Хмельницький національний університет
Факультет інформаційних технологій
Кафедра кібербезпеки

# КВАЛІФІКАЦІЙНА РОБОТА

Борусевича Павла Олеговича

на здобуття ступеня вищої освіти Бакалавра

Система захисту вебсерверів на основі аналізу атак і вразливостей

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

Шифр КРБКБ.220104.22.01.03 ПЗ

Виконав студент 4 курсу група КБ-22-1 Павло БОРУСЕВИЧ
Керівник канд. техн. наук, доцент Володимир ДЖУЛІЙ
Нормоконтролер д-р філософії Наталія ПЕТЛЯК

До захисту допускаю:
Завідувач кафедри кібербезпеки Юрій КЛЬОЦ

4 06 2026 р.

Хмельницький 2026

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Кібербезпеки

Рівень вищої освіти Бакалавр

Галузь знань 12 – Інформаційні технології

Спеціальність 125 – Кібербезпека

Освітня програма Кібербезпека

ЗАТВЕРДЖУЮ

Завідувач кафедри кібербезпеки

Юрій КЛЬОЦ

09 лютого 2026 р.

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

Борусевича Павла Олеговича

1 Тема роботи Система захисту вебсерверів на основі аналізу атак і вразливостей

Керівник роботи к.т.н., доцент, Джулій В.М.

Затверджено наказом ректора університету від 7 лютого 2025 № 23

2 Строк подання студентом кваліфікаційної роботи на кафедру 25 травня 2026

3 Вихідні дані до роботи Дослідити методи аудиту безпеки (DAST, OSINT) та стандарти вразливостей (OWASP, CWE). Розробити мовою Python автоматизований вебсканер із підсистемами пасивної мережевої розвідки та активного пошуку вразливостей (SQLi, XSS). Інтегрувати LLM для інтелектуальної генерації рекомендацій щодо усунення загроз. Створити графічний інтерфейс користувача, систему формування звітів (HTML/TXT) та провести практичну апробацію комплексу.

4 Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Вступ. Аналіз предметної області та існуючих систем аудиту безпеки вебдодатків. Дослідження класифікацій вебзагроз (OWASP, CWE), методів тестування безпеки (DAST, SAST) та ролі OSINT. Порівняльний аналіз сканерів-аналогів. Постановка задачі. Формування вимог до програмного продукту та обґрунтування модульної архітектури. Проєктування алгоритмів мережевої розвідки, картографування та пошуку вразливостей (SQLi, XSS). Програмна реалізація модулів сканера мовою Python. Розробка графічного інтерфейсу та підсистеми генерації звітів з інтеграцією LLM-моделі для надання рекомендацій. Практична апробація та тестування системи. Висновки.

5 Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

Загальна архітектура та зв'язки модулів програмного комплексу. Блок-схема алгоритму процесу динамічного сканування та виявлення вразливостей. Діаграма порівняння підходів до тестування безпеки (SAST, DAST, IAST).

6 Консультанти розділів кваліфікаційної роботи

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

7 Дата видачі завдання 09 лютого 2026 р.

КАЛЕНДАРНИЙ ПЛАН

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

Студент _____ Павло БОРУСЕВИЧ

Керівник кваліфікаційної роботи _____ Володимир ДЖУЛІЙ

# АНОТАЦІЯ

Автор роботи: Борусевич Павло Олегович

Керівник роботи: к.т.н., доцент, Джулій Володимир Миколайович

Пояснювальна записка: 56 с., 22 рис., 40 джерел

Графічна частина: 3 плакати

Ключові слова: кібербезпека, вразливість веб-сайтів, сканер вразливостей, DAST, OSINT, штучний інтелект, SQL-ін'єкція, XSS, Python, проект «Сканер вразливостей веб-сайтів».

У кваліфікаційній роботі проведено аналіз сучасних кіберзагроз та методів аудиту безпеки вебдодатків. Досліджено класифікацію вразливостей за міжнародними стандартами OWASP Top 10 та CWE, а також розглянуто існуючі підходи до тестування на проникнення (DAST, SAST) і методики розвідки на основі відкритих джерел (OSINT). Виконано проєктування архітектури комплексного сканера вразливостей, що поєднує пасивний збір інформації (виявлення WAF, картографування, фазинг директорій) та активний динамічний аналіз (пошук SQLi та XSS).

У роботі обґрунтовано вибір мови програмування Python та розроблено програмний комплекс, який інтегрує великі мовні моделі (LLM) для генерації інтелектуальних рекомендацій щодо усунення виявлених загроз. Розроблено графічний інтерфейс та підсистему автоматичної генерації інтерактивних звітів. Результатом роботи є програмний прототип системи аудиту безпеки «Web Vulnerability Scanner Project», що автоматизує процес виявлення вразливостей та підвищує загальний рівень захищеності серверної інфраструктури.

25.05.2026 _____

# ABSTRACT

Author: Borusevych Pavlo Olehovych

Supervisor: PhD, Assoc. Prof. Dzhulii Volodymyr Mykolaiovych

Explanatory note: 56 pages, 22 figures, 40 sources

Graphic part: 3 poster

Keywords: cybersecurity, web vulnerability, vulnerability scanner, DAST, OSINT, artificial intelligence, SQL injection, XSS, Python, Web Vulnerability Scanner Project.

This thesis analyzes modern cyber threats and methods for auditing the security of web applications. The classification of vulnerabilities according to the international OWASP Top 10 and CWE standards is examined, and existing approaches to penetration testing (DAST, SAST) and open-source intelligence (OSINT) methodologies are reviewed. The architecture of a comprehensive vulnerability scanner was designed, combining passive information gathering (WAF detection, mapping, directory phasing) and active dynamic analysis (SQLi and XSS detection).

The work justifies the choice of the Python programming language and develops a software suite that integrates large language models (LLMs) to generate intelligent recommendations for mitigating detected threats. A graphical user interface and a subsystem for the automatic generation of interactive reports have been developed. The result of the work is a software prototype of the "Web Vulnerability Scanner Project" security audit system, which automates the vulnerability detection process and increases the overall security level of the server infrastructure.

25.05.2026 _____

ЗМІСТ



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | КРБКБ.220104.22.01.03 ПЗ | | |
| Зм. | Аркуш | № докум. | Підпис | Дата | | | |
| Розробив | | Борусевич П.О. | | | Система захисту вебсерверів на основі аналізу атак і вразливостей *Пояснювальна записка* | Літ | Аркуш 6 / Аркушів 56 |
| Перевірив | | Джулій В.М. | | | | Н | |
| Н.контр. | | Петляк Н.С. | | | | ХНУ КБ-22-1 | |
| Затвер. | | Кльоц Ю.П. | | 4.06.25 | | | |

# ВСТУП

У сучасному цифровому світі вебдодатки та серверну інфраструктуру використовують для забезпечення безперебійної роботи критично важливих сервісів у всіх без винятку сферах суспільного життя: від систем електронної комерції та корпоративних порталів до державних реєстрів і систем управління критичною інфраструктурою. Проте експоненційне зростання кількості та складності вебзастосунків супроводжується пропорційним збільшенням кількості цілеспрямованих кібератак [1]. За даними міжнародних аналітичних звітів, такі класичні вразливості, як ін'єкції (SQLi), міжсайтовий скриптинг (XSS) та неправильне налаштування параметрів безпеки (Security Misconfiguration), продовжують залишатися головними векторами компрометації конфіденційних даних.

Незважаючи на стрімкий розвиток індустрії кібербезпеки, існуючі інструменти автоматизованого аудиту (сканери вразливостей) часто не здатні повною мірою задовольнити потреби сучасних розробників та системних адміністраторів. Комерційні рішення є занадто дороговартісними та складними у розгортанні, тоді як відкриті (Open Source) альтернативи генерують велику кількість хибнопозитивних спрацьовувань (False Positives) і, що найважливіше, видають виключно шаблонні, статичні рекомендації щодо усунення загроз. Такі звіти вимагають від фахівців значних витрат часу на додатковий аналіз контексту помилки та пошук релевантних методів її виправлення. У цьому контексті надзвичайно актуальним є розроблення сучасного автоматизованого сканера вразливостей, який би синергійно поєднував методи динамічного тестування безпеки (DAST) із розвідкою з відкритих джерел (OSINT) та можливостями великих мовних моделей (LLM) для надання інтелектуальних, контекстно-залежних рекомендацій щодо усунення виявлених загроз.

Мета і завдання дослідження. Мета дипломної роботи полягає у підвищенні ефективності та якості процесу аудиту безпеки вебсерверів шляхом проєктування та розробки комплексного автоматизованого сканера вразливостей з інтегрованим модулем штучного інтелекту для генерації адаптивної технічної звітності.

Для досягнення поставленої мети було сформульовано та послідовно вирішено такі основні завдання:

Проаналізувати сучасні вектори кібератак на вебдодатки та серверну інфраструктуру за міжнародними методологіями OWASP Top 10 та класифікатором CWE.

Дослідити існуючі методологічні підходи до тестування безпеки програмного забезпечення (SAST, DAST, IAST) та технічно обґрунтувати вибір динамічного аналізу (DAST) як основи для створюваного інструментарію.

Спроєктувати гнучку модульну архітектуру програмного комплексу, що включатиме підсистеми розвідки (OSINT), фазингу директорій, активного сканування та інтелектуальної генерації звітів.

Розробити функціональні програмні модулі мовою Python, реалізувавши надійні алгоритми автоматизованого виявлення критичних вразливостей (SQLi, XSS, Directory Traversal) із підтримкою оброблення сесійних маркерів для проведення автентифікованого сканування.

Інтегрувати модуль штучного інтелекту (LLM) для обробки зібраного контексту загроз та автоматичного формування точних технічних рекомендацій щодо усунення виявлених недоліків на рівні коду та конфігурації.

Провести практичну апробацію та тестування розробленої системи на спеціалізованих навчальних вразливих середовищах із подальшим аналізом результатів її роботи та згенерованої звітності.

Об’єкт дослідження – процес аудиту та забезпечення інформаційної безпеки вебсерверів і вебдодатків у сучасних корпоративних мережах.

Предмет дослідження - методи, алгоритми та програмні засоби автоматизованого виявлення вебвразливостей та формування адаптивних рекомендацій щодо їх усунення з використанням технологій штучного інтелекту.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ СИСТЕМ ЗАХИЩЕНОГО УПРАВЛІННЯ ФАЙЛАМИ

## 1.1 Класифікація сучасних атак на вебдодатки та серверну інфраструктуру за методологією OWASP Top 10 та CWE

У сучасній практиці забезпечення інформаційної безпеки стандартизація та класифікація загроз є фундаментальною умовою для ефективної комунікації між розробниками програмного забезпечення, фахівцями з кібербезпеки (DevSecOps) та розробниками автоматизованих засобів аналізу [4]. Зі стрімким розвитком вебтехнологій та масовим переходом до мікросервісних архітектур площина потенційних атак (Attack Surface) значно розширилася, що вимагає використання загальноприйнятих таксономій для ідентифікації та оцінки ризиків. Двома найавторитетнішими міжнародними стандартами класифікації вразливостей є методологія OWASP (Open Worldwide Application Security Project) та словник CWE (Common Weakness Enumeration) [5, 20, 21].

Методологія OWASP Top 10 являє собою базовий документ, який визначає десять найкритичніших ризиків безпеки вебдодатків на основі глобального консенсусу експертів, статистичних даних телеметрії та глибокого аналізу реальних інцидентів безпеки. Регулярне оновлення цього переліку відображає еволюцію векторів кібератак та зміну парадигм у проєктуванні архітектури додатків. З іншого боку, стандарт CWE є формальним ієрархічним переліком типів слабкостей програмного та апаратного забезпечення (Software Weaknesses). Використання ідентифікаторів CWE дає змогу максимально деталізувати конкретні технічні недоліки архітектури або вихідного коду, що призводять до виникнення високорівневих ризиків, описаних у методології OWASP.

Однією з найпоширеніших та найнебезпечніших категорій загроз є вразливості ін'єкційного типу (OWASP A03:2021-Injection). Ця категорія об'єднує широкий спектр вразливостей, що виникають унаслідок відсутності належної фільтрації, валідації та санітизації вхідних даних, отриманих від користувача через HTTP-параметри, заголовки, файли Cookie або безпосередньо від сторонніх

інтегрованих систем [6]. Класичним і найбільш руйнівним прикладом цієї категорії є впровадження SQL-коду (SQL Injection, CWE-89). Ця вразливість виникає, коли невалідовані дані безпосередньо використовуються для динамічного формування запитів до реляційних баз даних, таких як MySQL, PostgreSQL чи MSSQL. Шляхом цілеспрямованої модифікації параметрів HTTP-запиту зловмисник має змогу додати спеціальні синтаксичні конструкції (наприклад, одинарні лапки, логічні оператори або команди об'єднання), що змушує рушій бази даних виконати несанкціоновані команди. Наслідки такої атаки є критичними: від витоку конфіденційної інформації та несанкціонованої модифікації записів до обходу механізмів автентифікації або повного компрометування сервера баз даних [9, 33].

Механіку реалізації класичної атаки типу SQL-ін'єкція наочно продемонстровано на рисунку 1.1.

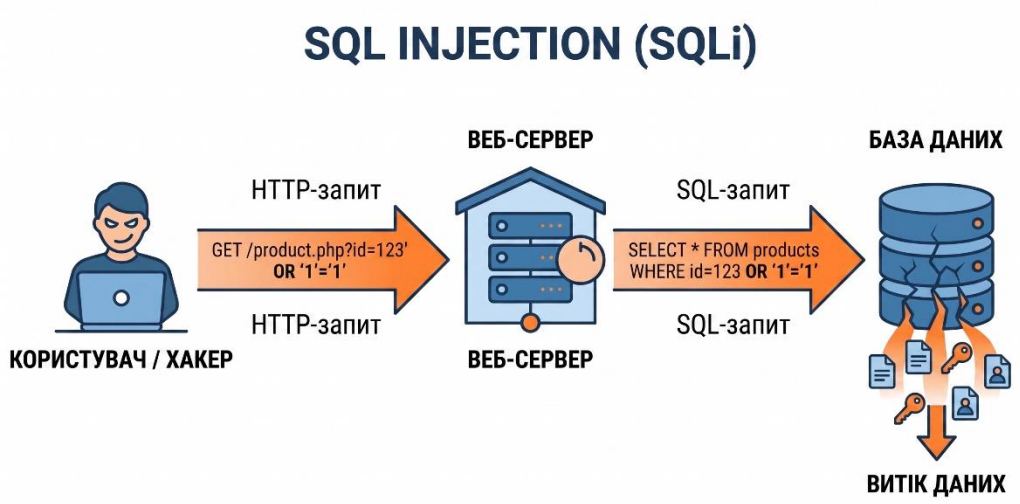



Рисунок 1.1 - Інфографіка принципу дії атаки SQL Injection (SQLi)

Як видно з наведеної схеми, процес атаки складається з кількох послідовних етапів. На першому етапі користувач (зловмисник) формує та надсилає до сервера спеціально модифікований HTTP-запит. У наведеному прикладі замість очікуваного цілочисельного ідентифікатора передається рядок GET /product.php?id=123' OR '1'='1', що містить класичну логічну тавтологію. На етапі обробки вебсервер, який не має надійних механізмів перевірки типів даних або не

використовує підготовлені вирази (Prepared Statements), приймає цей параметр і безпосередньо конкатенує його з шаблоном SQL-запиту у вихідному коді. У результаті до бази даних надходить логічно змінений рядок: SELECT * FROM products WHERE id=123 OR '1'='1'. Оскільки умова '1'='1' є завжди істинною (True), СУБД ігнорує початкове обмеження за ідентифікатором і виконує команду для всієї таблиці, що призводить до масового витоку даних (Data Exfiltration), які передаються назад хакеру через вебсервер.

До категорії ін'єкцій, згідно з сучасною редакцією OWASP, також віднесено міжсайтовий скриптинг (Cross-Site Scripting, CWE-79). Ця загроза полягає у можливості впровадження шкідливого клієнтського коду, найчастіше написаного мовою JavaScript, у вебсторінку, яку переглядають інші легітимні користувачі. Особливу небезпеку становить відбитий міжсайтовий скриптинг (Reflected XSS), за якого шкідливий скрипт передається через параметри URL і негайно повертається сервером клієнту без належного екранування спеціальних символів (Output Encoding). Експлуатація цієї вразливості дозволяє зловмисникам викрадати сесійні файли Cookie, здійснювати фішингові атаки або виконувати несанкціоновані дії від імені жертви [9, 34].

Окрім ін'єкцій, фундаментальною проблемою сучасної вебархітектури є порушення контролю доступу (OWASP A01:2021-Broken Access Control). Відповідно до останніх аналітичних звітів, саме порушення механізмів авторизації вийшло на перше місце серед причин компрометації вебдодатків [9].

Це стається у випадках, коли застосунок не перевіряє належним чином дозволи та привілеї користувача перед наданням доступу до критичних ресурсів або адміністративних функцій. Одним із найнебезпечніших проявів цієї категорії є обхід каталогу та локальне включення файлів (Path Traversal / LFI, CWE-22). Ця вразливість дає змогу зловмиснику вийти за межі визначеного кореневого каталогу вебсервера (Web Root) та зчитати локальні конфігураційні файли операційної системи (наприклад, файл /etc/passwd у середовищі Linux або C:\Windows\win.ini у Windows). Атака реалізується шляхом маніпуляції вхідними параметрами, що відповідають за завантаження або відображення файлів, з

використанням спеціальних послідовностей символів переходу (наприклад, ../ або %2e%2e%2f) [12].

Проте безпека вебдодатка залежить не лише від якості та надійності вихідного коду, але й від правильного налаштування серверного оточення, баз даних, фреймворків та мережевої інфраструктури в цілому. Саме ці аспекти охоплює категорія небезпечної конфігурації (OWASP A05:2021-Security Misconfiguration). Найпоширенішим прикладом є відсутність механізмів захисту на рівні протоколу (CWE-693). Вебсервери дуже часто розгортаються з базовими (default) конфігураціями, які не включають сучасних HTTP-заголовків безпеки (Security Headers). Відсутність таких заголовків, як Content-Security-Policy (CSP) для захисту від XSS, X-Frame-Options для запобігання Clickjacking, або Strict-Transport-Security (HSTS) для примусового шифрування трафіку, критично знижує стійкість застосунку до атак на стороні клієнта. Додатковими факторами ризику в межах цієї категорії є наявність доступних ззовні адміністративних інтерфейсів (наприклад, відкритих панелей phpMyAdmin), залишених тестових скриптів або загальнодоступних системних файлів резервних копій, що значно розширює загальну поверхню атаки цільового ресурсу.

