Хмельницький національний університет
Факультет інформаційних технологій
Кафедра кібербезпеки

КВАЛІФІКАЦІЙНА РОБОТА

Дейчука Ростислава Руслановича

на здобуття ступеня вищої освіти Бакалавра

Система проактивного сканування вразливостей хостинг-платформ

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

Шифр КРБКБ.220159.22.01.02 ПЗ

Виконав студент 3 курсу група КБс-22-1 Ростислав ДЕЙЧУК
Керівник д-р філософії Микола СТЕЦЮК
Нормоконтролер старший викладач Сергій МОСТОВИЙ

До захисту допускаю:
Завідувач кафедри кібербезпеки Юрій КЛЬОЦ

3 06 2025 р.

Хмельницький 2025

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Кібербезпеки

Рівень вищої освіти Бакалавр

Галузь знань 12 – Інформаційні технології

Спеціальність 125 – Кібербезпека

Освітня програма Кібербезпека

ЗАТВЕРДЖУЮ

Завідувач кафедри кібербезпеки

Юрій КЛЬОЦ

15 лютого 2025 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дейчуку Ростиславу Руслановичу

1 Тема роботи Система проактивного сканування вразливостей хостинг-платформ

Керівник роботи д-р філософії, Стецюк Микола Васильович

Затверджено наказом ректора університету від 15 лютого 2025 № 8

2 Строк подання студентом кваліфікаційної роботи на кафедру 2.06.2025

3 Вихідні дані до роботи Проєкт реалізовано у вигляді веб-додатку на основі ASP.NET. Основна мета – автоматизоване виявлення директорій веб-хостингів за допомогою словникових атак. Система підтримує зміну словників, вибір User-Agent'а, налаштування потоків та глибини сканування, а також модулі для аналізу HTML-тіла знайдених сторінок.

4 Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Аналіз сучасного стану безпеки веб-ресурсів, Методи виявлення прихованих директорій, Архітектура системи сканування, Реалізація модулів fuzzing, пошуку полів введення та завантаження файлів, Проведення тестування, Аналіз ефективності рішень.

5 Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Схема алгоритму сервісу відповідального за User-Agent. Схема алгоритму роботи бібліотеки AngleSharp. Схема алгоритму роботи рекурсії.

6 Консультанти розділів кваліфікаційної роботи

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

7 Дата видачі завдання 16 лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

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

Студент _____ Ростислав ДЕЙЧУК

Керівник кваліфікаційної роботи _____ Микола СТЕЦЮК

АНОТАЦІЯ

Тема кваліфікаційної роботи: Система проактивного сканування вразливостей хостинг-платформ.

Автор роботи: Дейчук Ростислав Русланович.

Керівник роботи: Стецюк Микола Васильович.

Пояснювальна записка: 70 с., 2 додатки, 52 рисунків, 6 таблиць, 40 джерел.

Графічна частина: 3 плакати, 12 презентаційних слайдів.

СКАНУВАННЯ ВРАЗЛИВОСТЕЙ, ФАЗЗИНГ, C#, WEB-БЕЗПЕКА, HTML-ПАРСИНГ, USER-AGENT, ПОТОКОВІСТЬ.

Кваліфікаційна робота бакалавра присвячена розробці програмного засобу AbyssScan – інструменту для автоматизованого сканування директорій веб-хостингів, що може застосовуватись у сфері тестування веб-ресурсів.

У роботі проаналізовано актуальні загрози веб-безпеці, оглянуто існуючі інструменти фаззингу та виявлено їх недоліки. На основі цього сформульовану технічні вимоги до власного рішення, спроєктовано архітектуру та реалізовано програму AbyssScan мовою C# із застосуванням .NET 8.

Серед функціоналу реалізовано: багатопоточне сканування з можливістю керування потоками, вбудовані словники, модуль парсингу HTML для виявлення точок введення даних, підміну User-Agent заголовків для маскування запитів.

Результатом вийшов готовий до використання інструмент із модульним підходом до налаштувань і функцією глибокого рекурсивного обходу структури сайту.

27.05.2025 _____

ABSTRACT

Subject of qualification work: Proactive vulnerability scanning system for hosting platforms.

Author: Deichuk Rostyslav Ruslanovych.

Head of work:Stetsiuk Mykola Vasylovych.

Explanatory note: 70 p., 2 appendices, 52 figures, 6 tables, 40 sources

Graphic part: 3 posters, 12 presentation slides.

VULNERABILITY SCANNING, FUZZING, C#, WEB SECURITY, HTML PARSING, USER-AGENT, MULTITHREADING.

The bachelor's qualification work is dedicated to the development of AbyssScan a software tool for automated scanning of web hosting directories, which can be used in the field of web resource testing.

The work analyzes current web security threats, reviews existing fuzzing tools, and identifies their shortcomings. Based on this analysis, technical requirements for a custom solution were formulated, the architecture was designed, and the AbyssScan application was implemented using the C# programming language and the .NET 8 framework.

The implemented functionality includes multithreaded scanning with user-defined thread control, built-in dictionaries, an HTML parsing module for detecting input points, and User-Agent header substitution for request masking.

As a result, a ready-to-use tool was developed, featuring a modular configuration approach and deep recursive scanning of the site structure.

27.05.2025 _____

ЗМІСТ

ВСТУП

Сказати, що безпека веб-ресурсів у сучасному цифровому світі набуває дедалі більшого значення – це не сказати нічого. У наш час, коли значна частина державних сервісів, бізнес-процесів, банківської системи, освіти та комунікації перенесена у веб-простір, захищеність сайтів та онлайн-додатків стає великою технічною проблемою. А на фоні нинішніх подій на теренах України, взагалі набуває стратегічного значення. Веб-сайти, веб-додатки, хмарні сервіси, CMS-платформи, REST API та інші елементи фронт- і бекенду – усе це стає об’єктами інтересу для потенційних атак.

Хакери намагаються знайти навіть найменші вразливості, щоб використати їх на свою користь: для викрадення конфіденційних даних, втручання у функціонування систем або для розповсюдження шкідливого ПЗ. За останні роки кількість кібер-атак по світу кратно зросла: тільки у сфері малого та середнього бізнесу фіксуються тисячі інцидентів за добу. Загрози, такі як несанкціонований доступ, витік персональних або фінансових даних, атаки типу “відмова в обслуговуванні” (DoS/DDoS), маніпуляція з куками, ін’єкції коду (XSS, SQLi), експлуатація CMS-вразливостей – лише частина тих небезпек, які несуть сучасні кіберзлочинці.

Одним із найбільших викликів є не лише кількість загроз, а й їхня динаміка: техніки атак постійно еволюціонують, стають складнішими, більш замаскованими і адаптивними. У таких умовах звісно звичайні методи ручного аудиту виявляються недостатньо швидкими або об’єктивними. Оскільки час реагування стає критичним фактором, особливо в умовах 0-day атак, фахівці з кібербезпеки віддають перевагу автоматизації пентесту.

Одним із найбільш ефективних напрямів є фаззинг директорій – тобто автоматичне сканування структури веб-ресурсу з метою виявлення “прихованих” директорій, адміністративних панелей, резервних файлів, тестових підкаталогів тощо. Це важлива частина розвідки (рекогностування) перед будь яким глибшим аналізом. Особливо небезпечними є ті директорії, які випадково залишаються

відкритими: наприклад, “/admin”, “/config”, “/debug”, “/.env” та інші. Доступ до таких папок може миттєво призвести до компроментації ресурсу.

Але є й проблема. Так як самостійно передбачити всі можливі назви директорій майже нереально – використовують словникові атаки – це перебір великої кількості URL-шляхів, сформованих із заздалегідь підготовлених списків. Щоб цей процес не займав години або дні, важливо обгорнути його у багатопоточність із затримками та адаптивним налаштуванням навантаження.

У відповідь на ці виклики виникають численні інструменти автоматизованої розвідки – такі як Dirbuster, Gobuster, Dirsearch, а також універсальні комплекси на зразок Burp Suite чи OWASP ZAP. Проте більшість із них або складні в налаштуванні, або мають вузьку спеціалізацію, або не дозволяють адаптувати поведінку під конкретну ціль без глибокого конфігурування. Саме тому актуальною є розробка власного гнучкого рішення, яке б відповідало конкретним вимогам безпеки, сучасним практикам та мало адаптивну структуру.

У даному дипломному проєкті буде реалізовуватись інструмент з назвою AbyssScan – мультифункціональний веб-додаток, який дозволяє проводити автоматизоване сканування директорій з урахуванням словникових сценаріїв, рекурсивної глибини, кількості потоків та штучних затримок. Крім того, додаток аналізує знайдені сторінки здійснюючи HTML-парсинг для виявлення форм, точок введення, полів завантаження файлів та інших можливих вразливих компонентів.

Сама система буде реалізовуватись на платформі ASP.NET Core MVC з використанням мовного середовища C# 8.0. Завдяки використанню багатошарової архітектури, додаток буде легко підтримуватись та розширюватись. Його компонентами будуть: контролери, сервіси, моделі та інтерфейси – відокремлені, що дозволятиме швидко вносити зміни або додавати новий функціонал без руйнування базової логіки.

Додаток призначений для локального використання – тобто запуску на машині фахівця без необхідності встановлення на сервер чи використання

зовнішніх сервісів. Це важливо з точки зору безпеки, адже так мінімізується передача даних третім лицям, що особливо актуально під час аудиту державних систем, внутрішніх сервісів компаній або систем з обмеженим доступом в Інтернет.

Метою даного дипломного проєкту є не лише розробка працюючого прототипу сканера, а й демонстрація комплексного підходу до розвідки веб-ресурсів, який охоплює перебір директорій, аналіз форм вводу, логування всіх дій у реальному часі та надання кінцевого результату користувачеві. Проєкт AbyssScan – це не “черговий сканер”, а інструмент з акцентом на зручність і контроль, призначений для фахівців з кібербезпеки та інших людей, які слідкують за кібер-гігієною своїх ресурсів.

1 АНАЛІЗ СУЧАСНОГО СТАНУ БЕЗПЕКИ ВЕБ-РЕСУРСІВ ТА МЕТОДІВ СКАНУВАННЯ ДИРЕКТОРІЙ

1.1 Огляд проблематики веб-безпеки

Так як об’єктом нашого дослідження служить веб-ресурс як такий, потрібно розібратись що ж він із себе представляє. Яка роль вебсайту? Роль вебсайту може відрізнятись залежно від його призначення та потреб аудиторії, яка з ним працює. Спектр ролей, які виконують вебресурси дуже широкий: від банального поширення інформації до підтримки або надання послуг. Якщо говорити про першу, це вебсайти які слугують платформою для обміну інформацією з різних тем, від новин і навчального контенту до деталей про продукти та послуги. Існують вебсайти, котрі мають в собі концепцію комунікації – вони сприяють комунікації між бізнесом та клієнтами, дозволяючи ставити запитання, залишати відгуки та отримувати підтримку. До речі, до таких вебсайтів частково відносяться також ті, які функціонують як інтернет-магазини, дозволяючи компаніям продавати товари та послуги безпосередньо споживачам – ця роль названа “електронною комерцією”. Також до ролей можна віднести формування бренду (вебсайт допомагає створити та зміцнити онлайн-ідентичність бренду), залучення аудиторії (вебсайти можуть залучати користувачів за допомогою інтерактивних функцій, таких як блоги, форуми тощо), демонстрація портфоліо (сайти, такі як Лінкедін і т.п.) і в кінці кінців вебсайти для освіти або розваг. Отже, ми бачимо, що веб-ресурси охоплюють широкий спектр надання послуг кінцевому користувачеві. Але і самі веб-ресурси можуть бути ціллю для атак зі сторони хакерів. [1]

Загрози для хостингів дуже різні. Наприклад, атака на відмову в обслуговувані (DoS) [2] спрямована на перевантаження ресурсів системи, щоб вона не могла обробляти легітимні запити. У разі розподіленої атаки відмови в обслуговуванні (DDoS) [3] використовується велика кількість заражених шкідливим ПЗ пристроїв, які одночасно надсилають запити до цільового сайту

(рисунок 1.1). Це призводить до того, що ресурси сайту вичерпуються, роблячи його недопустимим для користувачів.

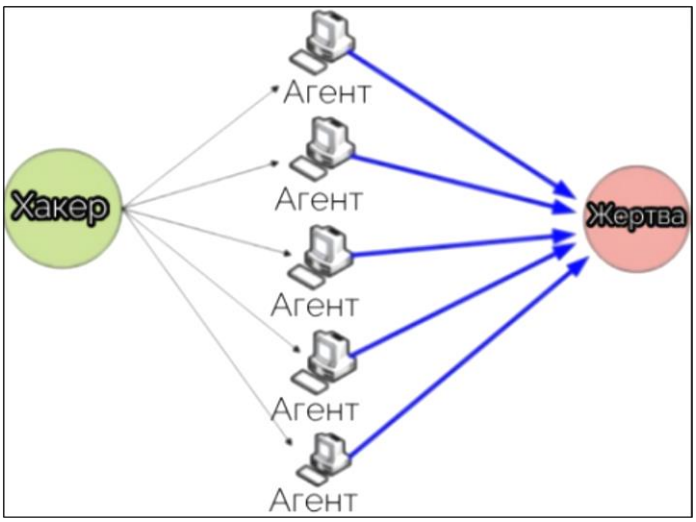


Рисунок 1.1– Схема DDoS атаки

Тип атак, відомий як SQL-ін’єкції [4] використовується через вразливості вебсайтів, які працюють з базами даних. Зловмисник вставляє шкідливі SQL-запити в поля вводу, наприклад, для логіну або паролю, змушуючи сервер виконувати команди, що надають доступ до конфіденційних даних або порушують функціональність системи (рисунок 1.2).

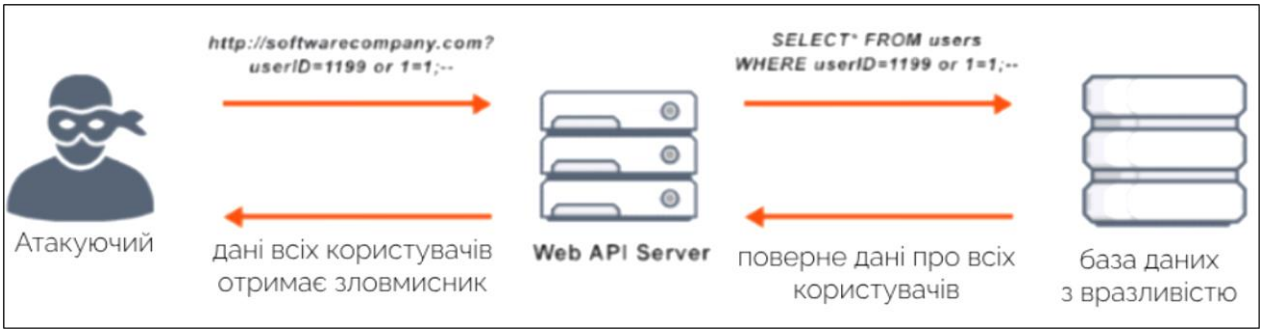


Рисунок 1.2 – Схема атаки SQL-ін’єкції

Як раз тип атаки, імітація якої є в основі нашого проекту – це інтерпретація URL [5]. Тобто хакери змінюють або підробляють адреси для отримання доступу до конфіденційних даних або адміністраторських привілеїв

на вебсайті. Наприклад, вони можуть вгадати URL-адресу для доступу до адміністративної панелі сайту. Якщо пароль слабкий – хакер без проблем стає адміністратором і маніпулює даними.

XSS або міжсайтовий скриптинг, схема атаки якого зображена на рисунку 1.3, працює так, що зловмисник вставляє шкідливий код у вебсайт, який виконується в браузері користувача .

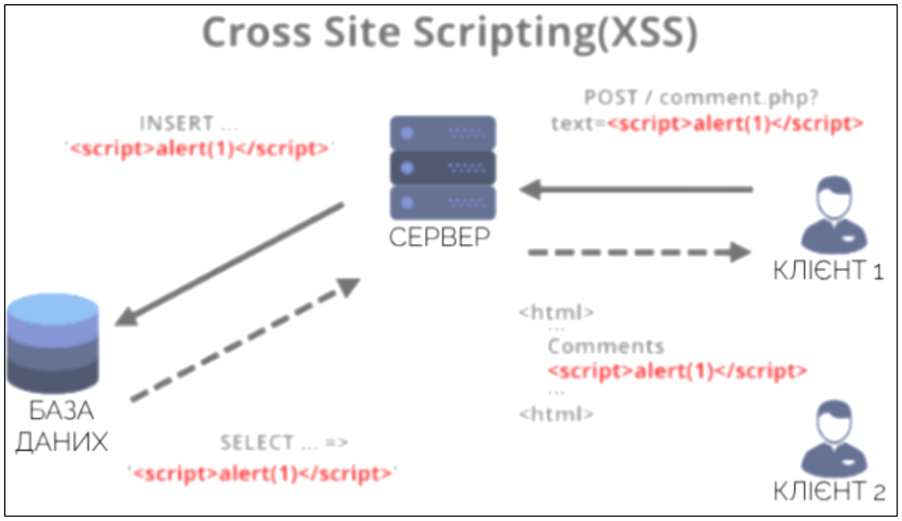


Рисунок 1.3 – Схема XSS атаки

Це може призвести до крадіжки даних або небажаних дій від мені користувача. [6]

1.2 Існуючі методи та інструменти сканування директорій

В загалі існує два основних підходи до сканування директорій – це ручний метод і автоматизований. Почнімо з ручного. Ручний перебір директорій є ключовим етапом розвідки і використовується для отримання максимально релевантної інформації про ціль перед експлуатацією вразливостей. Це дозволяє хакеру або пентестеру зібрати деталі про структуру вебсайту чи системи, визначити потенційно небезпечні точки входу або конфіденційні файли [7].

Ручний підхід більше підходить для досвідчених фахівців, які вміють аналізувати результати та адаптувати стратегію атаки залежно від отриманих даних. Хакер або пентестер має заздалегідь зрозуміти цілі атаки: це може бути атака на точку входу, наприклад, через SQL-ін'єкції, або пошук відкритих конфігураційних файлів, які можуть містити паролі, ключі доступу чи налаштування системи. Розглянемо модель ситуації ручного перебору. Пентестер проводить перевірку безпеки вебсайту компанії. Його мета – знайти конфіденційні файли, які випадково залишились у відкритому доступі.

Дії починаються з розвідки. Пентестер починає з аналізу структури сайту за допомогою URL-адрес. Він виявляє, що сайт використовує стандартну структуру директорій, наприклад: "https://example.com/admin" або "https://example.com/uploads". Діючи тільки в рамках ручного перебору директорій, пентестер намагається відкрити кожну підозрілу директорію. Виявляється, що в директорії https://example.com/uploads/ доступні файли з бекапами сайту (backup.zip).

Коли пентестер знаходить такий файл, він завантажує його собі на машину і розпаковує, а потім знаходить в ньому файл config.php, де зберігаються логін і пароль до бази даних. Отримавши доступ до бази даних, він проводить SQL-ін'єкцію, щоб отримати доступ до таблиць з конфіденційною інформацією.

Для ручного методу пошуку директорій, як вже говорилось вище, потрібно обладати знаннями стандартних шляхів збереження файлів, адже це допоможе шукати потенційно вразливі файли. Наприклад такі директорії, як "/admin", "/config", "/backup", "/temp" перевіряються досвідченими пентестерами в ручну в першу чергу. Також в нагоді можуть бути такі файли як ".git/", ".env", "test.php", "debug.log"; це файли-залишки від розробки або тестуваня.

Серед шляхів подальшого розвитку атак після ручного перебору не тільки вищезгадана SQL-ін'єкція. Це також може бути отримання доступу до API – якщо виявлено відкриті API-інтерфейси, можна використовувати їх для автоматизованого збору даних або обходу автентифікації [8]. Сюди ж зловживання відкритими файлами, такими як ".env" – він може містити секретні

ключі до сторонніх сервісів, як от AWS, що відкриває шлях до нових атак. Експлуатація конфіденційних даних також може бути можлива, якщо знайдено бекапи або лог-файли, вони можуть містити паролі, сесійні токени або інші чутливі дані.

Окрім ручного підбору директорій, існує і автоматичний підхід. Він також дозволяє досліджувати структуру URL для виявлення прихованих сторінок, директорій або файлів. Найпопулярніші інструменти які входять в кластер автоматизованих енумераторів директорій і вебсторінок є – Dirbuster, Dirsearch, Gobuster та частково Burp Suite. [9]

Розглянемо такий інструмент, як Gobuster [10]. Це утиліта для швидкого перебору директорій, піддоменів або віртуальних хостів, яка має в собі навіть можливість пошуку файлів із заданним розширенням. Інструмент працює за принципом грубого перебору або brute-force, надсилаючи HTTP-запити на сервер на основі попередньо сформованого словника. Він перевіряє наявність файлів або директорій, порівнюючи відповіді сервера з очікуваними кодами статусу (наприклад, 200 OK 403 або Forbidden). Для ефективного використання Gobuster необхідний текстовий файл-словник, який містить перелік можливих імен файлів і директорій. Існує багато готових словників, таких як common.txt, що містять популярні імена для швидкого сканування. Проте, за потреби, користувач сам може сформувати свій словник орієнтуючись на специфіку вебресурсу.

Крім того, що Gobuster дозволяє шукати приховані каталоги, він також може шукати файли за розширенням, а це значно підвищує точність бажаного результату. Наприклад, можна налаштувати перевірку лише “.php”, “.html”, “.txt” файлів.

Розширений режим виводу за параметрами надає користувачеві простір для масштабування свого сканування. Параметр “-e” дозволяє отримати повні адреси знайдених файлів і директорій; функція фільтрації кодів статусу за параметрами “-s” або “-b” включить або виключить деякі коди статусу зі сканування відповідно.

Блокування зі сторони веб-сайтів легко обходиться за допомогою параметра “--proxy”. Тоді запити виконуються через вказаний проксі, що імітує активність окремих користувачів. Функція --random-agent дозволяє додавати випадкові заголовки User-Agent для кожного запиту, що ускладнює детектування автоматизованого сканування.

На рисунку 1.4 видно, як розпочинається тестове сканування з використанням словника common.txt, цей словник вміщає 4614 рядків. Сканування відбувається на десяти потоках і пауза між запитами – 10 секунд. Результати і справді видають лише вказані формати файлів разом із статус-кодом. Текст статус кода підфарбований в окремий колір в залежності від коду. Сканування завершилось за три хвилини.

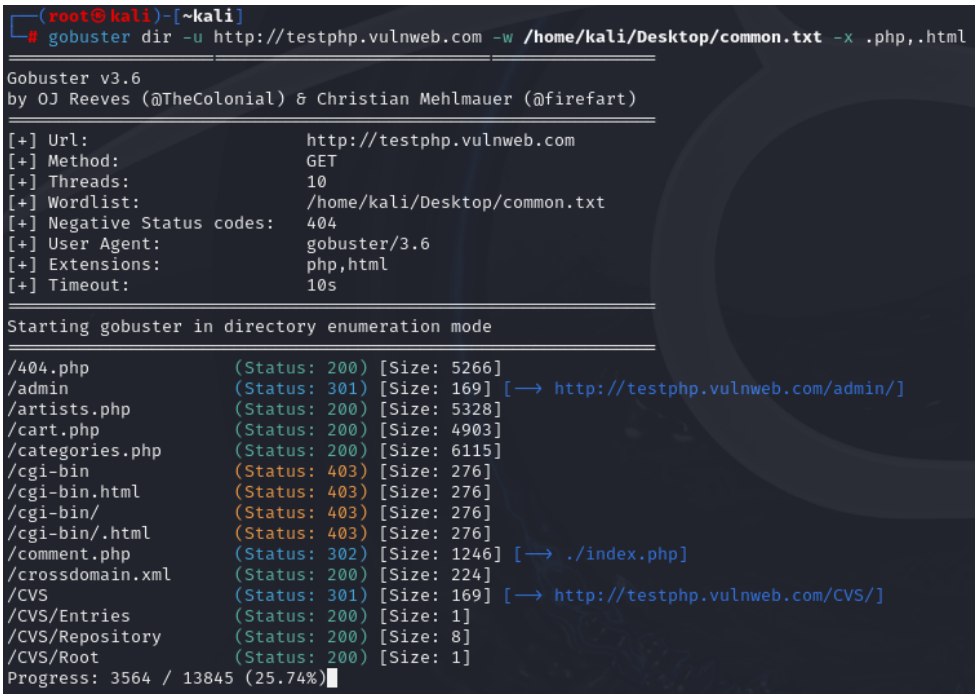


Рисунок 1.4 – Розпочали тестове сканування файлів з конкретним розширенням (“.html”, “.php”)

Наступний інструмент на огляді – це OWASP Dirbuster [11], який написаний на мові Java. Також призначений для сканування каталогів сайту, і так як і попередній інструмент Gobuster має багатопоточність. В Kali linux він йде з “коробки” але якщо користувач використовує інший дистрибутив Лінукс

або Віндоус, Dirbuster потрібно буде встановити. Його можна використовувати як і в консолі так і користуватись графічним інструментом (рисунок 1.5):

