звіту про результати сканування (templates/report_template.html)

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Security Scan Report - {{ target_url }}</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
rel="stylesheet">
    <style>
        body { padding: 20px; }
        .vulnerability { margin-bottom: 20px; }
        .port-info { margin-bottom: 10px; }
        .timestamp { color: grey; }
        .risk-high { border-left: 5px solid #dc3545; }
        .risk-medium { border-left: 5px solid #ffc107; }
        .risk-low { border-left: 5px solid #0dcaf0; }
        .risk-informational { border-left: 5px solid grey; }
        .risk-badge {
            padding: 5px 10px;
            border-radius: 4px;
            color: white;
            font-weight: bold;
        }
        .risk-badge.critical { background-color: #47040b; color: white; border: 1px
solid black;}
        .risk-badge.high { background-color: #dc3545; color: white; border: 1px
solid black;}
        .risk-badge.medium { background-color: #ffc107; color: black; border: 1px
solid black;}
        .risk-badge.low { background-color: #0dcaf0; border: 1px solid black;}
        .risk-badge.informational { background-color: grey; color: white; border:
1px solid black; }
    </style>
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js">
</script>
</head>
<body>
    <div class="container">
        <h1 class="mb-4">Security Scan Report</h1>
        <p class="timestamp">Generated on: {{ timestamp }}</p>
        <h2 class="mb-3">Target: {{ target_url }}</h2>
        <p><strong>Start of audit: </strong> {{ scan_start }}</p>
        <p><strong>End of audit: </strong> {{ scan_end }}</p>
        <p><strong>Total duration of audit: </strong> {{ scan_duration }}</p>
```

```
{% if port_scan_results %}
<div class="card mb-4">
    <div class="card-header">
        <h3 class="mb-0">Port Scan Results (Nmap)</h3>
    </div>
    <div class="card-body">
            <h4>Open Ports:</h4>
            <table class="table table-striped">
                <thead>
                    <tr>
                        <th>Port</th>
                        <th>Protocol</th>
                        <th>Service</th>
                        <th>State</th>
                        <th>Product</th>
                        <th>Version</th>
                        <th>Extra Info</th>
                    </tr>
                </thead>
                <tbody>
                    {% set mitre_cve_count = 0 %}
                    {% for port in port_scan_results.ports %}
                    <tr>
                        <td>{{ port.port }}</td>
                        <td>{{ port.protocol }}</td>
                        <td>{{ port.service }}</td>
                        <td>{{ port.state }}</td>
                        <td>{{ port.product }}</td>
                        <td>{{ port.version }}</td>
                        <td>{{ port.extrainfo }}</td>
                    </tr>
                    {% if port.vulscan_results %}
                        {% for source in port.vulscan_results %}
                            {% if source.source == 'MITRE CVE' %}
                            <tr>
                                <td colspan="7">
                                    <div class="mt-2">
                                        <strong>Vulnerabilities from MITRE CVE ({{
source.vulnerabilities|length }})</strong>
                                        {% set mitre_cve_count = mitre_cve_count +
source.vulnerabilities|length %}
                                        <div class="accordion" id="accordion-{{
loop.index0 }}-{{ loop.index }}">
                                            <div class="accordion-item">
                                                <h2 class="accordion-header"
id="heading-{{ loop.index0 }}-{{ loop.index }}">
                                                    <button class="accordion-button
collapsed" type="button" data-bs-toggle="collapse" data-bs-target="#collapse-{{
loop.index0 }}-{{ loop.index }}" aria-expanded="false" aria-controls="collapse-{{
loop.index0 }}-{{ loop.index }}">
```