Підсумовуючи вищевикладене, слід зазначити, що застосування стандартизованих ідентифікаторів CWE під час автоматизованого аудиту є необхідною вимогою для проєктування сучасних засобів сканування. Використання чіткої класифікації дає змогу не лише констатувати факт наявності тієї чи іншої вразливості, але й математично формалізувати результати сканування для їх подальшої безшовної передачі системам штучного інтелекту.

Саме формалізований та технічно точний контекст загрози виступає надійною основою для генерування релевантних рекомендацій щодо усунення виявлених недоліків архітектури програмного забезпечення, що становить головну інноваційну складову розроблюваного сканера безпеки.

На відміну від традиційних інструментів аудиту, які здебільшого обмежуються видачею статичних шаблонних попереджень, реалізований підхід дозволяє формувати динамічні та вузькоспрямовані інструкції з безпеки.

1.1 Порівняльний аналіз підходів до тестування безпеки:SAST,DAST та IAST.Обґрунтування вибору динамічного аналізу(DAST)

Забезпечення належного рівня захищеності сучасних інформаційних систем та вебдодатків вимагає комплексного, системного підходу до виявлення вразливостей на різних етапах життєвого циклу розроблення програмного забезпечення (Software Development Life Cycle - SDLC). Інтеграція процесів безпеки безпосередньо у цикл розробки (DevSecOps) зумовила появу та еволюцію різноманітних класів аналізаторів. У сучасній індустрії кібербезпеки домінують три основні методології автоматизованого тестування: статичний (SAST), динамічний (DAST) та інтерактивний (IAST) аналізи. Кожен із цих підходів має власну архітектурну філософію, унікальні переваги та специфічні обмеження, які визначають доцільність їх використання у тих чи інших сценаріях аудиту. [1,3]

Статичне тестування безпеки застосунків (Static Application Security Testing - SAST) є фундаментальною методологією, що базується на принципі «білого ящика» (White-Box). Її сутність полягає у глибокому синтаксичному та семантичному аналізі вихідного коду, байт-коду або бінарних файлів програми без їх фактичного виконання (компіляції чи запуску у середовищі). Аналізатори класу SAST досліджують код зсередини, перевіряючи його на відповідність правилам безпечного програмування, виявляючи використання небезпечних функцій або застарілих бібліотек. Беззаперечною перевагою цього підходу є можливість виявлення вразливостей на найбільш ранніх етапах розроблення, а також надзвичайно високе покриття коду з точною вказівкою на вразливий рядок або метод, що містить першопричину проблеми. Проте методологія SAST має і низку суттєвих концептуальних недоліків. Найбільшою проблемою є генерування великої кількості хибнопозитивних спрацьовувань (False Positives), оскільки статичний аналізатор не здатен врахувати реальний контекст виконання програми, структуру бази даних або наявність зовнішніх засобів захисту (наприклад, WAF). Крім того, SAST-інструменти мають жорстку прив'язку до конкретних мов програмування та фреймворків, а також абсолютно не здатні виявляти вразливості, пов’язані із помилками конфігурації серверного

середовища (відсутність HTTP-заголовків безпеки, відкриті порти чи некоректні налаштування вебсервера). [30]

Альтернативним та історично одним із перших методів автоматизованого аудиту є динамічне тестування безпеки застосунків (Dynamic Application Security Testing - DAST). Цей метод реалізує парадигму «чорного ящика» (Black-Box), за якої тестування проводиться шляхом безпосередньої взаємодії з повністю розгорнутим, скомпільованим і функціонуючим вебдодатком ззовні. DAST-сканер імітує дії реального потенційного зловмисника: він сканує інтерфейси, аналізує структуру сайту та надсилає спеціально сформовані HTTP-запити з тестовими шкідливими навантаженнями (Payloads, фазинг), після чого детально аналізує відповіді сервера на предмет аномалій або розкриття конфіденційних даних [35].

Переваги методології DAST роблять її універсальним інструментом для зовнішнього аудиту: вона є абсолютно незалежною від технологічного стека, мови програмування чи внутрішньої архітектури цільового сервера [30]. Більше того, саме DAST здатний виявляти проблеми налаштування інфраструктури (некоректна робота мережевих екранів, відкриті адміністративні директорії, застарілі або невалідні SSL-сертифікати), що лежить поза межами можливостей статичного аналізу. Ключовою цінністю DAST є низький відсоток хибнопозитивних результатів, оскільки будь-яка виявлена вразливість підтверджується фактичною та експлуатованою реакцією сервера (наприклад, розкриттям синтаксичної помилки бази даних під час атаки типу SQL-ін'єкція).

Еволюційним кроком у розвитку засобів тестування безпеки стало інтерактивне тестування безпеки застосунків (Interactive Application Security Testing - IAST). Це гібридний підхід, який поєднує переваги статичного (SAST) та динамічного (DAST) аналізу безпеки. На відміну від інших методів тестування, IAST працює безпосередньо всередині застосунку за допомогою спеціальних агентів інструментації, що інтегруються у середовище виконання програми, наприклад JVM або CLR.

Під час роботи такі агенти аналізують потоки даних, виконання функцій,

використання пам’яті та взаємодію між компонентами системи у режимі реального часу. Завдяки цьому технологія дозволяє не лише виявляти потенційні вразливості, але й точно визначати місце їх виникнення у програмному коді. Однією з основних переваг IAST є висока точність результатів та мінімальна кількість хибнопозитивних спрацювань. Метод особливо ефективний для виявлення SQL-ін’єкцій, XSS-атак, помилок автентифікації та інших критичних проблем безпеки.

Водночас технологія IAST має і певні недоліки. Її впровадження є технічно складним, оскільки потребує інтеграції агентів у середовище виконання застосунку та налаштування сумісності з конкретними платформами й фреймворками. Крім цього, робота агентів створює додаткове навантаження на серверні ресурси, що може впливати на продуктивність системи. Також підтримка технологій у багатьох IAST-рішеннях є обмеженою, що ускладнює використання методу в різних програмних середовищах.

Комплексне порівняння ключових характеристик, фаз застосування та принципів роботи описаних методологій наочно представлено на рисунку 1.2.


Рисунок 1.2 - Порівняльна характеристика методологій тестування безпеки SAST, DAST та IAST

У межах виконання даної кваліфікаційної роботи перед етапом проєктування архітектури автоматизованого сканера вразливостей постало завдання вибору базової методології тестування. За результатами порівняльного аналізу було ухвалено рішення побудувати ядро програмного комплексу виключно на базі методології динамічного аналізу (DAST). Цей архітектурний вибір є науково та практично обґрунтованим з огляду на низку вагомих чинників.

По-перше, вибір зумовлений специфікою цільового використання розроблюваного інструменту. DAST-сканер призначений для проведення зовнішнього незалежного аудиту безпеки вебсерверів. У реальних умовах аудитор (або пентестер) зазвичай не має доступу до вихідного коду програми, діючи в межах моделі зовнішнього порушника без попередніх привілеїв. Динамічний аналіз дозволяє ефективно оцінити захищеність системи саме з тієї позиції, з якої її бачить потенційний зловмисник, що робить результати аудиту максимально наближеними до реального рівня ризику. [16]

По-друге, DAST забезпечує комплексність перевірки, яка виходить за межі простого пошуку вразливостей у коді. Архітектура DAST дає змогу реалізувати та інтегрувати спеціалізовані модулі для проведення розвідки з відкритих джерел (OSINT), пошуку прихованих серверних директорій методом перебору (Brute-force) та глибокого аналізу конфігурації HTTP-заголовків. Проведення таких інфраструктурних перевірок є концептуально неможливим за умови використання виключно статичного аналізу вихідного коду (SAST).

По-третє, і це є найголовнішим аргументом у контексті теми дипломної роботи, методологія DAST ідеально підходить для інтеграції з алгоритмами штучного інтелекту. На відміну від SAST, який оперує абстрактними номерами рядків коду, DAST-сканер отримує реальний, експлуатований контекст вразливості. Це включає точний уніфікований локатор ресурсу (URL), вразливий вхідний параметр, конкретний сформований HTTP-запит та, що найважливіше, фактичний фрагмент HTTP-відповіді сервера із зафіксованою помилкою або витоком даних. Такий багатий емпіричний набір є оптимальними вхідними даними для модуля великої мовної моделі (LLM). Отримуючи експлуатований

контекст від DAST-ядра, ШІ-асистент (AI Advisor) здатен згенерувати високоточні, практичні та контекстно-залежні рекомендації щодо усунення загрози, трансформуючи розроблений інструмент зі звичайного сканера на інтелектуальну систему аудиту та консалтингу. Таким чином, застосування підходу DAST забезпечує створення універсального інструменту, здатного проводити результативний аудит сучасних вебдодатків незалежно від їхньої внутрішньої архітектури, імітуючи реальні вектори кібератак [31].

## 1.2 Роль розвідки з відкритих джерел (OSINT) у формуванні вектора атаки на вебсервери(субдомени,витоки даних).

У сучасних умовах забезпечення кібербезпеки розвідка з відкритих джерел (OSINT) є одним із перших та найбільш критичних етапів аналізу захищеності інформаційних систем. Процес збору інформації з метою визначення потенційних векторів атак на вебсервери доцільно розглядати у вигляді трирівневої воронки фільтрації та аналізу даних. Ця модель демонструє послідовне звуження масиву неструктурованих публічних даних до конкретних технічних вразливостей, які можуть бути використані зловмисником або фахівцем з тестування на проникнення [10]. Процес формування вектора атаки на основі OSINT-даних складається з трьох послідовних етапів.

Першим етапом є збір первинних даних, що передбачає пасивний та активний пошук інформації з глобальних мережевих джерел без безпосереднього деструктивного впливу на цільову систему. Основними об'єктами дослідження на цьому кроці є служби доменних імен (DNS), публічні репозиторії коду, такі як GitHub чи GitLab, та загальні пошукові системи. Аналіз ресурсних записів DNS дозволяє визначити діапазони IP-адрес організації, поштові сервери та сторонні сервіси, інтегровані в інфраструктуру. Дослідження відкритих репозиторіїв розробників організації часто дозволяє виявити випадково залишені чутливі дані, наприклад конфігураційні файли, токени доступу, API-ключі або паролі від баз

даних. Використання специфічних операторів пошуку в пошукових системах дає змогу виявити проіндексовані адміністративні панелі чи відкриті порти. Отримані на цьому етапі дані піддаються первинному аналізу та фільтрації з метою відсіювання нерелевантної інформації та структурування знайдених відомостей.

Другим етапом є ідентифікація мережевого периметра та виявлених активів. Результатом фільтрації первинних даних є формування точної карти зовнішніх цифрових активів організації, де ключовим завданням виступає ідентифікація субдоменів (Subdomain Enumeration). Організації часто приділяють підвищену увагу захисту лише основного доменного імені, тоді як субдомени, наприклад тестові середовища, застарілі сервіси або внутрішні API, залишаються менш захищеними. У процесі аналізу виявлених активів виконується співставлення субдоменів із реальними IP-адресами та хостинг-провайдерами, визначення топології мережі та логічного взаємозв'язку між різними серверами та базами даних організації, а також виявлення прихованих директорій та невикористовуваних сервісів, які продовжують функціонувати на вебсерверах.

Третім етапом є безпосереднє формування вектора атаки через визначення вразливостей. На цьому кроці структурована інформація про активи аналізується з метою пошуку конкретних технічних недоліків. Процес переходу від виявленого активу до вектора атаки базується на визначенні вразливостей програмного забезпечення, виявленні помилок конфігурації вебдодатків та пошуку витоків автентифікаційних даних. Ідентифікація версій вебсерверів та систем керування вмістом дозволяє знайти відомі невиправлені вразливості [40]. Помилки конфігурації створюють можливість реалізації атак класу SQL-ін'єкцій через небезпечну фільтрацію вхідних даних або міжсайтового скриптингу на застарілих субдоменах. Використання знайдених у відкритих джерелах паролів та ключів дозволяє скомпрометувати облікові записи адміністраторів або отримати доступ до баз даних вебсервера.

Таким чином, розвідка з відкритих джерел є структурованим процесом аналітичного звуження інформаційного поля. Модель доводить, що систематичний збір відкритих даних та аналіз субдоменів дає змогу з високою

точністю локалізувати слабкі місця в інфраструктурі вебсервера та сформувати дієвий вектор атаки.

## 1.3 Огляд існуючих автоматизованих сканерів вразливостей та обґрунтування розробки власного рішення

На сучасному ринку програмного забезпечення для забезпечення інформаційної безпеки представлено надзвичайно широкий спектр автоматизованих інструментів для проведення аудиту вебдодатків та інфраструктури (сканерів вразливостей). Вони є невід'ємною частиною конвеєра безперервної інтеграції та розгортання (CI/CD) і стандартом де-факто для проведення тестування на проникнення (Penetration Testing) [8, 36]. Усю сукупність існуючих рішень можна умовно поділити на дві великі категорії: комерційні продукти корпоративного рівня та інструменти з відкритим вихідним кодом (Open Source).

Комерційні рішення (наприклад, Burp Suite Professional, Nessus, Acunetix, IBM Security AppScan) є індустріальним стандартом у сфері кібербезпеки. Вони вирізняються високою точністю сканування, наявністю складних алгоритмів обходу брандмауерів вебдодатків (WAF), можливістю тестування розгалужених API та глибоким аналізом бізнес-логіки застосунків [9]. Проте впровадження та масштабування таких інструментів у повсякденну практику має низку суттєвих обмежень. По-перше, це висока вартість ліцензування: цінова політика корпоративних підписок робить ці інструменти практично недоступними для малого та середнього бізнесу, незалежних розробників та студентських наукових проєктів. По-друге, вони мають високий поріг входження — ефективне керування комерційними комплексами вимагає вузькоспеціалізованих знань і тривалого навчання. По-третє, процес сканування такими інструментами є вкрай ресурсоємним, створює критичне навантаження на мережеву інфраструктуру та вимагає потужних апаратних ресурсів з боку скануючої станції. Альтернативою

виступають інструменти з відкритим вихідним кодом (наприклад, OpenVAS, OWASP ZAP, Nikto, Wapiti). Вони є безкоштовними, легкодоступними та активно підтримуються світовою спільнотою експертів з кібербезпеки.

Незважаючи на доступність, ці сканери мають спільний архітектурний недолік: вони генерують звіти на основі статичних баз даних вразливостей (шаблонів) [16]. У разі виявлення загрози адміністратор отримує уніфікований текст із загальним описом проблеми (наприклад, «використовуйте параметризовані запити для уникнення SQL-ін'єкцій»), який абсолютно не враховує специфіки конкретного знайденого вектора атаки, мови програмування чи конфігурації цільового сервера.

Для обґрунтування необхідності створення нового програмного продукту було проведено комплексне порівняння лідерів ринку з концептом розроблюваної системи. Результати цього зіставлення за ключовими критеріями наведено на рисунку 1.4.

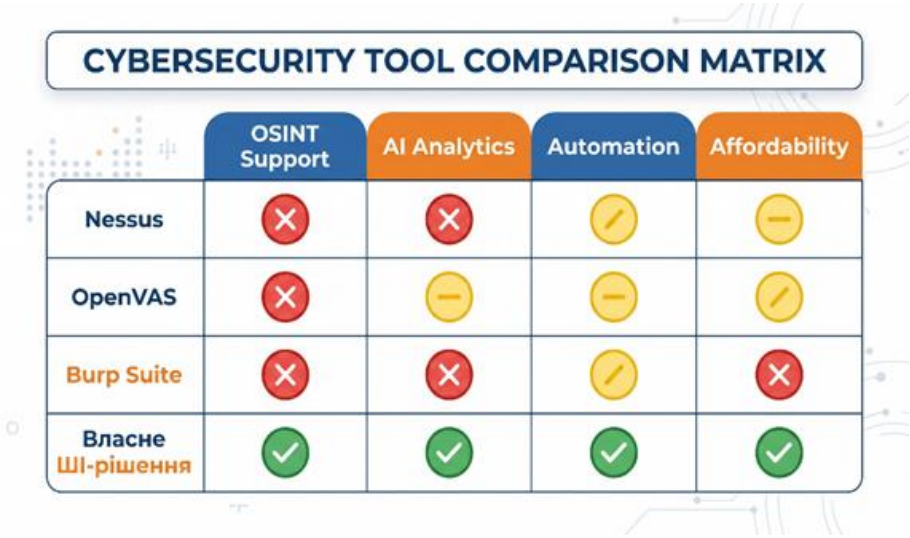



Рисунок 1.3 - Порівняльний аналіз функціональних можливостей засобів аудиту безпеки

## 1.4 Постановка задачі

На основі проведеного аналізу предметної області, дослідження векторів

кібератак та існуючих методологій тестування безпеки можна зробити висновок, що сучасний аудит вебдодатків потребує комплексного підходу. Класичні інструменти статичного (SAST) та динамічного (DAST) аналізу, хоча й є ефективними для пошуку типових технічних дефектів, часто генерують складні для інтерпретації звіти без урахування контексту конкретної інфраструктури. Крім того, традиційні сканери не завжди враховують інформацію, яку можна отримати методами пасивної розвідки (OSINT), що знижує загальну ефективність виявлення загроз на ранніх етапах.

Отже, виникає об’єктивна необхідність у створенні сучасного автоматизованого інструменту, який об'єднував би фази пасивного збору інформації, активного динамічного сканування та інтелектуального аналізу результатів.

Головною задачею даної кваліфікаційної роботи є проєктування та програмна реалізація системи аудиту безпеки вебресурсів - «Web Vulnerability Scanner Project». Розроблюваний програмний комплекс повинен вирішувати наступні конкретні завдання:

– проведення пасивної мережевої розвідки: сканування відкритих портів, пошук пов'язаних піддоменів (через відкриті джерела) та ідентифікація технологічного стеку цільового сервера;

– виявлення наявності та типу міжмережевих екранів вебдодатків (WAF) для уникнення блокування під час активної фази аудиту;

– автоматизоване картографування структури цільового сайту (Crawling) та перебір прихованих директорій (Directory Fuzzing);

– виконання активного динамічного аналізу (DAST) для виявлення критичних вразливостей, зокрема SQL-ін'єкцій (SQLi) та міжсайтового скриптингу (XSS);

– забезпечення програмної взаємодії (через API) з великими мовними моделями (LLM) для аналізу виявлених дефектів та генерації контекстно-залежних рекомендацій щодо їх усунення;

– формування зручних, структурованих та інтерактивних звітів (у форматах

TXT та HTML) за результатами сканування.