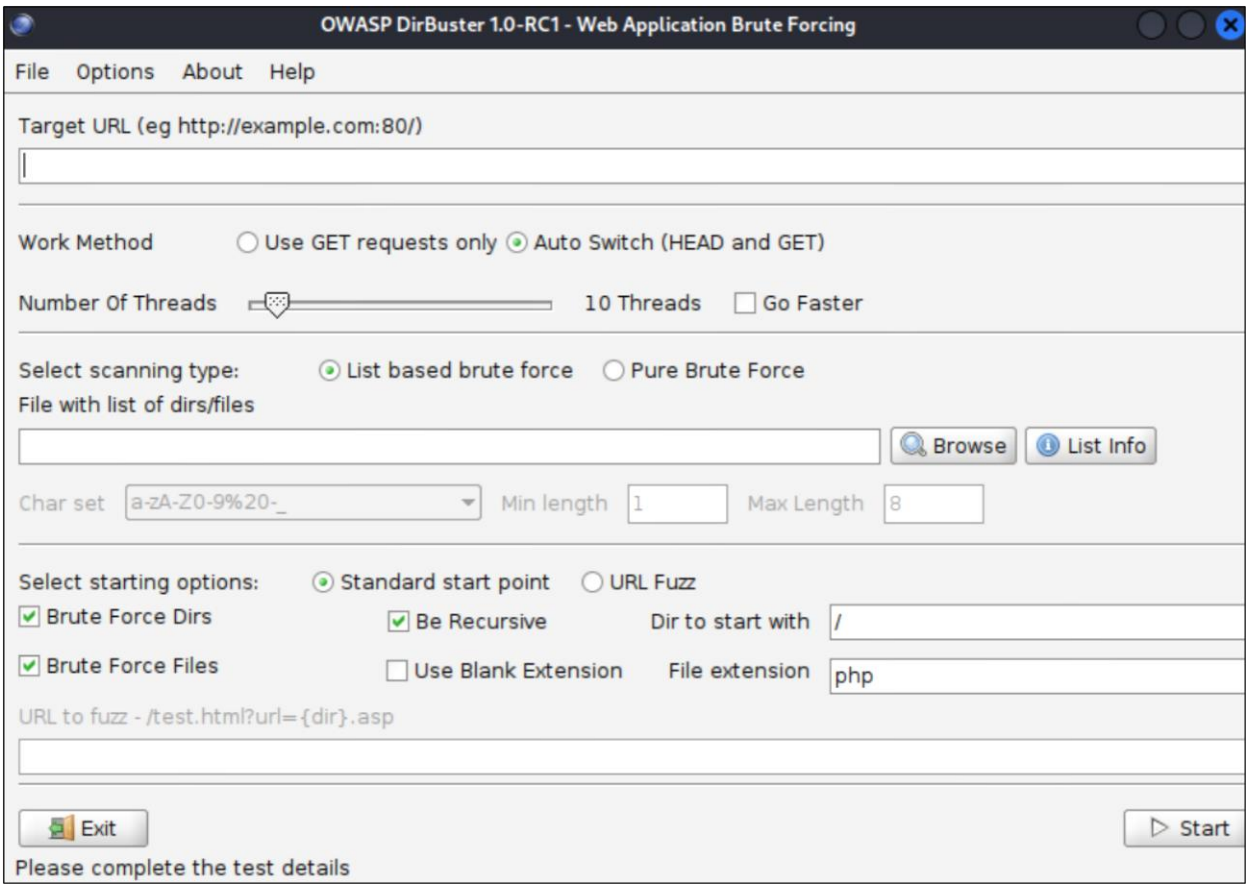


Рисунок 1.5 – Вигляд графічного інтерфейса Dirbuster

Після запуску сканування цілі з такими ж параметрами, як і в першому випадку (тільки додавши рекурсивність), ми можемо буквально дивитись за процесом сканування зі всіх сторін. У вікні “scan information” інструмент надає огляд усіх режимів сканування які ми запустили, включно з прогрес-баром до кожного. У вікні списку результатів можемо наблюдати таблицю з типом знайденого елементу (це директорія або файл), назва знайденого об’єкту, статус-код та розмір сторінки. Також Dirbuster будує дерево з директорій і файлів, котрі він шукає, це допомагає розуміти структуру веб-ресурсу, з яким ми працюємо. В одному з моментів інструмент навіть викинув помилку і очікував на наші дії, так він зреагував на знайдену SQL-ін’єкцію (рисунок 1.6):

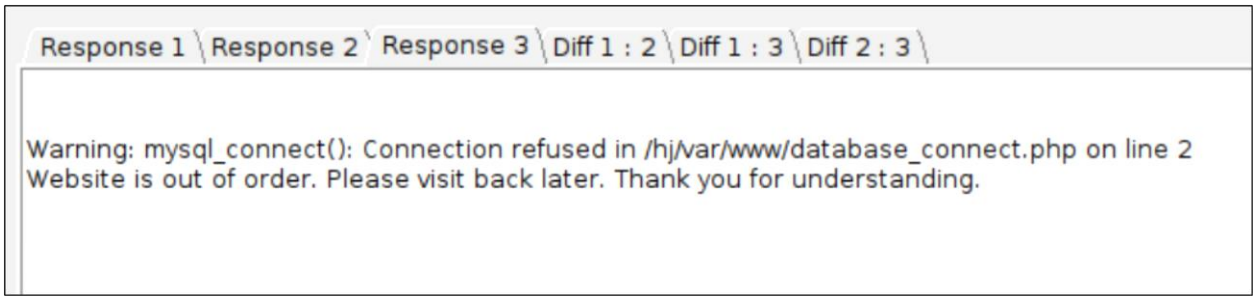


Рисунок 1.6 – Реакція Dirbuster на знайдену SQL-ін’єкцію

Також плюсом виявилось те, що кількість потоків можна міняти “на льоту”. Наприклад, помінявши з 10 потоків на 50, поточна швидкість 15 запитів в секунду відчутно змінилась – стало 240 запитів на секунду. В кінці сканування (рисунок 1.7) приємним бонусом була можливість складення звіту про сканування, де також можна вказати бажаний формат звіту – текстовий файл, “.xml” формат або “.csv”.

Рисунок 1.7 – Вікно по завершенню сканування

Настав час випробувати інструмент Burp Suite [12]. Цей інструмент написаний на Java і він не призначений суто для перебору директорій або пошуку файлів на сайтах. Але має в собі модуль названий “Intruder” (рисунок 1.8), в ньому можна вибирати потрібні точки в тілі сайту або в адресі і по ним робити перебір.

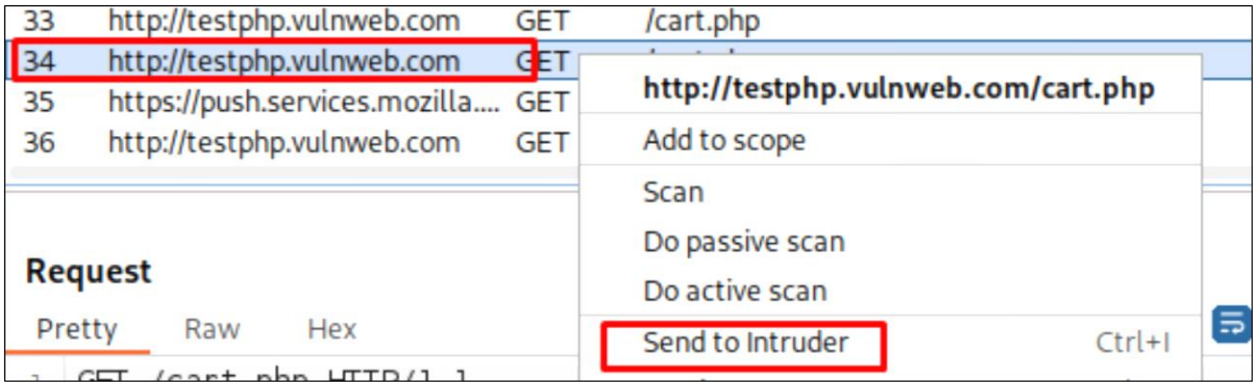


Рисунок 1.8 – Вибрали наш запит і відправляєм у вкладку “Intruder”

У випадку із Burp Suite знадобиться більше дій, так як тут ми працюємо лише через проксі інструмента. У розділі Proxy/HTTP HISTORY знаходимо запит, який тільки що відбувся у браузері, виділяємо його та після натиску правою кнопкою миші відправляємо у Intruder. Потім безпосередньо у останній вкладці налаштовуємо бажаний перебір, вказавши спеціальними символами з обох боків де буде наше корисне навантаження (рисунок 1.9):

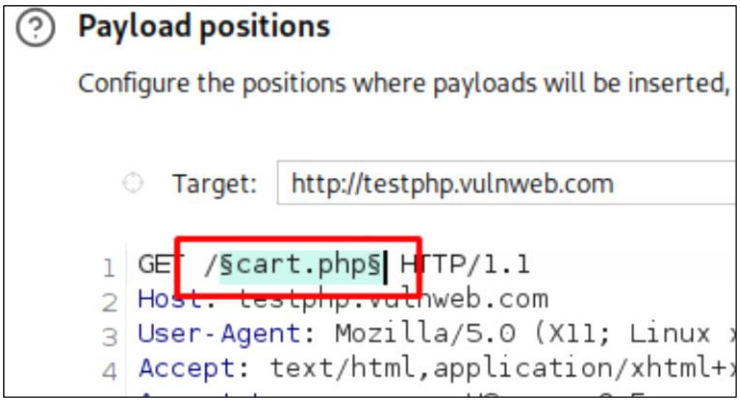


Рисунок 1.9 – Вибрали місце для корисного навантаження

Сканування тривало приблизно секунд 30, що показує нам дуже гарні результати збоку Burp Suite. В результатах розібратись також не важко, до кожного рядка корисного навантаження додається поле із статус-кодом та довжиною тіла запиту, яке повертає веб-сторінка. Ми також можемо переглядати вміст кожної знайденої сторінки та модифікувати словники (рисунок 1.10), чого немає у попередніх об’єктів дослідження.

Рисунок 1.10 – Додали суфікс до кожного рядку в словнику

Таблиця 1.1 — Порівняння інструментів для перебору директорій

Інструмент	Перевага	Недолік
GoBuster	швидкість роботи, можливість роботи з великими словниками, ефективність у простих сценаріях перебору директорій або файлів.	відсутність вбудованого графічного інтерфейсу. Також бракує підтримки глибокої рекурсивності та звітів у структурованому вигляді.
DirBuster	зрозумілий графічний інтерфейс, підтримка багатопоточності, гнучка настройка параметрів сканування. генерує звіти у різних форматах	відносно низька продуктивність. Також помітно, що інструмент може бути нестабільним при обробці великого обсягу запитів.
Burp Suite	вбудовані модулі, такі як Intruder, що дозволяють кастомізувати перебір на рівні запитів. Забезпечує глибокий аналіз HTTP/HTTPS-запитів і дозволяє переглядати вміст знайдених сторінок.	потребує значної кількості ручних дій для налаштування перебору. Немає автоматизованого аналізу рез-ів і обмеження в роботі з великими словниками через продуктивність проксі.

Проведений аналіз ручних і автоматизованих методі сканування директорій, а також огляд ключових інструментів таких як GoBuster, Dirbuster, Burp Suite дозволив оцінити їх ефективність, переваги та недоліки в контексті цього дипломного проекту. Ці всі утиліти покривають значну частину потреб під час сканування директорій. Однак їхні обмеження в автоматизації, аналізі та масштабуванні говорять про необхідність реалізації інтегрованих рішень, які б поєднували швидкість, адаптивність та простоту використання.

1.3 Основні вимоги до автоматизованих сканерів

Виходячи із висновків про огляд інструментів в минулому підрозділі, ми вже можемо зрозуміти, які основні вимоги повинен мати додаток автоматизованого сканування директорій хостингів. Серед основних проблем такого типу сканування є рекурсія, швидкість роботи та багатопоточність, словники та звітність про сканування. Рекурсія – це функція, яка викликає саму себе. Щоб зрозуміти краще, розберемо її на прикладі: нам потрібно знайти ключ в коробці але коли ми відкриваємо коробку, в ній ми бачимо ще одну, і так далі. З точки зору коду, ми можемо поступити за двома варіантами: ітеративний підхід [13] і рекурсивний підхід. В ітеративному підході ми використаємо цикл, для перегляду коробок, поки всі коробки не будуть переглянуті (рисунок 1.11).

Проаналізуємо два варіанта. В випадку із ітеративним підходом пошук триває доки є коробки в стеку, тобто навіть коли ми знайдемо ключ у коробці – ми все одно продовжимо перебирати залишок, знаючи що там нічого немає. Рекурсивний підхід викликає функцію кожен раз, якщо коробка не вміщає в собі ключа. Тоді ж якщо ключ в коробці знайдено, функція припиняє виклики, а повідомлення про ключ виводиться. Тепер уявімо, що коробка – це директорія, а ключ – це сторінка, яку повинен знайти сканер.

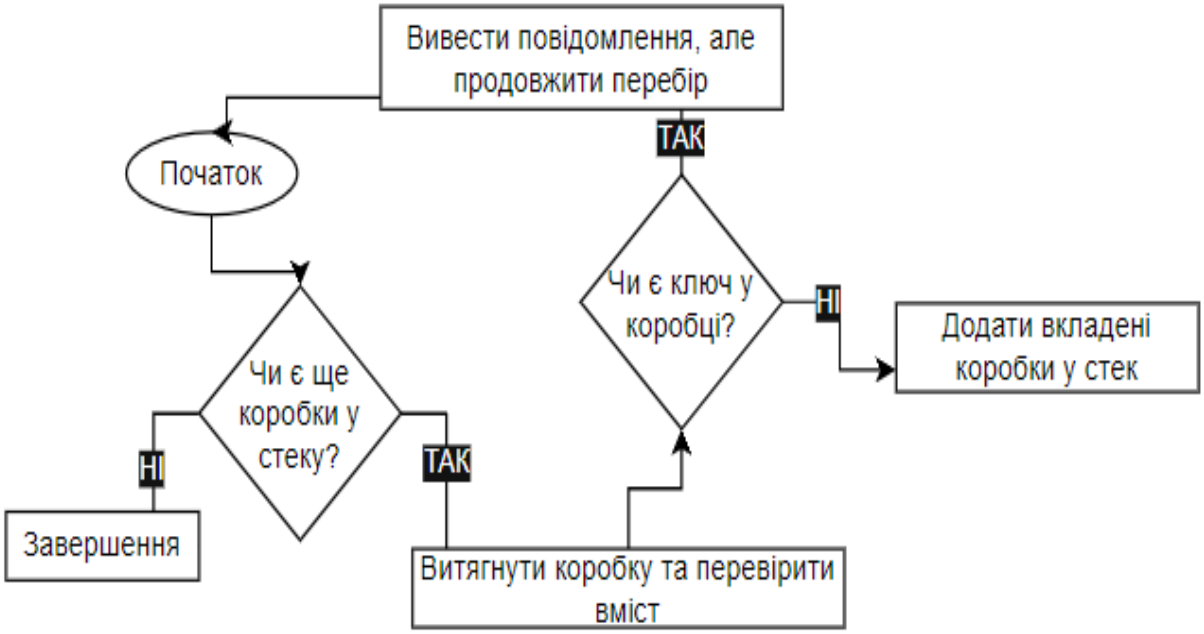


Рисунок 1.11 – Ітеративний підхід до задачі

В рекурсивному ж ми просто можемо викликати функцію в самій собі для обробки вкладених коробок (рисунок 1.12).

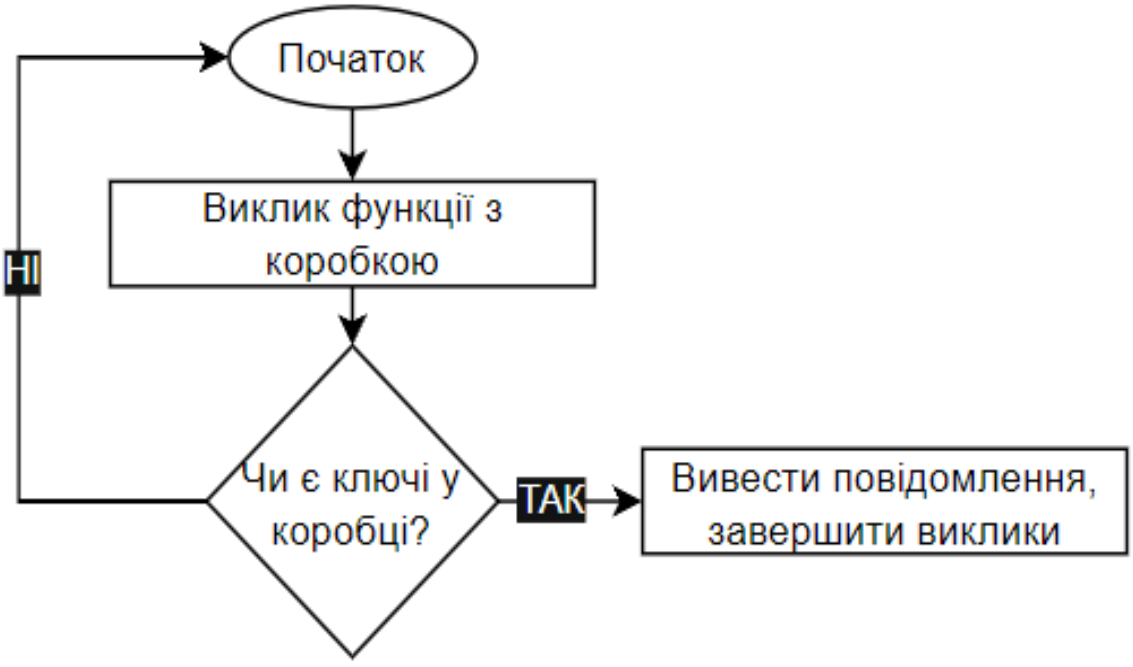


Рисунок 1.12 – Рекурсивний підхід до задачі

В такому разі рекурсія є підходящим варіантом, через, в першу чергу економію часу та відсутність ручної підтримки структури даних, як у випадку зі

стеком в ітеративному підході. Усе це автоматично обробляється стеком викликів.

Файлова система – це деревоподібна структура, тому рекурсія ідеально підходить для обходу таких структур, бо вона “імітує” вкладеність функціонально. Також, якщо ми захочемо додати функцію ручного вказання глибини пошуку (наприклад, глибина = 2 це пошук в глибину двох “/” в адресі), то це буде легко реалізувати через рекурсію. В ітеративному підході для цього потрібно було б створювати додаткову логіку. [14]

Так як сканування це процес, який залежить від наданого користувачем системі словника – а вони можуть бути різного об’єму, тоді повинна бути регуляція швидкості на рівні потоків. В .NET від Майкрософт ми можемо створювати код, який виконує декілька операцій одночасно. Операції, які можуть блокувати інші, зазвичай виконуються в окремих потоках, цей процес названий “багатопоточністю” або “вільною поточністю”. Багатопоточність – це програмний підхід, який дозволяє виконувати кілька потоків одночасно [15]. Потік – це послідовність інструкцій, яка виконується окремо але може використовувати спільні ресурси з іншими потоками [16]. Такий підхід використовують, щоб підняти ефективність використання апаратних ресурсів, підвищуючи продуктивність і швидкодію. Багатопоточність є трьох видів:

– крупнозерниста – коли потік виконується поки не блокується через подію з високою затримкою, наприклад, кеш-пропуск. Потім процесор перемикається на інший потік;

– тонкозерниста – потоки перемикаються кожного циклу процесора, зменшуючи залежність між інструкціями та забезпечуючи рівномірне використання ресурсів;

– одночасна – дозволяє процесору виконувати інструкції кількох потоків в той же час, максимально застосовуючи паралелізм у межах кількох потоків.

У програмуванні є апаратна підтримка керування потоками або програмна. У першому випадку використовуються конкретні регістри для кожного потоку, що надає можливість точного і швидкого перемикання між ними. У програмній

реалізації для керування потоками використовують бібліотеки і фреймворки, наприклад, такі як “System.Threading.Tasks” у C# або “std::thread” в мові C++.

На рисунку 1.13 показано концепцію багатопоточності у межах одного процесу, який виконується на одному процесорі. Виконується два потоки, які поділяють спільний адресний простір процесу (дані, пам’ять) але виконуються окремо один від одного. Хвилі, які йдуть від кожного потоку – це його активність. По ним зрозуміло, що паралелізм лише імітується, тобто CPU швидко перемикається між потоками, створюючи ілюзію їх одночасного виконання – це називається “мультиплексуванням”.

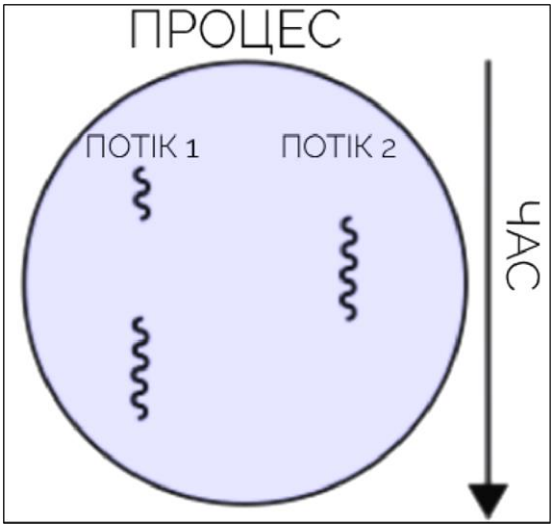


Рисунок 1.13 – Приклад процесу з двома потоками, що працює на одному процесорі

В своїй документації Microsoft пропонує декілька класів, які в мові C# взаємодіють із потоками – “System.Threading.Tasks.Parallel” [17], “System.Threading.Tasks.Task” [18], “Parallel LINQ (PLINQ)” [19]. Ці класи надають контроль над поведінкою потоків програми, щоб ми могли керувати потоками через них, а не самостійно. Потоки можна створювати як нові, тоді можна їх запускати. Для створення потрібно створити екземпляр класу “System.Threading.Thread” [20]. Для запуску використовується метод “Thread.Start”.

У нашому проекті, який взаємодіє із зовнішніми ресурсами через HTTP-запити, важливо мати механізм для зупинки виконання потоку. Для цього у .NET використовується спеціальний механізм – “ System.Threading.CancellationToken” [21] – він надає єдиний спосіб зупинки потоків у кооперативному режимі. Тобто, це інструмент, який надає можливість сигналізувати про потребу зупинити асинхронний процес. У випадку з нашим додатком, такий механізм необхідний для зупинки сканування директорій, якщо користувач натискає кнопку “стоп” або змінює ціль під час активного перебору.

Без CancellationToken ми не мали б контролю над процесом: додаток продовжував би надсилати запити навіть після того, як користувач вирішив змінити ціль чи завершити поточну сесію. Це б призвело до непотрібного навантаження на сервер, втрати ресурсів і поганого UX. Крім того, у складних випадках може виникнути ситуація “взаємного блокування” – коли система зависає через неочікувану паузу або повторюваний запит.

У контролері “FuzzingController.cs” є метод “StopFuzzing”, який при виклику гарантує, що якщо користувач натиснув кнопку “Стоп”, то процес одразу завершиться, не чекаючи завершення циклів або запитів. Все через те, що в цьому методі є “CancellationTokenSource”, який ініціюється під час запуску сканування, а у методі асинхронного сканування у сервісі “DirectoryScanner” токен передається у кожен запит.

Таке кооперативне скасування має свої переваги:

– система не “вбиває” потік насильно, а делікатно просить його зупинитись, код сам перевіряє токен і завершується;

– не виникає виняткових ситуацій через обрив HTTP-запиту;

– одночасно може скануватись 50+ директорій – миттєва зупинка критично знижу навантаження на мережу та процесор;

– також ми можемо логувати, скільки директорій встигли пройти до моменту зупинки.

Ще один важливий аспект – у нашому проекті сканування виконується у кількох потоках паралельно, що означає, що усі потоки мають отримати один і

той самий токен відміни, кожен потік періодично перевіряє токен і при методі “Cancel()” усі вони припиняють роботу максимально швидко. Наприклад, у сервісі “DirectoryScanner” усі запити запускаються через “Task.Run()” і кожен отримує той самий токен. Реалізація токенів відміни у нашій системі – це одна із ключових умов для стабільної роботи додатку, тому що вона дозволяє гарантувати що користувач має повний контроль над скануванням директорій та може зупиняти глибокі або довгі сканування без втрат. У поєднанні з логуванням, це забезпечує реактивний UX і високу надійність при пентестах, коли час реагування на зміну конфігурації критичний.

Якщо для відміни всього процесу використовуються токени відміни, то для призупинки потоку, іншими словами, введення його в паузу, використовується метод “Thread.Sleep” [22] – він може стати в нагоді, якщо, наприклад, ми захочемо робити затримку в секундах між потоками, щоб не викликати підозри у фаєрвола сайту. [23]

Таблиця 1.2 – Властивості потоків в C#

Властивість	Опис
1	2
IsAlive	Повертає true, якщо потік було запущено і він ще не завершився нормально або не був примусово зупинений.
IsBackground	Вказує, чи є потік фоновим. Фонові потоки не перешкоджають завершенню процесу, коли всі передні потоки завершено.
Name	Отримує або задає ім'я потоку. Найчастіше використовується для виявлення окремих потоків під час налагодження.

Кінець таблиці 1.2

1	2
Priority	Отримує або задає значення ThreadPriority, яке операційна система використовує для планування потоків.
ThreadState	Отримує значення ThreadState, яке містить поточний стан потоку.

Для того, щоб сервер веб-сайта розумів, яку версію сайту, в залежності від операційної системи видавати користувачеві – браузери використовують User-Agent [24]. Тобто цей агент просто ідентифікує браузер і його платформу для сервера, щоб забезпечити кращий досвід для кінцевого користувача. Наприклад, якщо сайт виявить User-agent Android – він поверне версію веб-сторінки відповідно для цієї ОС. Однак User-agent також є одним з атрибутів, які використовуються для браузерних відбитків. Далі розглянемо один з таких агентів на рисунку 1.14.

```
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/122.0.0.0 Safari/537.36
```

Рисунок 1.14 – Випадковий User-Agent

Рисунок 1.15 наглядно демонструє вміст любого UA. Перше, що ми бачимо в таких агентах – це жетон наслідування. Найбільш популярні UA мають з початку “Mozilla/5.0” – він зародився ще в 90-х роках, через так звані “війни браузерів” [25], тому що в ті часи лідером серед браузерів був Netscape і вебсайти створювали версії оптимізовані для цього популярного браузера, а запити з менш популярних могли бути відмовлені в обслуговувані, тоді майже всі браузери перейшли на такий жетон наслідування.

Частина заголовка присвячена операційній системі розкриває інформацію, відповідно про ОС користувача. Третій компонент – це механізм рендерингу, використовуваний браузером. Ці механізми відповідають за перетворювання HTML або CSS у візуальні та інтерактивні веб-сторінки. До речі, четверту частину заголовка UA також можна віднести до таких механізмів. П’ята та шоста частини агента відповідають за браузери та їх версії: актуальний та сумісний відповідно. Щоб приховати активність автоматизації в нашому додатку, використовуватимемо словники з такими агентами, з можливістю вибору агента для користувача.

Рисунок 1.15 – Анатомія User-Agent

Наступною вимогою до оптимального сканера директорій хостингів – це самі словники (wordlists). Зазвичай вони відіграють навіть важливішу роль аніж використовуваний інструмент. Іноді в скануванні великі списки, наприклад, такий як “rockyou.txt” можуть дати результат, а іноді потрібен більш специфічний список. Деякі директорії просто на просто можуть не існувати у наданих користувачем словниках. В Інтернеті існує колекція різних списків

названа “SecLists” – списки включають імена користувачів, паролі, пейлоади та багато іншого, але нас цікавлять URL або назви директорій чи сторінок. Задачею нашого проекту також являється надання кінцевому користувачеві різних типів словників в залежності від CMS, наприклад, якщо користувач буде сканувати сайт на Word Press, або в залежності від мови на якій він написаний – PHP або C# ASP.NET. Якщо сайт і його вміст дуже специфічний, можна надати користувачеві самому вносити зміни в поточні словники.