```
                                                    MITRE CVE ({{
source.vulnerabilities|length }})
                                                </button>
                                            </h2>
                                            <div id="collapse-{{ loop.index0
}}-{{ loop.index }}" class="accordion-collapse collapse" aria-labelledby="heading-
{{ loop.index0 }}-{{ loop.index }}" data-bs-parent="#accordion-{{ loop.index0 }}-{{
loop.index }}">
                                                <div class="accordion-body">
                                                    <ul class="list-group list-
group-flush">
                                                        {% for vuln in
source.vulnerabilities %}
                                                            <li class="list-
group-item">
                                                                <strong>{{
vuln.id }}</strong>: {{ vuln.description }}
                                                            </li>
                                                        {% endfor %}
                                                    </ul>
                                                </div>
                                            </div>
                                        </div>
                                    </div>
                                </td>
                            </tr>
                            {% endif %}
                        {% endfor %}
                    {% endif %}
                    {% endfor %}
                </tbody>
            </table>
    </div>
</div>
{% endif %}
    <div class="card mb-4">
        <div class="card-header">
            <h3 class="mb-0">Google Dorking Results</h3>
        </div>
        <div class="card-body">
                {% if google_dorking_results %}
                    {% for result in google_dorking_results %}
                        <div class="dorking-result">
                            <h4>{{ result.title }}</h4>
                            <p>{{ result.snippet }}</p>
                            <a href="{{ result.link }}" target="_blank">Read
more</a>
                        </div>
                    {% endfor %}
                {% else %}
```

```
                    <div class="alert alert-info">No Google Dorking results found
for this target.</div>
                {% endif %}
            </div>
        </div>

        {% if google_dork_results %}
        <div class="section">
            <h2>Google Dorking Results</h2>
            <div class="content">
                {% for dork_type, results in google_dork_results.items() %}
                    <h3>{{ dork_type | replace('_', ' ') | title }}</h3>
                    {% if results is string %}
                        <p class="text-danger">{{ results }}</p>
                    {% else %}
                        {% if results %}
                            <ul class="list-group">
                            {% for result in results %}
                                <li class="list-group-item">
                                    <h5><a href="{{ result.link }}"
target="_blank">{{ result.title }}</a></h5>
                                    <p>{{ result.snippet }}</p>
                                </li>
                            {% endfor %}
                            </ul>
                        {% else %}
                            <p>No results found</p>
                        {% endif %}
                    {% endif %}
                {% endfor %}
            </div>
        </div>
        {% endif %}

        {% if scan_results %}
        <div class="card">
            <div class="card-header d-flex justify-content-between align-items-
center">
                <h3 class="mb-0">Vulnerability Scan Results (ZAP)</h3>
                <div class="sort-controls">
                    <select id="sortOrder" class="form-select"
onchange="sortVulnerabilities(this.value)">
                        <option value="desc" {% if current_sort == 'desc'
%}selected{% endif %}>Highest Risk First</option>
                        <option value="asc" {% if current_sort == 'asc'
%}selected{% endif %}>Lowest Risk First</option>
                    </select>
                </div>
            </div>
            <div class="card-body" id="vulnerabilitiesContainer">
                {% if not scan_results %}
                    <div class="alert alert-info">No vulnerabilities found.</div>
```

```
                {% else %}
                    {% for result in scan_results %}
                    <div class="vulnerability card mb-3 risk-{{ result.risk.lower()
}}" data-risk="{{ result.risk }}">
                        <div class="card-header d-flex justify-content-between
align-items-center">
                            <h4 class="mb-0">{{ result.name }}</h4>
                            <span class="risk-badge {{ result.risk.lower() }}">{{
result.risk }}</span>
                        </div>
                        <div class="card-body">
                            <p><strong>Description:</strong> {{ result.description
}}</p>
                            {% if result.url %}
                            <p><strong>URL:</strong> {{ result.url }}</p>
                            {% endif %}
                            {% if result.solution %}
                            <p><strong>Solution:</strong> {{ result.solution }}</p>
                            {% endif %}
                        </div>
                    </div>
                    {% endfor %}
                {% endif %}
            </div>
        </div>
        {% endif %}

        {% if nuclei_results %}
        <div class="card">
            <div class="card-header d-flex justify-content-between align-items-
center">
                <h3 class="mb-0">Vulnerability Scan Results (Nuclei)</h3>
                <div class="sort-controls">
                    <select id="nucleiSortOrder" class="form-select"
onchange="sortNucleiVulnerabilities(this.value)">
                        <option value="desc" selected>Highest Risk First</option>
                        <option value="asc">Lowest Risk First</option>
                    </select>
                </div>
            </div>
            <div class="card-body" id="nucleiResultsContainer">
                {% if not nuclei_results %}
                    <div class="alert alert-info">No vulnerabilities found.</div>
                {% else %}
                    {% for item in nuclei_results %}
                    <div class="vulnerability card mb-3" data-severity="{{
item.severity | default('N/A') | lower }}">
                        <div class="card-header d-flex justify-content-between
align-items-start flex-wrap">
                            <h4 class="mb-2">{{ item.name | default('N/A') }}</h4>
                            <div class="d-flex gap-2 align-items-center flex-wrap">
                                {# CVSS Badge #}
```