Для забезпечення кросплатформності програмного рішення, високої продуктивності під час обробки мережевих запитів, а також спрощення інтеграції сучасних бібліотек для машинної обробки даних і роботи з моделями штучного інтелекту, розробку системи доцільно виконувати з використанням мови програмування Python [7]. Даний вибір обумовлений її широкими можливостями, розвиненою екосистемою бібліотек, а також високим рівнем підтримки мережевих, аналітичних та AI-технологій.

Окрему перевагу Python становить його гнучкість у реалізації модульної архітектури програмних систем, що дозволяє розділяти функціональні компоненти на незалежні логічні модулі. Такий підхід значно підвищує масштабованість системи, полегшує її подальший супровід, тестування та розширення функціоналу без необхідності суттєвих змін у вже реалізованих частинах коду.

Крім того, для підвищення ефективності обробки великої кількості мережевих запитів та паралельного виконання операцій у системі доцільно застосовувати принципи багатопотокової та асинхронної обробки даних. Це дозволяє оптимізувати використання системних ресурсів, зменшити затримки при роботі з мережею та забезпечити стабільну роботу системи навіть при високому навантаженні.

Використання Python у поєднанні з модульною та багатопотоковою архітектурою створює оптимальні умови для реалізації сучасних інтелектуальних систем, зокрема тих, що інтегрують механізми штучного інтелекту, аналізу вразливостей та автоматизованого формування рекомендацій з кібербезпеки.

Завдяки модульності забезпечується гнучкість програмного комплексу, що дозволяє легко додавати нові вектори перевірок (наприклад, модулі для виявлення SQL Injection чи XSS) без необхідності втручання в основне ядро. Багатопотоковість, у свою чергу, є критично важливою для методів динамічного аналізу (DAST), оскільки вона дає змогу паралельно надсилати payload-запити та обробляти відповіді сервера.

# 2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА ЛОГІКИ РОБОТИ КОМПЛЕКСНОГО СКАНЕРА ВРАЗЛИВОСТЕЙ

## 2.1 Формування вимог до програмного продукту та обґрунтування модульної архітектури

Процес проєктування надійного та ефективного програмного забезпечення розпочинається з чіткого визначення вимог до його функціональності, архітектурної будови та умов експлуатації. З огляду на високу складність та специфіку предметної області (аудит інформаційної безпеки вебдодатків), розроблюваний автоматизований сканер вразливостей повинен відповідати низці суворих критеріїв, які традиційно поділяються на функціональні та нефункціональні вимоги. Комплексний підхід до формування цих вимог гарантує, що кінцевий продукт не лише виконуватиме свої базові завдання з пошуку вразливостей, але й буде зручним в інтеграції, стійким до збоїв та готовим до подальшого масштабування.

Функціональні вимоги до системи До базових функціональних вимог, які визначають безпосередню поведінку програмного продукту та його можливості під час проведення аудиту, належать:

- Підсистема розвідки: можливість автоматизованого пасивного збору інформації (OSINT) про цільовий ресурс, що включає пошук пов'язаних субдоменів, збір публічних електронних адрес та ідентифікацію відкритих мережевих портів для визначення загальної площі атаки.
- Картування поверхні атаки (Attack Surface Mapping): здатність системи здійснювати глибокий аналіз структури вебдодатка шляхом автоматичного перебору прихованих адміністративних директорій та аналізу конфігурації HTTP-заголовків безпеки.
- Динамічне тестування (Active DAST): реалізація алгоритмів активного фазингу для виявлення критичних вразливостей (зокрема, SQL-ін'єкцій, міжсайтового скриптингу та обходу каталогів). Обов'язковою вимогою є підтримка оброблення сесійних маркерів (Cookies) для забезпечення можливості проведення автентифікованого аудиту закритих зон застосунку.

- Інтелектуальна аналітика: наявність інтегрованого підмодуля для взаємодії з прикладним програмним інтерфейсом (API) великої мовної моделі (LLM). Цей компонент повинен забезпечувати інтелектуальний аналіз виявлених загроз та автоматичне генерування контекстно-залежних технічних рекомендацій щодо їх усунення.

- Генерація звітності: автоматичне формування агрегованих підсумкових звітів у форматі HTML із візуальною категоризацією ризиків згідно з міжнародними стандартами OWASP та відображенням порад від штучного інтелекту.

Окрім безпосереднього функціоналу, програмний комплекс повинен відповідати критеріям продуктивності та надійності. Серед ключових нефункціональних вимог визначено кросплатформність - тобто можливість безперебійного функціонування в операційних системах сімейства Windows, macOS та GNU/Linux (зокрема, у спеціалізованих дистрибутивах на кшталт Kali Linux) без необхідності модифікації вихідного коду.

Важливою вимогою є стійкість до мережевих тайм-аутів та аномальних відповідей сервера (наприклад, помилок 429 Too Many Requests або 500 Internal Server Error). Система повинна коректно обробляти обриви з'єднання, не перериваючи загальний цикл сканування. Крім того, архітектура має забезпечувати здатність до легкого масштабування кодової бази для майбутнього розширення функціоналу.

Обґрунтування модульної архітектури Для задоволення зазначеного комплексу вимог застосування традиційної монолітної архітектури програмного забезпечення є вкрай неефективним. Монолітний підхід, за якого вся логіка програми зосереджена в єдиному масиві коду, суттєво ускладнює процес оновлення баз сигнатур, інтеграцію нових векторів атак та загальний супровід продукту [15]. Саме тому для реалізації програмного комплексу було обрано парадигму модульної архітектури.

Сутність спроєктованої архітектури полягає у строгому розподілі загального функціоналу на незалежні компоненти (модулі), які слабко пов'язані

між собою (Loose Coupling), але мають високу внутрішню зв'язність (High Cohesion). Усі ці модулі взаємодіють між собою через центральний вузол — ядро ініціалізації та оркестрації. Основні переваги модульного підходу для розроблюваного сканера полягають у наступному:

- Ізоляція процесів та відмовостійкість: виникнення виняткової ситуації в роботі одного компонента (наприклад, блокування запитів під час пошуку субдоменів з боку інтернет-провайдера) не призводить до критичного збою і зупинки роботи всього сканера або генератора звітів.
- Масштабованість та розширюваність: архітектура дає змогу розробнику або спільноті легко додавати перевірки на нові класи вразливостей (наприклад, Command Injection або XML External Entity) шляхом створення додаткового скрипта у відповідному каталозі, абсолютно не порушуючи цілісності базового ядра системи.
- Спрощення процесу налагодження (Debugging): оскільки кожен логічний етап аудиту (розвідка, фазинг, активне тестування, генерація документації) є відокремленим, це значно пришвидшує виявлення, локалізацію та усунення програмних дефектів на етапах розроблення та тестування. Крім того, така архітектура дає змогу проводити незалежне модульне тестування (Unit Testing) кожного компонента ізольовано, використовуючи тестові заглушки (mock-об'єкти) без необхідності ініціювати повний цикл мережевого сканування. Важливою перевагою є також підвищена відмовостійкість системи: у разі виникнення критичної помилки, мережевого таймауту або раптового блокування запитів з боку WAF під час роботи одного з модулів, ядро програми здатне коректно перехопити виключення (Exception Handling). Це гарантує, що результати попередніх фаз аудиту будуть безпечно збережені в пам'яті, а сканер зможе продовжити виконання інших незалежних завдань, запобігаючи аварійному завершенню (Crash) всього програмного комплексу.

Візуалізацію спроєктованої архітектури та механізмів взаємодії між ключовими компонентами системи наведено на рисунку 2.1.

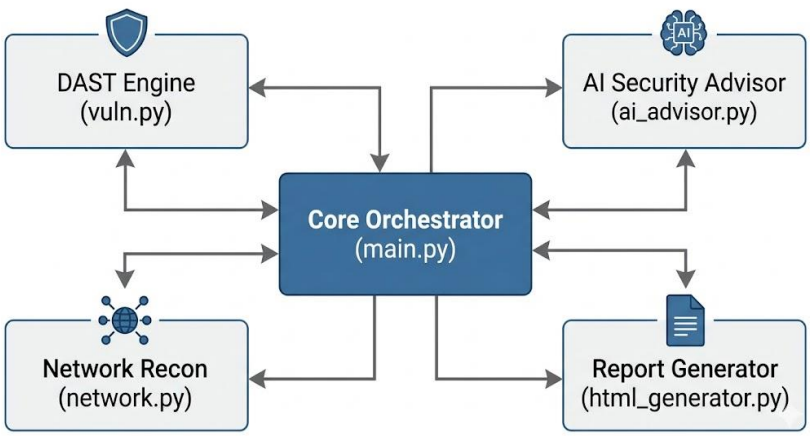



Рисунок 2.1 - Структурна схема взаємодії програмних модулів комплексу

Як видно з наведеної структурної схеми, архітектура програмного комплексу побудована за зіркоподібною топологією, де всі потоки даних та команди управління проходять через єдиний координаційний центр.

– Core Orchestrator (main.py): Це центральне ядро системи, яке виконує функції головного диспетчера. Хоча користувач взаємодіє з програмою через графічний інтерфейс (GUI), саме цей модуль відповідає за ініціалізацію змінних середовища, перевірку доступності цільового ресурсу, управління пулом потоків (Multithreading) та послідовний виклик усіх інших підсистем. Він акумулює результати роботи кожного модуля у єдиний структурований словник даних.

– Network Recon (network.py): Модуль пасивної розвідки, який запускається першим. Він відповідає за збір мережевої топології, сканування портів та ідентифікацію сервісів. Передає ядру початкову інформацію про стан інфраструктури цілі.

– DAST Engine (vuln.py): Критично важливий компонент активної фази. Отримавши від ядра карту доступних посилань (згенеровану краулером), цей рушій починає процес фазингу: ін'єктує шкідливі навантаження у параметри запитів та аналізує відповіді сервера на наявність SQLi, XSS та інших вразливостей.

– AI Security Advisor (ai_advisor.py): Інноваційний модуль, який підключається на фінальних етапах аналізу. Ядро системи передає йому знайдений контекст вразливостей, а модуль, своєю чергою, звертається до API

великої мовної моделі для отримання рекомендацій щодо виправлення коду та налаштувань.

– Report Generator (html_generator.py): Завершальна ланка конвеєра. Після того як ядро збере всі дані від рушія DAST та ШІ-асистента, цей модуль отримує команду на агрегацію даних та їх вбудовування у візуальний HTML-шаблон, формуючи кінцевий артефакт роботи програми - звіт з безпеки.

Таким чином, спроєктована модульна структура створює надійний, гнучкий та стійкий до відмов фундамент для побудови комплексного інструменту аудиту безпеки вебсерверів. Цей архітектурний підхід повністю відповідає сучасним вимогам інженерії програмного забезпечення та гарантує виконання всіх поставлених функціональних завдань.

## 2.2 Проєктування алгоритмів пасивного аналізу : виявлення Web Applicaton Firewall (WAF) та збір базової інформації

Будь-який процес автоматизованого аудиту інформаційної безпеки вебдодатків розпочинається з етапу пасивного аналізу та збору відбитків (Fingerprinting). Метою цього етапу є отримання максимально повної технічної картини про цільовий сервер та його інфраструктурну топологію без виконання деструктивних або надмірно агресивних дій, які можуть призвести до миттєвого блокування IP-адреси сканера системами захисту. У спроєктованій системі цей етап є критичним і структурно поділяється на дві логічно пов'язані складові: детектування мережевих екранів (WAF) та комплексний збір базової мережевої інформації.

Web Application Firewall (WAF) - це спеціалізований захисний комплекс (апаратний або програмний), призначений для глибокої фільтрації, моніторингу та блокування шкідливого HTTP-трафіку на прикладному рівні (наприклад, спроб SQL-ін'єкцій, міжсайтового скриптингу або експлуатації відомих CVE). Наявність WAF на цільовому сервері кардинально змінює підхід до процесу подальшого

динамічного сканування (DAST). Якщо сканер почне надсилати тестові навантаження (payloads) без урахування правил WAF, захисний екран негайно заблокує ці запити, відхилить з'єднання або занесе IP-адресу аудитора до «чорного списку». Це неминуче призведе до хибнонегативних результатів (False Negatives), за яких сканер повідомить про відсутність вразливостей там, де вони фактично існують, але доступ до них перекрито захисним екраном.

Для розв'язання цієї проблеми в межах програмного комплексу (модуль waf.py) було спроєктовано евристичний алгоритм детектування, який складається з таких послідовних кроків:

- Аналіз нормальної поведінки (Baseline Profiling): Система ініціює з'єднання та надсилає стандартний, легітимний HTTP-запит (GET) до головної сторінки цільового ресурсу. Отримана відповідь (HTTP-код статусу, розмір тіла відповіді, стандартні заголовки) зберігається як еталонна модель поведінки сервера.
- Пасивна ідентифікація за заголовками (Passive Fingerprinting): Алгоритм детально аналізує отримані HTTP-заголовки відповіді (Response Headers). Більшість сучасних комерційних WAF залишають характерні цифрові маркери. Наприклад, системи часто модифікують заголовок Server (встановлюючи значення на кшталт cloudflare, AkamaiGHost, f5) або додають специфічні файли Cookie (як-от __cfduid для Cloudflare або incapsula_ для Imperva), що дозволяє однозначно ідентифікувати вендора захисту.
- Активне провокування (Active Fuzzing): Якщо пасивний аналіз не виявив явних маркерів, алгоритм переходить до активної фази. Сканер навмисно надсилає запит із заздалегідь відомим шкідливим сигнатурним рядком (наприклад, корисним навантаженням /?id=1' OR '1'='1 або базовим XSS-вектором).
- Порівняльний аналіз відповіді: Якщо у відповідь на шкідливий запит сервер повертає специфічний код помилки доступу (найчастіше HTTP 403 Forbidden, 406 Not Acceptable або 501 Not Implemented), або ж структура повернутої сторінки кардинально відрізняється від еталонної (відображається

стандартна сторінка блокування WAF), алгоритм достовірно фіксує наявність захисного екрана.

Після підтвердження факту блокування система ініціює етап розпізнавання (fingerprinting) конкретного типу WAF. Для цього додатково аналізуються специфічні HTTP-заголовки (наприклад, Server, X-Powered-By, кукі-файли) та унікальні текстові сигнатури в тілі повернутої сторінки, що дозволяє класифікувати такі популярні рішення, як Cloudflare, ModSecurity чи AWS WAF.

Логіку роботи описаного алгоритму наочно відображено на рисунку 2.2.

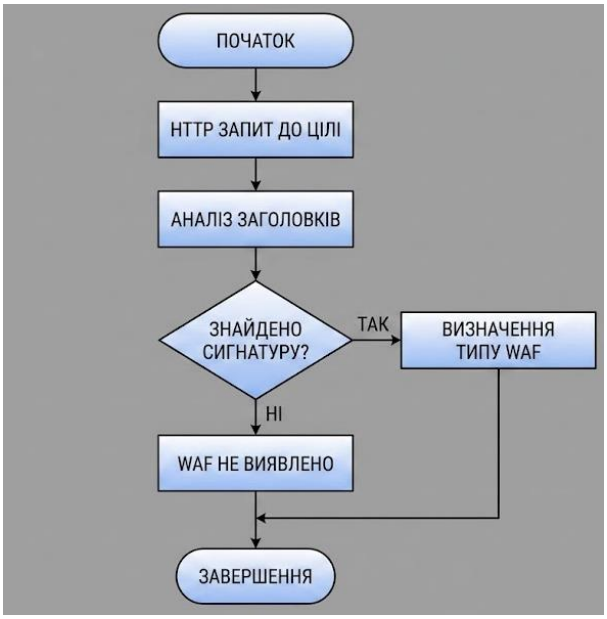



Рисунок 2.2 - Блок-схема алгоритму пасивного виявлення WAF

Як видно з наведеної структурної схеми, логіка детектування є максимально лінійною та оптимізованою для швидкодії. Процес ініціюється відправкою базового HTTP-запиту до цілі. Отримані дані проходять етап лексичного аналізу заголовків. Ключовим вузлом прийняття рішень є логічний оператор «Знайдено сигнатуру?». У разі позитивного результату (ТАК) система переходить до визначення конкретного типу WAF. Це критично важливо, оскільки знання вендора (наприклад, AWS WAF чи Sucuri) дозволяє аудитору в подальшому застосувати специфічні техніки обходу (Bypass techniques) саме для цієї системи. У випадку відсутності збігів із базою сигнатур (НІ) генерується статус «WAF НЕ ВИЯВЛЕНО», що є сигналом для безпечного запуску агресивніших модулів

динамічного сканування.

Паралельно з детектуванням WAF, ядро системи ініціює роботу модулів розвідки інфраструктури (зокрема, скриптів network.py та web.py). Їхнє завдання полягає у зборі технічних метаданих, які функціонують за такими алгоритмами:

- Алгоритм сканування портів (Port Scanning): Реалізовано за допомогою неблокуючих мережевих сокетів. Система швидко ітерує заданий масив критично важливих мережевих портів (наприклад, 80, 443 для вебсервісів, 21 для FTP, 22 для SSH, 3306 для MySQL, 8080 для альтернативних HTTP-серверів) і намагається встановити TCP-з'єднання. Аналіз відкритих портів дає змогу ідентифікувати суміжні сервіси. Наприклад, виявлення відкритого порту 3306 свідчить про те, що база даних доступна ззовні, що є серйозним порушенням архітектури безпеки та розширює поверхню атаки.
- Аналіз SSL/TLS-сертифікатів: У разі детектування безпечного HTTPS-з'єднання, система автоматично вилучає публічний сертифікат сервера. Алгоритм перевіряє термін дії сертифіката, тип застосованого шифрування (наприклад, RSA чи ECC) та належність довіреному центру сертифікації (CA). Використання застарілих протоколів (таких як TLS 1.0 або SSLv3) класифікується системою як самостійна вразливість відповідно до категорії OWASP A02:2021-Cryptographic Failures.
- Визначення технологічного стека (Technology Fingerprinting): Цей процес здійснюється шляхом глибокого синтаксичного парсингу HTTP-заголовка X-Powered-By та аналізу специфічних метатегів HTML-документа. Це дає змогу ідентифікувати мову програмування бекенду (наприклад, PHP, Python, ASP.NET) та тип вебсервера (Nginx, Apache, IIS). Отримана інформація є неоціненною для DAST-рушія: вона допомагає адаптувати вектори атак, відкидаючи нерелевантні тестові навантаження (наприклад, система не буде витрачати час на перевірку вразливостей, специфічних для MSSQL, якщо було виявлено сервер на базі Linux/MySQL).
- Усі дані, отримані на етапі пасивного аналізу, структуруються у форматі комплексного словника даних (Data Dictionary) і передаються до головного

оркестратора системи. Такий підхід до проєктування гарантує оптимізацію наступних, активних фаз аудиту, значно заощаджує мережеві ресурси та підвищує загальну точність виявлення вразливостей розробленим програмним комплексом.