Словники поділяються на кілька категорій залежно від завдань, які стоять перед сканером. Загальні словники включатимуть поширені назви директорій та файлів, наприклад “admin”, “login”, “config”. Спеціалізовані словники міститимуть назви директорій та файлів, характерних для певних CMS або мов програмування, наприклад: “wp-admin”, “wp-content”, “uploads”, “plugins”, “themes”, “wp-config.php” для Word Press; “administrator”, “components”, “configuration.php”, “index.php” для Joomla; “bin”, “config”, “.env”, “composer.json” для фреймворку PHP Symfony тощо. [26]

В теорії, навіть сканер також може мати можливість автоматично визначати, який словник найбільш підходить для конкретного сайту. Це може бути реалізовано через визначення CMS, мови програмування за допомогою аналізу HTML-коду або HTTP-заголовків; потім система пропонує найбільш релевантні словники, які підходять в окремому випадку.

1.4 Цілі проєкту та постановка задачі

Аналізуючи описані вище інструменти, підходи та вимоги до сканування директорій, можна зробити висновок, що незважаючи на наявність великої кількості утиліт для таких задач, а саме виявлення прихованих ресурсів, жодна з них не надає одночасно гнучкості налаштувань, зручного інтерфейсу, автоматизації та розширюваності в межах одного програмного продукту. Вони або складні в налаштуванні (наприклад, Burp Suite), або обмежені в функціоналі

(як Gobuster), або нестабільні при великих навантаженнях (Dirbuster). Саме ці обмеження й стали вихідною точкою для формулювання ідеї цього дипломного проєкту.

У межах кваліфікаційної роботи передбачено створення власного веб-додатку, який дозволить не просто перебирати директорії згідно словника, а й аналізувати вміст знайдених сторінок, керувати конфігурацією сканування в реальному часі, а також отримувати результати у зручному виді. Проєкт отримав назву AbyssScan. Додаток в плані цільової аудиторії орієнтується на фахівців з кібербезпеки, етичних хакерів, тестувальників безпеки веб-додтків. За мету проекта взято розробку багатофункціонального веб-застосунку, який дозволятиме здійснити автоматизоване, контрольоване та адаптивне сканування директорій веб-ресурсів із можливістю гнучного налаштування параметрів, логування та подальшого аналізу знайденого HTML-вмісту.

На кінець цього розділу, опишемо основні задачі, які вирішуватимуться у межах проєкту: в основу поставленої цілі в першу чергу лягає реалізація багатопоточного словникового сканера директорій з підтримкою глибини (рекурсії) та затримками між запитами; застосування системи підміни User-Agent з можливістю вибору агента з попередньо сформованого набору; аналіз HTML-контента знайдених сторінок з метою виявлення форм для введення тексту та завантаження файлів; забезпечення можливості зупинки поточної сесії сканування; забезпечення простого і зрозумілого веб-інтерфейсу для користувача, без встановлення на сервер або хмару.

2 ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ СКАНЕРА ДЛЯ ПЕРЕБОРУ ДИРЕКТОРІЙ

2.1 Архітектура та структура проекту

В фундаменті будь якого цифрового продукту стоїть архітектура. Саме вона проектується однією з перших. Типів архітектур є багато і всі вони підходять окремо під певні задачі. Для того, щоб зрозуміти, яка з них найближче підійде до задачі цього проекту, потрібно ознайомитись з деякими видами архітектур - взагалі їх розділяють на три основні види: моноліт, мікросервіси та серверлес. [27]

Один з перших видів – це моноліт [28] (до слівно перекладається як “суцільна маса/брила ”). Це рішення (рисунок 2.1) в якому всі компоненти системи містяться в одному блоці з єдиною точкою розгортання. Переваги: швидке і просте впровадження, зручне налагодження. Недоліки: щойно доводиться вносити якісь зміни – треба міняти всю систему; один збій може зупинити всю програму.

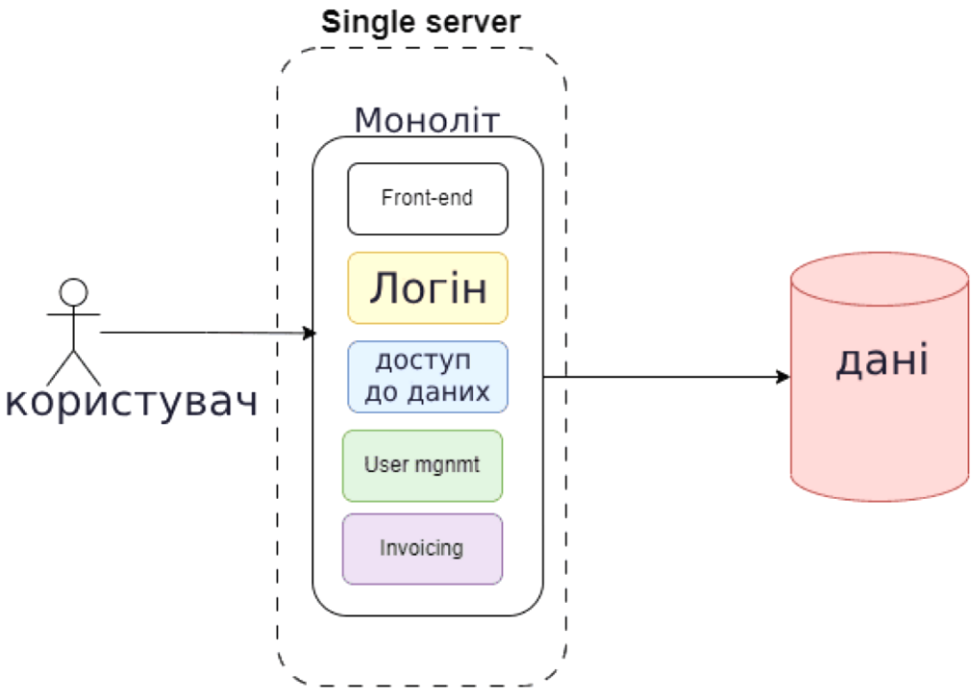


Рисунок 2.1 – Схема архітектури “моноліт”

Інша, навпаки, сьогодні досить популярна архітектура – це мікросервісна [29] (рисунок 2.2). Це підхід, коли систему розділено на невеликі автономні сервіси, що взаємодіють через повідомлення. Серед переваг: назалежне оновлення й розвиток кожної частини, можливість реалізації на різних технологіях. Також ця архітектура має свої недоліки: складність розробки та підтримки, зокрема потреба організувати взаємодію багатьох підсистем, а також значні вимоги до командної роботи.

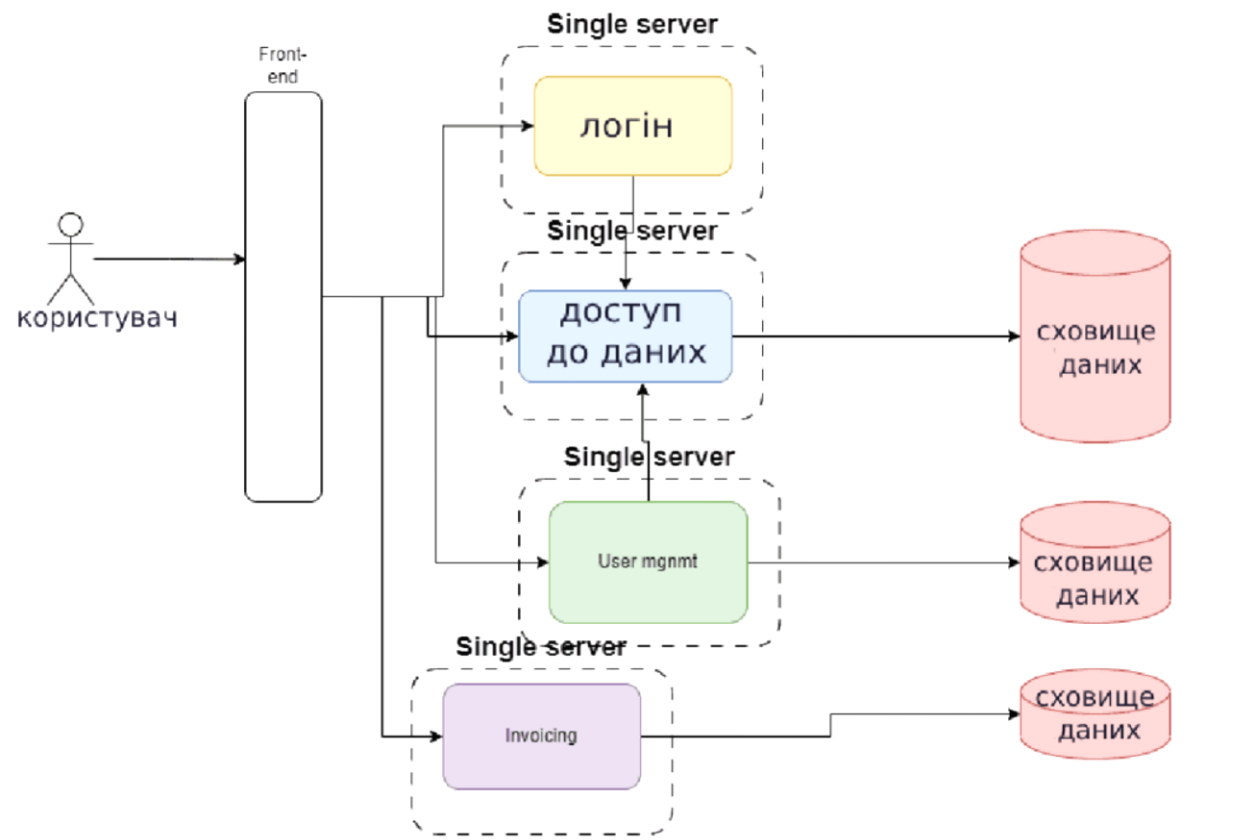


Рисунок 2.2 – Схема архітектури “мікросервіс”

Третій вид архітектури – серверлес [30]. Модель, при якій основні процеси розгортаються й масштабуються на хмарних платформах, а розробник зосереджується майже виключно на коді і логіці. Серед плюсів: висока гнучкість, відсутність необхідності вручну керувати інфраструктурою. Недоліки: складність налагодження та ризик “каскадних” помилок через взаємозалежність компонентів.

Рисунок 2.3 – Схема архітектури “serverless” на прикладі онлайн-магазину

Виходячи з завдання даного цьому додатку, яке детальніше обговорене в підрозділі 1.3, а саме: багатопоточність, налаштування глибини пошуку (рекурсії), затримка між запитами тощо – було вибрано багатошарову архітектуру для проекту, також відома як “onion architecture” (цибуле-подібна). Вона забезпечує більш підтримувані додатки, оскільки акцентує увагу на поділі відповідальностей у системі. Але перед тим як розглянути її детально, важливо визначити контекст її застосування. Ця архітектура не підходить для малих вебсайтів, вона ідеальна для додатків зі складною поведінкою. Два головних принципи такої архітектури:

– використання інтерфейсів для визначення контрактів поведінки;

– відокремлення інфраструктурної логіки від основної бізнес-логіки.

Цибулева архітектура була свого часу замінником для традиційної шарової архітектури (рисунок 2.4).

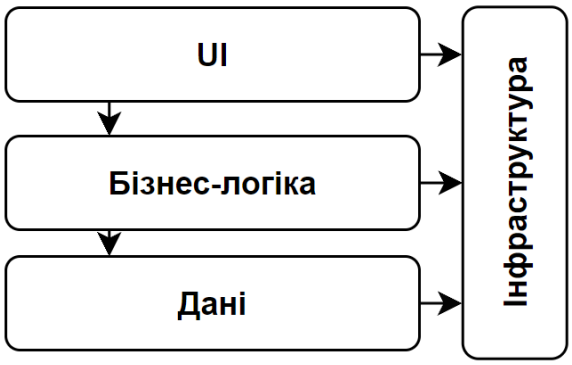


Рисунок 2.4 – Діаграма традиційної шарової архітектури

В останній кожен шар залежить від нижнього шару, і кожен шар зазвичай залежить від спільної інфраструктури та служб. Найбільший недолік традиційної архітектури – це надмірна зв'язаність серед шарів, тобто тут ми можемо піддаватись каскадним змінам, в випадку якщо захочемо змінити нижні шари (наприклад, доступ до даних). Щоб зменшити проблему зв'язаності, запропонували новий архітектурний підхід – цибулева архітектура, її зображено на рисунку 2.5.

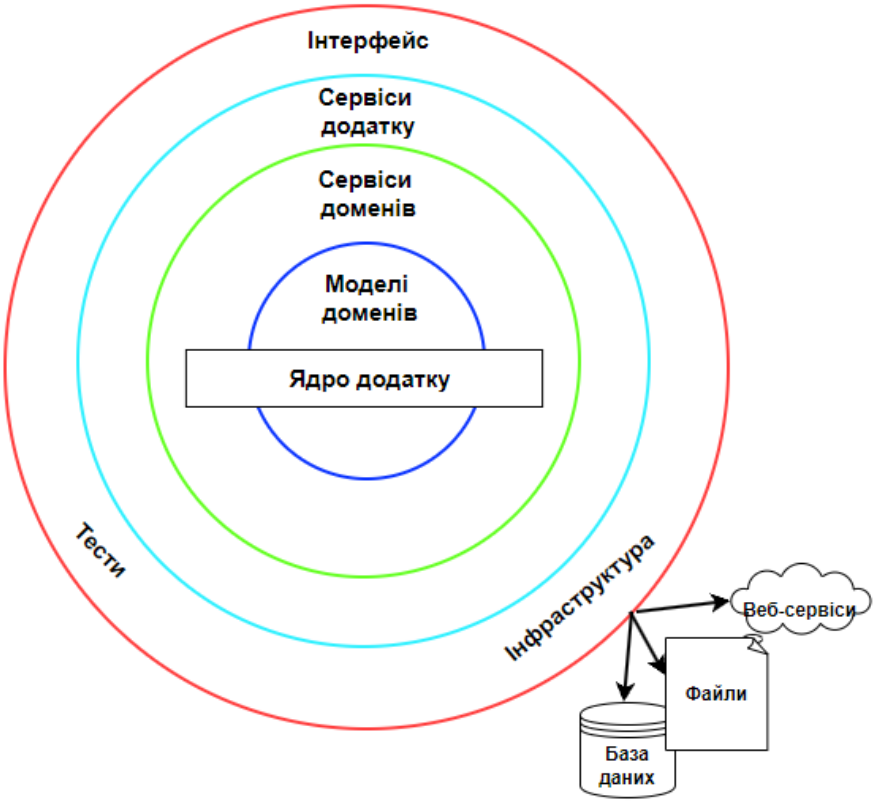


Рисунок 2.5 – Приклад цибулевої архітектури

Головна ідея – контроль залежностей. Всі залежності мають спрямовуватись до центру, а код не може залежати від шарів, які розташовані далі ядра. Стає зрозуміло, що ця архітектура використовує принцип "Dependency inversion principle, DIP" – принцип інверсії залежностей. Вона орієнтована на об'єктно-орієнтоване програмування (ООП), ставлячи об'єкти в центр уваги. [31]

У центрі знаходиться доменна модель, яка представляє стан і поведінку бізнес-логіки, а навколо неї розташовуються інші шари поведінки, котрі

розширюють функціональність. З цього і виходить аутентична назва – цибулева архітектура. Шари, які мають потенційну властивість змінюватись, це інтерфейс користувача, інфраструктура і тести – вони повинні бути відокремлені від ядра. Наприклад, клас який реалізовує інтерфейс сервісу з пошуком директорій знаходитемиться у зовнішньому шарі. З огляду на те, що наш додаток має:

– свій набір доменної логіки (перебір директорій, пошук форм, робота з User-Agent, словники і т. д.);

– потребу розділити контролери і конкретні реалізації сервісів;

– можливість зміни в майбутньому або словників, конфігурації або деталі реалізації.

Нам важливо аби внутрішня логіка не залежала від деталей реалізації інфраструктури. Тож цибулева архітектура дає змогу чітко розділювати ці аспекти. Таким чином, умовно можна поділити проект на декілька шарів (рисунок 2.6). Завдяки такому підходу, впершу чергу залежності спрямовані всередину, тобто контролери (зовнішній шар) залежать від сервісів, сервіси залежать від інтерфейсів і моделей у ядрі, а самі моделі не залежать від інфраструктури чи користувацького інтерфейсу; також можна легко підміняти будь яку реалізацію, наприклад, можна замінити клас парсера або інший, не зачіпаючи внутрішню логіку сервісу, який відповідальний за сканування форм.

Відповідаючи на питання чому саме так, можна виділити три основні фактори. Наприклад, якщо ми захочемо змінити метод зчитування словників і замість текстових файлів використовувати базу даних – достатньо реалізувати інший інтерфейс “IDictionaryProvider”. Код у сервісі “DirectoryScanner” не зміниться, оскільки він знає лиш про інтерфейс. Другий фактор – зрозуміла підтримка. Інтуїтивно зрозуміло де шукати логіку перебору (сервіси), де лежать контракти (ядро), де знаходиться логіка зчитування словників (інфраструктура), а контролери лише координують ці виклики. Останній фактор – гнучкість і масштабування: якщо систему потрібно буде розширити ми можемо створювати нові сервіси, які використовують уже наявні інтерфейси з ядра. Це не ламає те що вже працює.

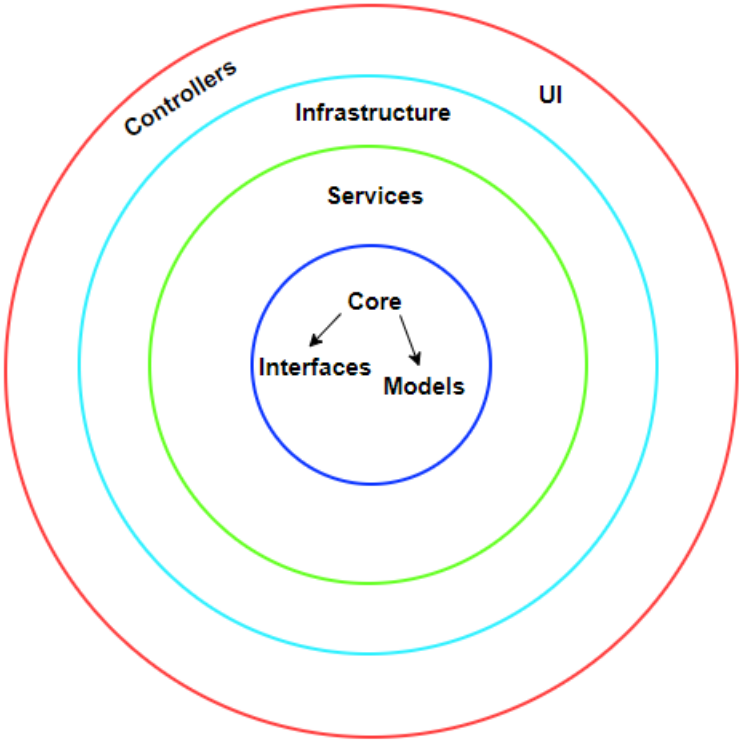


Рисунок 2.6 – Цибулева архітектура нашого додатку

Отож, такий вид архітектури повністю вписується під задачі цього проекту. Він дає нам можливість тримати бізнес-логіку незалежною від деталей реалізації (як-от методи парсингу, файли словників або конфігурація). Завдяки цьому код простіше підтримувати, тестувати і розвивати.

2.2 Використані технології і середовища

Вибір технологій та середовищ розробки, це те, про що варто задуматись розробнику після того, коли додаток вже спроектований.

Все починається з мови програмування та фреймворку: у нас це C# та .NET версії 8.0. Додаток потребує обробки великої кількості запитів для сканування кількох веб-директорій одночасно (багатопоточність, рекурсивний обхід та паралельний перебір). У .NET є високорівневі конструкції, такі як “async/await”, “Paralell.ForEachAsync” або бібліотека “Task” та оптимізований “Thread Pool”,

що спрощує написання ефективного багатопоточного коду. За даними офіційної [32] документації Майкрософт, нові версії .NET 7 та 8 містять покращення в асинхронній моделі виконання, що позитивно впливає на продуктивність під час обробки великої кількості однотипних операцій введення-виведення.

Наш додаток має веб-інтерфейс, тому логічно, що ми обрали саме фреймворк ASP.NET для цього, він достатньо легкий і в той же час гнучкий, має вбудований мінімалістичний сервер Kestrel, який спрощує початкове налаштування. Також ASP.NET дозволяє зручно налаштовувати шари залежностей завдяки вбудованому IoC контейнеру (Dependency injection). Згідно із результатами дослідження JetBrains [33], саме наявність великої кількості якісних пакетів через NuGet є одним із факторів, який стимулює вибір .NET серед розробників. За допомогою цієї зручної інтеграції з NuGet-бібліотеками у проекті використовується кілька сторонніх бібліотек, таких як AngleSharp для парсингу HTML, Net.Http для HTTP-запитів, Microsoft.Logging для логування тощо.

Варто більше описати кожну з цих використаних бібліотек. Наприклад, бібліотека AngleSharp займається обробкою і вилученням даних із HTML-документів. Вона також дозволяє працювати з таблицями, атрибутами, внутрішнім текстом елементів та JS-контентом. Нижче, на рисунку 2.7, графічно пояснюється алгоритм роботи AngleSharp. [34]

Якщо описати рисунок 2.7 на словах, то можна розбити процес на п’ять кроків. Перед початком парсингу потрібно отримати HTML-код сторінки, для цього використовують HTTP-запит через бібліотеку HttpClient, потім формується DOM-структура – на цьому етапі отриманий HTML-код необхідно перетворити на об’єктну модель документа (DOM), щоб можна було взаємодіяти з його структурою; далі AngleSharp дозволяє виконувати пошук елементів у цій структурі: визначаються цільові елементи, які потрібно знайти, потім виконується вибір елементів за допомогою методів пошуку, а в кінці виконується перевірка – якщо елемент знайдено, витягується його текстовий вміст або атрибути.

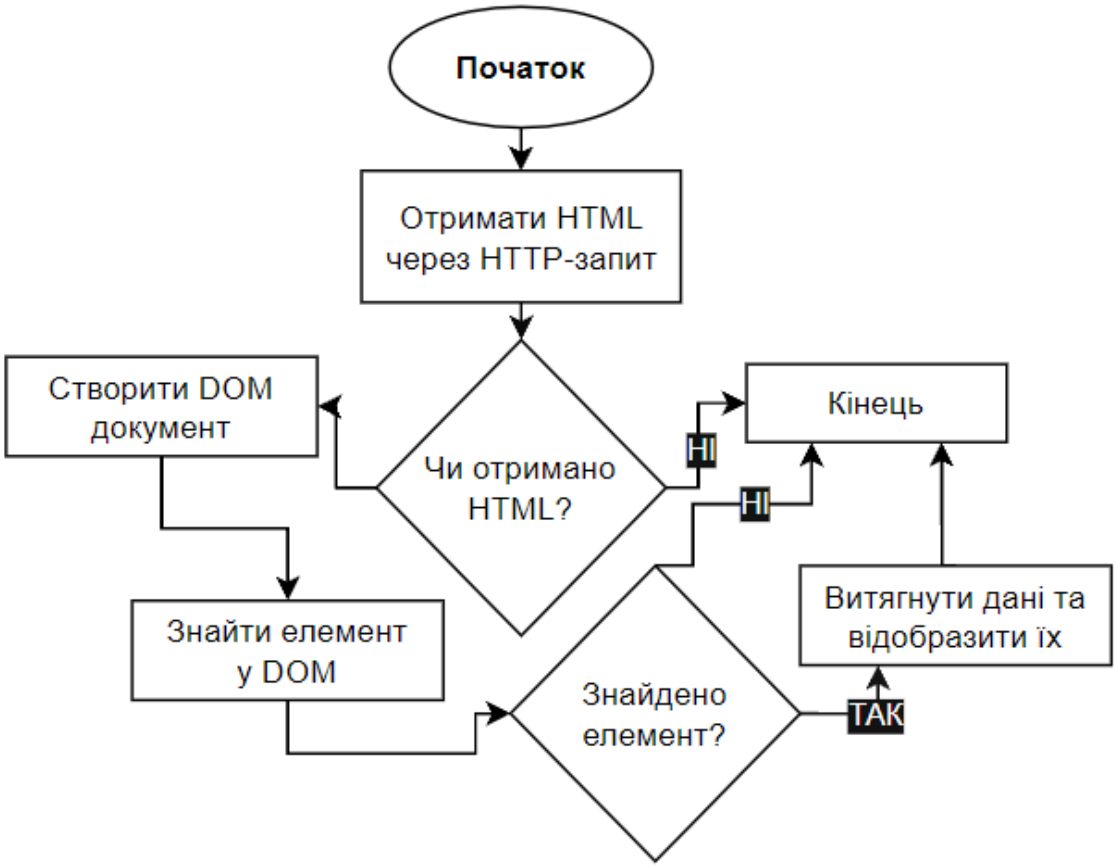


Рисунок 2.7 – Алгоритм роботи бібліотеки AngleSharp

Четвертим етапом є аналіз отриманої інформації: формується структурований результат, виконується очищення тексту від зайвих пробілів або спеціальних символів і за необхідності проводиться додаткова обробка. Останній, п’ятий етап передбачає вивід або збереження результатів.

Бібліотека, яка виконує запит до веб-ресурсу називається HttpClient [35], на її основі реалізований клас HttpRequester. Подібно до опису бібліотеки вище, також можна виділити п’ять основних кроків. Кожен крок відповідає конкретній задачі під час відправлення запиту та отримання відповіді:

– перед тим як виконати запит, цей клас потрібно ініціалізувати та налаштувати бажані параметри: можна встановити таймаут (наприклад, 30 секунд), після якого запит вважається невдалим якщо сервер не відповів; заголовки які будемо відсилати на сервер, в нашому випадку це User-Agent;

– далі йде етап формування запиту, формується HttpRequestMessage – об’єкт який містить: цільову адресу, метод та тіло запиту, в нашому випадку підставляється слово з словника для перебору;

– наступна дія це отримання та аналіз відповіді. Коли сервер повернув відповідь, клас HttpClient формує HttpResponseMessage, який складається з статус-коду відповіді, заголовку та тіла повідомлення;

– отримані дані аналізуються та надсилаються на подальшу обробку в нашу бізнес-логіку.