```
                            {% set cvss = item.cvss_score | default('N/A') %}
                            {% if cvss != 'N/A' %}
                                {% set cvss_val = cvss | float %}
                                <span class="risk-badge
                                    {% if cvss_val >= 9 %}critical
                                    {% elif cvss_val >= 7 %}high
                                    {% elif cvss_val >= 4 %}medium
                                    {% elif cvss_val > 0 %}low
                                    {% else %}informational{% endif %}">
                                    CVSS: {{ "%.1f"|format(cvss_val) }}
                                </span>
                            {% else %}
                                <span class="risk-badge informational">CVSS:
N/A</span>
                            {% endif %}

                            {# Severity Badge #}
                            {% set sev = item.severity | default('N/A') | lower
%}
                            <span class="risk-badge
                                {% if sev == 'critical' %}critical
                                {% elif sev == 'high' %}high
                                {% elif sev == 'medium' %}medium
                                {% elif sev == 'low' %}low
                                {% elif sev == 'informational' %}informational
                                {% else %}informational{% endif %}">
                                {{ item.severity | default('N/A') | title }}
                            </span>
                        </div>
                    </div>
                    <div class="card-body">
                        <p><strong>CVE ID:</strong>
                            {% if item.cve_id %}
                                {{ item.cve_id | join(', ') }}
                            {% else %}
                                N/A
                            {% endif %}
                        </p>

                        <p><strong>CWE ID:</strong>
                            {% if item.cwe_id %}
                                {{ item.cwe_id | join(', ') }}
                            {% else %}
                                N/A
                            {% endif %}
                        </p>

                        <p><strong>Matched URL:</strong> {{ item.matched |
default('N/A') }}</p>
                        <p><strong>Description:</strong> {{ item.description |
default('N/A') }}</p>
```

```
                            <p><strong>Impact:</strong> {{ item.impact |
default('N/A') }}</p>

                            <p><strong>References:</strong>
                                {% if item.references %}
                                    <ul class="mb-0 ps-3">
                                        {% for ref in item.references %}
                                            <li><a href="{{ ref }}"
target="_blank">{{ ref }}</a></li>
                                        {% endfor %}
                                    </ul>
                                {% else %}
                                    N/A
                                {% endif %}
                            </p>
                        </div>
                    </div>
                    {% endfor %}
                {% endif %}
            </div>
        </div>
        {% endif %}

        {% if wapiti_results %}
            <h3>Vulnerability Scan Results (Wapiti)</h3>
            <table class="table table-bordered">
                <thead>
                    <tr>
                        <th>Vulnerability</th>
                        <th>Description</th>
                        <th>Solution</th>
                        <th>References</th>
                    </tr>
                </thead>
                <tbody>
                    {% for classification, details in
wapiti_results['classifications'].items() %}
                        <tr>
                            <td>{{ classification }}</td>
                            <td>{{ details['desc'] }}</td>
                            <td>{{ details['sol'] }}</td>
                            <td>
                                {% for ref_name, ref_url in details['ref'].items()
%}
                                    <a href="{{ ref_url }}" target="_blank">{{
ref_name }}</a><br>
                                {% endfor %}
                            </td>
                        </tr>
                    {% endfor %}
                </tbody>
            </table>
```

```
        {% endif %}
    </div>
    <script>
        // Store the current sort order for use when downloading the report
        let currentSortOrder = '{{ current_sort }}';

        function sortVulnerabilities(order) {
            currentSortOrder = order;
            const container = document.getElementById('vulnerabilitiesContainer');
            const vulnerabilities =
Array.from(container.getElementsByClassName('vulnerability'));

            const riskOrder = {
                'High': 4,
                'Medium': 3,
                'Low': 2,
                'Informational': 1
            };

            vulnerabilities.sort((a, b) => {
                const riskA = riskOrder[a.dataset.risk] || 0;
                const riskB = riskOrder[b.dataset.risk] || 0;
                return order === 'desc' ? riskB - riskA : riskA - riskB;
            });