## 2.3 Алгоритмічне забезпечення модуля картування поверхні атаки(Web Crawler) та пошук прихованих директорії

Після завершення етапу пасивного збору технічної інформації архітектура системи передбачає перехід до критично важливого процесу визначення повної площі (поверхні) потенційної атаки (Attack Surface Mapping). Динамічне тестування (DAST) виключно головної сторінки вебдодатка є малоефективним, оскільки переважна більшість критичних вразливостей зазвичай прихована у внутрішніх розділах, панелях авторизації або параметризованих запитах, на які немає прямих посилань. Для автоматизації процесу ідентифікації всіх доступних точок входу (Endpoints) у програмному комплексі спроєктовано два взаємодоповнюючі алгоритми: лексичний аналіз внутрішніх посилань (Web Crawling) та активний фазинг директорій (Directory Enumeration).

Вебкраулер (або «вебпавук») - це програмний модуль, який автоматично здійснює обхід сторінок цільового вебсервера з метою збору всіх доступних гіперпосилань. Алгоритмічно цей процес базується на класичній моделі обходу графа у ширину (Breadth-First Search) [13]. Роботу модуля спроєктовано за наступним ітеративним циклом:

- Ініціалізація та завантаження: алгоритм отримує від ядра базовий URL, надсилає початковий HTTP-запит і завантажує структуру DOM (Document Object Model) стартової сторінки.
- Синтаксичний аналіз (Парсинг): за допомогою регулярних виразів та бібліотек синтаксичного аналізу здійснюється автоматичний пошук усіх тегів, що містять посилання. Зокрема, аналізуються атрибути href у тегах <a> та атрибути action у тегах <form>, які визначають обробники даних на стороні сервера.
- Нормалізація посилань: оскільки вебсторінки часто використовують

відносні шляхи (наприклад, /about або ../config), алгоритм автоматично перетворює їх на абсолютні уніфіковані локатори ресурсів (URL) шляхом конкатенації з базовим доменом цілі.

- Контроль області аудиту (Scope Control): це критично важливий етап фільтрації, на якому алгоритм відкидає всі посилання, що ведуть на зовнішні ресурси (соціальні мережі, сторонні сервіси). Сканування жорстко обмежується виключно цільовим доменом для запобігання несанкціонованому тестуванню чужих систем.
- Ітеративне збереження: виявлені унікальні посилання додаються до черги на подальше сканування, доки не буде досягнуто заданого користувачем ліміту глибини обходу або вичерпано всі доступні внутрішні сторінки.

Результатом роботи модуля є вичерпний масив валідних посилань вебдодатка, які згодом передаються до активного DAST-модуля (vuln.py) для пошуку SQL-ін'єкцій та XSS.

Алгоритмічне забезпечення пошуку прихованих ресурсів (Brute-force)

Вебкраулер здатний ідентифікувати лише ті сторінки, на які існують явні гіперпосилання у коді вебдодатка. Проте системні адміністратори часто залишають на серверах критично важливі ресурси без жодних посилань, помилково вважаючи це достатнім заходом захисту («Security through obscurity»). До таких ресурсів належать адміністративні інтерфейси (наприклад, /admin), конфігураційні файли середовища (.env), резервні копії баз даних (backup.sql) або архіви вихідного коду (.git/) [32].

Для виявлення цих прихованих об'єктів спроєктовано алгоритм перебору за спеціалізованими словниками (Dictionary-based attack), що реалізований у модулі directories.py. Такий підхід дозволяє знаходити критично важливі дані: забуті файли конфігурацій, резервні копії, неавторизовані адміністративні панелі або старі версії API, які не пов’язані прямими посиланнями з головною сторінкою. Принцип ієрархічної побудови карти поверхні атаки, що поєднує публічні та приховані ресурси, наочно продемонстровано на рисунку 2.3.


Рисунок 2.3 - Карта поверхні атаки веб-додатку: ієрархічна структура об'єктів

Технічний опис алгоритму перебору (відповідно до рис. 2.3):

- Генерування запитів: система ітерує завантажений словник (Wordlist), що містить сотні найпоширеніших назв технічних директорій, динамічно формуючи нові адреси (наприклад, [target.com/.env](https://target.com/.env)).

- Аналіз HTTP-відповідей: алгоритм надсилає тисячі запитів за допомогою багатопотокового пулу (ThreadPool) і аналізує коди стану:

- Код 200 OK: фіксується як успішне знаходження прихованого файлу чи каталогу.

- Код 403 Forbidden: також реєструється у звіті, оскільки підтверджує фізичне існування ресурсу на сервері, що є цінною інформацією для подальшого аналізу прав доступу.

- Евристичний захист від «Soft 404»: алгоритм включає перевірку на хибнопозитивні результати. Якщо сервер налаштований повертати 200 OK для будь-якого запиту, система порівнює хеш-суми відповідей із еталонною сторінкою помилки та відфільтровує нерелевантні дані.

Синхронна робота методів лексичного краулінгу та активного перебору за словником забезпечує максимальне покриття площі вебдодатка. Це гарантує, що жоден прихований параметр, конфігураційний файл чи забутий адміністративний

інтерфейс не залишиться поза увагою системи автоматизованого аудиту безпеки.

## 2.4 Логіка пасивного DAST-модуля:методи автоматичного виявлення SQL-ін’єкцій (SQLi) та Cross-Site Scripting(XSS)

Після завершення етапів розвідки та картографування цільового ресурсу програмний комплекс переходить до ключової фази аудиту безпеки - активного динамічного тестування (Active DAST). Завданням спроєктованого модуля vuln.py є автоматизована імітація кібератак на виявлені точки входу (Endpoints) з метою підтвердження наявності критичних вразливостей. Алгоритмічна основа модуля базується на методі фазингу (Fuzzing) - автоматизованій підстановці спеціально сформованих тестових навантажень (Payloads) у вхідні параметри вебдодатка та подальшому аналізі реакції сервера.

Процес пошуку SQL-ін'єкцій реалізовано за методом аналізу помилок (Error-based SQLi), який є найбільш надійним для автоматизованих систем. Алгоритм функціонує за такою схемою:

- Декомпозиція цільового URL: алгоритм розбирає отримане посилання, відокремлюючи базовий шлях від рядка запиту (Query String), та здійснює вилучення всіх GET-параметрів та їхніх початкових значень.
- Генерування векторів атаки: для кожного виявленого параметра система по черзі підставляє набір спеціальних символів та синтаксичних конструкцій, таких як одинарні та подвійні лапки (', "), логічні оператори (OR '1'='1) та команди UNION SELECT.
- Надсилання мутованих запитів: сформовані HTTP-запити надсилаються до цільового сервера за допомогою бібліотеки requests.
- Евристичний аналіз відповідей: система аналізує тіло HTTP-відповіді на наявність характерних повідомлень про синтаксичні помилки СУБД (наприклад, syntax error, mysql_fetch_array(), ORA-01756). Якщо у відповіді знайдено збіг із внутрішнім словником помилок баз даних, вразливість вважається підтвердженою та маркується як високий ризик.

- Алгоритм виявлення міжсайтового скриптингу (CWE-79)

- Логіку виявлення вразливостей типу Reflected XSS спроєктовано за принципом перевірки відображення (Reflection Check):

- Ін'єкція маркерів: замість стандартних значень параметрів алгоритм підставляє тестові вектори, що містять безпечний, але унікальний JavaScript-код, наприклад: <script>alert('XSS_Test')</script>.

- Аналіз цілісності навантаження: після отримання відповіді алгоритм перевіряє вихідний код сторінки. Якщо впроваджений тестовий вектор присутній у документі у первісному вигляді, без застосування механізмів екранування символів < та >, це свідчить про те, що браузер користувача виконає цей шкідливий скрипт. Вразливість фіксується у звіті з відповідною класифікацією OWASP.

Суттєвим архітектурним обмеженням багатьох автоматизованих сканерів є неможливість аналізу закритих частин вебдодатка (адміністративних панелей, профілів), оскільки неавторизовані запити автоматично переадресовуються на сторінку входу. Для розв'язання цієї проблеми у логіку модуля vuln.py закладено підтримку автентифікованого сканування.

Алгоритм передбачає можливість отримання та збереження сесійних маркерів (Cookies), зокрема ідентифікаторів сесій PHP (PHPSESSID) або JWT-токенів. Перед початком активного фазингу модуль інтегрує ці маркери у HTTP-заголовки кожного згенерованого запиту.

Це дає змогу сканеру легітимно обходити механізми контролю доступу та проводити глибокий аудит захищених сторінок від імені авторизованого користувача, що суттєво розширює поверхню тестування та підвищує загальну ефективність аудиту.

У разі успішного підтвердження будь-якої із зазначених вразливостей, модуль динамічного аналізу формує формалізований об'єкт із описом загрози та передає його до ядра системи для подальшого опрацювання модулем штучного інтелекту.

## 2.5 Висновок

У другому розділі було проведено комплексне проєктування архітектури та деталізоване обґрунтування логіки функціонування інтелектуального автоматизованого сканера вразливостей. Результати етапу проєктування дозволили сформувати надійний технологічний фундамент для подальшої програмної реалізації системи.

По-перше, на основі специфіки аудиту вебсерверів було сформувано вичерпний перелік функціональних та нефункціональних вимог до програмного продукту. Визначено, що система повинна не лише забезпечувати виявлення вразливостей за методологією DAST, але й виконувати пасивну OSINT-розвідку, підтримувати автентифіковані сесії та генерувати адаптивну звітність за допомогою технологій штучного інтелекту.

По-друге, науково обґрунтовано вибір модульної архітектури зіркоподібної топології, де центральне ядро оркестрації (main.py) координує роботу незалежних компонентів. Такий підхід забезпечує системі необхідну відмовостійкість, дозволяє паралельно виконувати мережеві завдання та гарантує легкість інтеграції нових модулів перевірки у майбутньому.

По-третє, розроблено алгоритмічне забезпечення етапів розвідки та картографування поверхні атаки. Спроєктовано логіку евристичного виявлення мережевих екранів (WAF) та гібридний метод обходу структури сайту, що поєднує лексичний аналіз посилань вебпавуком (Web Crawler) із багатопотоковим перебором прихованих директорій за словниками.

Це дозволяє мінімізувати кількість «сліпих зон» під час проведення аудиту.

По-четверте, деталізовано внутрішню логіку активного рушія DAST для детектування SQL-ін'єкцій та XSS-атак. Реалізований підхід на основі аналізу помилок СУБД та перевірки відображення маркерів забезпечує високу точність виявлення загроз за стандартами CWE-89 та CWE-79. Окрему увагу приділено проєктуванню модуля AI Advisor, який на основі інженерії підказок (Prompt Engineering) трансформує знайдений технічний контекст у практичні інструкції з виправлення коду.

Спроєктована в цьому розділі архітектурна модель системи, а також детально описані алгоритми взаємодії її функціональних модулів, формують фундаментальну основу для подальшого етапу практичної реалізації програмного забезпечення. Запропонована структура відображає логіку взаємодії компонентів системи, принципи обробки даних та механізми обміну інформацією між окремими модулями, що забезпечує цілісність і узгодженість усієї архітектури.

Зазначена модель фактично виступає формалізованою інструкцією для етапу програмної реалізації, оскільки визначає як загальні підходи до побудови системи, так і конкретні правила реалізації її ключових функціональних елементів. Це дозволяє мінімізувати ризики виникнення архітектурних помилок на етапі кодування та забезпечує відповідність кінцевого програмного продукту первинно спроєктованій концепції.

У наступному розділі буде детально розглянуто процес практичної реалізації запропонованої системи із використанням мови програмування Python, включаючи опис ключових програмних модулів, особливостей їх взаємодії, а також результати тестування системи в умовах моделювання роботи на потенційно вразливих середовищах. Детальний розбір охопить логіку роботи підсистеми збору інформації, модулів динамічного аналізу (DAST) для виявлення критичних вразливостей (зокрема SQL Injection та XSS), а також алгоритми взаємодії з цільовим сервером. Значну увагу буде приділено інноваційному аспекту розробки - інтеграції інтелектуальних компонентів для автоматичного аналізу знайдених загроз та формування експертних рекомендацій щодо їх усунення.

Окрім цього, буде продемонстровано роботу підсистеми генерації комплексної звітності у зручних для користувача форматах. Практична апробація розробленого програмного комплексу на спеціалізованих тестових стендах дозволить об’єктивно оцінити його продуктивність, точність виявлення аномалій та загальну відмовостійкість, що у підсумку підтвердить доцільність використання створеного рішення у реальних завданнях з аудиту інформаційної безпеки.

# 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ АУДИТУ БЕЗПЕКИ ВЕБРЕСУРІВ

## 3.1 Вибір технологічного стеку та реалізація програмних модулів на мові Python

Вибір інструментальних засобів розроблення є одним із найважливіших етапів проєктування програмного забезпечення, оскільки від нього залежить не лише швидкість реалізації, а й подальша стабільність, продуктивність та гнучкість системи. Для реалізації комплексного автоматизованого сканера вразливостей було обрано високорівневу об'єктно-орієнтовану мову програмування Python. Такий вибір є науково обґрунтованим і базується на низці вагомих чинників:

- Індустріальний стандарт у галузі кібербезпеки: Python має найбагатшу у світі екосистему спеціалізованих бібліотек для роботи з мережевими протоколами, криптографією та синтаксичним аналізом неструктурованих даних [7, 22]. Переважна більшість сучасних Open-Source інструментів для пентестингу (наприклад, sqlmap, Nmap scripts) розроблена саме цією мовою.

- Кросплатформність: Використання інтерпретатора Python дає змогу безперешкодно запускати розроблений програмний код в операційних системах сімейства Windows, macOS та дистрибутивах Linux (зокрема у спеціалізованих ОС на кшталт Kali Linux) без необхідності перекомпіляції коду під конкретне апаратне забезпечення.

- Швидкість прототипування: Лаконічний та інтуїтивно зрозумілий синтаксис мови дозволяє зосередитися на реалізації складних алгоритмів фазингу (Fuzzing) та евристичного аналізу відповідей сервера, мінімізуючи часові витрати на низькорівневе керування пам'яттю.

Основою взаємодії сканера з цільовими вебсерверами є зовнішня бібліотека requests. Вона забезпечує високорівневий та надійний інтерфейс для відправлення HTTP-запитів різних типів (GET, POST), керування заголовками та обробки кодів стану відповідей [24].

Ключовою особливістю реалізації модуля активного аналізу (vuln.py) є

використання об'єкта requests.Session(). На відміну від виконання поодиноких запитів, об'єкт сесії дає змогу автоматично зберігати та передавати сесійні маркери (файли Cookie, наприклад PHPSESSID чи JWT) між усіма послідовними запитами сканера. Саме це програмне рішення дозволило реалізувати в системі методологію автентифікованого сканування (Authenticated DAST), що забезпечує можливість глибокого аудиту закритих зон вебдодатка (адміністративних панелей, профілів), які є недоступними для анонімних користувачів [11].

Для забезпечення ефективної роботи системи було реалізовано такі технічні підходи:

- Синтаксичний аналіз: Для обробки HTML-відповідей сервера (у модулях crawler.py для пошуку посилань та waf.py для ідентифікації захисних екранів) було використано вбудовану бібліотеку регулярних виразів re та стандартні модулі парсингу URL-адрес (urllib.parse). Це гарантує високу точність вилучення параметрів запитів та нормалізацію відносних посилань [22, 23].

- Інтеграція зі штучним інтелектом: Взаємодію з прикладним програмним інтерфейсом (API) великої мовної моделі у модулі ai_advisor.py реалізовано шляхом передачі структурованих JSON-об'єктів. Використання бібліотеки json дозволяє системі формалізувати знайдений технічний контекст вразливості та отримувати від LLM-моделі релевантні інструкції з виправлення помилок у форматі, придатному для автоматичного вбудовування у HTML-звіт.

- Поєднання потужних можливостей мови Python із модульною архітектурою дозволило створити гнучкий та масштабований інструментарій, здатний ефективно виконувати складні завдання з автоматизованого аудиту безпеки вебсерверів. Завдяки незалежності окремих компонентів, розроблений комплекс може бути легко адаптований під нові вектори кібератак без необхідності повної переробки вихідного коду.

Це забезпечує тривалий життєвий цикл продукту та можливість його потенційної інтеграції в сучасні конвеєри безперервної розробки та розгортання безпечного коду. Крім того, використання широкої екосистеми відкритих бібліотек Python значно спрощує подальше масштабування системи та додавання

нових аналітичн модулів.

## 3.2 Розробка графічного інтерфейсу (GUI) та організація багатопотокового виконання завдань

Для забезпечення високої ефективності взаємодії кінцевого користувача з програмним комплексом та спрощення процесу управління складними етапами аудиту було розроблено сучасний графічний інтерфейс користувача (Graphical User Interface - GUI). Оскільки програмні засоби для тестування на проникнення часто мають високий поріг входження, під час проєктування інтерфейсу пріоритет було надано принципам мінімалізму, ергономіки та технічної інформативності.

Архітектура та компоненти графічного інтерфейсу На рисунку 3.1 представлено зовнішній вигляд розробленого графічного інтерфейсу. Програма реалізована з використанням кросплатформних бібліотек мови Python, що гарантує ідентичність відображення елементів управління незалежно від обраної операційної системи [25].