Зрештою, ця бібліотека використовує концепцію “налаштуй – сформуй – відправ – отримай – оброби”, і такий цикл повторюється до кожної директорії чи URL, які ми скануємо. Завдяки асинхронній природі викликів та можливості тонкого налаштування ми можемо одночасно сканувати багато ресурсів або ж працювати в ощадливому режимі з однією-двома паралельними нитками.

Для виводу інформації у консоль прямо на фронтенді ми використали пакет “Microsoft.Extensions.Logging”. За допомогою техніки логування ми фіксуємо попередження, помилки та інші важливі дані під час роботи програми. Завдяки логам можна контролювати поведінку застосунку, відстежувати проблеми і краще розуміти процес його виконання. Плюс цього пакету у тому, що є можливість додавання різних рівней важливості для повідомлення. Усього є шість рівнів логів, які допомагають класифікувати та впорядковувати повідомлення:

– найдетальніший рівень – Trace, часто його використовують для глибокого діагностування;

– debug відповідає за діагностичну інформацію, корисну під час розробки але зазвичай не відображається в продакшні;

– інформаційний рівень стандартний, повідомлення описують нормальний перебіг виконання;

– warning вказує на нетипову ситуацію, яка потенційно може призвести до проблеми;

– error означає помилку, яка вже вплинула або вплине на роботу застосунку;

– critical найсерйозніший рівень, він повідомляє коли стан аварійний або є критична несправність.

Використовуючи “Microsoft.Extensions.Logging” ми можемо виводити для користувача дії системи прямо на фронтенд, у нашому випадку логування буде гарним інструментом, щоб виводити інформацію при скануванні директорій і для підказок користувачеві. [36]

2.3 Опис функціональних вимог

Наша система має певні функції та механізми для коректної та продуктивної роботи користувача з додатком. Серед них є, наприклад, такі як: робота зі словниками для перебору сторінок, можливість зміни User-Agent при надсиланні запиту, логіка перебору директорій, а також аналіз форм введення та завантаження файлів. Усі ці частини тісно взаємодіють між собою утворюючи єдиний великий механізм.

Для роботи зі словниками був створений клас на рівні сервісів, який називається “FileDictionaryProvider”. Самі словники містять зарання сформований перелік потенційних назв директорій чи файлів. Алгоритм цього сервісу дозволяє швидко перемикатись між різними словниками для різних сценаріїв чи технологій. З початку зчитується конфігурація з конфігураційного файлу config.json, там налаштовані шляхи до словників - кожен словник має унікальне ім’я та свій фізичний шлях. Коли користувач на стороні інтерфейса вибирає потрібний словник, метод “SwitchDictionary” оновлює внутрішній шлях класу на відповідний файл. Паралельно метод “ValidateFileExists” піклується про те, щоб файл точно існував. Читає рядки слів у файлі метод “GetEntriesAsync”, він обробляє файл фільтруючи порожні рядки та зберігає їх у вигляді колекції рядків.

Для того, щоб сервер сприймав запити сканера як реальні запити веббразуера або інших клієнтських застосунків в додатку використовується

функція підміни User-agent. Це допомагає уникнути деякі базові блокування. За це відповідає сервіс “UserAgentProvider”. Він має досить простий алгоритм (рисунок 2.8), який графічно можна поділити на декілька кроків.

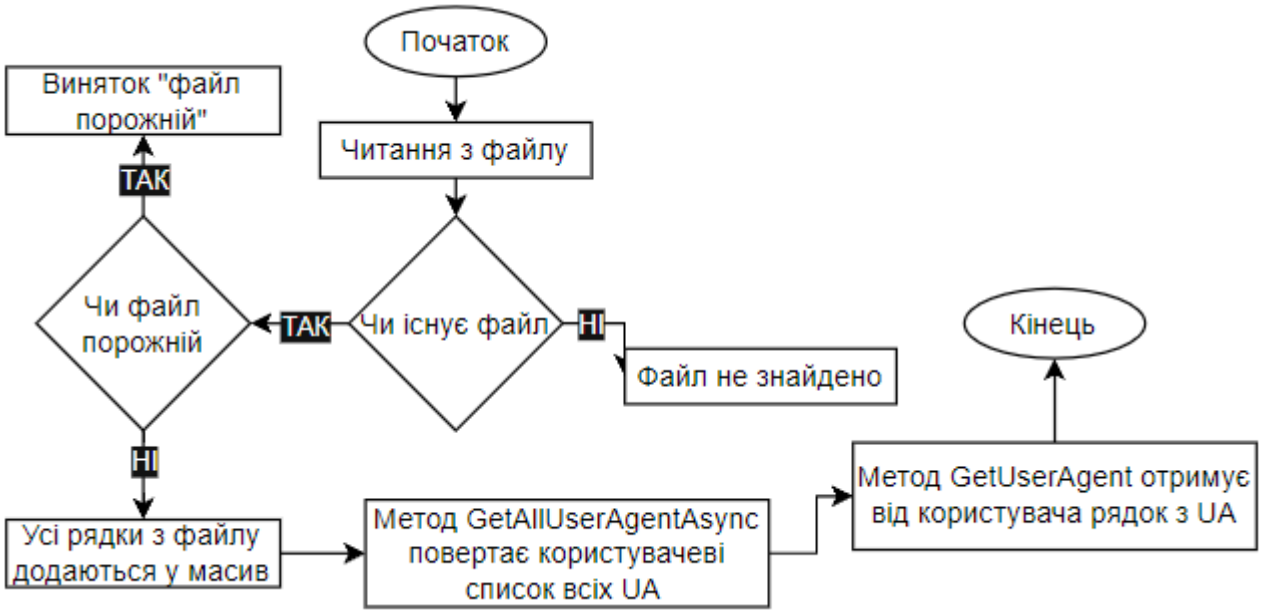


Рисунок 2.8 – Алгоритм роботи сервіса, відповідального за отримання User-agent

Реалізація цієї функції дає змогу імітувати різні браузери або інструменти і як наслідок уникати можливого блокування, а також коректно отримувати HTML-версії сторінок, які сервер віддає певним типам клієнтів.

Основа нашого додатку – це логіка перебору директорій, його “рушійна” сила, яка, використовуючи словники, формує список потенційних URL, надсилає запити й аналізує відповіді. Її програмна частина поєднує в собі декілька елементів, таких як багатопоточність, управління глибиною рекурсії, контроль дублікатів, гнучкий user-agent та словниковий підхід. Завдяки цим всім аспектам сервіс може досить швидко й гнучко знаходити потенційно приховані директорії на сервері та формувати повну картину URL-адрес, які повертають позитивний статус. Але рекомендую розглянути цикл життя цього механізма більш детально. Сервіс “DirectoryScanner” реалізує інтерфейс “IDirectoryScanner”, в нього є контракти з двома методами: перший вміє запускати сканування з різною сигнатурою, а другий повертає вже знайдені директорії. Загальна логіка

перебору влаштована так, що на вхід подається базовий URL (тобто той, який вставив користувач), вказуються глибина (рекурсія) перебору і словник шляхів, потім виконується багатопоточна перевірка чи існує файл/директорія на цьому шляху (за відповідним статус-кодом) і якщо код успішний – шлях вважається знайденим. На стороні контролера фрагмент виклику сервісу виглядає так, як на рисунку 2.9.

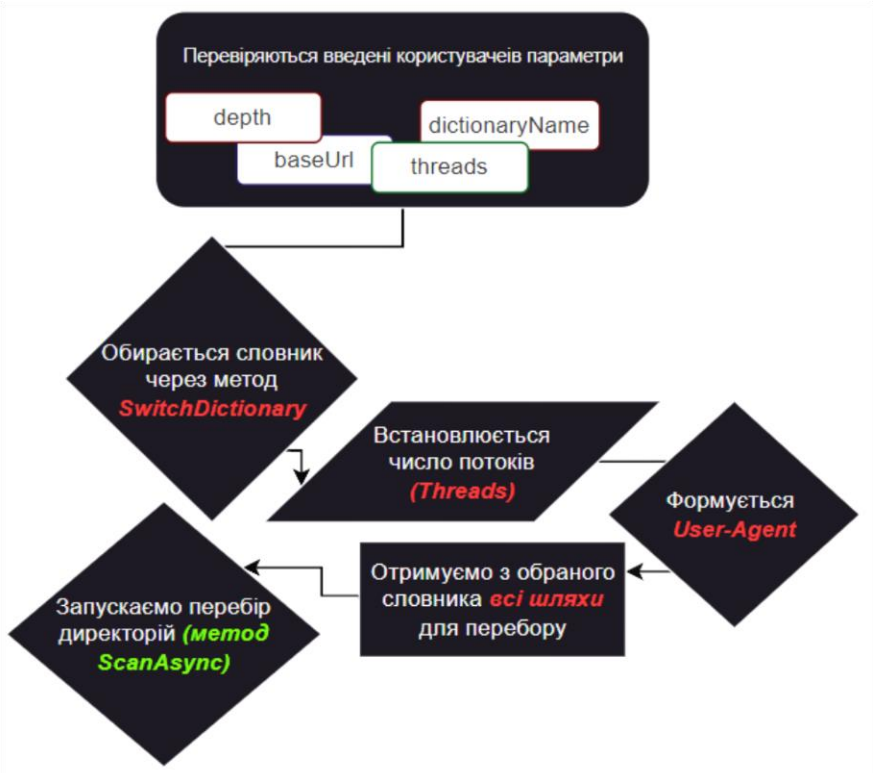


Рисунок 2.9 – Логіка на стороні контролера

Одне із завдань пентестера – виявити всі місця введення: поля для введення тексту, завантаження файлів та інші типи форм, аби перевірити чи існують у цих точках можливі вразливості (наприклад XSS, SQL-ін’єкції або небезпечне завантаження файлів). Для такої мети було створено сервіс “FormScanner” а також відповідний метод у контролері “ActionsController” для централізованого керування логікою:

– завантаження HTML зі сторінки, яку вибрав користувач;

– парсинг отриманого HTML для виявлення <form>;

– аналіз структури та атрибутів (input, textarea, select, file тощо);

– повернення результатів у форматі, що зручно інтерпретується під час пентесту.

Основний метод у цьому сервісі – це “FindFormAsync”, з назви зрозуміло що він асинхроний і займається безпосередньо пошуком форм. Його функціонал можна розділити на кілька кроків, які графічно показані на рисунку 2.10:

Рисунок 2.10 – Графічне представлення функціоналу методу

З першим щаблем піраміди все ясно – якщо отриманий HTML-контент порожній, викликається виняток та додається лог із повідомленням про помилку. Другий крок – парсинг, і цим займається парсер, який був обговорений вище. Потім в “розібраному” HTML коді шукаються теги “<form>” у документі, для кожного такого елемента зберігаються ключові атрибути – це “action” та “method”. Четвертий щабель піраміди аналізує атрибути які стосуються полів введення, такі як “input”, “textarea”, “select” і для кожного зчитується інформація про нього – від назви до типу. Перед останнім кроком є аналогічна функція але по відношенню до полів завантаження файлів. Останнім кроком тут є

формування результатів – на кожну знайдену форму створюється об’єкт “FormFound”, де є: повна адреса, куди надсилаються дані; назва метода (GET або POST тощо); список полів введення і список файлових полів.

Отже, такий сервіс надає ще більше автоматизованості для кінцевого користувача забираючи нудний пошук форм на кожній знайденій сторінці у свої обов’язки – він миттєво виводить результати у зведеному списку. Користувачеві просто достатньо вказати певний перелік адрес, а система сама завантажить HTML (багатопоточно), знайде усі “<form>” і акуратно винесе дані про кожне поле.

3 ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ ДОДАТКУ

3.1 Опис інтерфейсу користувача

Крім програмної частини веб-додатку, ще є інтерфейс – це те, що бачить користувач, з чим спілкується при взаємодії із додатком. Інтерфейс – це буквально проксі між користувачем та основною логікою системи. Він повинен бути легким, зрозумілим та інформативним в одночас. Для перевірки цих всіх потреб існують “юзабіліті” евристики Якоба Нільсона [37] – це правила, які потрібно знати дизайнерам перед тим як проектувати інтерфейс. Вони допомагають визначити потенційні незручності для користувача і запобігти їм ще на стадії розробки.

Основна евристика – видимість стану системи. Уявімо, що ви граєте у гру, отримуєте певний прогрес або рівень і через деякий час виходите з гри. Потім ви повертаєтесь але гра не дає знати вам, як користувачеві, на якому ви зараз рівні, який у вас поточний прогрес. Це дизорієнтує і викликає когнітивне навантаження, тому що користувач може не запам’ятовувати рівень у іграх. Або ж ви завантажуєте файл з інтернету, але при цьому функції перегляду об’єму який вже завантажився або час до його закінчення відсутні. Щоб позбавитись цього дизкомфорту існує евристика видимості стану системи.

В позитивному випадку, якщо ця евристика дотримана в інтерфейсі додатку, користувач отримує більше інформації і планує свої майбутні дії з упевненістю. В моєму веб-додатку AbyssScan цього правила було дотримано на різних рівнях. На рисунку 3.1 видно, що впершу чергу користувач орієнтується де він і які на сторінці існують дії, наприклад, ми бачимо зверху текст “Налаштування перебору директорій” – крім того, що це інтуїтивно підказує користувачеві які дії можна проводити йому на поточній сторінці, це ще слугує назвою сторінки.

В цілому, наш додаток проводить багато процесів “під капотом”, ці процеси користувач не бачить. Серед них безпосередньо проведення сканування директорій і сторінок хостингів – основна задача цього проекту. Але за

допомогою логування ми не тільки слідкуємо за діями системи але ще й даємо знати що відбувається з його скануванням користувачеві. На рисунку 3.1 бачимо, що логування відобразило інформацію про стан деяких URL – “досягнуто максимальну глибину для URL”, а також по завершенню сканування дається інформація того, скільки директорій знайдено – “Перебір директорій закінчений. Знайдено 5 директорій”.

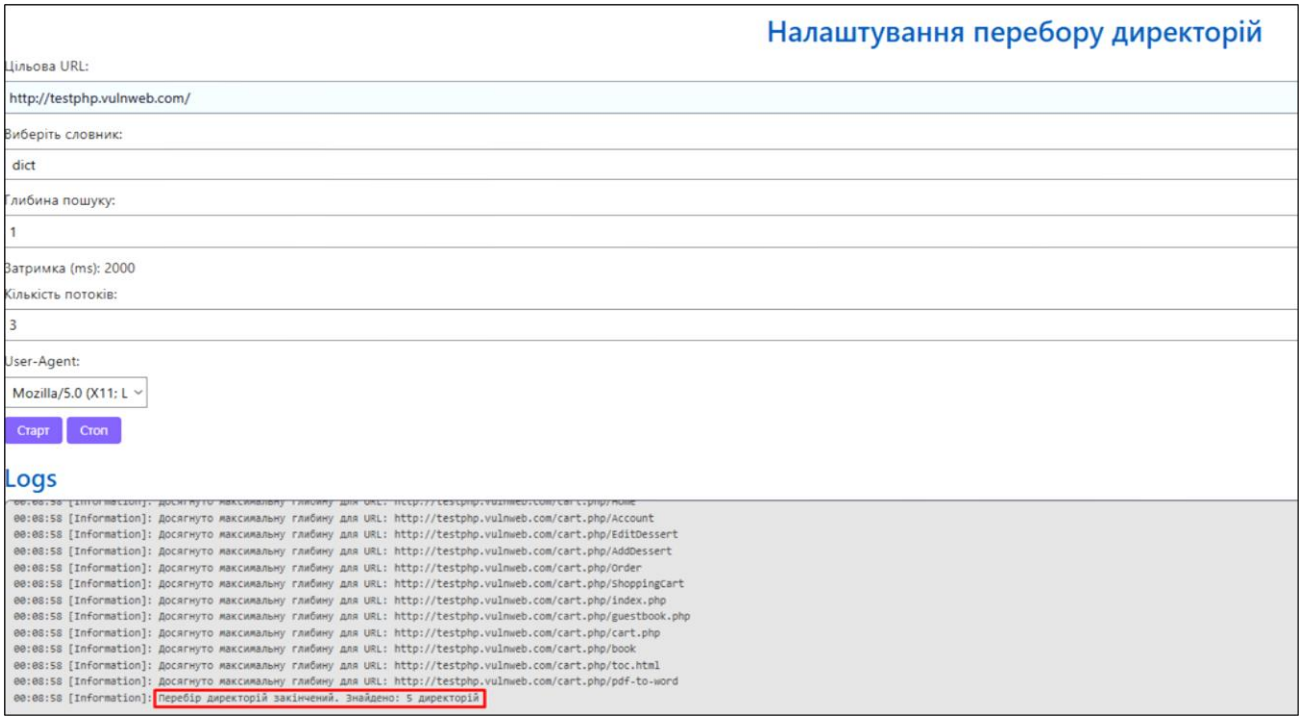


Рисунок 3.1 – Сторінка сканування і сам процес

Також коли ми запускаємо сканування (рисунок 3.2), логування надає загальну інформацію про характеристику поточного сканування: про затримку, кількість потоків, глибину та вибраний словник і цільовий хостинг.

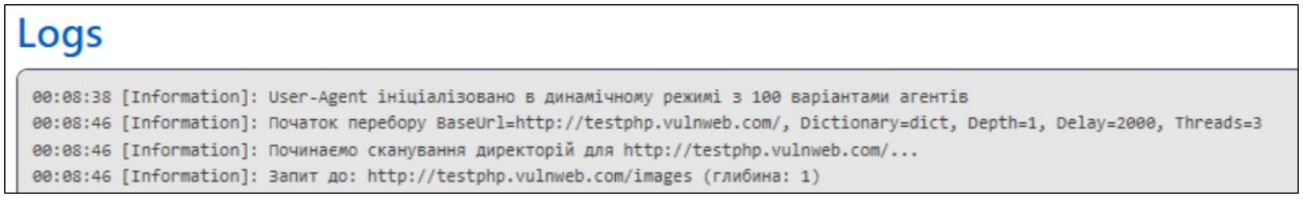


Рисунок 3.2 – Інформація в логах при запуску сканування

Кольора грають не меньш важливу роль у цій евристиці – вони допомагають швидше орієнтуватись в інформації користувачеві, як показано на рисунку 3.3. Тобто, отримані статус-коди діляться на позитивні та негативні, серед позитивних це перенаправлення, успішний запит, доступ заборонено і тому подібні, серед негативних усі інші. Повернута інформація включає в себе назву повного посилання, повернутий статус-код та дію (пропускаємо або додано).

Рисунок 3.3 – Приклад інформації яка повертається під час сканування

Сторінка, де користувач після сканування може проводити точкове сканування (парсинг) вже знайдених директорій також включає в себе правило про видимість стану системи. Рисунок 3.4 нижче наглядно це демонструє, ми бачимо зверху назву сторінки, яка інтуїтивно підводить користувача до можливих дій які проводяться на цій сторінці, а також кожен блок завчасно сповіщає про тип даних, які в ньому містяться – це допомагає кінцевому користувачеві.

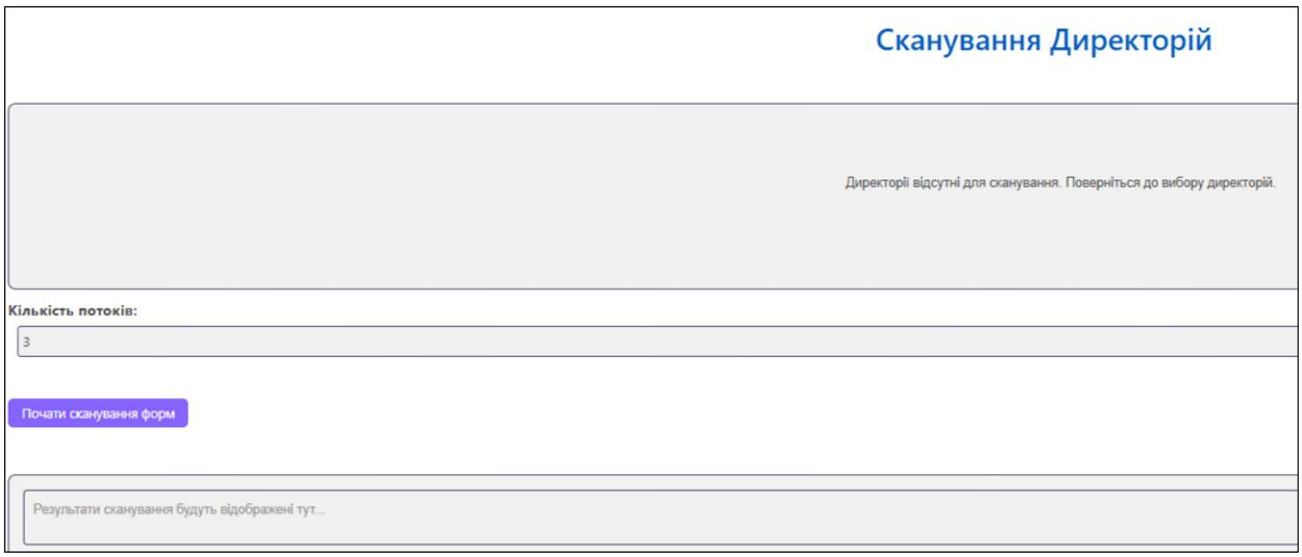


Рисунок 3.4 – Сторінка парсингу окремих директорій

Другий принцип евристик Нільсона, полягає в тому щоб користувач контролював деякі процеси системи, тобто йому можна змінювати свої дії або відмінити щось. Це правило так і називається – контроль і свобода користувача.

Суть евристики: іноді користувачі можуть робити помилкові дії або у них з'являється бажання змінити свій попердній крок. Дизайнери інтерфейсу додатку повині це врахувати і надавати простий механізм "відміни" або повернення в минулий стан. В контексті нашого проекту ця евристика дотримана за допомогою функції "Stop fuzzing" (рисунок 3.5) – за допомогою неї користувач може швидко перервати процес перебору директорій по якійсь причині: достатньо результатів, неправильний словник або неправильна URL-адреса.

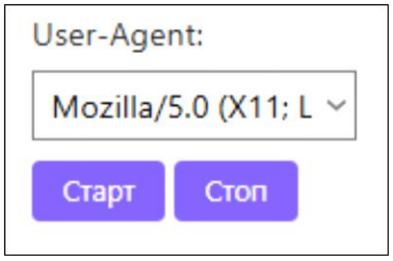


Рисунок 3.5 – Можливість відмінити процес через кнопку "стоп"

Якщо говорити про можливість змінити свій попередній крок, в контексті проекту AbyssScan можна привести в приклад бокову панель, котра вміщає в собі переходи на інші існуючі сторінки у веб-додатку (рисунок 3.6). Таким чином користувач може легко переміщатись у проекті:

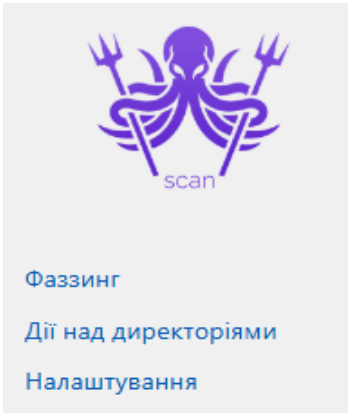


Рисунок 3.6 – Бокова панель

Третє правило евристик Нільсона – послідовність і стандарти [38]. Ця евристика вимагає у межах одного додатку дотримуватись єдиної термінології, розташуваня елементів і стилю. Для користувача це полегшує адаптованість до системи й зменшує потенційну плутанину. Приклад дотриманості цій евристиці є на рисунку 3.5 – кнопки “стоп” і “старт” мають один стиль, тож користувач одразу бачить “дію-пару”. Також можна відмітити однакову структуру і принцип логування. Евристика “Error prevention” – запобігання помилкам також здійснюється у проекті, через валідацію введення даних: якщо користувач вводить цільовий URL без “https://” або взагалі пустий рядок, інтерфейс відреагує і виведене три різних повідомлення на кожну окрему ситуацію через логування (рисунок 3.7). Плюс у тому, що в методі який реалізовує цей функціонал застосовується слово “return” і користувач ні за що не продовжить свої “неправильні” дії, а буде отримувати повідомлення до тих пір, доки триває помилка.

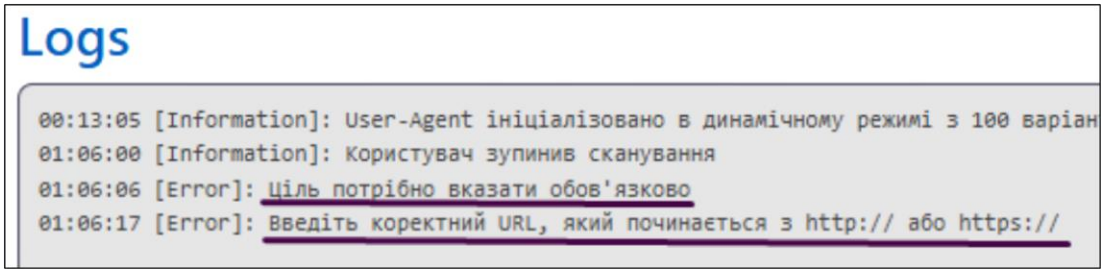


Рисунок 3.7 – Види попереджувальних повідомлень

Ще один не менш важливий приклад – це вибір словника через випадаючий список (рисунок 3.8).

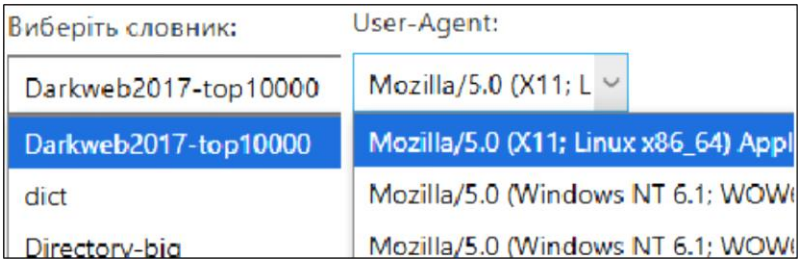


Рисунок 3.8 – Випадаючі списки для вибору словників та UA

Саме це унеможливлює помилкові звернення до невірних файлів, а разом із цим друкарські помилки, які може здійснити користувач.Сюди ж ситуація із вибором User-agent, це також реалізовано через випадаючий список.

Отже, у цьому веб-додатку реалізовані деякі евристики і завчасно звернуто увагу інтерфейс з точки зору майбутніх дій користувача.