            // Clear and re-append sorted elements
            vulnerabilities.forEach(v => container.appendChild(v));

            // Store the sort preference in localStorage
            localStorage.setItem('reportSortOrder', order);
        }

        // Initialize sorting immediately when the page loads
        document.addEventListener('DOMContentLoaded', () => {
            // Always sort initially by the server-provided sort order
            sortVulnerabilities(currentSortOrder);

            // Then check if there's a stored preference
            const storedOrder = localStorage.getItem('reportSortOrder');
            if (storedOrder && storedOrder !== currentSortOrder) {
                document.getElementById('sortOrder').value = storedOrder;
                sortVulnerabilities(storedOrder);
            }
        });

        // Handle sort order change
        function sortNucleiVulnerabilities(order) {
        const container = document.getElementById('nucleiResultsContainer');
        const cards = Array.from(container.querySelectorAll('.vulnerability'));

        const severityRank = {
            'critical': 5,
```

```
            'high': 4,
            'medium': 3,
            'low': 2,
            'informational': 1,
            'info': 1,
            'n/a': 0,
            'unknown': 0
        };

        cards.sort((a, b) => {
            const aVal = severityRank[a.dataset.severity] || 0;
            const bVal = severityRank[b.dataset.severity] || 0;
            return order === 'asc' ? aVal - bVal : bVal - aVal;
        });

        // Update DOM in the correct order
        cards.forEach(card => container.appendChild(card));
    }
    </script>
</body>
</html>
```

Лістинг А.4 – Базовий шаблон сторінки (templates/base.html)

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Web Security Scanner</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
rel="stylesheet">
    <link href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/5.15.4/css/all.min.css" rel="stylesheet">
</head>
<body>
    <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
        <div class="container">
            <a class="navbar-brand" href="{{ url_for('index') }}">Web Security
Scanner</a>
            <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
data-bs-target="#navbarNav">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse" id="navbarNav">
                <ul class="navbar-nav">
                    <li class="nav-item">
                        <a class="nav-link" href="{{ url_for('index') }}">New
scan</a>
                    </li>
```

```
                    <li class="nav-item">
                        <a class="nav-link" href="{{ url_for('completed_scans')
}}">Completed scans</a>
                    </li>
                </ul>
            </div>
        </div>
    </nav>

    {% block content %}{% endblock %}

    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js">
</script>
</body>
</html>
```

Лістинг А.5 – Сторінка зі списком завершених сканувань (templates/completed_scans.html)

```
{% extends "base.html" %}

{% block content %}
<div class="container mt-5">
    <h2 class="mb-4">Completed scans</h2>

    <div id="alertContainer"></div>

    <div class="table-responsive">
        <table class="table table-striped">
            <thead>
                <tr>
                    <th>Target URL</th>
                    <th>Scan date</th>
                    <th>Actions</th>
                </tr>
            </thead>
            <tbody>
                {% for scan in scans %}
                <tr id="scan-{{ scan[0] }}">
                    <td>{{ scan[1] }}</td>
                    <td>{{ scan[2] }}</td>
                    <td>
                        <div class="btn-group">
                            <a href="{{ url_for('get_report', filename=scan[3]) }}"
class="btn btn-primary btn-sm" target="_blank">
                                <i class="fas fa-eye"></i> View HTML
                            </a>
                            <a href="{{ url_for('download_pdf', filename=scan[3])
}}" class="btn btn-success btn-sm">
```

```
                                <i class="fas fa-download"></i> Download PDF
                            </a>
                            <button class="btn btn-danger btn-sm delete-scan" data-
scan-id="{{ scan[0] }}">
                                <i class="fas fa-trash"></i> Delete
                            </button>
                        </div>
                    </td>
                </tr>
                {% endfor %}
            </tbody>
        </table>
    </div>
</div>

<script>
document.addEventListener('DOMContentLoaded', function() {
    const alertContainer = document.getElementById('alertContainer');

    function showAlert(message, type = 'success') {
        const alert = document.createElement('div');
        alert.className = `alert alert-${type} alert-dismissible fade show`;
        alert.innerHTML = `
            ${message}
            <button type="button" class="btn-close" data-bs-
dismiss="alert"></button>
        `;
        alertContainer.appendChild(alert);

        setTimeout(() => {
            alert.classList.remove('show');
            setTimeout(() => alert.remove(), 150);
        }, 3000);
    }

    document.querySelectorAll('.delete-scan').forEach(button => {
        button.addEventListener('click', function() {
            const scanId = this.dataset.scanId;
            const row = document.getElementById(`scan-${scanId}`);

            if (confirm('Are you sure you want to delete this report?')) {
                fetch(`/delete_scan/${scanId}`, {
                    method: 'POST'
                })
                .then(response => response.json())
                .then(data => {
                    if (data.error) {
                        throw new Error(data.error);
                    }
                    row.remove();
                    showAlert('Report successfully deleted');
                })
```

```
                .catch(error => {
                    showAlert(error.message, 'danger');
                });
            }
        });
    });
});
</script>
{% endblock %}
```