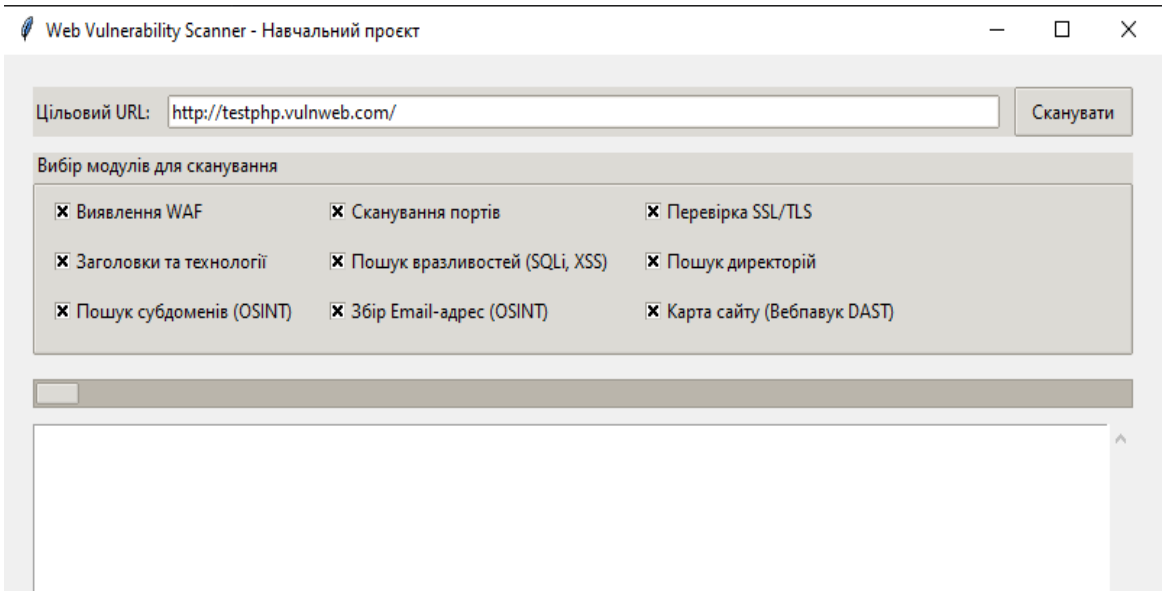



Рисунок 3.1 - Графічний інтерфейс сканера вразливостей

3.3 Практична апробація розробленого сканера: тестування на навчальних вразливих ресурсах та аналіз згенерованих звітів

Завершальним етапом розроблення інтелектуального сканера є його практична апробація, метою якої є підтвердження працездатності спроєктованих алгоритмів та оцінка точності виявлення вразливостей у реальних умовах. Відповідно до етичних норм кібербезпеки та законодавства, тестування проводилося в ізольованому лабораторному середовищі. Це дозволило детально проаналізувати реакцію системи на вразливості категорій OWASP Top 10 без ризику впливу на публічні мережеві ресурси.

Для проведення тестів було обрано метод розгортання локальної інфраструктури на базі стека WAMP/LAMP. Як платформу для тестування використано програмний комплекс XAMPP, що забезпечує функціонування вебсервера Apache та бази даних MySQL [29]. Такий підхід дозволив симулювати реальну серверну архітектуру, на якій функціонує цільовий вебдодаток. Стан розгорнутого тестового середовища під час проведення аудиту продемонстровано на рисунку 3.2.

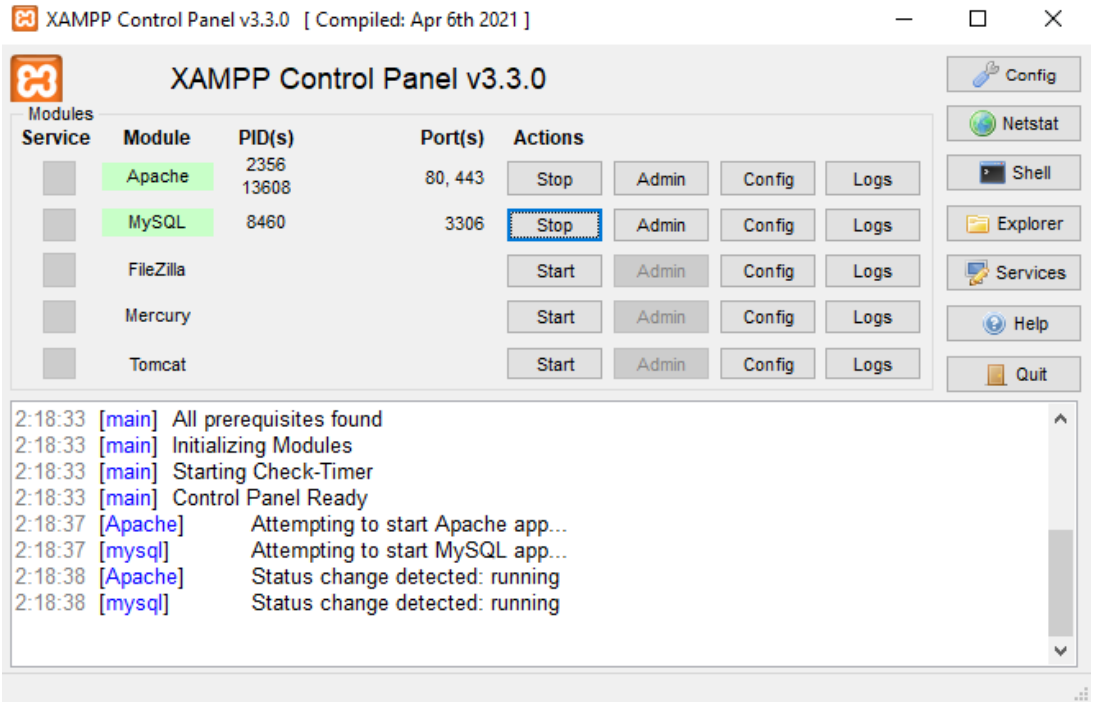
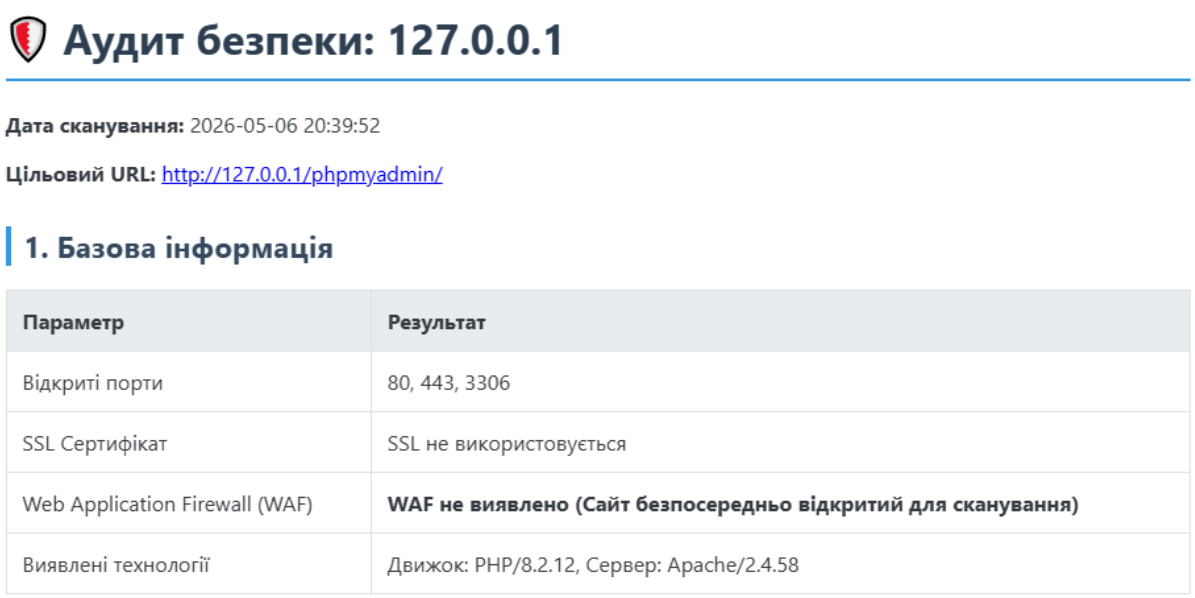



Рисунок 3.2 - Запуск локального тестового середовища в XAMPP Control Panel

Як видно з контрольної панелі XAMPP (рис. 3.2), для проведення аудиту було активовано модулі Apache та MySQL. Використання стандартних мережевих портів (80, 443 для вебсервісу та 3306 для СУБД) дозволило перевірити ефективність модуля network.py щодо автоматичного виявлення відкритих сервісів. Запуск сканування проводився щодо локального ресурсу 127.0.0.1, що є критично важливим для перевірки стійкості алгоритмів до роботи в мережах із мінімальними затримками.

Після завершення активної фази сканування програма автоматично згенерувала інтерактивний звіт у форматі HTML. Структура звіту побудована за принципом від загального до конкретного, починаючи з блоку ідентифікації цілі. На рисунку 3.3 представлено початковий фрагмент звіту з результатами напівпасивного аналізу.



## Аудит безпеки: 127.0.0.1

**Дата сканування:** 2026-05-06 20:39:52

**Цільовий URL:** http://127.0.0.1/phpmyadmin/

### 1. Базова інформація

| Параметр | Результат |
|---|---|
| Відкриті порти | 80, 443, 3306 |
| SSL Сертифікат | SSL не використовується |
| Web Application Firewall (WAF) | **WAF не виявлено (Сайт безпосередньо відкритий для сканування)** |
| Виявлені технології | Движок: PHP/8.2.12, Сервер: Apache/2.4.58 |

Рисунок 3.3 - Результати збору базової інформації та перевірки наявності WAF

Аналіз результатів пасивного аналізу (відповідно до рис. 3.3):

- Детектування інфраструктури: Сканер успішно ідентифікував відкриті порти (80, 443, 3306), що підтверджує коректну роботу мережевого модуля.

- Виявлення WAF: Алгоритм пасивного відбитка встановив статус «WAF не виявлено», що відповідає конфігурації локального сервера Apache без встановленого модуля ModSecurity. Це дозволило системі перейти до

агресивніших методів тестування.

- Fingerprinting технологій: Система точно визначила версію серверного ПЗ (Apache 2.4.58) та мови програмування (PHP 8.2.12). Отримання такої деталізації є критичним для модуля штучного інтелекту, оскільки дозволяє йому генерувати поради саме для вказаних версій технологій.

Другим етапом апробації стала перевірка ефективності модуля картування поверхні атаки (Web Crawler). Результати обходу структури сайту автоматично агрегуються у розділі «Карта сайту», як показано на рисунку 3.4.

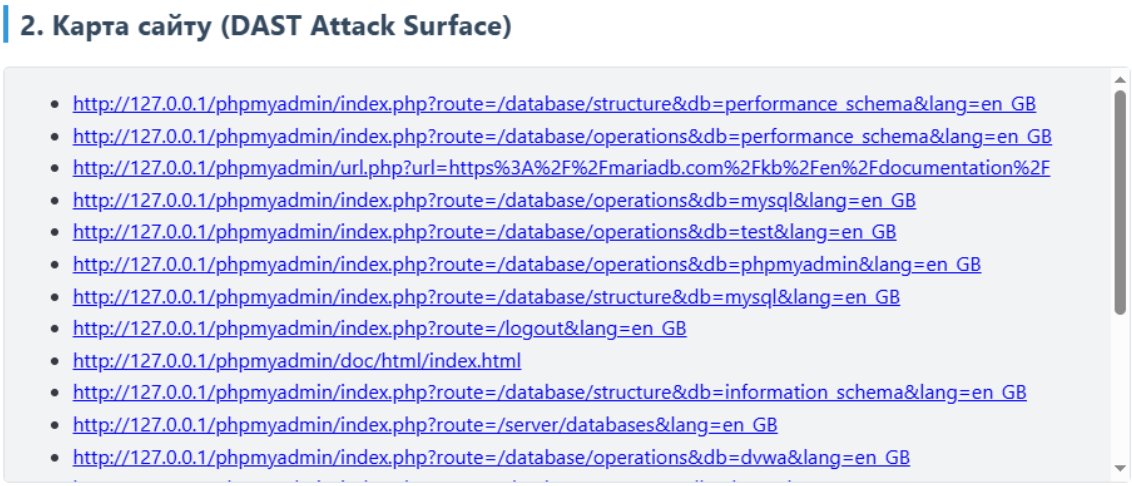



Рисунок 3.4 - Візуалізація карти поверхні атаки (DAST Attack Surface)

Наведений фрагмент (рис. 3.4) демонструє здатність сканера автоматично виявляти внутрішні посилання та параметризовані URL-адреси. Програма успішно просканувала структуру додатку phpmyadmin, виявивши складні маршрути із декількома GET-параметрами (наприклад, route, db, lang). Кожен із цих параметрів автоматично став об'єктом для подальшої ін'єкції тестових навантажень модулем vuln.py для пошуку вразливостей типу SQLi та XSS.

Окрему увагу під час тестування було приділено модулю пошуку прихованих ресурсів методами перебору (Brute-force). Результати роботи цієї підсистеми представлено на рисунку 3.5.

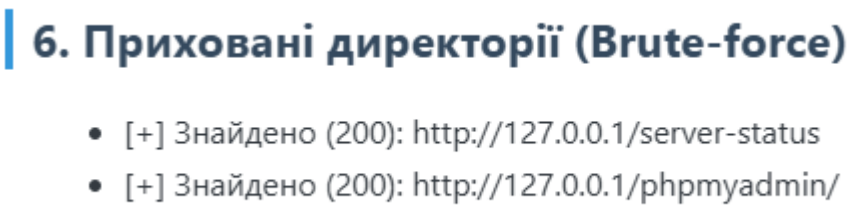



Рисунок 3.5 - Список прихованих директорій, виявлених шляхом фазингу

Завдяки використанню багатопотокового виконання (Multithreading) та оптимізованих словників, сканер зміг виявити ресурси, на які немає прямих посилань у вихідному коді сторінок. Зокрема, було успішно ідентифіковано системну директорію /server-status та вхід до адміністративного інтерфейсу /phpmyadmin/. Виявлення таких об'єктів підтверджує високу ефективність модуля directories.py та його здатність розширювати видиму площу атаки за межі стандартного краулінгу.

## 3.4 Розробка рекомендацій щодо усунення виявлених вразливостей для адміністраторів вебсерверів

Виявлення вразливостей є лише першим кроком у процесі аудиту безпеки. Для забезпечення реальної захищеності інфраструктури критично важливим є надання системним адміністраторам та розробникам чітких, технічно обґрунтованих інструкцій щодо усунення знайдених недоліків. У розробленому програмному комплексі «Web Vulnerability Scanner Project» це завдання вирішується за допомогою підсистеми інтелектуального аналізу загроз (модуль ai_advisor.py), яка виступає в ролі експертної системи.

На відміну від класичних сканерів, які використовують статичні бази шаблонів, розроблена система генерує рекомендації динамічно, враховуючи контекст знайденої вразливості. Після завершення активної фази сканування (модуль vuln.py), оркестратор формує масив виявлених дефектів та передає їх до ШІ-радника. Використовуючи спеціально сформований запит (Prompt

Engineering), система звертається до великої мовної моделі (LLM) через API, ставлячи їй завдання виступити в ролі експерта з кібербезпеки та надати покроковий план дій українською мовою.

Для найбільш критичних векторів атак, що виявляються під час тестування, система формує наступні базові рекомендації:

– Щодо усунення SQL-ін'єкцій (SQLi): Адміністраторам та розробникам категорично не рекомендується використовувати пряму конкатенацію рядків при формуванні SQL-запитів до бази даних. Основною рекомендацією є обов'язковий перехід на використання параметризованих запитів (Prepared Statements) або об'єктно-реляційних відображень (ORM). Крім того, необхідно впровадити строгу типізацію та валідацію всіх вхідних даних від користувача виключно на стороні бекенду.

– Щодо запобігання міжсайтовому скриптингу (XSS): Для нейтралізації XSS-загроз система рекомендує впровадити контекстно-залежне екранування вихідних даних (Context-Aware Output Encoding) перед їх рендерингом у браузері користувача. Додатковим, але критично важливим ешелоном захисту є налаштування HTTP-заголовка Content Security Policy (CSP) на стороні вебсервера, який жорстко регламентує, з яких саме джерел дозволено завантаження та виконання скриптів.

– Щодо захисту прихованих директорій та витоків даних: У разі виявлення модулем фазингу (directories.py) відкритих конфігураційних файлів (наприклад, .env), резервних копій або адміністративних панелей, система радить негайно перенести такі файли за межі публічної кореневої вебдиректорії (вище каталогу public_html або www).

Для адміністративних інтерфейсів рекомендується налаштувати обмеження доступу за IP-адресою (IP Whitelisting) та впровадити додаткову HTTP-автентифікацію безпосередньо на рівні вебсервера (Apache або Nginx).

Згенеровані ШІ-радником інструкції автоматично інтегруються у фінальний інтерактивний звіт модулем html_generator.py. Завдяки цьому адміністратор вебсервера отримує не просто констатацію факту наявності прогалини в безпеці,

але й готовий, адаптований під конкретну ситуацію алгоритм дій (Mitigation Plan). Такий підхід суттєво скорочує час реакції на інцидент та дозволяє оперативно підвищити загальний рівень захищеності вебдодатка ще до моменту можливої експлуатації вразливості зловмисниками.

## 3.5 Архітектура та функціональне призначення модулів програмного комплексу

Модуль динамічного аналізу та автентифікованого сканування (vuln.py, waf.py).

Програмна реалізація ядра сканера вразливостей зосереджена у двох ключових модулях, що відповідають за безпеку самого процесу аудиту та безпосереднє виявлення дефектів захисту: waf.py та vuln.py. Ці модулі реалізують методологію динамічного тестування (DAST), імітуючи дії зовнішнього зловмисника, але з можливістю використання привілейованого доступу для глибшого аналізу.

Основним завданням модуля waf.py є запобігання блокуванню сканера під час проведення аудиту. Перш ніж ініціювати активну фазу фазингу, система виконує ідентифікацію наявності та типу Web Application Firewall. У розробленому рішенні це реалізовано через сигнатурний аналіз HTTP-заголовків та файлів Cookie.

Фрагмент програмного коду, що відповідає за базу сигнатур мережевих екранів, представлено нижче на рис. 3.6.

```
waf_signatures = {
    "Cloudflare": ["cloudflare", "__cfduid", "cf-ray"],
    "AWS WAF": ["x-amzn-requestid", "x-amzn-trace-id"],
    "Akamai": ["x-akamai-request-id", "akamai"],
    "Sucuri": ["x-sucuri-id", "x-sucuri-cache"],
    "Imperva / Incapsula": ["incapsula", "x-iinfo"],
    "F5 BIG-IP": ["big-ip", "f5"],
}
```

Рисунок 3.6 - Реалізація словника сигнатур для детектування WAF

Аналіз механізму детектування (відповідно до рис. 3.6):

- Пасивне розпізнавання: Система використовує словник waf_signatures, де ключами є назви вендорів, а значеннями - масиви унікальних ідентифікаторів, що зустрічаються в заголовках відповідей сервера.

- Вендори захисту: Алгоритм здатен розпізнавати провідні рішення ринку, такі як Cloudflare (через маркер __cfduid), AWS WAF (x-amzn-requestid), Akamai, Sucuri, Imperva та F5 BIG-IP.

- Логіка роботи: Модуль ітерує отримані від сервера заголовки та шукає збіги із базою сигнатур. У разі виявлення WAF, сканер автоматично знижує швидкість надсилання запитів, щоб імітувати поведінку легітимного користувача та уникнути спрацювання антифлуд-фільтрів.