3.2 Функціональне тестування веб-додатку

Тестування будь якої системи йде в ногу з контролем якості цієї системи. Якщо говорити в більш ширшому сенсі – тестування це техніка, яка покликана превентивно або явно знайти дефекти або їх можливість, і як наслідок привести до їх завчасного усунення. Це те, чому присвячений цей підрозділ, а саме – функціональному тестуванню. Функціональне ж тестування базується на функціях системи та її особливостях, воно на сам перед розглядає зовнішню поведінку додатку [39].

Я розділю тестування на модулі, тобто кожен модуль – це сервіс проекту. Це з урахуванням того факту, що ми маємо сервісну архітектуру проекту в рази спрощує процес тестування. Зосередимось на таких модулях: налаштування словників, запуск перебору директорій, керування User-agent, пошук форм, використаня багатопоточності у межах логіки, отримання логів.

Перший на черзі до тестування – модуль налаштування словників, який реалізований в сервісі “FileDictionaryProvider”. Його функціональна вимога в тому, щоб користувач сам вибирав потрібний йому словник з потенційними назвами директорій для перебору. Нижче у таблиці 3.1 зображено два use-case з очікуваними результатами.

Use-case – в контексті тестування [40] необхідний для розуміння потенційних дій збоку користувачів для розуміння вектора руху під час тестування. Іншими словами – це певний сценарій процесу, який описує “діалог” між кінцевим користувачем та системою.

Таблиця 3.1 – Характеристика тестування для словників

Use-case	Переключення словника	Перевірка вмісту словника
Кроки	А) В інтерфейсі вибрати словник; Б) Зберегти зміни; В) Виконати короткий перебір.	У меню “Settings” відкрити перегляд словника, переконатись, що там правильно відображаються всі записи
Очікуваний результат	Жодних помилок “File not found” і відповідність назв директорій словнику	Словник відображається без “порожніх” записів, без дублікатів

Розпочали з юз-кейсу який передбачає вибір словника і коротке сканування. На рисунку 3.9 бачимо, що сканування пройшло успішно і директорії відповідають тим, які є в словнику:

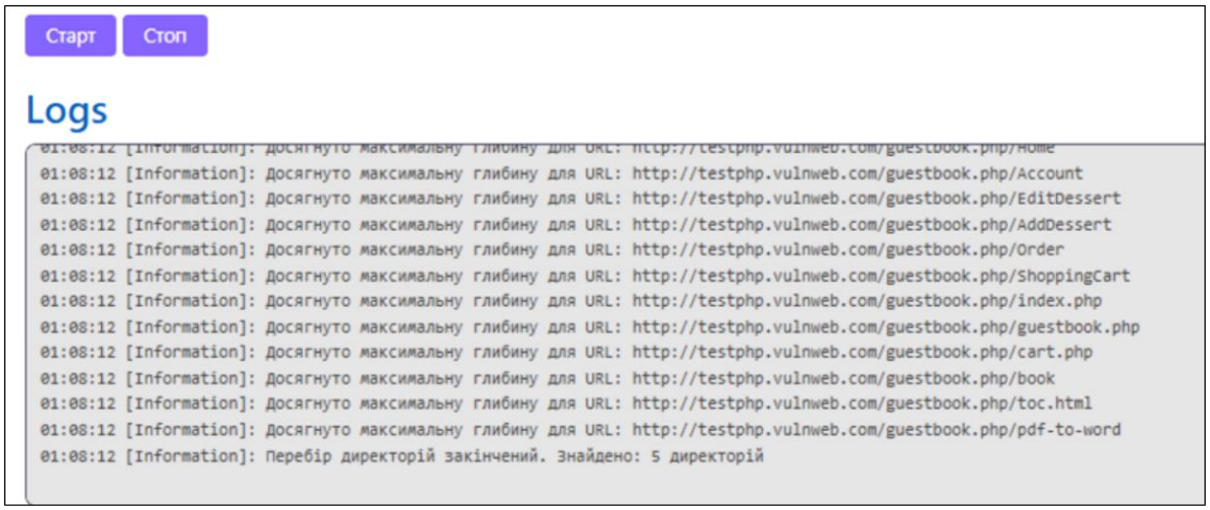


Рисунок 3.9 – Короткий перебір

Після того, як перший тест пройшов успішно переходимо до другого – перевірка вмісту словника. Для цього переходимо на сторінку додатку

“налаштування” і там вибираємо довільний словник для перевірки чи коректний його вміст (рисунок 3.10):

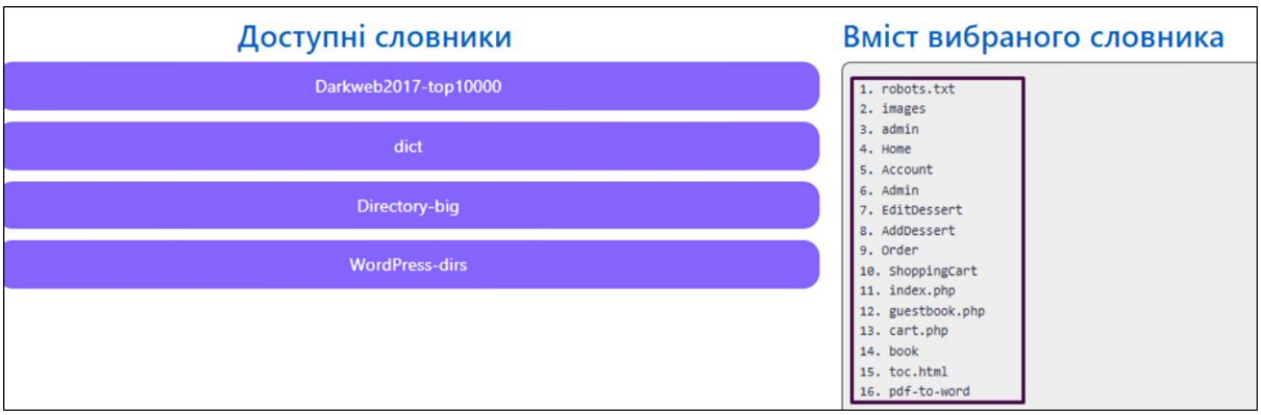


Рисунок 3.10 – Огляд словника

Для достовірності можемо зрівняти вивід із конкретним файлом цього словника у нашому проекті (рисунок 3.11). Впевнюємось, що цей тест також є успішним.

Рисунок 3.11 – Файл “dict.txt”

Другий модуль відповідає за сам запуск перебору директорій і реалізований через сервіс “DirectoryScanner”. Функціонал вимагає, щоб користувач міг вільно задавати глибину, кількість потоків, словник та запускати

сканування. Система в результаті перебирає усі можливі комбінації шляхів і збирає ті, де статус-код відповіді “успішний” або інформативний. Таблиця 3.2 описує use-case для цього випадку:

Таблиця 3.2 – Характеристика тестування для сканування

Use-case	Базове сканування	Рекурсивний перебір	Обробка статус-кодів
Кроки	А) Ввести у полі ціль; Б) Вибрати глибину, потоки, словник; В) Почати сканування.	А) Ввести ціль; Б) Глибина більше 1; В) Потоки 5; Г) Запуск сканування.	А) Завчасно знати що сайт відповідає певним статус-кодом; Б) перевірити чи сканування враховує і виводить це.
Очікуваний результат	У логах відобразиться ціль і запит, а по завершенні повідомлень: “знайдено X директорій”.	Логіка заходить у глиб – якщо одна директорія була успішною, програма почне сканувати всередині.	Відоме посилання на директорію заноситься до знайдених, бо його статус-код входить в число “успішних”.

На рахунок тесту базових директорій, можна відіслатись до рисунка 3.1 і побачити, що воно проходить успішно, так як у логах відображається ціль і запити, а в кінці ми отримали повідомлення з кількістю знайдених директорій.

На рисунку 3.12 показано успішність другого юз-кейса, який передбачає рекурсивний обхід для кожної знайденої сторінки або директорії. Я вибрав глибину 2. У цьому прикладі сканування знайшло з початку “admin/”, а потім робить рекурсивний перебір в середині цієї директорії. Також помітно, що

знайдено директорію “images/” – це значить що після цього система буде проходитись паралельно і в цій директорії:

```
01:15:04 [Information]: Запит до: http://testphp.vulnweb.com/admin (глибина: 1)
01:15:06 [Information]: URL http://testphp.vulnweb.com/robots.txt повернув 404, пропускаємо...
01:15:06 [Information]: Запит до: http://testphp.vulnweb.com/Home (глибина: 1)
01:15:06 [Information]: Додано: http://testphp.vulnweb.com/admin зі статусом 200
01:15:06 [Information]: Запит до: http://testphp.vulnweb.com/admin/robots.txt (глибина: 2)
01:15:06 [Information]: Додано: http://testphp.vulnweb.com/images зі статусом 200
01:15:06 [Information]: Запит до: http://testphp.vulnweb.com/images/robots.txt (глибина: 2)
01:15:08 [Information]: URL http://testphp.vulnweb.com/Home повернув 404, пропускаємо...
01:15:08 [Information]: Запит до: http://testphp.vulnweb.com/Account (глибина: 1)
01:15:09 [Information]: URL http://testphp.vulnweb.com/admin/robots.txt повернув 404, пропускаємо..
```

Рисунок 3.12 – Результати скану

Останній тест заключається у тому, щоб порівняти реальний статус-код із тим, який прийшов на відповідь системі під час сканування. У результатах (рисунок 3.13) сканування ми бачимо знайдені сторінки, наприклад, “//” повернулась з кодом 200:

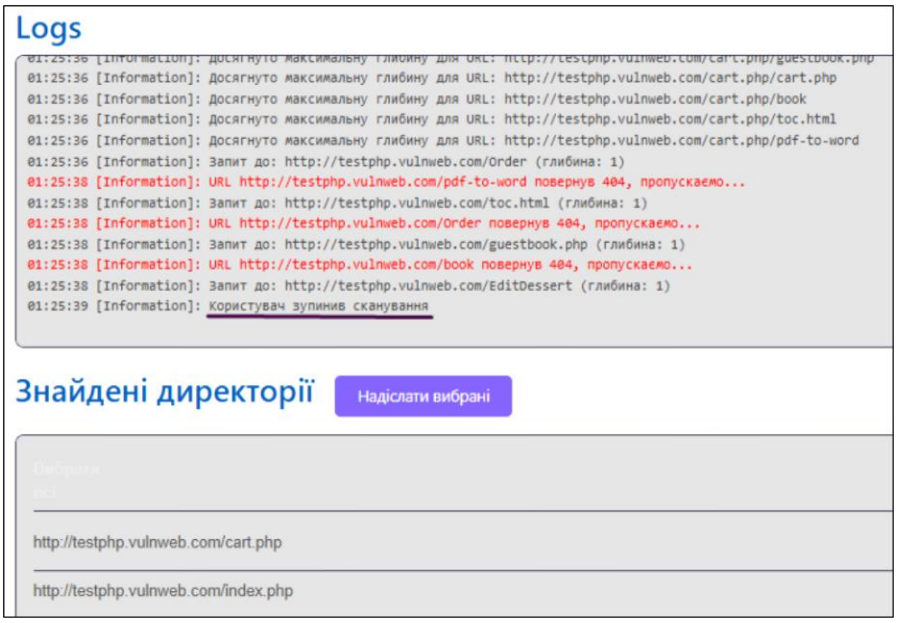


Рисунок 3.13 – Завчасне закінчення сканування

Перевіримо відповідність коду сторінки “/images/” (рисунок 3.14) в браузері. Бачимо, що вона дійсно відповідає тому, що вивела система веб-додатку у процесі сканування.

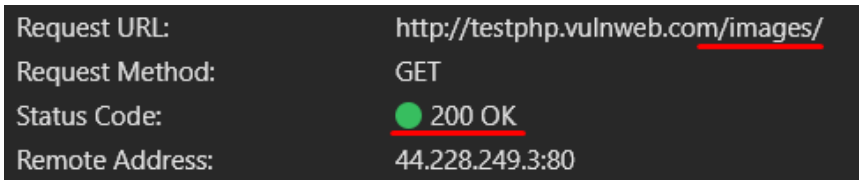


Рисунок 3.14 – Статус сторінки в браузері

Або ж візьмемо для порівняння сторінку “/cgi-bin”, котра повернула статус-код 403 (Forbidden). На рисунку 3.15 бачимо результат який вивівся в логах:

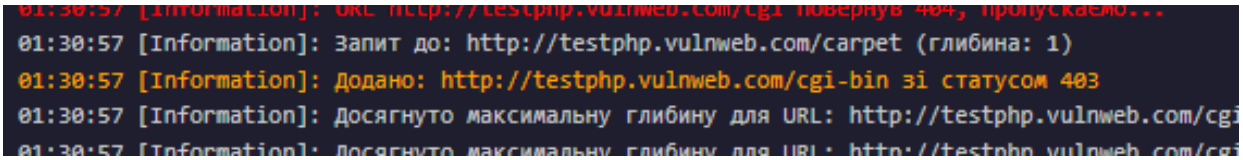


Рисунок 3.15 – Сторінка з кодом 403

Також переглянемо відповідність у бразузері, це зображено нижче на рисунку 3.16:

Рисунок 3.16 – Статус сторінки

Наступний функціональний тест стосується пошуку форм. Цьому модулю потрібно знайти на сторінці усі форми та їх поля (включно з “input, textarea, select”), визначити метод (GET/POST), “action” і зібрати списки полів для вводу або завантаження файлів. Основні кроки і результати які ми апріорі очікуємо зображені у таблиці 3.3. Для початку тестування на наявність форм з простими полями, тобто поля, в які можна вводити лише певні символи ми будемо

використовувати парсинг HTML-тіла вже раніше знайдених сторінок (рисунок 3.17):

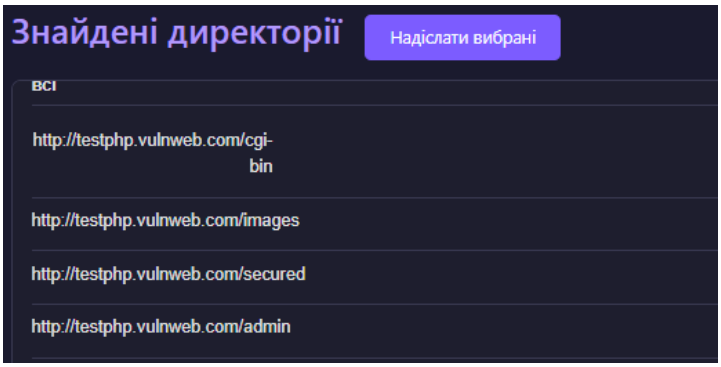


Рисунок 3.17 – Знайдені директорії

Таблиця 3.3 – Характеристика тестування пошуку форм

Use-case	Форми з простими полями	Файл завантаження
Кроки	А) Запустити скан вмісту сторінки.	А) Знайти сторінку де точно є завантаження файлів; Б) Запустити сканування сторінки.
Очікуваний результат	У результатах має бути метод, дія і поля.	Переконатись що в результати виводиться форма завантаження файлу.

Після того як знайдені директорії були перенесені у вкладку “Дії над директоріями” я почав проводити їх сканування, на пошук простих форм. На рисунку 3.18 видно, що вивелись деякі об’єкти які вміщають опис дії, методу, кількості і назв (разом з атрибутами) для знайдених полів вводу. Отже, цей тест можна вважати успішним:

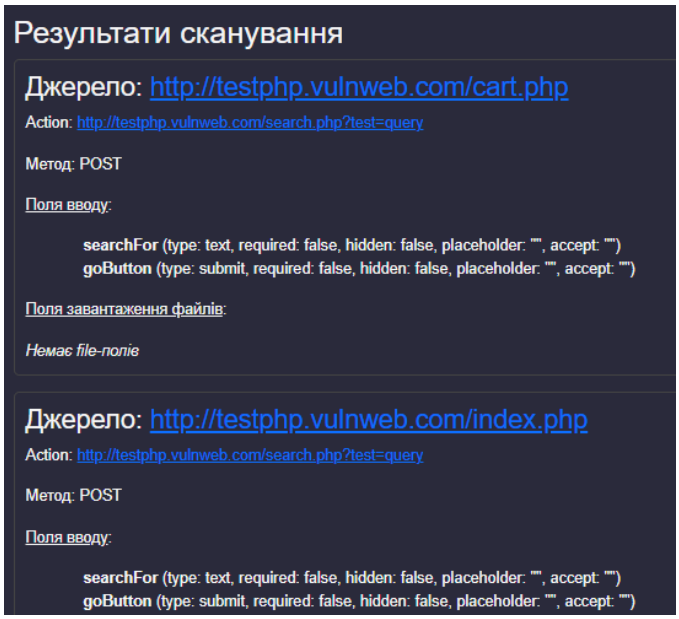


Рисунок 3.18 – Перші результати парсингу

Наступне тестування стосується форм, які приймають файли завантаженя. Для цього я вибрав специфіний ресурс, де конвертується файл в інший формат: логічно, що він точно вміщає в собі форму завантаження файлів (рисунок 3.19):

Рисунок 3.19 – Ресурс, який виступить “жертвою”

Для цього завдання я з початку запустив фаззинг директорій цього сайту, а потім знайдені директорії надіслав у модуль парсингу HTML-тіла. Ми переконані на сто відсотків, що результат повинен знайти і вивести поле завантаження файлу. Після сканування сторінки, отримано результати, які справді крім методу, дії і полей вводу, також показують поля завантаження файлів: вони містять багато цікавої інформації – від типу до переліку форматів

завантажених файлів. Цей факт грає важливу роль у майбутній експлуатації веб-додатку (рисунок 3.20).

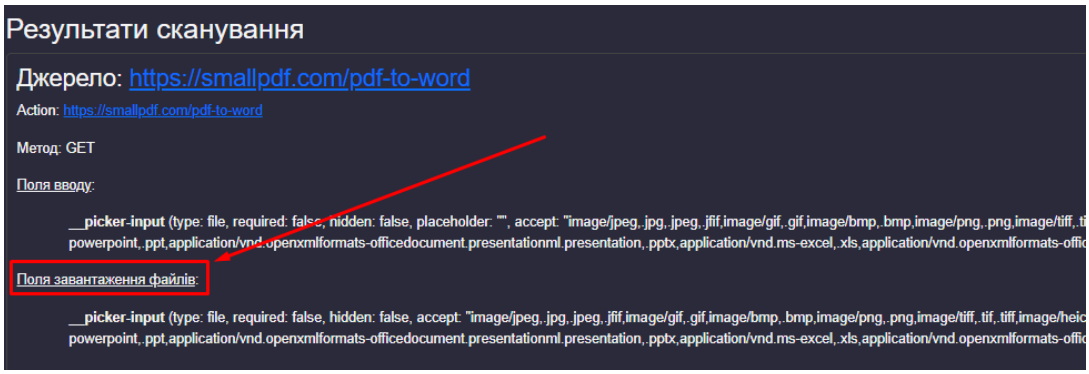


Рисунок 3.20 – Другі результати тестування

Останній функціональний тест, який потрібно провести стосується модулю, відповідального за керування User-agent – це сервіс “UserAgentProvider”. Додаток повинен надавати користувачеві можливість обирати певний UA серед загального переліку. У запитах, які відправляє сервіс сканування директорій підставляється саме цей рядок.

Таблиця 3.4 – Характеристика тестування User-agent

Use-case	Задано користувачем статичний UA
Кроки	А) У формі “Start Fuzzing” обрати довільний UA; Б) Запустити перебір; В) Перехопити запит через Network DevTools в браузері та перевірити, що в заголовках проставлено той самий UA.
Очікуваний результат	Кожен запит DirectoryScanner має відповідний UA.

Для тестування цього функціоналу переходимо на сторінку, де запускається сканування і вибираємо будь який user-agent (рисунок 3.21):

Рисунок 3.21 – Вибрали UA

Після того як запускаємо сканування, потрібно перейти у в браузері на вкладку “переглянути код елемента” і там знайти інструмент для моніторингу завантаження сторінки та всіх її файлів – “Network”. Тоді натиснути по відправленому запиті і переглянути його корисне навантаження (Payload). Це відображено на рисунку 3.22.

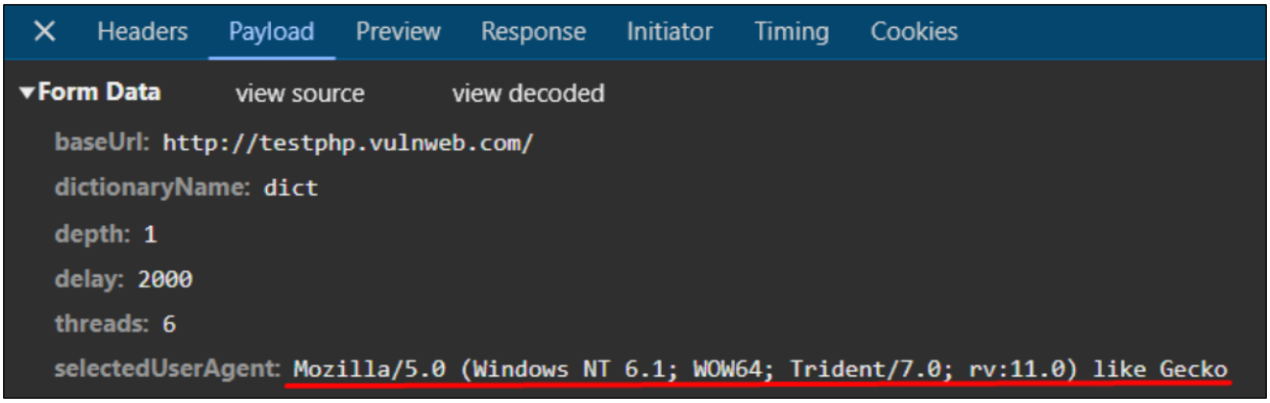


Рисунок 3.22 – Серед всієї інформації помічаємо вибраний UA

Проведене тестування показало, що кожен із розглянутих модулів у веб-додатку – від робочої логіки з файлами-словниками й налаштування глибини перебору директорій до підміни заголовка UA та пошуку форм ведення/завантаження файлів – готовий до роботи та коректно виконує заявлені функції.

Сервіс “FileDictionaryProvider” підтвердив свою здатність гнучно перемикати словники та коректно відображати їхній вміст у додатку. Під час

тестування “DirectoryScanner” було перевірено варіативність глибини рекурсії, правильність обробки статус-кодів, а також здатність переривати тривале сканування на вимогу користувача – ці всі етапи продемонстрували стабільну роботу та відповідність очікуваним результатам.

Щодо блоку “FormScanner”, практика з пошуком прости полів для завантаження файлів підтвердила, що застосунок вміє парсити HTML-код та правильно формувати інформацію про знайдені елементи форм – визначає “action”, “method” і набір полів. Окремо було перевірено механізм обрання та підстановки User-agent (через сервіс “UserAgentProvider”), який дозволяє підміняти заголовки запитів і таким чином обходити деякі обмеження серверів або працювати з різними версіями контенту. Усі тести підтвердили правильність функціонування цього модуля: обраний UA успішно відображається в заголовках запитів.

Загалом результати функціонального тестування показали узгодженість фактичної поведінки системи із вимогами та сценаріями використання. Наявність логів (і окремо – можливість зупиняти сканування) суттєво спрощує контроль за процесом і допомагає вчасно виявляти потенційні помилки. За підсумками, можна стверджувати, що AbyssScan функціонує відповідно до задуманої архітектури й відповідає заданим вимогам: користувач може безперешкодно змінювати словники, налаштовувати різні параметри перебору, підміняти UA, знаходити цільові поля форм та аналізувати успішні чи заборонені директорії.

3.3 Порівняльна оцінка інструментів

Для того, щоб об’єктивно оцінити ефективність створеного додатку AbyssScan – було вирішено провести порівняння з існуючими популярними інструментами, які описувались в першому розділі. Вони виконують аналогічні задачі з автоматизованого сканування директорій. До цього порівняння я

включив GoBuster, OWASP DirBuster та модуль Intruder з Burp Suite, які представляють різні підходи до реалізації даної задачі – від CLI-утиліт до повноцінних GUI-систем.

Перед тим як оцінювати кожен з цих інструментів, варто розробити методику та критерії для оцінки. Будемо виходити з двох головних чинників, які важливі для будь якого користувача – це комфорт у користуванні, контроль процесу та швидкість. І як результат, виникають декілька кількісних та якісних критеріїв, серед яких: швидкість сканування, інтерфейс, гнучкість налаштувань, якість результатів, можливості аналізу HTML-вмісту, кастомізація та масштабованість та логування і звітність.

Почнемо з кількісного критерія – це швидкість, час проходження словника певного об’єму за якийсь час. Вихідними параметрами у нас будуть: словник на 960 рядків, сканування з десятьма потоками та сама ціль – тестовий сайт “testphp.vulnweb.com”. Перший на черзі для тестування інструмент GoBuster, який працює через термінал. Для замірювання часу роботи, перед основною командою запуску сканування, я також додав слово “time” – це виклик спеціальної утиліти, яка зафіксує час роботи. І як результат, на рисунку 3.23 видно у полі “real” 17 секунд і 64 мілісекунд – це зафіксований час сканування.

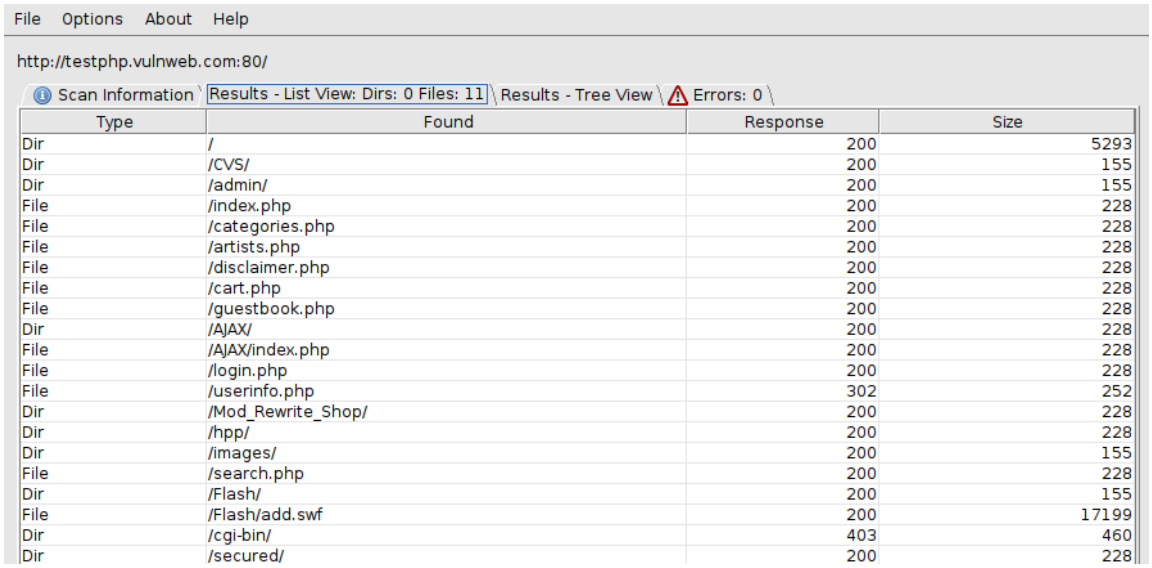
Також у процесі сканування було виявлено шість директорій з двома статусами: 301 та 403 - /admin, /images, /cgi-bin тощо. Це свідчить про правильну обробку відповідей сервера.