Лістинг А.6 – Основна логіка взаємодії на стороні клієнта (static/js/main.js)

```
document.addEventListener('DOMContentLoaded', function () {
    const scanForm = document.getElementById('scanForm');
    const scanStatus = document.getElementById('scanStatus');
    const scanResults = document.getElementById('scanResults');
    const resultsContent = document.getElementById('resultsContent');
    const startScanBtn = document.getElementById('startScan');

    const fullWapitiCheckbox = document.getElementById('selectAllWapitiModules');
    const defaultWapitiCheckbox =
document.getElementById('selectDefaultWapitiModules');
    const wapitiModuleCheckboxes = document.querySelectorAll('.wapiti-module');

    const DEFAULT_WAPITI_MODULES = [
        'cookieflags', 'csp', 'exec', 'file', 'http_headers',
        'permanentxss', 'redirect', 'sql', 'ssl', 'ssrf', 'upload', 'xss'
    ];

    function updateSelectButtonsState() {
        const checkedModules = Array.from(wapitiModuleCheckboxes)
            .filter(cb => cb.checked)
            .map(cb => cb.value);

        const allModules = Array.from(wapitiModuleCheckboxes).map(cb => cb.value);

        if (checkedModules.length === allModules.length) {
            fullWapitiCheckbox.checked = true;
            defaultWapitiCheckbox.checked = false;
        }
        else if (
            checkedModules.length === DEFAULT_WAPITI_MODULES.length &&
            DEFAULT_WAPITI_MODULES.every(mod => checkedModules.includes(mod))
        ) {
            defaultWapitiCheckbox.checked = true;
            fullWapitiCheckbox.checked = false;
        }
        else {
            fullWapitiCheckbox.checked = false;
            defaultWapitiCheckbox.checked = false;
        }
```

```
    }


    const fullZapCheckbox = document.getElementById('selectAllZapModules');
    const zapModuleCheckboxes = document.querySelectorAll('.zap-module');

    fullZapCheckbox.addEventListener('change', function () {
        zapModuleCheckboxes.forEach(cb => cb.checked = this.checked);
    });

    zapModuleCheckboxes.forEach(cb => {
        cb.addEventListener('change', () => {
            const allChecked = Array.from(zapModuleCheckboxes).every(cb =>
cb.checked);
            fullZapCheckbox.checked = allChecked;
        });
    });

    fullWapitiCheckbox.addEventListener('change', function () {
        if (this.checked) {
            wapitiModuleCheckboxes.forEach(cb => cb.checked = true);
            defaultWapitiCheckbox.checked = false;
        } else {
            wapitiModuleCheckboxes.forEach(cb => cb.checked = false);
        }
    });

    defaultWapitiCheckbox.addEventListener('change', function () {
        if (this.checked) {
            wapitiModuleCheckboxes.forEach(cb => {
                if (DEFAULT_WAPITI_MODULES.includes(cb.value)) cb.checked = true;
            });
            fullWapitiCheckbox.checked = false;
        } else {
            wapitiModuleCheckboxes.forEach(cb => {
                if (DEFAULT_WAPITI_MODULES.includes(cb.value)) cb.checked = false;
            });
        }
    });

    wapitiModuleCheckboxes.forEach(cb => {
        cb.addEventListener('change', updateSelectButtonsState);
    });

    scanForm.addEventListener('submit', function (e) {
        e.preventDefault();

        startScanBtn.disabled = true;
        startScanBtn.innerHTML = '<span class="spinner-border spinner-border-sm me-
2" role="status" aria-hidden="true"></span>Scanning...';
        scanStatus.style.display = 'block';
        scanResults.style.display = 'none';
```

```
        const selectedVulns =
Array.from(document.querySelectorAll('input[name="vuln_types"]:checked')).map(cb =>
cb.value);
        const selectedWapitiModules =
Array.from(document.querySelectorAll('input[name="wapiti_modules"]:checked')).map(c
b => cb.value);
        const includePortScan =
document.getElementById('include_port_scan').checked;
        const includeGoogleDork =
document.getElementById('include_google_dork')?.checked || false;
        const includeNuclei = document.getElementById('include_nuclei')?.checked ||
false;

        const data = {
            target_url: document.getElementById('target_url').value,
            vuln_types: selectedVulns,
            wapiti_modules: selectedWapitiModules,
            include_port_scan: includePortScan,
            include_google_dork: includeGoogleDork,
            include_nuclei: includeNuclei
        };

        fetch('/scan', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
            },
            body: JSON.stringify(data)
        })
            .then(response => response.json())
            .then(data => {
                scanStatus.style.display = 'none';
                startScanBtn.disabled = false;
                startScanBtn.innerHTML = 'Start Scan';

                if (data.error) throw new Error(data.error);

                scanResults.style.display = 'block';

                if (data.report_url) {
                    const viewReportBtn = document.getElementById('viewReportBtn');
                    const downloadPdfBtn =
document.getElementById('downloadPdfBtn');
                    if (viewReportBtn) viewReportBtn.href = data.report_url;
                    if (downloadPdfBtn) downloadPdfBtn.href =
data.report_url.replace('/reports/', '/download_pdf/');
                }

                if (data.message) {
                    resultsContent.innerHTML = `<div class="alert alert-
success">${data.message}</div>`;
```

```
                }
            })
            .catch(error => {
                scanStatus.style.display = 'none';
                startScanBtn.disabled = false;
                startScanBtn.innerHTML = 'Start Scan';
                resultsContent.innerHTML = `<div class="alert alert-danger">Error
during scan: ${error.message}</div>`;
                scanResults.style.display = 'block';
            });
    });
});
```