Модуль vuln.py є виконавчим механізмом DAST-ядра. Його програмна логіка базується на створенні динамічних об'єктів запитів, що містять шкідливі навантаження для пошуку SQL-ін'єкцій та XSS. Реалізацію алгоритмів перевірки цих вразливостей продемонстровано на рисунку 3.7

```
# 1. SQL Injection (CWE-89)
for payload in sqli_payloads:
    test_url = f"{base_url}?{param_name}={original_value}{payload}"
    try:
        res = requests.get(test_url, timeout=5)
        if any(error in res.text.lower() for error in sqli_errors):

            results.append(f"[🔴 Високий Ризик] SQL Injection у параметрі '{param_name}'\n    └─ Класифікація: CWE-89 | OWASP A03:2
            break
    except requests.RequestException:
        pass

# 2. XSS (CWE-79)
test_url_xss = f"{base_url}?{param_name}={urllib.parse.quote(xss_payload)}"
try:
    res = requests.get(test_url_xss, timeout=5)
    if xss_payload in res.text:
        # ДОДАНО КЛАСИФІКАЦІЮ OWASP ТА CWE
        results.append(f"[⚠️ Середній Ризик] Reflected XSS у параметрі '{param_name}'\n    └─ Класифікація: CWE-79 | OWASP A03:2021-Injecti
except requests.RequestException:
    pass
```

Рисунок 3.7 - Програмна реалізація алгоритмів виявлення SQL Injection та Reflected XSS

Аналіз програмної логіки (відповідно до рис. 3.7):

- Пошук SQL Injection (CWE-89): Алгоритм ітерує завантажений масив корисних навантажень (sqli_payloads). Для кожного вектора формується тестовий

URL, після чого виконується HTTP GET-запит. У разі виявлення у відповіді сервера ключових слів, що свідчать про помилку бази даних (масив sqli_errors), система фіксує вразливість із позначкою «Високий Ризик» та класифікацією за стандартами CWE та OWASP.

- Пошук Reflected XSS (CWE-79): Система використовує метод відображення маркерів. Тестове навантаження піддається URL-кодуванню за допомогою urllib.parse.quote для коректної передачі в параметрі. Якщо після виконання запиту оригінальний скриптовий рядок виявляється у незмінному вигляді у тілі відповіді сервера (res.text), це підтверджує відсутність фільтрації та наявність вразливості.

- Автентифіковане сканування: Програмний код спроєктовано для роботи з об'єктом сесії, що дозволяє інтегрувати сесійні маркери у кожен запит. Це дає змогу проводити аудит закритих частин вебдодатка, забезпечуючи максимальну повноту тестування.

- Синергія модулів waf.py та vuln.py дозволяє проводити аудит максимально приховано для систем захисту та максимально повно щодо внутрішньої структури вебдодатка, забезпечуючи високу точність виявлення вразливостей.

Інтеграція LLM для інтелектуального аналізу загроз (ai_advisor.py)

Однією з найбільш інноваційних та практично значущих складових розробленого програмного комплексу є модуль інтелектуальної підтримки прийняття рішень - ai_advisor.py. У той час як традиційні сканери вразливостей лише констатують наявність технічної проблеми, інтеграція великих мовних моделей (LLM - Large Language Models) дозволяє трансформувати сухий технічний звіт у персоналізовану інструкцію для розробника. Це значно скорочує час між виявленням загрози та її успішним усуненням (Time-to-Remediate).

Для реалізації цього функціоналу було обрано хмарну інфраструктуру Groq, яка забезпечує надвисоку швидкість генерації відповідей завдяки використанню спеціалізованих LPU-процесорів [19, 26]. Програмну реалізацію ключової функції модуля представлено на рисунку 3.8

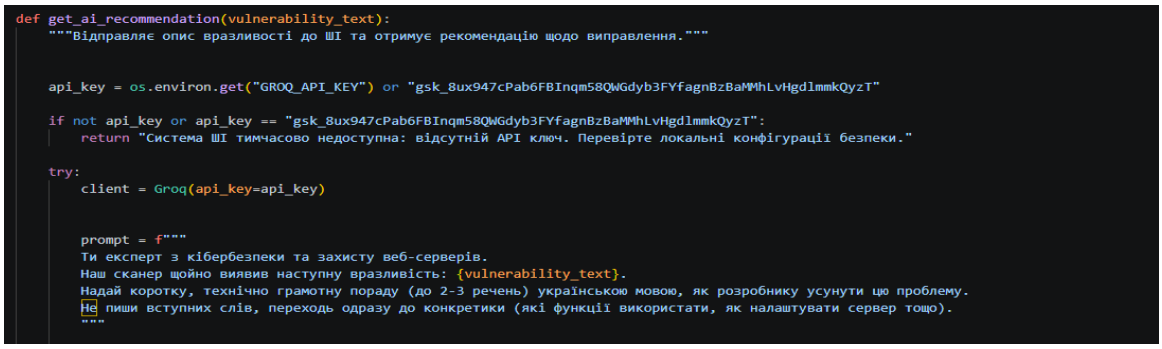



Рисунок 3.8 - Програмна реалізація модуля взаємодії з LLM та інженерія підказок (Prompt Engineering)

Технічний аналіз реалізації (відповідно до рис. 3.8) :

- Безпека та автентифікація: Функція get_ai_recommendation використовує безпечний метод отримання API-ключа через змінні середовища операційної системи (os.environ.get), що є стандартом безпечної розробки. У коді передбачено механізм валідації: якщо ключ відсутній або є недійсним, система не зупиняє роботу всього сканера, а повертає інформативне повідомлення про недоступність ШІ-модуля.

- Інженерія підказок (Prompt Engineering): Якість порад від штучного інтелекту безпосередньо залежить від точності сформульованого запиту (prompt). У розробленому модулі застосовано методику Role Prompting: системі призначається роль експерта з кібербезпеки та захисту вебсерверів [27].

- Контроль якості та стислості: Для того, щоб звіт залишався лаконічним та придатним для швидкого читання, у промпті встановлено жорсткі обмеження:

- Обсяг: порада не повинна перевищувати 2–3 речення.

- Мова: відповідь має бути виключно українською мовою.

- Стиль: заборонено використання вступних слів; система має одразу надавати назви конкретних функцій або налаштувань сервера.

- Контекстуалізація: У змінну {vulnerability_text} ядро системи передає вичерпний контекст виявленої вразливості: тип атаки (наприклад, SQLi), вразливий параметр, версію вебсервера та мову програмування бекенду. Це гарантує, що ШІ порадить, наприклад, використання саме prepared statements для Python/Flask, а не загальну пораду для PHP.

Інтеграція такого модуля дозволяє розв'язати проблему "інформаційного шуму" у звітах безпеки. Завдяки ai_advisor.py кінцевий користувач отримує не просто перелік помилок, а інтелектуально оброблений контент, який поєднує можливості точного машинного сканування із когнітивними здатностями сучасного штучного інтелекту до аналізу коду. Такий підхід робить розроблений інструмент незамінним помічником у циклі безпечної розробки програмного забезпечення.

Підсистема розвідки та збору інформації (network.py, osint.py, subdomains.py, web.py)

Підсистема розвідки є фундаментальним компонентом програмного комплексу, оскільки саме на основі зібраних нею даних формується подальша стратегія активного тестування.

Її архітектура складається з чотирьох спеціалізованих модулів, що реалізують комбінацію методів OSINT (розвідка з відкритих джерел) та активного мережевого аналізу.

Першим етапом розширення поверхні атаки є ідентифікація суміжних активів цілі через пошук субдоменів. Програмна реалізація модуля базується на методі перебору за словником найпоширеніших технічних назв. Фрагмент списку ідентифікаторів представлено на рисунку 3.9.

Такий підхід зумовлений тим, що адміністратори часто розміщують на окремих піддоменах тестові середовища, застарілі версії API або внутрішні панелі керування (наприклад, dev, test, admin, staging). Ці суміжні ресурси, як правило, мають нижчий рівень захисту та рідше покриваються основними правилами Web Application Firewall (WAF), що робить їх ідеальною точкою входу для потенційного зловмисника.

Усі виявлені активні субдомени автоматично додаються до загальної карти цілі для їх подальшого сканування. Завдяки цьому формується комплексна база даних інфраструктури, яка слугує надійним фундаментом для наступних етапів аудиту, зокрема сканування відкритих портів та ідентифікації запущених сервісів. Після структурування зібраних мережевих артефактів система ініціює спрямоване

тестування кожного виявленого вузла.

Такий системний підхід виключає поширену проблему, коли аудиту піддається виключно головна сторінка вебдодатка, залишаючи поза увагою слабо захищені суміжні сервіси або забуті API-інтерфейси. Зрештою, багаторівневе картографування поверхні атаки дозволяє максимально наблизити алгоритм роботи розробленого сканера до реальних дій зловмисника під час етапу розвідки (Reconnaissance), що значно підвищує загальну об'єктивність, глибину перевірки та репрезентативність фінального звіту про стан кіберзахисту цільової системи.

```
COMMON_SUBDOMAINS = [
    "www",
    "api",
    "dev",
    "test",
    "mail",
    "admin",
    "blog",
    "staging",
    "app",
    "portal",
    "beta",
    "cdn",
    "m",
    "shop",
]
```

Рисунок 3.9 - Програмна реалізація словника для пошуку субдоменів

Аналіз реалізації (відповідно до рис. 3.9): Модуль використовує масив COMMON_SUBDOMAINS, що включає стандартні технічні префікси, такі як api, dev, staging, vpn та mail. Алгоритм ітеративно надсилає DNS-запити для кожного елемента списку, намагаючись резолвити IP-адресу. Виявлення таких піддоменів часто дозволяє знайти тестові або застарілі версії додатків, які зазвичай мають нижчий рівень захисту, ніж основний ресурс.

Для забезпечення високої продуктивності етапу розвідки, надсилання DNS-запитів реалізовано з використанням багатопотоковості (або асинхронності), що дозволяє обробляти сотні ідентифікаторів за лічені секунди. Крім того, алгоритм оснащено механізмом обробки виключень (наприклад, NXDOMAIN), а також

логікою виявлення "Wildcard DNS" записів. Це запобігає масовим хибним спрацьовуванням (False Positives), коли сервер налаштований повертати однакову IP-адресу для будь-якого неіснуючого піддомену.

Модуль мережевого аналізу (network.py)

Для визначення доступних сервісів інфраструктури використовується модуль network.py, реалізацію якого наведено на рисунку 3.10.

```
def scan_ports(hostname):
    ports_to_scan = [21, 22, 80, 443, 3306, 8080]
    open_ports = []

    for port in ports_to_scan:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.settimeout(1.0)
        result = sock.connect_ex((hostname, port))

        if result == 0:
            open_ports.append(port)
        sock.close()

    return open_ports
```

Рисунок 3.10 - Реалізація функції багатопотокового сканування портів

Технічний аналіз (відповідно до рис. 3.10): Функція scan_ports використовує стандартну бібліотеку Python socket для спроби встановлення TCP-з'єднання. Система ітерує масив критичних портів:

- 21, 22: перевірка доступності FTP та SSH.
- 80, 443, 8080: ідентифікація вебсерверів.
- 3306: детектування відкритого доступу до СУБД MySQL. Використання методу connect_ex дозволяє ефективно отримувати код результату: значення 0 підтверджує, що порт відкритий і сервіс готовий до взаємодії.

Для збору контактних даних та ідентифікації потенційних векторів соціальної інженерії або пошуку витоків реалізовано модуль osint.py. Його ключову функцію продемонстровано на рисунку 3.11.

Зібрана у такий спосіб корпоративна чи персональна інформація становить

критичну цінність для оцінки загального рівня захищеності організації. Публічно доступні email-адреси співробітників найчастіше стають первинною мішенню для цілеспрямованих фішингових атак (Spear Phishing) та спроб підбору паролів (Credential Stuffing). Документування цих знахідок дозволяє завчасно попередити адміністраторів про надмірне розголошення чутливих даних у публічному просторі.

```
def extract_emails(url):

    try:
        response = requests.get(url, timeout=5)


        email_pattern = r"[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}"


        found_emails = set(re.findall(email_pattern, response.text))


        valid_emails = [email for email in found_emails if not email.endswith((

        return list(valid_emails)
    except requests.RequestException:
        return []
```

Рисунок 3.11 - Алгоритм вилучення адрес електронної пошти за допомогою регулярних виразів

Принцип роботи (відповідно до рис. 3.11): Функція extract_emails виконує HTTP-запит до цільової сторінки та проводить глибокий лексичний аналіз її контенту. Для пошуку адрес застосовується складний регулярний вираз (r"[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}"), що дозволяє точно ідентифікувати структуру електронної пошти у вихідному коді сторінки. Використання структури set гарантує унікальність зібраних адрес у фінальному звіті.

Завершальним етапом пасивної розвідки є глибокий аналіз конфігурації вебсервера та перевірка налаштувань безпеки на рівні протоколу HTTP, що реалізовано у спеціалізованому модулі web.py (рис. 3.12). На цьому етапі система

здійснює процедуру «захоплення банерів» (Banner Grabbing), аналізуючи HTTP-відповіді для ідентифікації типу та версії серверного програмного забезпечення.

```
def check_security_headers(url):

    security_headers = [
        'Content-Security-Policy',
        'X-Frame-Options',
        'X-XSS-Protection',
        'Strict-Transport-Security'
    ]
    results = {}
```

Рисунок 3.12 - Перелік критичних HTTP-заголовків безпеки для аудиту

Аналіз функціоналу (відповідно до рис. 3.12): Функція check_security_headers аналізує відповідь сервера на наявність механізмів превентивного захисту. До списку обов'язкової перевірки включено:

- Content-Security-Policy: захист від XSS та ін'єкцій даних [34].
- X-Frame-Options: запобігання атакам типу Clickjacking.
- Strict-Transport-Security: примусове використання шифрованого HTTPS-з'єднання. Відсутність будь-якого з цих заголовків класифікується системою як вразливість категорії Security Misconfiguration, про що користувач отримує відповідне попередження у звіті.

Синергійна робота цих чотирьох модулів дозволяє системі сформувати вичерпний профіль цілі, що є критично важливим для наступних етапів активного фазингу та інтелектуального аналізу загроз.

Модулі картографування та фазингу (crawler.py, directories.py)

Для проведення якісного динамічного тестування (DAST) необхідно виявити всі можливі точки входу - сторінки, форми та параметри вебдодатка. У розробленій системі це завдання виконується двома взаємодоповнюючими модулями: crawler.py (автоматичний обхід видимих посилань) та directories.py (пошук прихованих ресурсів методом перебору).

Модуль crawler.py призначений для автоматичної побудови карти сайту шляхом рекурсивного переходу за виявленими гіперпосиланнями. Логіку роботи

основної функції модуля наведено на рисунку 3.13.

```python
def crawl_site(base_url, max_pages=20):

    visited = set()
    to_visit = [base_url]
    internal_links = set()


    parsed_base = urllib.parse.urlparse(base_url)
    base_domain = parsed_base.netloc

    while to_visit and len(internal_links) < max_pages:
        current_url = to_visit.pop(0)

        if current_url in visited:
            continue

        visited.add(current_url)

        try:

            response = requests.get(current_url, timeout=5)


            if 'text/html' not in response.headers.get('Content-Type', ''):
                continue

            soup = BeautifulSoup(response.text, 'html.parser')
            internal_links.add(current_url)


            for link in soup.find_all('a', href=True):
                href = link['href']

                full_url = urllib.parse.urljoin(current_url, href)
                parsed_url = urllib.parse.urlparse(full_url)


                if parsed_url.netloc == base_domain and parsed_url.scheme in ['http', 'https']:
                    clean_url = full_url.split('#')[0]

                    if clean_url not in visited and clean_url not in to_visit:
                        to_visit.append(clean_url)

        except requests.RequestException:
            pass
    return list(internal_links)
```

Рисунок 3.13 - Реалізація рекурсивного вебкраулера для збору внутрішніх посилань

Технічний аналіз (відповідно до рис. 3.13):

- Синтаксичний аналіз: Функція crawl_site використовує бібліотеку BeautifulSoup із парсером html.parser для вилучення всіх тегів <a> з атрибутом href.

- Керування станом: Для уникнення зациклення та повторного сканування одних і тих самих сторінок використовується структура даних set (змінна visited).

- Фільтрація області (Scope): За допомогою методів бібліотеки urllib.parse система порівнює мережеву адресу кожного знайденого посилання з доменом цілі (base_domain). Це гарантує, що сканер не вийде за межі досліджуваного ресурсу і не буде проводити несанкціоноване тестування сторонніх сайтів.

- Обмеження інтенсивності: Параметр max_pages дозволяє контролювати глибину обходу, що важливо для оптимізації часу сканування великих ресурсів.

Модуль пошуку прихованих директорій (directories.py)

Оскільки багато критичних об'єктів (файли конфігурації, архіви баз даних, адміністративні інтерфейси) часто не мають прямих посилань на публічних сторінках, для їх виявлення реалізовано модуль directories.py. Програмну реалізацію методу фазингу за словником представлено на рисунку 3.14.

```python
def check_single_path(base_url, path):
    target_url = urllib.parse.urljoin(base_url, path)
    try:

        response = requests.head(target_url, timeout=3, allow_redirects=False)

        if response.status_code in [200, 403, 301, 302]:
            return f"[+] Знайдено ({response.status_code}): {target_url}"
    except requests.RequestException:
        pass
    return None

def find_hidden_directories(url):
    results = []
    parsed_url = urllib.parse.urlparse(url)
    base_url = f"{parsed_url.scheme}://{parsed_url.netloc}/"


    wordlist = [
        "admin/", "administrator/", "login/", "wp-login.php",
        "wp-admin/", "admin.php", "cpanel/", "phpmyadmin/",
        "db/", "database.sql", "backup.zip", "backup.sql",
        ".git/", ".env", "config.php", "config.bak",
        "test/", "api/", "server-status", "robots.txt"
    ]


    with concurrent.futures.ThreadPoolExecutor(max_workers=10) as executor:

        futures = [executor.submit(check_single_path, base_url, path) for path in wordlist]

        for future in concurrent.futures.as_completed(futures):
            result = future.result()
            if result:
                results.append(result)

    if not results:
        results.append("Прихованих директорій або критичних файлів зі стандартного списку не знайдено.")

    return results
```

Рисунок 3.14 - Реалізація багатопотокового пошуку прихованих ресурсів методом Brute-force

Аналіз реалізації (відповідно до рис. 3.14):

- Словниковий перебір: Функція find_hidden_directories використовує список wordlist, який містить найбільш критичні для безпеки шляхи: .env (файли конфігурації середовища), .git/ (репозиторії вихідного коду), phpmyadmin/ (інтерфейси баз даних) та backup.sql.