Рисунок 3.23 – запуск сканування через інструмент GoBuster

В цілому інструмент GuBuster продемонстрував високу швидкість перебору. Утиліта практично миттєво обробляє словники малого розміру за рахунок низькорівневої реалізації на мові Go та мінімального оверхеду. Однак інтерфейс повністю консольний і результати подаються у вигляді простого текстового логу без візуалізації або HTML-аналізу, хоча, можна витягнути результат через спеціальну команду.

Для подального порівняння візьмемо DirBuster – він графічний, і як інструмент написаний на мові Java, а це дозволить гнучно налаштувати перебір та спостерігати за деревом директорій у реальному часі. Інструмент було запущено з тим самим словником “small.txt” на 960 рядків, із встановленими 10 потоками та без рекурсії. Бойовим майданчиком залишився той же сайт – “testphp.vulnweb.com”. Для вимірювання часу запуску було використано команду “date” у термінали під час запуску сканування та при його закінченні. Отже, маємо такий час: 41 секунд сканування – майже в 2,5 рази довше, ніж GoBuster.

Проте цей інструмент продемонстрував розширені можливості візуалізації: знайдені директорії та файли були відображені у вигляді таблиці та дерева, що помітно спрощує аналіз. Нижче на рисунку 3.24 показано, що серед знайденого були тіж самі директорії, що і у випадку з GoBuster, а також файли /login.php, /search.php тощо.


File Options About Help

http://testphp.vulnweb.com:80/

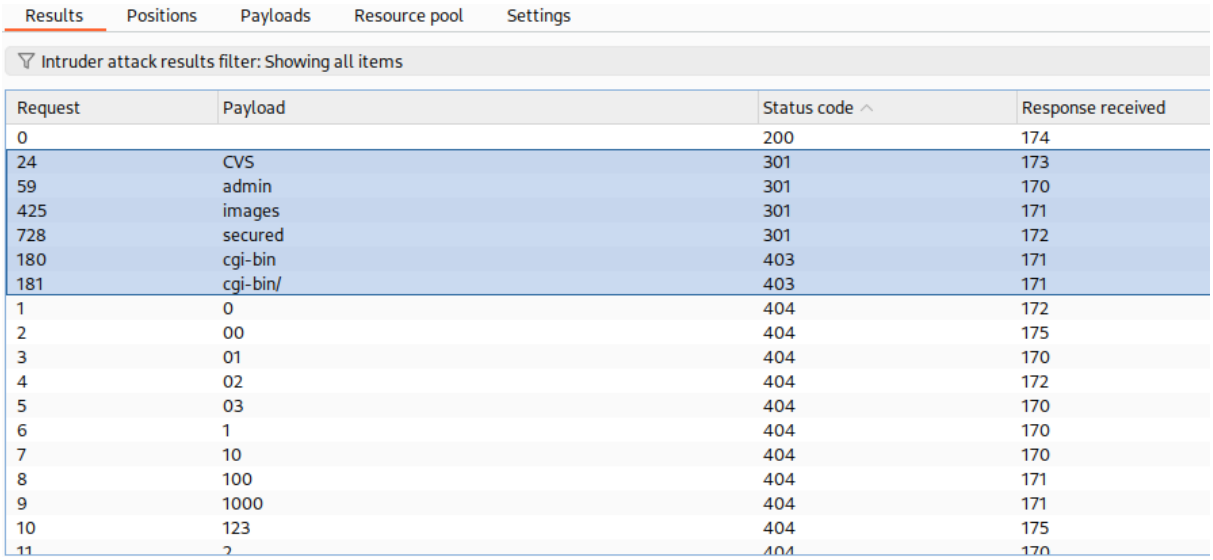
Scan Information | Results - List View: Dirs: 0 Files: 11 | Results - Tree View | Errors: 0

Type	Found	Response	Size
Dir	/	200	5293
Dir	/CVS/	200	155
Dir	/admin/	200	155
File	/index.php	200	228
File	/categories.php	200	228
File	/artists.php	200	228
File	/disclaimer.php	200	228
File	/cart.php	200	228
File	/guestbook.php	200	228
Dir	/AJAX/	200	228
File	/AJAX/index.php	200	228
File	/login.php	200	228
File	/userinfo.php	302	252
Dir	/Mod_Rewrite_Shop/	200	228
Dir	/hpp/	200	228
Dir	/images/	200	155
File	/search.php	200	228
Dir	/Flash/	200	155
File	/Flash/add.swf	200	17199
Dir	/cgi-bin/	403	460
Dir	/secured/	200	228

Рисунок 3.24 – вікно з результатами в DirBuster

Наступний інструмент для тестування – Burp Suite. А саме його модуль “Intruder”, котрий займається фаззингом параметрів. Хоча й Burp Suite не є класичним сканером директорій, проте він може бути адаптований під ці цілі шляхом ручного налаштування позиції підстановки у URL.

Після перехоплення базового запиту через проксі, його було відправлено у вкладку Intruder. У якості словника використано все той же “small.txt”, а режим атаки – Sniper з однією точкою підстановки в кінці URL. Час було зафіксовано за допомогою команди date у терміналі: старт сканування відбувся в 14:46:07, а завершення 14:46:25, отже – сканування зайняло 18 секунд, що є кращим результатом ніж DirBuster, але трохи повільніше ніж GoBuster. До недоліків можна віднести високу ручну складність налаштування. З рисунку 3.25 видно, що знайшлись ті ж самі шість директорій, що й раніше:



Request	Payload	Status code	Response received
0		200	174
24	CVS	301	173
59	admin	301	170
425	images	301	171
728	secured	301	172
180	cgi-bin	403	171
181	cgi-bin/	403	171
1	0	404	172
2	00	404	175
3	01	404	170
4	02	404	172
5	03	404	170
6	1	404	170
7	10	404	170
8	100	404	171
9	1000	404	171
10	123	404	175
11	2	404	170

Рисунок 3.25 – результати сканування через Burp Suite

Фінальним етапом порівняльного тестування стала перевірка продуктивності розробленого в рамках цього проєкту додатку – AbyssScan. Параметри експерименту були ідентичні до попередніх. Процес сканування запускався безпосередньо з веб-інтерфейсу додатку, а час фіксувався прямо через логи системи: старт був в 22:14:21, а завершення в 22:14:38. Відповідно

маємо тривалість сканування – 17 секунд, що повністю відповідає продуктивності інструмента GoBuster.

У результатах сканування було виявлено 5 директорій, серед яких ті, які ви бачите на рисунку 3.26. Це дещо менше ніж у попередніх інструментів, але це пов’язано з особливостями фільтрації результатів. Проте, з точки зору юзабіліті та контролю наш проєкт має низку переваг:

– повноцінне логування подій у реальному часі;

– миттєве припинення сканування за командою користувача;

– зміна словників без перезапуску додатку;

– вбудований HTML-парсер для аналізу знайдених об’єктів;

– миттєве налаштування сканування.

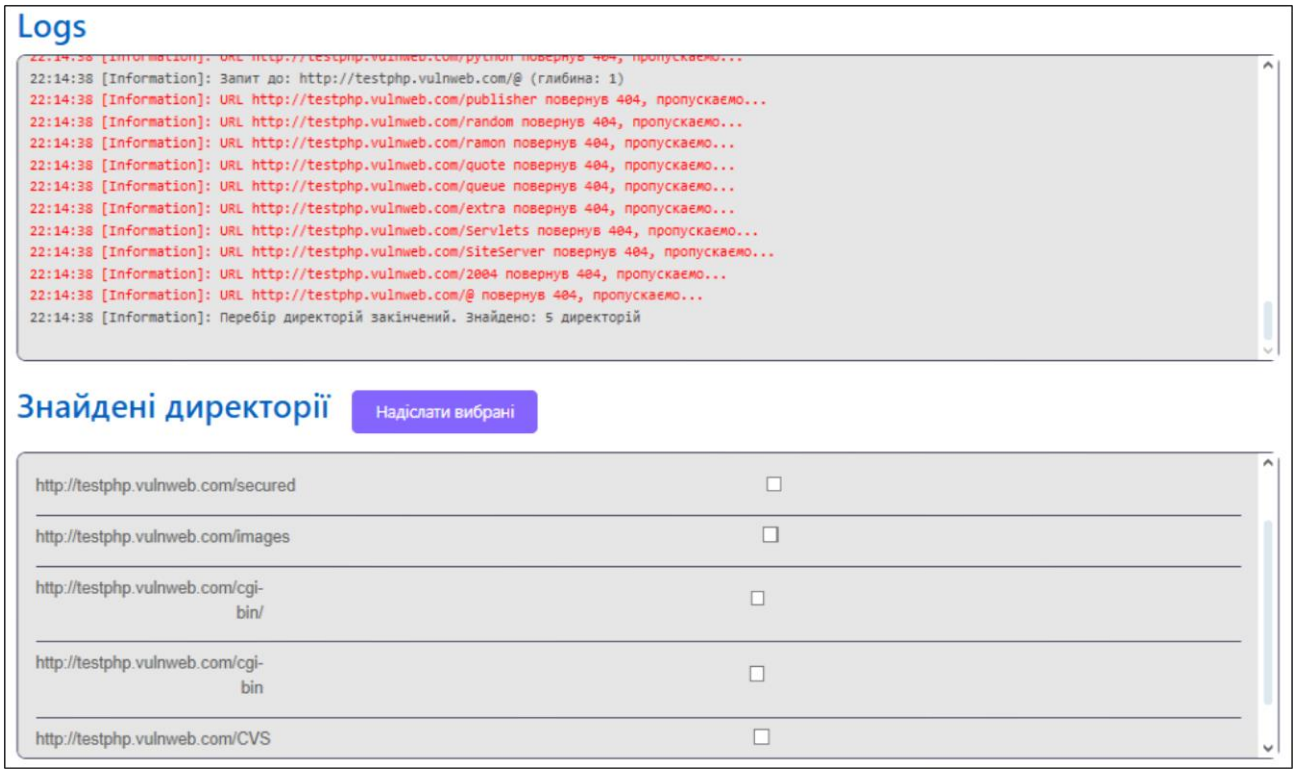


Рисунок 3.26 – вивід результатів сканування директорій у нашому додатку

Нижче у таблиці 3.1 узагальнено ключові показники кожного з досліджуваних інструментів.

Таблиця 3.1 – порівняльна оцінка інструментів сканування директорій

Критерій	GoBuster	DirBuster	Burp Suite	AbyssScan
Час сканування	17,06 сек	41 сек	18 сек	17 сек
Знайдено директорій	6	6+ (в т.ч. .php файли)	6	5
Інтерфейс	CLI	GUI	напів-GUI	повноцінний Web UI
Гнучкість налаштувань	потоки, словник	потоки, словник, рекурсія	ручне конфігурування	потоки, словник, UA, глибина
HTML-аналіз (форми)	–	–	частково	+
Зупинка сканування	–	+	–	+
Логування	обмежене	обмежене	обмежене	+
Звітність / експорт результатів	текстовий файл	CSV/XML	таблиця Intruder	вивід у UI
Можливість розширення	–	обмежена	+	Є (.NET Core, open-архітектура)

Як видно із таблиці, проєкт AbyssScan поєднує в собі найкращі риси інших додатків, має реальний потенціал і здорову конкуренцію між топовими інструментами в сфері фаззингу.

ВИСНОВКИ

У процесі дослідження, проєктування та реалізації нашого додатку вдалось досягти поставленої мети – створити інструмент для автоматизованого сканування директорій хостингів, який може стати практичним засобом для попереднього аналізу структури веб-ресурсів. Під час реалізації було виконано низку важливих кроків:

– проведено аналіз сучасного стану веб-безпеки, а також огляд ключових методів і інструментів, які використовуються в практиці пентестингу. Це дозволило сформувати чіткі вимоги до системи: швидкість, масштабованість, гнучкість у налаштуваннях і мінімальна участь користувача при проведенні типових сканувань;

– на основі цього аналізу було спроектовано архітектуру, яка дозволяє ізольовано реалізовувати окремі функції, підтримувати масштабування системи без переробки ядра, а також забезпечити просте тестування кожного модуля;

– реалізовано багатопоточний сканер директорій, що використовує словниковий підхід до перебору шляхів. Це дозволяє знаходити навіть URL, які не мають жодного посилання на сайті але можуть бути відкрити ми випадково або через байдужість розробників;

– підтримка словників з можливістю швидкого перемикання забезпечує адаптивність додатку під різні CMS та мови програмування вебресурсів. Додатково реалізовано модуль роботи з User-Agent, що дозволяє імітувати запити від імені різних браузерів або пристроїв;

– реалізовано парсинг HTML-сторінок для виявлення форм введення (текстових або файлів), а це відкриває можливість попереднього аналізу на наявність XSS, SQL-ін’єкцій та інших точок потенційної вразливості.

У третьому розділі записки вже були відомі практичні результати, наприклад, функціональне тестування продемонструвало стабільну роботу всіх основних модулів. Система коректно перемикає словники, а обробка директорій відбувається з врахуванням глибини та поточних потоків. Крім того, знайдені

сторінки парсяться без помилок і статус коди відповідають реальним HTTP-відповідям від цільового ресурсу. Логування забезпечує візуальний контроль процесу сканування у реальному часі, а функція зупинки процесу працює без втрати стабільності.

Тестування було проведене на демо-майданчику з вразливостями, де система змогла за лічені секунди виявити всі цільові директорії зі словника. Це в свою чергу доводить нам, що система є ефективною в умовах, наближених до реального пентест-середовища.

До того ж, завдяки використанню багатошарової архітектури (onion architecture) система є гнучкою у масштабуванні. При зміні логіки словників, API або парсера, не потрібно вносити зміни в ядро.

В цьому проєкті, на фоні підбиття підсумків, можна навіть сформувати реальні напрямки подальшого розвитку. Зараз користувач отримує результат нативно – через інтерфейс, але у перспективні це може бути у виді звітування у форматі PDF/CSV після завершення сканування. Так само один User-agent користувач вибирає умовно на одне сканування, але в ідеалі система могла б підтримувати ротацію UA – тобто, один UA на один поток, це значно ускладнить детекцію з боку захисних систем цілей.

Як підсумок, цей реалізований додаток – це повноцінна система, яка спрощує процес початкової розвідки сайту, дозволяє знаходити помилки конфігурації та відкриті директорії та дає змогу оцінити наявність форм, точок введення та полів завантаження.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Яка роль веб-сайтів. Quora. URL: https://www.quora.com/What-is-the-role-of-a-website (дата звернення: 02.01.2025).

2. What is Denial-of-service. Cloudflare.URL: https://www.cloudflare.com/learning/ddos/glossary/denial-of-service/ (дата звернення: 02.01.2025)

3. Розподілена атака на відомву в обсглуговуванні (DDoS). Eset. URL: https://www.eset.com/ua/support/information/entsyklopediya-zahroz/ddos-ataka/?srsltid=AfmBOoox8tmC2MQO8Ht13tWpcBNM2rZdqTFsE8mvzUSqGoWiWccpKLK6 (дата звернення: 02.01.2025)

4. What is SQL-injection. OWASP. URL: https://owasp.org/www-community/attacks/SQL_Injection (дата звернення: 02.01.2025)

5. Типи кібер-атак. Fortinet. URL: https://www.fortinet.com/resources/cyberglossary/types-of-cyber-attacks (дата звернення: 02.01.2025).

6. What is XSS injection. OWASP. URL: https://owasp.org/www-community/attacks/xss/ (дата звернення: 02.01.2025)

7. Інструменти для перебору директорій. Medium. URL: https://medium.com/@old.noisy.speaker/cybersecurity-web-fuzzing-50dc006f5e6f (дата звернення: 02.01.2025)

8. Обхід аутентифікації через API. Snyk. URL: https://learn.snyk.io/lesson/broken-authentication/?ecosystem=csharp (дата звернення: 02.01.2025)

9. Directory enumeration. Medium. URL: https://medium.com/@infosecnubes/directory-enumeration-1ba91012dcf8 (дата звернення: 02.01.2025)

10. Інструкція для користування інструментом GoBuster. Hackercool. URL: https://www.hackercoolmagazine.com/beginners-guide-to-gobuster-

tool/?srsltid=AfmBOooXXOjRHL5Wyzg7LcbZq3tC4t_DTtP8wmy2LNMBlXy_ONcbIEgB (дата звернення: 03.01.2025)

11. Guide to DirBuster. Hackercool. URL: https://www.hackercoolmagazine.com/beginners-guide-to-dirbuster/?srsltid=AfmBOopfBZL8eCi3n0DKBSK09Y1Z-8Ap2SrUsWUImTolorH0Ev5j7Kmy (дата звернення: 04.01.2025)

12. What is BurpSuite. GeeksForGeeks. URL: https://www.geeksforgeeks.org/what-is-burp-suite/ (дата звернення: 05.01.2025)

13. Що таке ітеративний метод. ScienceDirect. URL: https://www.sciencedirect.com/topics/mathematics/iteration-method (дата звернення: 05.01.2025)

14. Як працює рекурсія. FreeCodeCamp. URL: https://www.freecodecamp.org/news/how-recursion-works-explained-with-flowcharts-and-a-video-de61f40cb7f9 (дата звернення: 06.01.2025).

15. Багатопоточне програмування. IBM. URL: https://www.ibm.com/docs/en/aix/7.1.0?topic=concepts-multithreaded-programming (дата звернення: 06.01.2025)

16. Процеси та потоки. Microsoft. URL: https://learn.microsoft.com/en-us/windows/win32/procthread/processes-and-threads (дата звернення: 06.01.2025)

17. Клас Parallel у C#. Microsoft. URL: https://learn.microsoft.com/en-us/dotnet/api/system.threading.tasks.parallel?view=net-9.0 (дата звернення: 06.01.2025)

18. Клас Task у C#. Microsoft. URL: https://learn.microsoft.com/en-us/dotnet/api/system.threading.tasks.task?view=net-9.0 (дата звернення: 06.01.2025)

19. Parallel LINQ в C#. Microsoft. URL: https://learn.microsoft.com/en-us/dotnet/standard/parallel-programming/introduction-to-plinq (дата звернення: 06.01.2025)

20. Клас Thread у C#. Microsoft. URL: https://learn.microsoft.com/en-us/dotnet/api/system.threading.thread?view=net-9.0 (дата звернення: 07.01.2025)

21. Токен відміни – що це і як працює в C#. Microsoft. URL: https://learn.microsoft.com/en-us/dotnet/api/system.threading.cancellationtoken?view=net-9.0 (дата звернення: 08.01.2025)

22. Using threads and threading. Microsoft. URL: https://learn.microsoft.com/en-us/dotnet/standard/threading/using-threads-and-threading (дата звернення: 09.01.2025)

23. Метод Thread.Sleep.Microsoft. URL: https://learn.microsoft.com/en-us/dotnet/api/system.threading.thread.sleep?view=net-9.0 (дата звернення: 09.01.2025)

24. Що таке User-agent. Developer-mozilla. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/User-Agent (дата звернення: 09.01.2025)

25. Історія виникнення User-agent. WebAIM. URL: https://webaim.org/blog/user-agent-string-history/ (дата звернення: 10.01.2025)

26. Директорії WordPress. Infosec. URL: https://notes.qazeer.io/web-applications/cms_and_softwares/wordpress (дата звернення: 11.01.2025)

27. Що таке монолітна архітектура. IBM. URL: https://www.ibm.com/think/topics/monolithic-architecture (дата звернення: 12.01.2025)

28. Мікросервісна архітектура та її плюси. MA. URL: https://microservices.io/patterns/microservices.html (дата звернення: 13.01.2025)

29. Як вибрати архітектуру для веб-додатку. Blog. URL: https://blog.ithillel.ua/ru/articles/web-application-architecture (дата звернення: 13.01.2025)

30. Building apps with serverless architectures. AWS. URL: https://aws.amazon.com/lambda/serverless-architectures-learn-more/?nc1=h_ls (дата звернення: 14.01.2025)

31. Цибулева архітектура і її особливості. Jeffrey Palermo blog. URL: https://jeffreypalermo.com/2008/07/the-onion-architecture-part-1/ (дата звернення: 03.02.2025).

32. Що нового у технології .NET версій 7-8. Learn-microsoft. URL: https://learn.microsoft.com/en-us/dotnet/core/whats-new/dotnet-8/overview (дата звернення: 04.02.2025).

33. Дослідження JetBrains – чому розробники частіше вибирають технологію .NET. JetBrains. URL: https://www.jetbrains.com/lp/devecosystem-2023/ (дата звернення: 04.02.2025).

34. AngleSharp documentation. GitHub. URL: https://github.com/AngleSharp/AngleSharp/blob/devel/docs/README.md (дата звернення: 04.02.2025)

35. HTTP client in C#. Medium. URL: https://medium.com/@iamprovidence/http-client-in-c-best-practices-for-experts-840b36d8f8c4 (дата звернення: 04.02.2025)

36. Пакет Microsoft.Extensions.Logging – що це. Iron PDF. URL: https://ironpdf.com/blog/net-help/microsoft-extensions-logging-csharp/ (дата звернення: 04.02.2025).

37. 10 евристик юзабіліті Якоба Нільсона. UX Pub. URL: https://ux.pub/editorial/10-ievristik-iuzabiliti-iakoba-nilsiena-proiliustrovanikh-dizain-rishienniami-revolut-bdo (дата звернення: 10.02.2025).

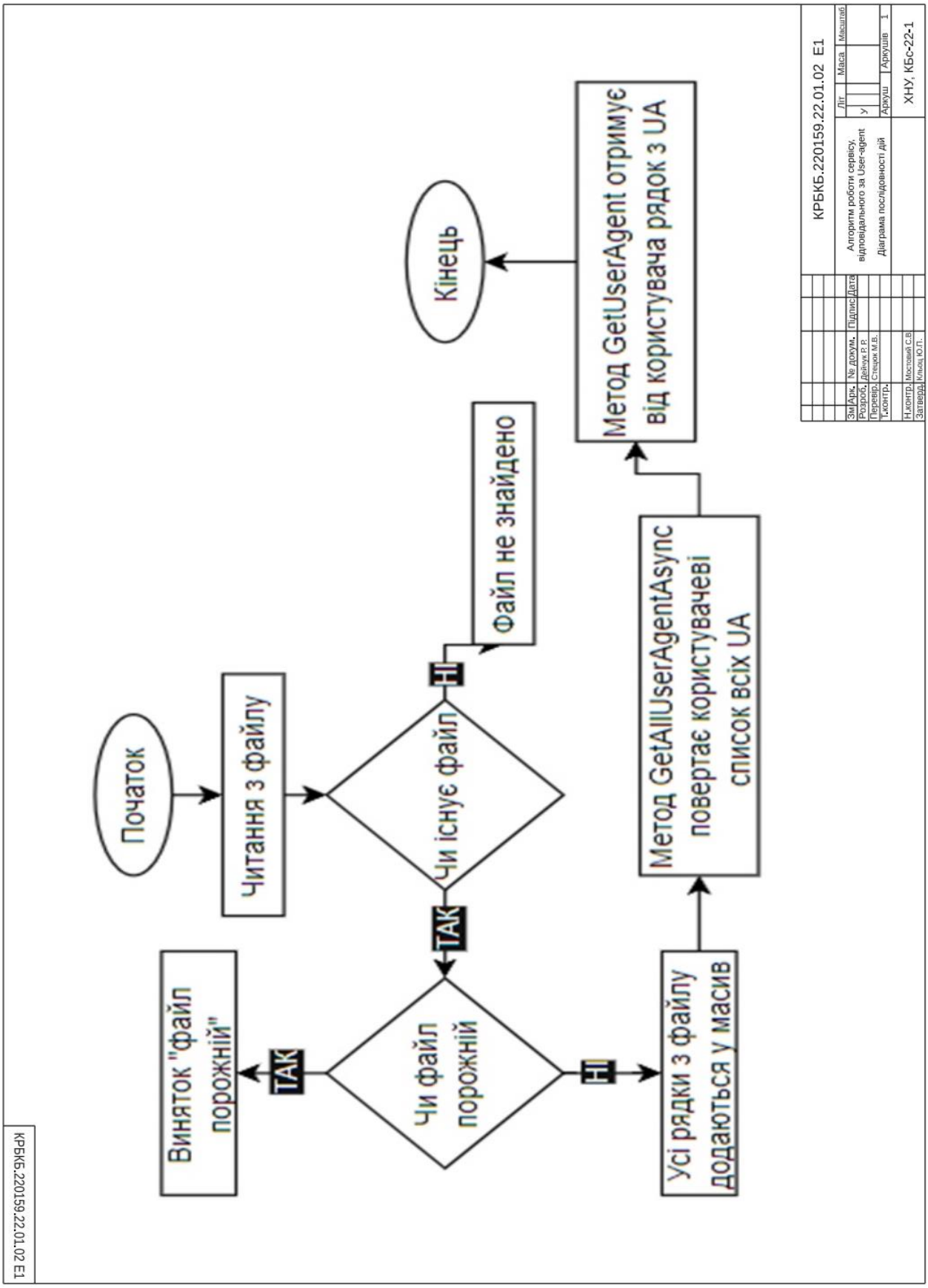
38. Heuristic consistency and standarts. The decision lab. URL: https://thedecisionlab.com/reference-guide/design/nielsens-heuristics (дата звернення: 10.02.2025)

39. Основи тестування для нетестувальників. Studylib. URL: https://studylib.net/doc/25850923/testirovshhik (дата звернення: 15.02.2025).