Лістинг А.7 – Файл стилів інтерфейсу користувача (static/css/style.css)

```
.vulnerability-options {
    max-height: 300px;
    overflow-y: auto;
    padding: 10px;
    border: 1px solid #dee2e6;
    border-radius: 4px;
}

.form-check {
    margin-bottom: 10px;
}

#scanStatus {
    transition: all 0.3s ease;
}

.card {
    box-shadow: 0 2px 4px rgba(0,0,0,0.1);
}

.btn-group {
    gap: 10px;
}

.alert {
    animation: fadeIn 0.3s ease;
}

@keyframes fadeIn {
    from {
        opacity: 0;
        transform: translateY(-10px);
    }
    to {
        opacity: 1;
        transform: translateY(0);
```

```
    }
}

.cvss-badge, .severity-badge {
    padding: 4px 8px;
    border-radius: 4px;
    font-weight: bold;
    display: inline-block;
    font-size: 0.9em;
    border: 1px solid #00000033;
}

.severity-critical, .cvss-critical {
    background-color: #4b0303;
}

.severity-high, .cvss-high {
    background-color: #dc3545;
    color: white;
}

.severity-medium, .cvss-medium {
    background-color: #ffc107;
    color: black;
}

.severity-low, .cvss-low {
    background-color: #0dcaf0;
    color: white;
}

.severity-info, .cvss-info {
    background-color: grey;
    color: white;
}
```