- Оптимізація продуктивності: Для прискорення процесу (оскільки він вимагає сотень послідовних запитів) застосовано ThreadPoolExecutor [39]. Це дозволяє одночасно перевіряти 10 різних шляхів, що значно скорочує загальний час роботи модуля.

- Евристика відповідей: Функція check_single_path надсилає легковагові HEAD-запити та аналізує коди стану. Система фіксує не лише успішні відповіді (200 OK), а й перенаправлення (301, 302) та помилки доступу (403 Forbidden), оскільки вони однозначно підтверджують фізичне існування ресурсу на сервері.

Результати роботи цих модулів агрегуються у загальну базу точок входу. Зібраний масив посилань автоматично передається до модуля vuln.py для подальшої активної перевірки на наявність SQL-ін'єкцій та вразливостей типу XSS.

Підсистема генерації адаптивної звітності (generator.py, html_generator.py)

Кінцевим етапом роботи будь-якої автоматизованої системи аудиту є представлення зібраних технічних даних у форматі, придатному для подальшого аналізу фахівцями. Підсистема звітності розробленого програмного комплексу призначена для консолідації результатів з усіх модулів (мережевого сканера, краулера, DAST-рушія та AI-асистента) та їхнього перетворення на структуровані звіти. Архітектура підсистеми передбачає підтримку двох форматів: текстового (TXT) для швидкого ознайомлення та інтерактивного (HTML) для детального вивчення результатів аудиту.

Модуль generator.py відповідає за формування легковагових звітів, які зручно використовувати для внутрішнього логування або перегляду в консольному середовищі. Логіку формування такого звіту представлено на рисунку 3.15.

```python
def generate_txt_report(data):
    filename = f"scan_report_{datetime.datetime.now().strftime('%Y%m%d_%H%M%S')}.txt"

    with open(filename, "w", encoding="utf-8") as f:
        f.write("="*50 + "\n")
        f.write(" ЗВІТ ПРО СТАН БЕЗПЕКИ ВЕБРЕСУРСУ \n")
        f.write("="*50 + "\n\n")

        f.write(f"Дата сканування: {datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n")
        f.write(f"Цільовий URL: {data['url']}\n\n")

        f.write("[1] ВІДКРИТІ ПОРТИ:\n")
        if data['ports']:
            for port in data['ports']:
                f.write(f" - Порт {port}: Відкрито\n")
        else:
            f.write(" - Відкритих портів не знайдено (або заблоковано фаєрволом)\n")

        f.write("\n[2] SSL СЕРТИФІКАТ:\n")
        f.write(f" - {data['ssl']}\n")

        f.write("\n[3] ЗНАЙДЕНІ СУБДОМЕНИ:\n")
        if data.get('subdomains'):
            for subdomain in data['subdomains']:
                f.write(f" - {subdomain}\n")
        else:
            f.write(" - Субдомени не знайдено.\n")

        f.write("\n[4] ТЕХНОЛОГІЇ:\n")
        for tech in data['tech']:
            f.write(f" - {tech}\n")

        f.write("\n[5] HTTP ЗАГОЛОВКИ БЕЗПЕКИ:\n")
        for header, status in data['headers'].items():
            f.write(f" - {header}: {status}\n")

        f.write("\n[6] ВИЯВЛЕНІ ВРАЗЛИВОСТІ В ПАРАМЕТРАХ:\n")
        for vuln in data['vulns']:
            f.write(f" - {vuln}\n")

        f.write("\n[7] ПРИХОВАНІ ДИРЕКТОРІЇ ТА ФАЙЛИ (Brute-force):\n")
        for directory in data.get('directories', []):
            f.write(f" - {directory}\n")
```

Рисунок 3.15 - Реалізація функції generate_txt_report для створення структурованих текстових звітів

Аналіз механізму формування TXT-звіту (відповідно до рис. 3.15):

- Динамічне іменування: Система автоматично створює унікальне ім'я файлу, використовуючи поточну дату та час за шаблоном scan_report_%Y%m%d_%H%M%S.txt. Це дозволяє уникнути перезапису попередніх результатів та забезпечує зручне сортування історії аудитів.

- Секціонування даних: Звіт розділений на сім логічних блоків, що відповідають етапам сканування:

- Відкриті порти: вивід списку доступних сервісів.
- SSL Сертифікат: інформація про стан шифрування.
- Знайдені субдомени: результати роботи модуля OSINT.
- Технології: визначений стек ПЗ цільового сервера.
- HTTP заголовки безпеки: перелік перевірених механізмів захисту.
- Виявлені вразливості: деталізація знайдених SQLi та XSS атак.
- Приховані директорії: результати роботи модуля фазингу (Brute-force).
- Валідація контенту: У кожному блоці реалізована логічна перевірка наявності даних. Якщо модуль було пропущено користувачем або результатів не виявлено, система записує інформативне повідомлення (наприклад, «Відкритих портів не знайдено»).

Для забезпечення високого рівня наочності, зручності навігації та можливості швидкого аналізу результатів проведеного аудиту було розроблено окремий програмний модуль html_generator.py. Даний компонент є ключовою частиною системи формування звітності, оскільки відповідає за перетворення зібраних технічних даних у зрозумілий для користувача формат у вигляді інтерактивної HTML-сторінки.

Головною метою модуля є автоматизація формування підсумкової документації, доступної для перегляду в будь-якому сучасному веббраузері. Згенерований HTML-звіт є повністю автономним: усі CSS-стилі, візуальна розмітка та колірне маркування рівнів ризику інтегруються безпосередньо в єдиний файл [35]. Це усуває залежність від стороннього ПЗ, забезпечує безпечне передавання результатів аудиту та дозволяє чітко структурувати технічні рекомендації, згенеровані підсистемою штучного інтелекту.

Фрагмент програмної логіки, що відповідає за підготовку вхідних даних для генерації HTML-шаблону, наведено на рисунку 3.16.

```python
def generate_html_report(data):

    parsed_url = urllib.parse.urlparse(data['url'])
    domain = parsed_url.netloc
    if not domain:
        domain = "unknown_site"

    domain = domain.replace(":", "_")
    date_str = datetime.datetime.now().strftime('%Y-%m-%d_%H-%M')
    filename = f"Аудит_{domain}_{date_str}.html"

    # --- БАЗОВА ІНФОРМАЦІЯ ---
    if data.get('ports') is None:
        ports_str = "<em>Пропущено користувачем</em>"
    elif not data.get('ports'):
        ports_str = "Не знайдено відкритих портів"
    else:
        ports_str = ', '.join(map(str, data.get('ports')))

    if data.get('tech') is None:
        tech_str = "<em>Пропущено користувачем</em>"
    elif not data.get('tech'):
        tech_str = "Технології не визначено"
    else:
        tech_str = ', '.join(data.get('tech'))

    waf_status = data.get('waf', '<em>Пропущено користувачем</em>')
    ssl_status = data.get('ssl', '<em>Пропущено користувачем</em>')
```

Рисунок 3.16 - Обробка логіки відображення даних у модулі

Технічний аналіз реалізації (відповідно до рис. 3.15):

- Нормалізація контексту: Функція проводить попередню обробку URL-адреси цілі для формування коректного заголовка та імені файлу, замінюючи символи, що не допускаються у назвах файлів (наприклад, двокрапки).

- Обробка виняткових станів: Оскільки користувач може вибірково відключати окремі модулі перед початком сканування, система використовує метод data.get(). Це дозволяє безпечно обробляти відсутні ключі в словнику даних. Якщо модуль не запускався, у звіті відображається позначка «Пропущено користувачем», стилізована за допомогою HTML-тегу .

- Генерація інтерактивних списків:

- Для краулера: Програма автоматично перетворює знайдені внутрішні посилання на клікабельні теги <a> з атрибутом target='_blank', що дозволяє адміністратору миттєво відкрити вразливу сторінку в новій вкладці браузера.

- Для субдоменів: Результати OSINT-розвідки ітеруються та вбудовуються у марковані списки <ul>/<li>.

- Стилізація та адаптивність: Кожна категорія даних (порти, технології,

статуси WAF) агрегується у рядки, розділені комою, що дозволяє компактно відображати великі масиви інформації на сторінці звіту.

Застосування такої підсистеми звітності дозволяє трансформувати сирі мережеві дані у структурований та зрозумілий технічний актив, придатний для подальшого аналізу й документування результатів перевірки безпеки. Формування звітів у декількох форматах забезпечує зручність роботи для різних категорій користувачів системи. Текстовий формат TXT є корисним для швидкого перегляду результатів, автоматизованої обробки даних та інтеграції з іншими інструментами аналізу, тоді як HTML-звіт забезпечує більш наочне та структуроване представлення інформації із збереженням форматування, таблиць і виділенням критичних вразливостей.

Наявність такої інтерактивної системи звітності суттєво оптимізує комунікацію між автоматизованим інструментарієм та кінцевим користувачем — фахівцем з кібербезпеки, розробником чи системним адміністратором. Завдяки деталізованим інструкціям від ШІ-модуля, звіт виступає не просто констатацією знайдених вразливостей, а готовим планом дій (Mitigation Plan) для їх оперативного усунення. Отримані результати формують фундаментальну базу для проведення подальших етапів аудиту, комплексної оцінки рівня захищеності вебресурсу та верифікації ефективності застосованих патчів безпеки.

Крім того, систематичне збереження звітів дозволяє створювати надійний аудиторський слід, здійснювати ретроспективний аналіз і відстежувати динаміку захищеності інформаційної системи в рамках концепції безперервного моніторингу безпеки [14].

Регулярне порівняння результатів сканувань допомагає об'єктивно оцінювати ефективність застосованих патчів та виявляти нові вразливості після оновлення коду. Такий підхід оптимізує процеси управління ризиками та дозволяє легко інтегрувати розроблений інструмент у парадигму безпечної розробки DevSecOps.

# ВИСНОВКИ

У ході виконання  бакалаврської роботи було проведено повний цикл розробки інтелектуальної системи для автоматизованого пошуку вразливостей вебсерверів. Основні результати та досягнення проєкту можна підсумувати наступним чином:

По-перше, було детально вивчено сучасний стан кібербезпеки у сфері вебтехнологій. Аналіз статистичних даних та звітів OWASP показав, що незважаючи на розвиток засобів захисту, такі вразливості як SQL-ін'єкції та XSS-атаки залишаються вкрай небезпечними. Це підтвердило актуальність створення інструменту, який дозволив би навіть розробнику-початківцю або адміністратору малого бізнесу швидко перевірити свій ресурс на базові дірки в захисті.

По-друге, було обґрунтовано архітектуру майбутнього сканера. Я обрав модульний підхід, що дозволило зробити програму гнучкою. Кожен блок системи - будь то мережевий сканер, модуль пошуку директорій чи рушій для аналізу вразливостей - працює незалежно. Це дозволяє легко оновлювати програму в майбутньому, додаючи нові типи перевірок без переписування всього коду.

По-третє, практична реалізація була виконана мовою Python, яка є стандартом у сфері кібербезпеки. В ході розробки було вирішено проблему повільної роботи мережевих запитів шляхом впровадження багатопотоковості. Це дозволило програмі виконувати сотні перевірок за лічені секунди, не перевантажуючи при цьому інтерфейс користувача. Особливу увагу було приділено створенню зручного графічного вікна (GUI), щоб робота зі сканером була інтуїтивно зрозумілою і не потребувала глибоких знань консольних команд.

Окремим важливим досягненням стала інтеграція штучного інтелекту.

На відміну від більшості існуючих сканерів, які просто видають помилку, мій проєкт аналізує контекст знайденої вразливості та за допомогою ШІ формує конкретну пораду українською мовою. Це значно спрощує життя адміністратору, оскільки він одразу отримує готове рішення або приклад коду для виправлення проблеми.

Практичне використання сканера в локальному середовищі XAMPP на

навчальних вразливих сайтах підтвердила його ефективність. Система успішно ідентифікувала версії серверного ПЗ, знаходила приховані адміністративні панелі та точно вказувала на вразливі параметри в коді сторінок. Програма показала стабільну роботу та високу швидкість обробки даних.

Таким чином, мету та завдання кваліфікаційної роботи було виконано в повному обсязі. У процесі дослідження було здійснено комплексний аналіз сучасного ландшафту кіберзагроз, стандартів класифікації вразливостей (OWASP Top 10, CWE) та методологій тестування безпеки. На основі цих досліджень було спроєктовано та програмно реалізовано мовою Python багаторівневу автоматизовану систему аудиту безпеки, яка вдало поєднує в собі класичні методи динамічного сканування (DAST) і розвідки з відкритих джерел (OSINT) із передовими технологіями штучного інтелекту.

Розроблений програмний комплекс довів свою ефективність завдяки гнучкій модульній архітектурі, яка забезпечує послідовне виконання етапів збору інформації, картографування структури цілі та активного виявлення критичних недоліків (зокрема, SQL-ін'єкцій та XSS). Визначальною інноваційною складовою розробки стала успішна інтеграція великих мовних моделей (LLM), що трансформувало систему зі звичайного сканера в інтелектуального помічника, здатного автономно формувати адаптивні рекомендації щодо усунення знайдених загроз [19].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Закон України «Про основні засади забезпечення кібербезпеки України» від 05.10.2017 № 2163-VIII.

2. ISO/IEC 27002:2022. Information Security, Cybersecurity and Privacy Protection – Information Security Controls. – Geneva: ISO, 2022.184 p.

3. ISO/IEC 27001:2022. Інформаційні технології. Методи захисту. Системи управління інформаційною безпекою. – Geneva: ISO, 2022.34 p.

4. Stallings W., Brown L. Computer Security: Principles and Practice. – 4th ed. – Hoboken: Pearson, 2018.816 p.

5. Zalewski M. The Tangled Web: A Guide to Securing Modern Web Applications. – San Francisco: No Starch Press, 2011.320 p.

6. OWASP Testing Guide v4.2 [Електронний ресурс] / OWASP Foundation. – 2021. – Режим доступу: https://owasp.org/www-project-web-security-testing-guide/ (дата звернення: 07.05.2026).

7. Seitz J. Black Hat Python: Python Programming for Hackers and Pentesters. – 2nd ed. – San Francisco: No Starch Press, 2021.216 p.

8. Kim P., Bahati D. The Hacker Playbook 3: Practical Guide to Penetration Testing. – Seattle: Secure Planet LLC, 2018.282 p.

9. Stuttard D., Pinto M. The Web Application Hacker’s Handbook: Finding and Exploiting Security Flaws. – Wiley, 2011.768 p.

10. SANS Institute. Open-Source Intelligence (OSINT) Gathering [Електронний ресурс]. – 2020. – Режим доступу: https://www.sans.org/reading-room/whitepapers/hackers/paper/37172 (дата звернення: 07.05.2026).

11. Grinberg M. Flask Web Development: Developing Web Applications with Python. – 2nd ed. – Sebastopol: O’Reilly Media, 2018.316 p.

12. Weidman G. Penetration Testing: A Hands-On Introduction to Hacking. – San Francisco: No Starch Press, 2014. 528 p.

13. Cormen T. H., Leiserson C. E., Rivest R. L., Stein C. Introduction to Algorithms. – 3rd ed. – Cambridge: MIT Press, 2009. 1312 p.

14. Scarfone K., Mell P. Guide to Intrusion Detection and Prevention Systems

(IDPS). NIST Special Publication 800-94. – Gaithersburg: NIST, 2007. 127 p.

15. Bass L., Clements P., Kazman R. Software Architecture in Practice. – 3rd ed. – Boston: Addison-Wesley, 2012. 624 p.

16. Engebretson P. The Basics of Hacking and Penetration Testing. – 2nd ed. – Waltham: Syngress, 2013. 223 p.

17. Halfond W. G., Viegas J., Orso A. A Classification of SQL Injection Attacks and Countermeasures // Proc. IEEE Symposium on Secure Software Engineering. – 2006. P. 13–15.

18. Grossman J. et al. XSS Attacks: Cross Site Scripting Exploits and Defense. – Burlington: Syngress, 2007. 480 p.

19. Gupta B. B., Agrawal D. P., Haoxiang W. Computer and Cyber Security: Principles, Algorithm, Applications, and Perspectives. – Boca Raton: CRC Press, 2018.666 p.

20. Про OWASP Top 10 : проект OWASP [Електронний ресурс]. – Режим доступу: https://owasp.org/www-project-top-ten/ (дата звернення: 07.05.2026).

21. CWE – Common Weakness Enumeration : офіційний сайт проекту MITRE [Електронний ресурс]. - Режим доступу: https://cwe.mitre.org/ (дата звернення: 07.05.2026).

22. Python Documentation : офіційна документація мови програмування Python [Електронний ресурс]. - Режим доступу: https://docs.python.org/3/ (дата звернення: 07.05.2026).

23. Beautiful Soup Documentation : бібліотека для парсингу HTML [Електронний ресурс]. - Режим доступу: https://www.crummy.com/software/BeautifulSoup/bs4/doc/ (дата звернення: 07.05.2026).

24. Requests: HTTP for Humans : документація бібліотеки Requests [Електронний ресурс]. - Режим доступу: https://requests.readthedocs.io/ (дата звернення: 07.05.2026).

25. Tkinter Documentation : посібник з розробки графічних інтерфейсів [Електронний ресурс]. - Режим доступу:

https://docs.python.org/3/library/tkinter.html (дата звернення: 07.05.2026).

26. Groq Cloud Documentation : документація API для прискорення виводу великих мовних моделей [Електронний ресурс]. - Режим доступу: https://console.groq.com/docs/ (дата звернення: 07.05.2026).

27. Prompt Engineering Guide : посібник з ефективної розробки підказок для LLM [Електронний ресурс]. - Режим доступу: https://www.promptingguide.ai/ (дата звернення: 07.05.2026).

28. ModSecurity Reference Manual [Електронний ресурс] / Trustwave. – 2022. – Режим доступу: https://github.com/SpiderLabs/ModSecurity/wiki/Reference-Manual-(v2.x) (дата звернення: 07.05.2026).

29. XAMPP Documentation : посібник з розгортання локальних серверних середовищ [Електронний ресурс]. - Режим доступу: https://www.apachefriends.org/docs/ (дата звернення: 07.05.2026).

30. NIST SP 800-115. Technical Guide to Information Security Testing and Assessment [Електронний ресурс]. - Режим доступу: https://csrc.nist.gov/publications/detail/sp/800-115/final (дата звернення: 07.05.2026).