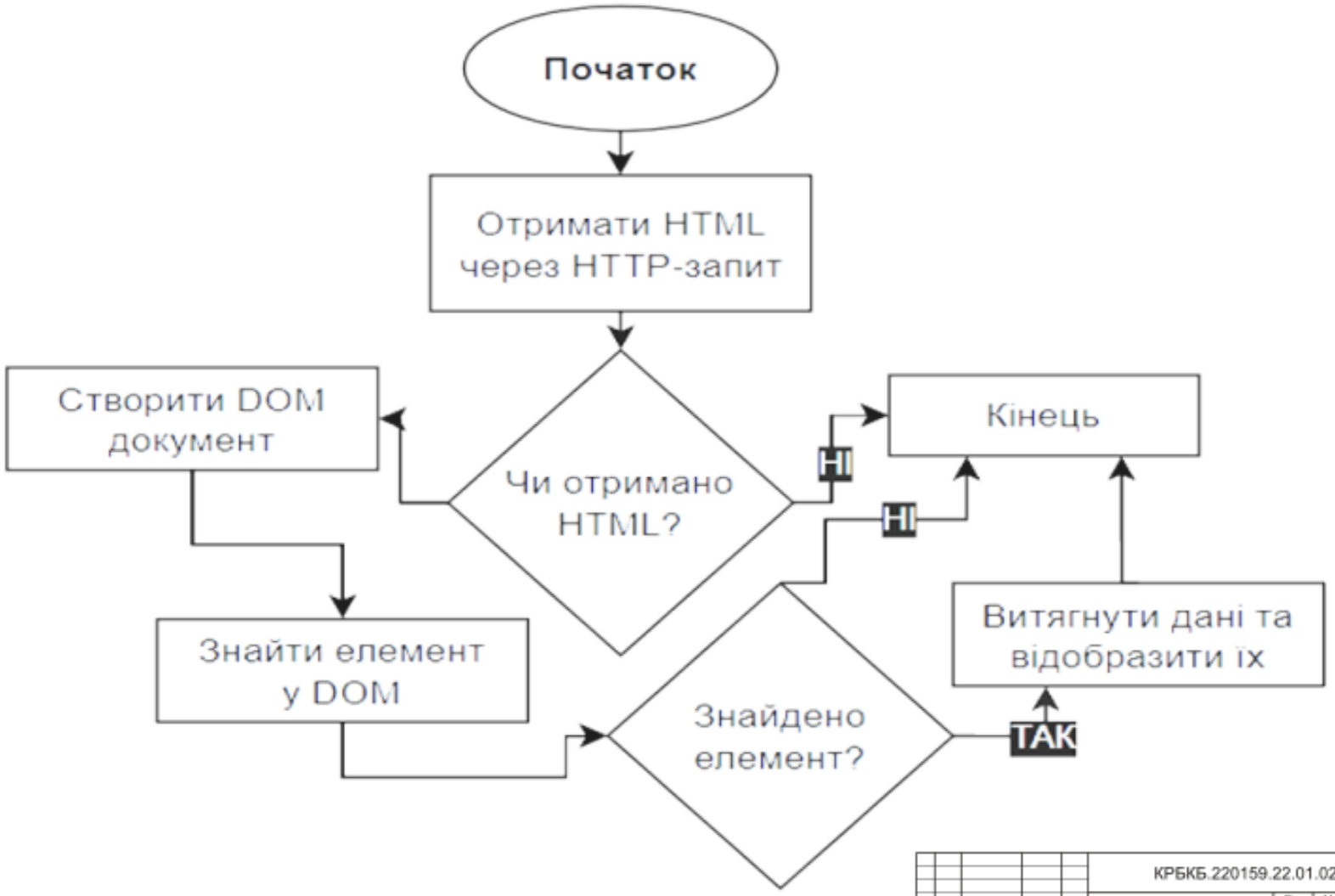
40. Що таке use-case та для чого потрібне. QA TestLab. URL: https://training.qatestlab.com/blog/technical-articles/what-is-a-use-case-and-what-are-they-for/ (дата звернення: 23.03.2025).

ДОДАТОК А

Копія графічної частини

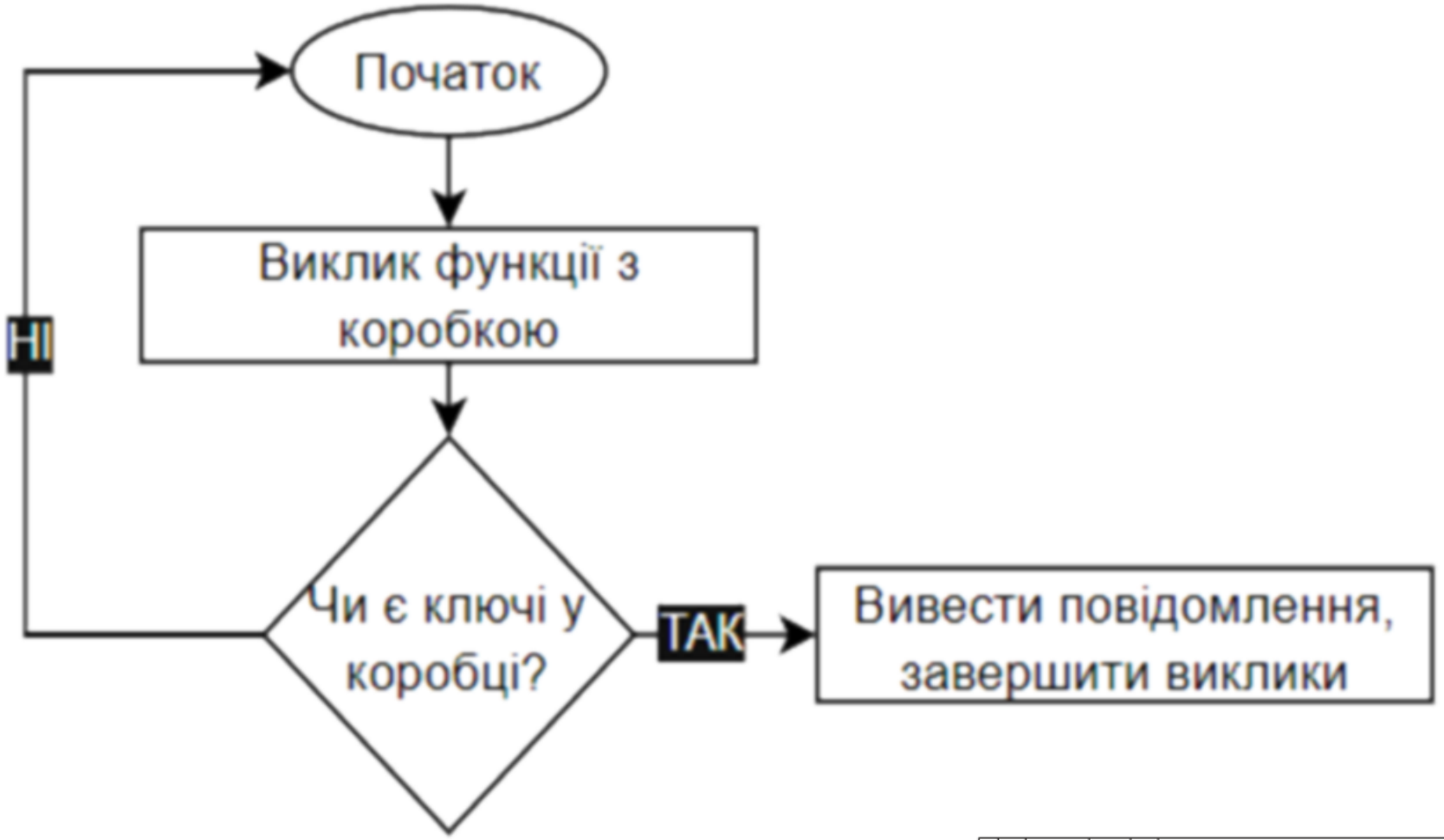


КРБКБ.220159.22.01.02 Е1



					КРБКБ.220159.22.01.02Е1		
Зм.	Арк.	№докум.	Підпис	Дата	Алгоритм роботи бібліотеки AngleSharp	Літ.	Маса	Масштаб
Розроб.		Дейнуй Р.Р.				У		
Перевір.		Степаненко М.В.						
Т.контр.						Аркуш	Аркушів 1	
Н.контр.		Мостовий С.Ю.						
Затверд.		Кільчевський Ю.П.				ХНУ,КБс-22-1		

КРБКБ.220159.22.01.02 Е1



					КРБКБ.220159.22.01.02 Е1	
Зм.	Арк.	№докум.	Підпис	Дата	Алгоритм роботи рекурсії Діаграма послідовності дій	Літ. / Маса / Масштаб У
Розроб.		Дайчук Р.Р.				
Перевір.		Стецюк М.В.				Аркуш / Аркушів 1
Т.контр.						
Н.контр.		Мостовий С.В.				ХНУ, КБс-22-1
Затверд.		Кльоц Ю.П.				