31. OWASP Web Security Testing Guide v4.2 [Електронний ресурс] / OWASP Foundation. – 2021. – Режим доступу: https://owasp.org/www-project-web-security-testing-guide/ (дата звернення: 07.05.2026).

32. PortSwigger Web Security Academy [Електронний ресурс] / PortSwigger Ltd. – Режим доступу: https://portswigger.net/web-security (дата звернення: 07.05.2026).

33. SQL Injection Prevention Cheat Sheet : офіційний посібник OWASP [Електронний ресурс]. - Режим доступу: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html (дата звернення: 07.05.2026).

34. Cross-Site Scripting Prevention Cheat Sheet : офіційний посібник OWASP [Електронний ресурс]. - Режим доступу: https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html (дата звернення: 07.05.2026).

35. HTML Living Standard: офіційна специфікація стандартів веб-розмітки та інтеграції стилів [Електронний ресурс]. – Режим доступу: https://html.spec.whatwg.org/ (дата звернення: 07.05.2026).

36. EC-Council. Certified Ethical Hacker (CEH) Exam Guide [Електронний ресурс] / EC-Council. – Режим доступу: https://www.eccouncil.org/programs/certified-ethical-hacker-ceh/ (дата звернення: 07.05.2026).

37. PyInstaller Manual : документація щодо створення виконуваних файлів [Електронний ресурс]. - Режим доступу: https://pyinstaller.org/en/stable/ (дата звернення: 07.05.2026).

38. Mell P., Grance T. The NIST Definition of Cloud Computing. NIST Special Publication 800-145 [Електронний ресурс] / NIST. – 2011. – Режим доступу: https://csrc.nist.gov/publications/detail/sp/800-145/final (дата звернення: 07.05.2026).

39. concurrent.futures : документація модуля багатопотоковості Python [Електронний ресурс]. - Режим доступу: https://docs.python.org/3/library/concurrent.futures.html (дата звернення: 07.05.2026).

40. CVE - Common Vulnerabilities and Exposures : база даних відомих вразливостей [Електронний ресурс]. - Режим доступу: https://cve.mitre.org/ (дата звернення: 07.05.2026)

ДОДАТОК А

Фрагмент коду програми

ФАЙЛ GUI.PY

```
    class ScannerGUI:
        def __init__(self, root):
            self.root = root
            self.root.title("Web Vulnerability Scanner ")
            self.root.geometry("800x680")
            self.root.configure(padx=20, pady=20)

            self.url_frame = ttk.Frame(self.root)
            self.url_frame.pack(fill=tk.X, pady=(0, 10))

            ttk.Label(self.url_frame, text="Цільовий
URL:").pack(side=tk.LEFT, padx=(0, 10))

            self.url_entry = ttk.Entry(self.url_frame, width=50)
            self.url_entry.pack(side=tk.LEFT, expand=True, fill=tk.X)
            self.url_entry.insert(0, "http://testphp.vulnweb.com/")


            self.context_menu = tk.Menu(self.root, tearoff=0)
            self.context_menu.add_command(label="Вставити",
command=self.paste_from_menu)
            self.context_menu.add_command(label="Очистити",
command=self.clear_entry)

            self.url_entry.bind("<Button-3>", self.show_context_menu)

            self.url_entry.bind("<Control-KeyPress>",
self.keyboard_shortcuts)


            self.scan_btn = ttk.Button(self.url_frame, text="Сканувати",
command=self.start_scan)
            self.scan_btn.pack(side=tk.LEFT, padx=(10, 0))


            self.options_frame = ttk.LabelFrame(self.root, text=" Вибір
модулів для сканування ")
            self.options_frame.pack(fill=tk.X, pady=(0, 15), ipadx=10,
ipady=5)

            self.var_waf = tk.BooleanVar(value=True)
            self.var_ports = tk.BooleanVar(value=True)
            self.var_ssl = tk.BooleanVar(value=True)
            self.var_headers = tk.BooleanVar(value=True)
            self.var_vulns = tk.BooleanVar(value=True)
```

```
            self.var_dirs = tk.BooleanVar(value=True)
            self.var_subs = tk.BooleanVar(value=True)
            self.var_emails = tk.BooleanVar(value=True)
            self.var_crawler = tk.BooleanVar(value=True)


            ttk.Checkbutton(self.options_frame, text="Виявлення WAF",
variable=self.var_waf).grid(row=0, column=0, sticky=tk.W, padx=10, pady=5)
            ttk.Checkbutton(self.options_frame, text="Сканування
портів", variable=self.var_ports).grid(row=0, column=1, sticky=tk.W,
padx=10, pady=5)
            ttk.Checkbutton(self.options_frame, text="Перевірка
SSL/TLS", variable=self.var_ssl).grid(row=0, column=2, sticky=tk.W,
padx=10, pady=5)

            ttk.Checkbutton(self.options_frame, text="Заголовки та
технології", variable=self.var_headers).grid(row=1, column=0, sticky=tk.W,
padx=10, pady=5)
            ttk.Checkbutton(self.options_frame, text="Пошук вразливостей
(SQLi, XSS)", variable=self.var_vulns).grid(row=1, column=1, sticky=tk.W,
padx=10, pady=5)
            ttk.Checkbutton(self.options_frame, text="Пошук директорій",
variable=self.var_dirs).grid(row=1, column=2, sticky=tk.W, padx=10,
pady=5)

            ttk.Checkbutton(self.options_frame, text="Пошук субдоменів
(OSINT)", variable=self.var_subs).grid(row=2, column=0, sticky=tk.W,
padx=10, pady=5)
            ttk.Checkbutton(self.options_frame, text="Збір Email-адрес
(OSINT)", variable=self.var_emails).grid(row=2, column=1, sticky=tk.W,
padx=10, pady=5)
            ttk.Checkbutton(self.options_frame, text="Карта сайту
(Вебпавук DAST)", variable=self.var_crawler).grid(row=2, column=2,
sticky=tk.W, padx=10, pady=5)

            self.progress = ttk.Progressbar(self.root,
mode='indeterminate')
            self.progress.pack(fill=tk.X, pady=(0, 10))

            self.log_area = scrolledtext.ScrolledText(self.root,
wrap=tk.WORD, state=tk.DISABLED, font=("Consolas", 10))
            self.log_area.pack(expand=True, fill=tk.BOTH)

        def paste_from_menu(self):
            try:
                self.url_entry.event_generate("<<Paste>>")
            except Exception:
                pass

        def clear_entry(self):
            self.url_entry.delete(0, tk.END)
```

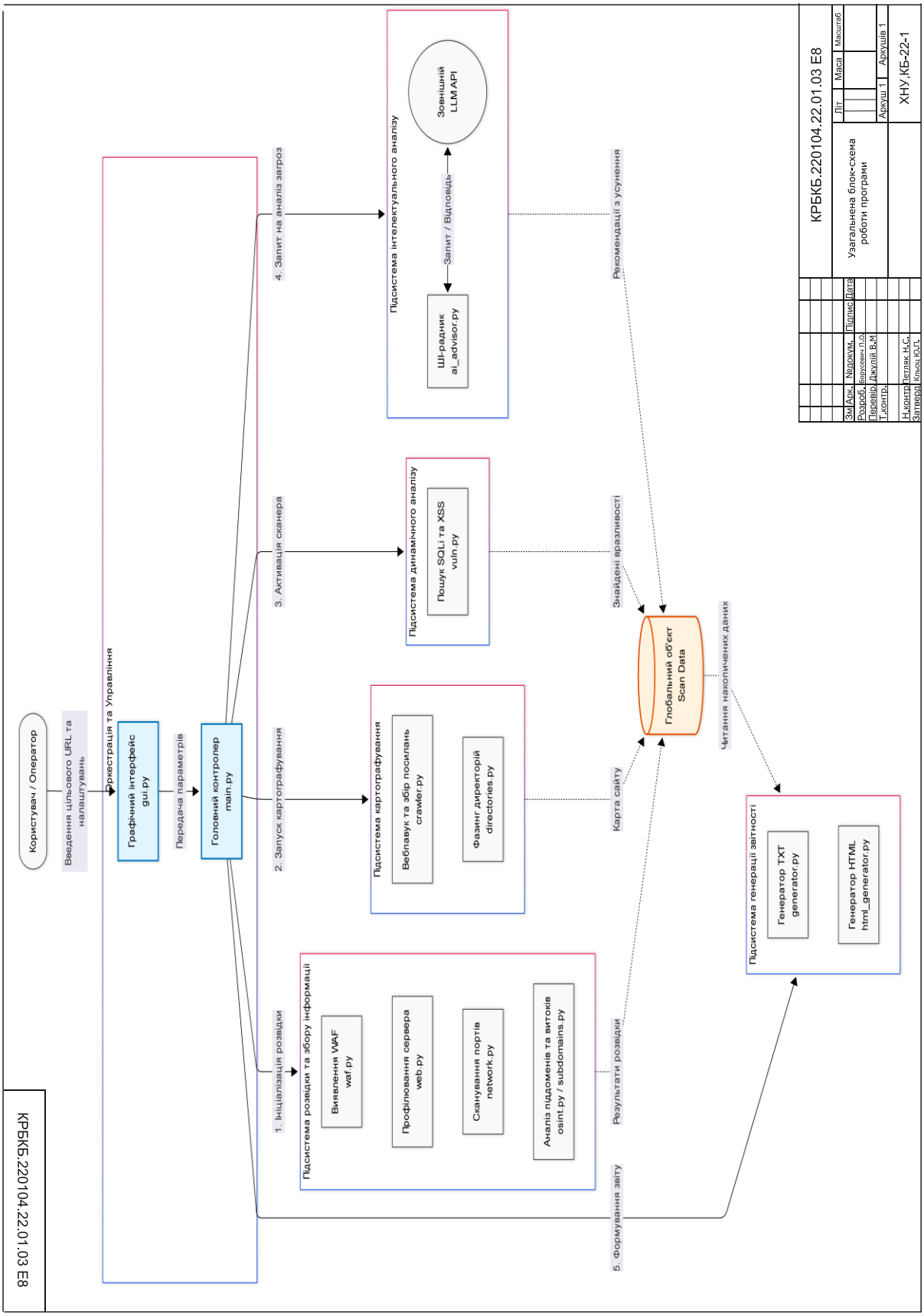
```
        def show_context_menu(self, event):
            self.url_entry.focus_set()
            self.context_menu.tk_popup(event.x_root, event.y_root)

        def keyboard_shortcuts(self, event):

            if getattr(event, 'keycode', 0) == 86:
                self.url_entry.event_generate("<<Paste>>")
                return "break"

        def log(self, message):
            self.log_area.config(state=tk.NORMAL)
            self.log_area.insert(tk.END, message + "\n")
            self.log_area.see(tk.END)
            self.log_area.config(state=tk.DISABLED)

        def start_scan(self):
            url = self.url_entry.get().strip()
            if not url.startswith("http"):
                messagebox.showerror("Помилка", "URL має починатися з
http:// або https://")
                return

            self.scan_btn.config(state=tk.DISABLED)
            self.progress.start(10)
            self.log_area.config(state=tk.NORMAL)
            self.log_area.delete(1.0, tk.END)
            self.log_area.config(state=tk.DISABLED)

            threading.Thread(target=self.run_scan, args=(url,),
daemon=True).start()

        def run_scan(self, target_url):
            self.log(f"[*] Початок сканування: {target_url}\n")

            parsed = urllib.parse.urlparse(target_url)
            hostname = parsed.netloc or parsed.path
            domain_only = parsed.netloc.split(':')[0]

            waf_info = "Пропущено користувачем"
            open_ports = None
            ssl_info = "Пропущено користувачем"
            headers_info = None
            tech_info = None
            vuln_info = None
            dir_info = None
            subdomains_info = None
            emails_info = None
            crawler_info = None

            try:
                if self.var_waf.get():
```

```
                    self.log("-> Перевірка наявності WAF...")
                    waf_info = detect_waf(target_url)
                    self.log(f"   {waf_info}")

                if self.var_ports.get():
                    self.log("-> Сканування портів...")
                    open_ports = scan_ports(hostname)

                if self.var_ssl.get():
                    self.log("-> Перевірка SSL...")
                    ssl_info = check_ssl(hostname) if
target_url.startswith("https") else "SSL не використовується"

                if self.var_crawler.get():
                    self.log("-> Запуск Вебпавука (картування поверхні
атаки)...")
                    crawler_info = crawl_site(target_url)
                    if crawler_info:
                        self.log(f"   Знайдено внутрішніх сторінок:
{len(crawler_info)}")

                if self.var_headers.get():
                    self.log("-> Перевірка заголовків безпеки та
технологій...")
                    headers_info = check_security_headers(target_url)
                    tech_info = detect_technologies(target_url)

                if self.var_vulns.get():
                    self.log("-> Пошук вразливостей (SQLi, XSS)...")
                    vuln_info = scan_vulnerabilities(target_url)

                if self.var_dirs.get():
                    self.log("-> Пошук прихованих директорій та адмін-
панелей...")
                    dir_info = find_hidden_directories(target_url)

                if self.var_subs.get():
                    self.log("-> Пошук субдоменів (API + Brute-
force)...")
                    subdomains_info = find_subdomains(domain_only)

                if self.var_emails.get():
                    self.log("-> Збір Email-адрес зі сторінки...")
                    emails_info = extract_emails(target_url)


                self.log("\n[+] Сканування завершено! Формування
звіту...")

                report_data = {
                    "url": target_url,
                    "waf": waf_info,
```

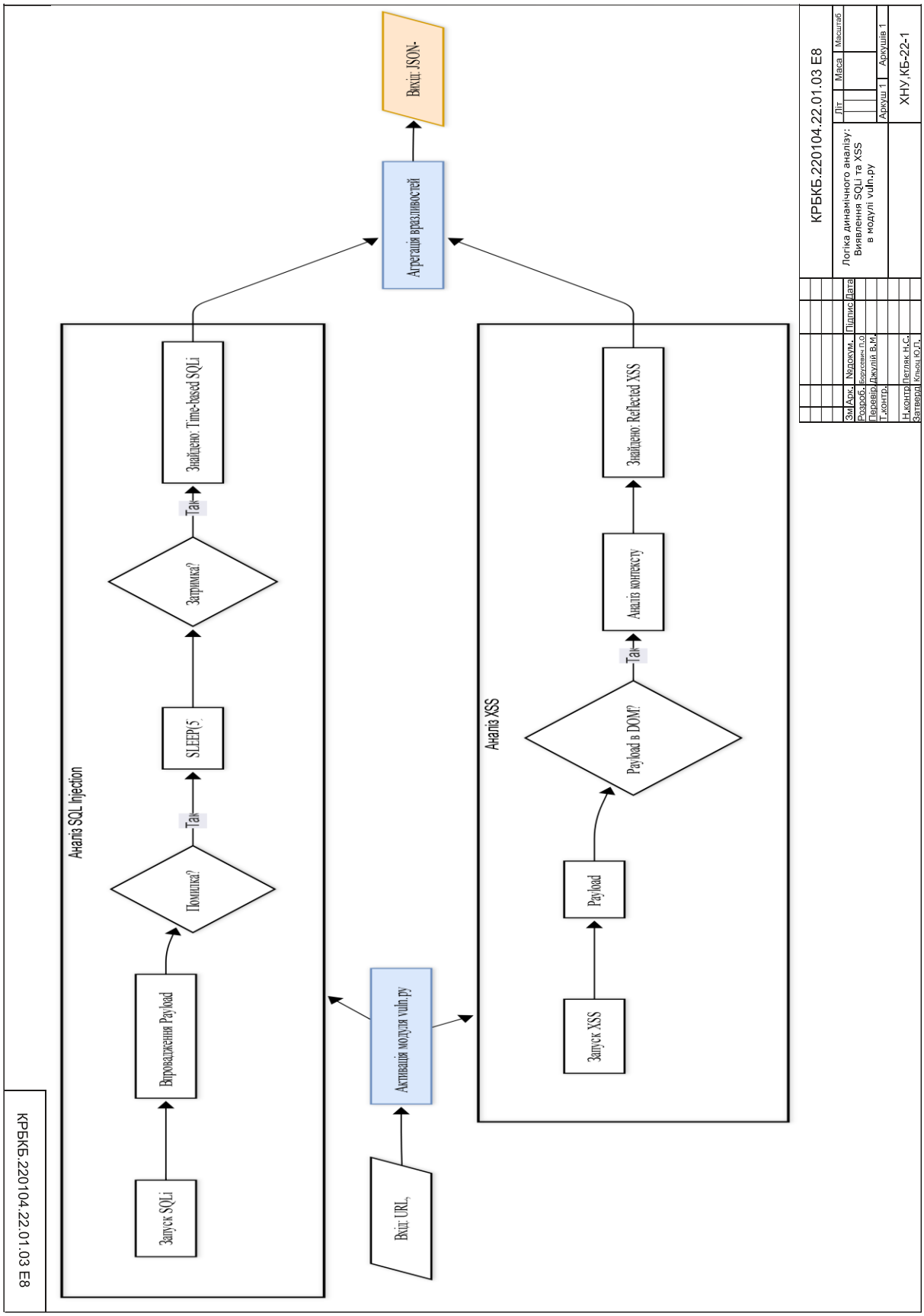
```
                    "ports": open_ports,
                    "ssl": ssl_info,
                    "headers": headers_info,
                    "tech": tech_info,
                    "vulns": vuln_info,
                    "directories": dir_info,
                    "subdomains": subdomains_info,
                    "emails": emails_info,
                    "crawler": crawler_info
                }

                report_filepath = generate_html_report(report_data)
                self.log(f"\n[!] Успіх! Звіт
збережено:\n{report_filepath}")

                self.root.after(0, lambda:
self.show_success_and_open(report_filepath))

            except Exception as e:
                self.log(f"\n[!] Виникла помилка під час сканування:
{e}")
                messagebox.showerror("Помилка", str(e))

            finally:
                self.progress.stop()
                self.scan_btn.config(state=tk.NORMAL)

        def show_success_and_open(self, filepath):
            if messagebox.askyesno("Готово", f"Сканування
завершено!\n\nЗвіт збережено.\nВідкрити його у браузері зараз?"):
                webbrowser.open(f"file://{filepath}")

    if __name__ == "__main__":
        root = tk.Tk()
        style = ttk.Style()
        if "clam" in style.theme_names():
            style.theme_use("clam")
        app = ScannerGUI(root)

        root.mainloop()
```

# ДОДАТОК Б

## Копія графічої частини «Узагальнена блок-схема роботи прогами»



Копія креслення «Логіка динамічного аналізу: Виявлення SQLi та XSS в модулі vuln.py»



Копія креслення «Методологія інтелектуального аналізу та Prompt Engineering (модуль ai_advisor.py) »