ДОДАТОК Б
Лістинг коду

```
public class ActionsController : Controller
 {
    private readonly IFormScanner _formScanner;
    private readonly ILogger<ActionsController> _logger;

    public ActionsController(IFormScanner formScanner, ILogger<ActionsController> logger)
    {
       _formScanner = formScanner;
       _logger = logger;
    }

    public IActionResult Index()
    {
       if (TempData["SelectedDirectories"] is string dirsRaw)
       {
          var dirs = dirsRaw.Split(';').ToList();
          return View(dirs);
       }

       return View(new List<string>());
    }

    [HttpPost]
    public IActionResult ProcessSelectedDirectories([FromBody] DirectoriesRequest request)
    {
       if (request.Directories is null || !request.Directories.Any())
       {
          return BadRequest(new { error = "Не передано жодної директорії." });
       }

       TempData["SelectedDirectories"] = string.Join(";", request.Directories);
       TempData["Threads"] = request.Threads;

       _logger.LogInformation("Отримано {Count} директорій для подальшого сканування. Перейдіть у вкладку 'Дії
над директоріями' для подальших дій", request.Directories.Count);

       return Json(new
       {
          message = "Директорії успішно отримано",
          count = request.Directories.Count,
          threads = request.Threads
       });
    }

    [HttpPost("/Actions/ScanDirectories")]
    public async Task<IActionResult> ScanDirectories([FromBody] DirectoriesRequest request)
    {
       if (request == null)
       {
          return BadRequest(new { error = "Запит не може бути порожнім" });
       }

       if (request.Directories == null || !request.Directories.Any())
       {
          return BadRequest(new { error = "Не передано жодної директорії" });
       }

       if (request.ScanType == null || !request.ScanType.Any())
```

```
    {
        return BadRequest(new { error = "Не обрано типу сканування" });
    }

    if (request.Threads <= 0)
    {
        return BadRequest(new { error = "Кількість потоків повинна бути більша нуля" });
    }

    var results = new List<object>();
    var options = new ParallelOptions { MaxDegreeOfParallelism = request.Threads };

    try
    {
        await Parallel.ForEachAsync(request.Directories, options, async (directory, cancellationToken) =>
        {
            var content = await FetchPageContentAsync(directory);

            try
            {
                var forms = await _formScanner.FindFormsAsync(content, directory);
                lock (results)
                {
                    results.AddRange(forms.Select(form => new
                    {
                        Source = directory,
                        ScanType = "forms",
                        Action = form.Action,
                        Method = form.Method,
                        InputFields = form.InputFields,
                        FileFields = form.FileFields
                    }));
                }
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Помилка під час сканування директорії {Directory}", directory);
            }
        });

        return Json(results);
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Помилка під час обробки запиту");
        return StatusCode(500, new { error = "Помилка під час обробки запиту" });
    }
}

private async Task<string> FetchPageContentAsync(string url)
{
    using var httpClient = new HttpClient();
    try
    {
        var response = await httpClient.GetAsync(url);
        response.EnsureSuccessStatusCode();

        var content = await response.Content.ReadAsStringAsync();
        return content;
    }
    catch (Exception ex)
    {
        _logger.LogInformation(ex, "Не вдалось отримати контент сторінки {Url}", url);
```

```
            return string.Empty;
        }
    }
}

public class FuzzingController : Controller
{
    private readonly ILogger<FuzzingController> _logger;
    private readonly IThrottlingService _throttlingService;
    private readonly IDirectoryScanner _directoryScanner;
    private readonly IDictionaryProvider _dictionaryProvider;
    private readonly IUserAgentProvider _userAgentProvider;
    private readonly IHttpRequester _httpRequester;
    private static CancellationTokenSource _cts = new();
    private static string _baseUrl = "";

    public FuzzingController(IHttpRequester httpRequester, ILogger<FuzzingController> logger, IThrottlingService throttlingService, IDirectoryScanner directoryScanner, IDictionaryProvider dictionaryProvider, IUserAgentProvider userAgentProvider)
    {
        _logger = logger;
        _throttlingService = throttlingService;
        _directoryScanner = directoryScanner;
        _dictionaryProvider = dictionaryProvider;
        _userAgentProvider = userAgentProvider;
        _httpRequester = httpRequester;
    }

    public async Task<IActionResult> Index()
    {
        var directories = _dictionaryProvider.GetAvailableDictionaries();

        var allUserAgents = await _userAgentProvider.GetAllUserAgentsAsync();

        ViewBag.Dictionaries = directories;
        ViewBag.UserAgents = allUserAgents;

        return View();
    }

    [HttpPost]
    public async Task<IActionResult> StartFuzzing(string selectedUserAgent, string baseUrl, string dictionaryName, int depth, int delay, int threads)
    {
        if (string.IsNullOrWhiteSpace(baseUrl))
        {
            _logger.LogError("Ціль потрібно вказати обов'язково");
            return BadRequest("Ціль потрібно вказати обов'язково");
        }

        if (!Regex.IsMatch(baseUrl, @"^(https?:\/\/).+"))
        {
            _logger.LogError("Введіть коректний URL, який починається з http:// або https://");
            return BadRequest("Введіть коректний URL, який починається з http:// або https://");
        }

        if (threads < 1) threads = 1;
        if (threads > 100) threads = 100;

        _cts = new CancellationTokenSource();
        var cancellationToken = _cts.Token;

        _baseUrl = baseUrl;
```

```
        _logger.LogInformation("Початок перебору BaseUrl={BaseUrl}, Dictionary={Dictionary}, Depth={Depth}, Delay={Delay}, Threads={Threads}",
            baseUrl, dictionaryName, depth, delay, threads);

        _dictionaryProvider.SwitchDictionary(dictionaryName);

        if (_directoryScanner is DirectoryScanner ds) {
            ds.Threads = threads;
        }

        var userAgent = _userAgentProvider.GetUserAgent(selectedUserAgent);

        var throttlingService = new ThrottlingService();

        var paths = await _dictionaryProvider.GetEntriesAsync();

        try
        {
            if (!string.IsNullOrEmpty(userAgent) && _httpRequester is HttpRequester  hr)
            {
                hr.FixedUserAgent = userAgent;
            }

            var result = await _directoryScanner.ScanAsync(selectedUserAgent, baseUrl, paths, depth, cancellationToken, throttlingService);

            var found = result.Distinct().ToList();

            if (!found.Any())
            {
                _logger.LogWarning("Жодної директорії не знайдено");
                return Json(new
                {
                    message = "Сканування завершено. Директорії не знайдено",
                    found = new List<string>()
                });
            }

            var msg = $"Перебір директорій закінчений. Знайдено: {found.Count} директорій";
            _logger.LogInformation(msg);

            return Json(new
            {
                message = msg,
                found = found
            });
        }
        catch (OperationCanceledException)
        {
            _logger.LogWarning("Сканування було зупинено користувачем.");
            return Json(new
            {
                message = "Сканування зупинено користувачем.",
                found = new List<string>()
            });
        }
    }


    [HttpPost]
    public IActionResult StopFuzzing()
    {
```

```
        _cts.Cancel();
        _logger.LogInformation("Користувач зупинив сканування");

        var found = _directoryScanner.GetFoundDirectories().Distinct().ToList();

        var msg = $"Сканування зупинено. Знайдено: {found.Count} директорій";

        return Json(new
        {
            message = msg,
            found = found
        });
    }

    [HttpGet]
    public IActionResult ShowFoundDirectories()
    {
        var found = _directoryScanner.GetFoundDirectories().Distinct().ToList();

        return View(found);
    }

    [HttpGet]
    public IActionResult GetLogs()
    {
        var logs = InMemoryLogProvider.GetLogs();
        return Json(new { logs = logs });
    }

    public IActionResult Privacy()
    {
        return View();
    }

    [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
    public IActionResult Error()
    {
        return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });
    }
}

public class SettingsController : Controller
{
    private readonly IDictionaryProvider _dictionaryProvider;
    private readonly ILogger<SettingsController> _logger;

    public SettingsController(IDictionaryProvider dictionaryProvider, ILogger<SettingsController> logger)
    {
        _dictionaryProvider = dictionaryProvider;
        _logger = logger;
    }

    public IActionResult Index()
    {
        var fuzzingDictionaries = _dictionaryProvider.GetFuzzingDictionaries();
        return View(fuzzingDictionaries);
    }

    [HttpGet]
    public async Task<IActionResult> GetDictionaryContent(string dictName)
    {
        try
        {
```

```
            _dictionaryProvider.SwitchDictionary(dictName);
            var entries = (await _dictionaryProvider.GetEntriesAsync()).ToList();
            return Json(new { success = true, entries = entries });
        }
        catch (Exception ex)
        {
            _logger.LogError("Помилка при виборі словника {Name}: {Message}.", dictName, ex.Message);
            return Json(new { success = false, error = ex.Message });
        }
    }

    [HttpGet]
    public IActionResult ShowDictionary(string dictName)
    {
        try
        {
            _dictionaryProvider.SwitchDictionary(dictName);
            var entries = _dictionaryProvider.GetEntriesAsync();
            ViewBag.DictionaryName = dictName;
            return View("ShowDictionary", entries);
        }
        catch (Exception ex)
        {
            _logger.LogError("Помилка при виборі словника {Name}: {Message}", dictName, ex.Message);
            ModelState.AddModelError("", ex.Message);
            return RedirectToAction("Index");
        }
    }
}

public interface IDictionaryProvider
{
    Task<IEnumerable<string>> GetEntriesAsync();
    IEnumerable<string> GetAvailableDictionaries();
    IEnumerable<string> GetFuzzingDictionaries();
    public void SwitchDictionary(string dictionaryName);
}

public interface IDirectoryScanner
{
    Task<IEnumerable<string>> ScanAsync(
        string selectedUserAgent, string baseUrl, IEnumerable<string> paths, int depth,
        CancellationToken cancellationToken, IThrottlingService throttlingService);
    Task<IEnumerable<string>> ScanAsync(
        string selectedUserAgent, string baseUrl, int depth, CancellationToken cancellationToken,
        IThrottlingService throttlingService);
    IEnumerable<string> GetFoundDirectories();


public interface IFormScanner
{
    Task<IEnumerable<FormFound>> FindFormsAsync(string htmlContent, string baseUrl);
}

public interface IHttpRequester
{
    Task<HttpResponseMessage> SendRequestAsync(string selectedUserAgent, string url, HttpContent? content = null,
int? customDelay = null, CancellationToken cancellationToken = default);
}

public interface IThrottlingService
 {
     Task ApplyDelayAsync();
```

```
 }

public interface IUserAgentProvider
{
   string GetUserAgent(string selectedUserAgent);
   Task<string[]> GetAllUserAgentsAsync();
}

public class DirectoriesRequest
{
   public List<string> Directories { get; set; } = new();
   public int Threads { get; set; } = 1;

   public string ScanType { get; set; }

}

public class DirectoryNode
{
   public string Url { get; set; }
   public List<DirectoryNode> Children { get; set; } = new List<DirectoryNode>();

   public IEnumerable<DirectoryNode> GetLeafNodes()
   {
      if (!Children.Any())
      {
         return new[] { this };
      }

      return Children.SelectMany(c => c.GetLeafNodes());
   }
}

public class FieldDescriptor
{
   public string Name { get; set; }
   public string Type { get; set; }
   public bool IsRequired {  get; set; }
   public bool IsHidden { get; set; }
   public string Placeholder { get; set; }
   public string Accept { get; set; }
}

public class FormFound //описує знайдену форму на сторінці
{
   public string Action {  get; set; }
   public string Method { get; set; }
   public List<FieldDescriptor> InputFields { get; set; } = new();
   public List<FieldDescriptor> FileFields { get; set; } = new();

   public IEnumerable<FieldDescriptor> GetAllFields() => InputFields.Concat(FileFields);
}

public class FormTestResult
{
   public FormFound Form {  get; set; }
   public string Payload {  get; set; }
   public string Response {  get; set; }
   public System.Net.HttpStatusCode StatusCode { get; set; }
}

Config.json
{
```

```
  "UserAgent": {
    "FilePath": "Infrastructure/Dictionaries/user-agents.txt"
  },
  "Dictionaries": {
    "Fuzzing": {
      "Darkweb2017-top10000": "Infrastructure/Dictionaries/darkweb2017-top10000.txt",
      "Directory-big": "Infrastructure/Dictionaries/directory-big.txt",
      "WordPress-dirs": "Infrastructure/Dictionaries/WordPress-dirs.txt",
      "dict": "Infrastructure/Dictionaries/dict.txt"
    }
  }
}

public class HtmlParser
{
    private readonly IBrowsingContext _browsingContext;

    public HtmlParser()
    {
        var config = Configuration.Default.WithDefaultLoader();
        _browsingContext = BrowsingContext.New(config);
    }

    public async Task<IDocument> ParseDocumentAsync(string htmlContent)
    {
        if (string.IsNullOrWhiteSpace(htmlContent))
            throw new ArgumentException("HTML не може бути пустим", nameof(htmlContent));

        return await _browsingContext.OpenAsync(req => req.Content(htmlContent));
    }
}

public class DirectoryScanner : IDirectoryScanner
{
    private readonly IHttpRequester _httpRequester;
    private readonly IDictionaryProvider _dictionaryProvider;
    private readonly ILogger<DirectoryScanner> _logger;
    private readonly IEnumerable<HttpStatusCode> _successfulCodes;
    private ConcurrentBag<string> _foundDirectories = new ConcurrentBag<string>();
    private ConcurrentDictionary<string, bool> _visited = new ConcurrentDictionary<string,
bool>(StringComparer.OrdinalIgnoreCase);
    public int Threads { get; set; } = 3;
    private readonly int _maxDepth;
    private readonly IEnumerable<HttpStatusCode> _successfulCode;

    public DirectoryScanner(
        IHttpRequester httpRequester,
        IDictionaryProvider dictionaryProvider,
        ILogger<DirectoryScanner> logger,
        int maxDegreeOfParallelism = 3,
        int maxDepth = 1,
        IEnumerable<HttpStatusCode>? successfulCodes = null)
    {
        _httpRequester = httpRequester ?? throw new ArgumentNullException(nameof(httpRequester));
        _dictionaryProvider = dictionaryProvider ?? throw new ArgumentNullException(nameof(dictionaryProvider));
        _logger = logger ?? throw new ArgumentNullException(nameof(logger));
        this.Threads = maxDegreeOfParallelism;
        _maxDepth = maxDepth;
        _successfulCodes = successfulCodes ?? new[]
        {
            HttpStatusCode.OK,
            HttpStatusCode.Forbidden,
            HttpStatusCode.Unauthorized,
```

```
            HttpStatusCode.MovedPermanently,
            HttpStatusCode.Found
        };
    }

    public async Task<IEnumerable<string>> ScanAsync(string selectedUserAgent, string baseUrl, IEnumerable<string> paths, int depth,CancellationToken cancellationToken, IThrottlingService throttlingService)
    {
        return await ScanInternalAsync(selectedUserAgent, baseUrl, paths, depth, cancellationToken);
    }

    public async Task<IEnumerable<string>> ScanAsync(string selectedUserAgent, string baseUrl, int depth, CancellationToken cancellationToken, IThrottlingService throttlingService)
    {
        var paths = (await _dictionaryProvider.GetEntriesAsync());
        return await ScanInternalAsync(selectedUserAgent, baseUrl, paths, depth, cancellationToken);
    }

    private async Task<IEnumerable<string>> ScanInternalAsync(string selectedUserAgent,string baseUrl, IEnumerable<string> paths, int maxDepth, CancellationToken cancellationToken)
    {
        if (string.IsNullOrWhiteSpace(baseUrl))
            throw new ArgumentException("Base URL не може бути пустим", nameof(baseUrl));

        if (!Uri.TryCreate(baseUrl, UriKind.Absolute, out var baseUri))
            throw new ArgumentException("Некоректний формат Base URL", nameof(baseUrl));

        _logger.LogInformation("Починаємо сканування директорій для {BaseUrl}...", baseUrl);

        _visited.Clear();
        _foundDirectories = new ConcurrentBag<string>();
        _visited.TryAdd(baseUrl, true);

        var urls = paths
            .Select(p => new Uri(baseUri, p).ToString())
            .Distinct(StringComparer.OrdinalIgnoreCase)
            .ToList();

        _visited.TryAdd(baseUrl, true);

        var semaphore = new SemaphoreSlim(Threads, Threads);
        var tasks = new List<Task>();

        foreach (var url in urls)
        {
            cancellationToken.ThrowIfCancellationRequested();

            tasks.Add(Task.Run(async () =>
            {
                await semaphore.WaitAsync(cancellationToken);
                try
                {
                    await ProcessUrlAsync(selectedUserAgent, baseUrl, url, baseUri, 1, maxDepth, paths, cancellationToken);
                }
                finally
                {
                    semaphore.Release();
                }
            }, cancellationToken));
        }

        await Task.WhenAll(tasks);
```

```
        var result = _foundDirectories.Distinct(StringComparer.OrdinalIgnoreCase).ToList();
        return result;
    }

    private async Task ProcessUrlAsync(string selectedUserAgent,string baseUrl, string url, Uri baseUri, int currentDepth,
int maxDepth, IEnumerable<string> paths, CancellationToken cancellationToken)
    {
        if (!_visited.TryAdd(url, true))
        {
            return;
        }

        if (maxDepth > 0 && currentDepth > maxDepth)
        {
            _logger.LogInformation("Досягнуто максимальну глибину для URL: {Url}", url);
            return;
        }

        if (maxDepth <= 0)
        {
            _logger.LogInformation("Глибина повина бути більша нуля");
            return;
        }

        _logger.LogInformation("Запит до: {Url} (глибина: {Depth})", url, currentDepth);

        cancellationToken.ThrowIfCancellationRequested();
        var response = await _httpRequester.SendRequestAsync(selectedUserAgent, url, null, null, cancellationToken);

        if (IsSuccessfulStatus(response.StatusCode))
        {
            _foundDirectories.Add(url);
            _logger.LogInformation("Додано: {url} зі статусом {StatusCode}", url, (int)response.StatusCode);

            if (!url.EndsWith("/"))
            {
                url += "/";
            }

            foreach (var path in paths)
            {
                var newUri = new Uri(new Uri(url), path).ToString();
                await ProcessUrlAsync(selectedUserAgent, baseUrl, newUri, baseUri, currentDepth + 1, maxDepth, paths,
cancellationToken);
            }
        }
        else
        {
            _logger.LogInformation("URL {Url} повернув {StatusCode}, пропускаємо...", url, (int)response.StatusCode);
        }
    }

    private bool IsSuccessfulStatus(HttpStatusCode statusCode)
    {
        return statusCode == HttpStatusCode.OK ||                // 200: Успішний запит
               statusCode == HttpStatusCode.Unauthorized ||      // 401: Потрібна аутентифікація
               statusCode == HttpStatusCode.Forbidden ||         // 403: Доступ заборонено
               statusCode == HttpStatusCode.MovedPermanently ||  // 301: Постійне перенаправлення
               statusCode == HttpStatusCode.Found ||             // 302: Тимчасове перенаправлення
               statusCode == HttpStatusCode.SeeOther ||          // 303: Альтернативний ресурс
               statusCode == HttpStatusCode.NotModified ||       // 304: Ресурс не змінювався
               statusCode == HttpStatusCode.TemporaryRedirect || // 307: Тимчасове перенаправлення
               statusCode == HttpStatusCode.PermanentRedirect;   // 308: Постійне перенаправлення
```

```
    }

    public IEnumerable<string> GetFoundDirectories() => _foundDirectories;
}

public class FileDictionaryProvider : IDictionaryProvider
{
    private string _currenyDictionaryPath;
    private readonly ILogger<FileDictionaryProvider> _logger;
    private readonly Dictionary<string, string> _fuzzingDictionaries;

    public FileDictionaryProvider(IConfiguration configuration, ILogger<FileDictionaryProvider> logger)
    {
        _logger = logger ?? throw new ArgumentNullException(nameof(logger));

        _fuzzingDictionaries = configuration.GetSection("Dictionaries:Fuzzing").Get<Dictionary<string, string>>()
            ?? throw new Exception("Словники для фазингу відсутні");

        _currenyDictionaryPath = _fuzzingDictionaries.First().Value;
        ValidateFileExists(_currenyDictionaryPath);
    }

    public IEnumerable<string> GetAvailableDictionaries()
    {
        _logger.LogDebug("Отримання списку доступних словників");
        return _fuzzingDictionaries.Keys;
    }

    public IEnumerable<string> GetFuzzingDictionaries()
    {
        _logger.LogDebug("Отримання списку словників для фаззингу...");
        return _fuzzingDictionaries.Keys;
    }

    public void SwitchDictionary(string dictionaryName)
    {
        if (!_fuzzingDictionaries.TryGetValue(dictionaryName, out var dictionaryPath))
        {
            _logger.LogWarning("Словник з назвою {Name} не знайдений!", dictionaryName);
        }

        _currenyDictionaryPath = dictionaryPath;

        ValidateFileExists(_currenyDictionaryPath);
    }

    public async Task<IEnumerable<string>> GetEntriesAsync()
    {
        _logger.LogDebug("Читання словника з {Path}", _currenyDictionaryPath);
        var lines = await File.ReadAllLinesAsync(_currenyDictionaryPath);
        return lines
            .Where(line => !string.IsNullOrWhiteSpace(line))
            .Select(line => line.Trim());
    }

    private void ValidateFileExists(string filePath)
    {
        if (!File.Exists(filePath))
        {
            _logger.LogError("Файл словника {Path} не існує", filePath);
            throw new FileNotFoundException($"Файл словника {filePath} не існує");
        }
```

```
    }
}

public class FormScanner : IFormScanner
{
    private readonly HtmlParser _htmlParser;
    private readonly ILogger _logger;
    private readonly IHttpRequester _httpRequester;

    public FormScanner(HtmlParser htmlParser, ILogger<FormScanner> logger, IHttpRequester httpRequester)
    {
        _htmlParser = htmlParser ?? throw new ArgumentNullException(nameof(htmlParser));
        _logger = logger ?? throw new ArgumentNullException(nameof(logger));
        _httpRequester = httpRequester ?? throw new ArgumentNullException(nameof(_httpRequester));
    }

    public async Task<IEnumerable<FormFound>> FindFormsAsync(string htmlContent, string baseUrl)
    {
        if (string.IsNullOrEmpty(htmlContent))
        {
            _logger.LogError("HTML контент не може бути пустим");
            throw new ArgumentNullException("HTML content не може бути пустим", nameof(htmlContent));
        }

        // Парсимо HTML
        var htmlDocument = await _htmlParser.ParseDocumentAsync(htmlContent);

        // Знаходимо всі <form>
        var formElements = htmlDocument.QuerySelectorAll("form");
        var forms = new List<FormFound>();

        // Для кожної знайденої форми створюємо наш FormFound
        foreach (var formElement in formElements)
        {
            // Визначаємо Action (повна або відносна адреса)
            var formAction = formElement.GetAttribute("action") ?? string.Empty;
            var absoluteActionUrl = string.IsNullOrEmpty(formAction)
                ? baseUrl
                : new Uri(new Uri(baseUrl), formAction).ToString();

            // method="GET" або "POST" (за замовчуванням GET)
            var method = formElement.GetAttribute("method")?.ToUpper() ?? "GET";

            // --- Обробка інпутів (input, textarea, select) ---
            var inputDescriptors = new List<FieldDescriptor>();

            // Відберемо всі елементи, що можуть бути полями введення (включно із select і textarea)
            var inputElements = formElement.QuerySelectorAll("input, textarea, select");
            foreach (var e in inputElements)
            {
                // "name" або "id" (якщо name не задано)
                var nameAttr = e.GetAttribute("name") ?? e.GetAttribute("id") ?? "unknown-field";

                // Type: якщо input не має type, то припускаємо "text". Якщо це textarea => "textarea", select => "select"
                var tagName = e.TagName.ToLower();
                var typeAttr = e.GetAttribute("type")?.ToLower()
                            ?? (tagName == "textarea" ? "textarea"
                              : tagName == "select" ? "select"
                              : "text");

                var isRequired = e.HasAttribute("required");
                var isHidden = typeAttr == "hidden";
                var placeholder = e.GetAttribute("placeholder") ?? "";
```

```
            var accept = e.GetAttribute("accept") ?? "";

            inputDescriptors.Add(new FieldDescriptor
            {
                Name = nameAttr,
                Type = typeAttr,
                IsRequired = isRequired,
                IsHidden = isHidden,
                Placeholder = placeholder,
                Accept = accept
            });
        }

        // --- Окремо обробляємо FileFields (input[type=file]) ---
        // Якщо хочемо фізично розділяти, можна окремо зберігати:
        var fileElements = formElement.QuerySelectorAll("input[type=file]");
        var fileDescriptors = new List<FieldDescriptor>();

        foreach (var fe in fileElements)
        {
            var nameAttr = fe.GetAttribute("name") ?? fe.GetAttribute("id") ?? "unknown-file";
            var typeAttr = "file";
            var isRequired = fe.HasAttribute("required");
            var placeholder = fe.GetAttribute("placeholder") ?? "";
            var accept = fe.GetAttribute("accept") ?? "";

            fileDescriptors.Add(new FieldDescriptor
            {
                Name = nameAttr,
                Type = typeAttr,
                IsRequired = isRequired,
                IsHidden = false, // файл не буває hidden
                Placeholder = placeholder,
                Accept = accept
            });
        }

        // Створюємо саму форму
        var form = new FormFound
        {
            Action = absoluteActionUrl,
            Method = method,
            InputFields = inputDescriptors,
            FileFields = fileDescriptors
        };

        forms.Add(form);
    }

    return forms;
  }
}

public class HttpRequester : IHttpRequester
{
  private readonly HttpClient _httpClient;
  private readonly IThrottlingService _throttlingService;
  private readonly IUserAgentProvider _userAgentProvider;

  public HttpRequester(HttpClient httpClient, IUserAgentProvider userAgentProvider, IThrottlingService
throttlingService)
  {
    _httpClient = httpClient ?? throw new ArgumentNullException(nameof(httpClient));
```

```
        _userAgentProvider = userAgentProvider ?? throw new ArgumentNullException(nameof(userAgentProvider));
        _throttlingService = throttlingService ?? throw new ArgumentNullException(nameof(throttlingService));
    }

    public string FixedUserAgent { get; set; }

    public async Task<HttpResponseMessage> SendRequestAsync(string selectedUserAgent, string url, HttpContent content = null, int? customDelay = null, CancellationToken cancellationToken = default)
    {
        if (string.IsNullOrWhiteSpace(url))
            throw new ArgumentNullException("URL не може бути порожнім", nameof(url));

        if (customDelay.HasValue && customDelay.Value >= 0)
        {
            await Task.Delay(customDelay.Value, cancellationToken);
        }
        else
        {
            await _throttlingService.ApplyDelayAsync();
        }

        var userAgent = _userAgentProvider.GetUserAgent(selectedUserAgent); //налаштування юзер-агента

        var request = new HttpRequestMessage(content == null ? HttpMethod.Get : HttpMethod.Post, url)
        {
            Content = content
        };

        request.Headers.UserAgent.Clear();
        request.Headers.UserAgent.ParseAdd(userAgent);

        var response = await _httpClient.SendAsync(request, cancellationToken);

        return response;
    }
}

public class InMemoryLogProvider : ILoggerProvider
 {
    private static ConcurrentQueue<string> _logs = new ConcurrentQueue<string>();

    public ILogger CreateLogger(string categoryName)
    {
        return new InMemoryLogger(categoryName);
    }

    public void Dispose() { }

    public static IEnumerable<string> GetLogs()
    {
        return _logs.ToArray();
    }

    internal class InMemoryLogger : ILogger
    {
        private readonly string _categoryName;

        public InMemoryLogger(string categoryName)
        {
            _categoryName = categoryName;
        }

        public bool IsEnabled(LogLevel logLevel) => true;
```

```
        public IDisposable BeginScope<TState>(TState state) => null;

        public void Log<TState>(LogLevel logLevel, EventId eventId, TState state, Exception exception,
            Func<TState, Exception, string> formatter)
        {
            if (!IsEnabled(logLevel))
                return;

            var message = $"{DateTime.Now:HH:mm:ss} [{logLevel}]: {formatter(state, exception)}"; //{_categoryName} -
видалив з рядкка

            InMemoryLogProvider._logs.Enqueue(message);

            while (InMemoryLogProvider._logs.Count > 5000)
            {
                InMemoryLogProvider._logs.TryDequeue(out _);
            }
        }
    }
}

public class ThrottlingService : IThrottlingService
{
    private readonly int _delay = 0;

    public ThrottlingService()
    {
    }

    public async Task ApplyDelayAsync()
    {

        await Task.Delay(_delay);
    }
}

public class UserAgentProvider : IUserAgentProvider
{
    private readonly string[] _userAgents;
    private readonly ILogger<UserAgentProvider> _logger;

    public UserAgentProvider(string userAgentFilePath, ILogger<UserAgentProvider> logger)
    {
        _logger = logger ?? throw new ArgumentNullException(nameof(logger));

        if (!File.Exists(userAgentFilePath))
        {
            _logger.LogError("User-agent файл не знайдено за шляхом: {Path}", userAgentFilePath);
            throw new FileNotFoundException($"User-Agent файл не знайдено");
        }

        _userAgents = File.ReadAllLines(userAgentFilePath)
            .Where(line => !string.IsNullOrWhiteSpace(line))
            .ToArray();

        if (_userAgents.Length == 0)
        {
            _logger.LogError("Файл {Path} не містить жодного валідного User-agent", userAgentFilePath);
            throw new InvalidOperationException("Не знайдено жодного агента");
        }
```

```
        _logger.LogInformation("User-Agent ініціалізовано в динамічному режимі з {Count} варіантами агентів",
_userAgents.Length);
    }

    public async Task<string[]> GetAllUserAgentsAsync()
    {
        return await Task.FromResult(_userAgents);
    }

    public string GetUserAgent(string selectedUserAgent)
    {
        if (string.IsNullOrWhiteSpace(selectedUserAgent))
        {
            _logger.LogError("Статичний User-agent не може бути пустим");
            throw new ArgumentException("Статичний UA не може бути пустим", nameof(selectedUserAgent));
        }

        _logger.LogDebug("Використовується статичний User-Agent: {User-Agent}", selectedUserAgent);
        return selectedUserAgent.Trim();
    }
}

@{
    ViewData["Title"] = "Сканування Директорій";
    var directories = Model as List<string>;
}

<h2 style="margin-left: 930px; margin-top: 30px">Сканування Директорій</h2>

<div style="background: #e3e3e3;
            border: 1px solid #3a3a52;
            border-radius: 8px;
            padding: 15px;
            margin-top: 50px;
            height: 200px;
            overflow-y: auto;
            color: #444444;
            font-family: Arial, sans-serif;
            font-size: 14px;">
    <ul>
        @if (directories != null && directories.Any())
        {
            foreach (var dir in directories)
            {
                <li style="color: #444444; margin-top: 10px">@dir</li>
            }
        }
        else
        {
            <p style="color: #444444; margin-left: 850px; margin-top: 60px">
                Директорії відсутні для сканування.
                Поверніться до вибору директорій.
            </p>
        }
    </ul>
</div>

<div style="margin-bottom: 15px; margin-top: 10px">
    <label for="threads" style="color: #444444; font-weight: bold;">Кількість потоків:</label>
    <input type="number" id="threads" min="1" max="100" value="3"
           style="padding: 5px; font-size: 14px; border-radius: 4px;
                  border: 1px solid #3a3a52; background: #e3e3e3; color: #444444;
                  margin-left: 10px;" />
```

```
</div>

<div style="margin-bottom: 15px;">
    <button id="startScanButton"
            style="padding: 5px 15px; margin-top: 20px; font-size: 14px; border: none;
                   border-radius: 6px; background-color: #7c5cff; color: white; cursor: pointer;">
        Почати сканування форм
    </button>
</div>

<div id="results"
     style="background: #e3e3e3;
            border: 1px solid #3a3a52;
            border-radius: 8px;
            padding: 15px;
            margin-top: 50px;
            height: 580px;
            overflow-y: auto;
            color: #444444;
            font-family: Arial, sans-serif;
            font-size: 14px;">
    <div id="scanningLogs"
         style="border: 1px solid #3a3a52; border-radius: 4px; padding: 10px; background: #e3e3e3;">
        <p style="color: #888;">Результати сканування будуть відображені тут...</p>
        <div id="spinner" style="display: none; text-align: center; margin-top: 20px;">
            <p style="color: #444444;">Сканування виконується...</p>
        </div>
    </div>
</div>

@section Scripts {
    <script>
        document.getElementById("startScanButton")
            .addEventListener("click", () => {
                startScanning("/Actions/ScanDirectories", "forms");
            });

        function startScanning(endpoint, scanType) {
            const directories = @Html.Raw(Json.Serialize(directories));
            const threads = document.getElementById("threads").value;

            if (!directories || directories.length === 0) {
                alert("Список директорій порожній. Поверніться до вибору директорій.");
                return;
            }

            fetch(endpoint, {
                method: "POST",
                headers: { "Content-Type": "application/json" },
                body: JSON.stringify({
                    directories,
                    threads: parseInt(threads),
                    scanType
                })
            })
            .then(response => {
                if (!response.ok) throw new Error("Помилка на сервері");
                return response.json();
            })
            .then(data => {
                spinner.style.display = "none";
                displayResults(data);
            })
```

```
    .catch(error => {
        spinner.style.display = "none";
        console.error("Помилка:", error);
        alert("Помилка при запуску сканування.");
    });
}

function displayResults(data) {
    const resultsDiv = document.getElementById("scanningLogs");
    resultsDiv.innerHTML = "";

    if (!data || data.length === 0) {
        resultsDiv.innerHTML = "<p style='color: #575757;'>Результати відсутні</p>";
        return;
    }

    resultsDiv.innerHTML += "<h3 style='color: #575757;'>Результати сканування</h3>";

    data.forEach(result => {
        const source    = result.source  || "Невідомо";
        const actionUrl = result.action  || "#";
        const method    = result.method  || "GET";

        const inputFields  = Array.isArray(result.inputFields) ? result.inputFields : [];
        const fileFields   = Array.isArray(result.fileFields)  ? result.fileFields  : [];

        const inputFieldsHtml = inputFields.map(f => {
            return `
                <li>
                    <strong>${f.name}</strong>
                    (type: ${f.type},
                    required: ${f.isRequired},
                    hidden: ${f.isHidden},
                    placeholder: "${f.placeholder}",
                    accept: "${f.accept}")
                </li>
            `;
        }).join("");

        const fileFieldsHtml = fileFields.map(f => {
            return `
                <li>
                    <strong>${f.name}</strong>
                    (type: ${f.type},
                    required: ${f.isRequired},
                    hidden: ${f.isHidden},
                    accept: "${f.accept}")
                </li>
            `;
        }).join("");

        resultsDiv.innerHTML += `
          <div style="margin-bottom: 15px; padding: 10px; border: 1px solid #444; border-radius: 5px;">
            <h4>Джерело: <a href="${source}" target="_blank">${source}</a></h4>
            <p>Action: <a href="${actionUrl}" target="_blank">${actionUrl}</a></p>
            <p>Метод: ${method}</p>

            <p><u>Поля вводу</u>:</p>
            ${
              inputFieldsHtml
                ? `<ul style="margin-left:20px;">${inputFieldsHtml}</ul>`
                : "<em>Немає полів вводу</em>"
            }
```

```
                <p><u>Поля завантаження файлів</u>:</p>
                ${
                  fileFieldsHtml
                    ? `<ul style="margin-left:20px;">${fileFieldsHtml}</ul>`
                    : "<em>Немає file-полів</em>"
                }
              </div>
            `;
        });
    }
  </script>
}

@{
  ViewData["Title"] = "Fuzzing Configuration";
  var directories = ViewBag.Dictionaries as IEnumerable<string>;
  var userAgents = ViewBag.UserAgents as IEnumerable<string>;
}

<h2 style="margin-left: 850px; margin-top: 30px">Налаштування перебору директорій</h2>

<form id="fuzzingForm">
  <div>
    <label for="baseUrl">Цільова URL:</label>
    <input type="text" id="baseUrl" name="baseUrl" placeholder="http://example.com" required/>
  </div>
  <div>
    <label for="dictionaryName">Виберіть словник:</label>
    <select id="dictionaryName" name="dictionaryName">
      @foreach (var dict in directories ?? Enumerable.Empty<string>())
      {
        <option value="@dict">@dict</option>
      }
    </select>
  </div>

  <div>
    <label for="depth">Глибина пошуку:</label>
    <input type="number" id="depth" name="depth" min="1" value="1" />
  </div>
  <div>
    <label for="delay">Затримка (ms): 2000 </label>
    <input type="hidden" id="delay" name="delay" min="0" value="0" />
  </div>
  <div>
    <label>Кількість потоків:</label>
    <input type="number" name="threads" min="1" max="100" value="3"/>
  </div>
  <div>
    <label for="selectedUserAgent">User-Agent:</label>
    <select id="selectedUserAgent" name="selectedUserAgent" style="width:160px;">
      @foreach (var ua in userAgents ?? Enumerable.Empty<string>())
      {
        <option value="@ua">@ua</option>
      }
    </select>
  </div>
  <div>
    <button type="button" id="startButton" style="padding: 5px 15px; font-size: 14px; border: none;
    border-radius: 4px; background-color: #7c5cff;
    color: white; cursor: pointer;">
      Старт
```

```
        </button>
        <button type="button" id="stopButton" style="padding: 5px 15px; font-size: 14px; border: none;
        border-radius: 4px; background-color: #7c5cff;
        color: white; cursor: pointer;">
            Стоп
        </button>
    </div>
</form>

<h3>Logs</h3>
<div id="logsContainer" style="background:#d9d9d9;
    border:1px solid #3a3a52; border-radius:8px; padding:10px;
    margin-bottom:20px; height:260px; overflow-y:auto; color:#b5bdb7;
    font-family:monospace; font-size:14px;">
    <div id="logs">
        <p style="text-align:center; font-style:italic; color:#626363;">Поки що немає логів</p>
    </div>
</div>

<h3 style="display: inline-block; margin-right: 15px;">Знайдені директорії</h3>

<button type="button" id="submitSelected"
        style="padding: 8px 20px; font-size: 14px; border: none;
        border-radius: 5px; background-color: #7c5cff;
        color: white; cursor: pointer; transition: background-color 0.3s;">
    Надіслати вибрані
</button>

<div id="foundDirsContainer"
     style="background: #d9d9d9;
            border: 1px solid #3a3a52;
            border-radius: 8px;
            padding: 15px;
            margin-top: 15px;
            height: 260px;
            overflow-y: auto;
            color: #f5f5f5;
            font-family: Arial, sans-serif;
            font-size: 14px;">

    <form id="directoryForm">
        <div style="margin-bottom: 10px; display: flex; align-items: center; justify-content: flex-end; padding: 5px 0;
border-bottom: 1px solid #3a3a52;">
            <label for="selectAll" style="color: #e0e0e0; font-weight: bold; cursor: pointer; margin: 0;">
                Вибрати всі
            </label>
            <input type="checkbox" id="selectAll" style="cursor: pointer;" />
        </div>
        <ul id="foundDirs" style="list-style: none; padding: 0; margin: 0; text-align: right;">
            <li style="text-align: center; font-style: italic; color: #888;">
                Поки що немає знайдених директорій
            </li>
        </ul>
    </form>
</div>

@section Scripts {
    <script src="https://d3js.org/d3.v6.min.js"></script>
    <script>
        const startButton = document.getElementById("startButton");
        const stopButton = document.getElementById("stopButton");
        const logsDiv = document.getElementById('logs');
        const foundDirsUL = document.getElementById('foundDirs');
```

```
    const submitSelectionButton = document.getElementById('submitSelected');
    const selectAllCheckbox = document.getElementById('selectAll');

    function colorizeLogLine(line) {
      if (line.includes(" 200")) {
        return `<span style="color:green">${line}</span>`;
      } else if (line.includes(" 401")) {
        return `<span style="color:purple">${line}</span>`;
      } else if (line.includes(" 403")) {
        return `<span style="color:orange">${line}</span>`;
      } else if (line.includes(" 301")) {
        return `<span style="color:blue">${line}</span>`;
      } else if (line.includes(" 302")) {
        return `<span style="color:teal">${line}</span>`;
      } else if (line.includes(" 303")) {
        return `<span style="color:darkcyan">${line}</span>`;
      } else if (line.includes(" 304")) {
        return `<span style="color:gray">${line}</span>`;
      } else if (line.includes(" 307")) {
        return `<span style="color:darkblue">${line}</span>`;
      } else if (line.includes(" 308")) {
        return `<span style="color:navy">${line}</span>`;
      } else if (line.includes(" 404")) {
        return `<span style="color:red">${line}</span>`;
      } else {
        return `<span style="color:#3d3d3d">${line}</span>`;
      }
    }

    function renderFoundDirectories(dirs) {
      if (!dirs || dirs.length === 0) {
        foundDirsUL.innerHTML = `
          <li style="text-align:center; font-style:italic; color:#888;">
            Директорій не знайдено
          </li>`;
        return;
      }

      let html = "";
      dirs.forEach((dir, index) => {
      html += `
        <li style="display: flex; align-items: center; justify-content: flex-end; padding: 8px 0; border-bottom: 1px solid
#3a3a52;">
          <label for="dir_${index}" style="
              color: #575757;
              font-weight: normal;
              margin-right: 10px;
              text-align: right;">
            ${dir}
          </label>
          <input type="checkbox" id="dir_${index}" name="selectedDirs"
              value="${dir}" style="cursor: pointer;" />
        </li>`;
      });

      foundDirsUL.innerHTML = html;
    }

    selectAllCheckbox.addEventListener("change", function (event) {
      const checkboxes = document.querySelectorAll("#foundDirs input[type='checkbox']");
      checkboxes.forEach(cb => {
        cb.checked = event.target.checked;
      });
```

```
});

submitSelectionButton.addEventListener("click", () => {
    const selectedDirs = Array.from(
        document.querySelectorAll("#foundDirs input[type='checkbox']:checked")
    ).map(cb => cb.value);

    if (selectedDirs.length === 0) {
        alert("Будь ласка, виберіть хоча б одну директорію.");
        return;
    }

    fetch('/Actions/ProcessSelectedDirectories', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({ directories: selectedDirs, threads: 3 })
    })
    .then(response => {
        if (!response.ok) throw new Error("Помилка під час обробки запиту");
        return response.json();
    })
    .then(data => {
        alert("Вибрані директорії успішно передано!\n\n" + JSON.stringify(data, null, 2));
    })
    .catch(error => {
        console.error("Помилка:", error);
        alert("Не вдалося передати вибрані директорії.");
    });
});

startButton.addEventListener("click", function () {
    const formData = new FormData(document.getElementById("fuzzingForm"));

    fetch('/Fuzzing/StartFuzzing', {
        method: 'POST',
        body: formData
    })
    .then(response => {
        if (!response.ok) throw new Error("Помилка на сервері");
        return response.json();
    })
    .then(data => {
        logsDiv.innerHTML = `<span style="color:#888;">Сканування розпочато...</span><br/>`;
        logsDiv.innerHTML += `<span style="color:#888;">${data.message}</span><br/>`;

        renderFoundDirectories(data.found);
    })
    .catch(error => {
        console.error('Error:', error);
        logsDiv.innerText = "Помилка при запуску сканування.";
    });
});

stopButton.addEventListener("click", function () {
    fetch('/Fuzzing/StopFuzzing', {
        method: 'POST'
    })
    .then(response => response.json())
    .then(data => {
        logsDiv.innerHTML += `<br/><span style="color:#888;">Сканування зупинено.</span><br/>`;
        renderFoundDirectories(data.found);
```

```
            })
            .catch(error => {
                console.error('Error:', error);
                logsDiv.innerText = "Помилка при зупинці сканування.";
            });
        });

        setInterval(function () {
            fetch('/Fuzzing/GetLogs')
                .then(response => response.json())
                .then(data => {
                    if (data.logs && data.logs.length > 0) {
                        let colored = data.logs
                            .map(line => colorizeLogLine(line))
                            .join('\n');
                        logsDiv.innerHTML = '<pre>' + colored + '</pre>';
                    } else {
                        logsDiv.innerHTML = '<p>Поки немає логів...</p>';
                    }
                })
                .catch(err => console.error('Error fetching logs:', err));
        }, 2000);
    </script>
}

@model IEnumerable<string>
@{
    ViewData["Title"] = "Словники для фаззингу";
}

<h2 style="text-align: center; margin-top: 30px;">Словники для фаззингу</h2>

<div style="display: flex; justify-content: center; margin-top: 20px;">
    <div style="width: 30%; margin-right: 20px;">
        <h3 style="margin-left: 220px">Доступні словники</h3>
        <ul style="list-style-type: none; padding: 0;">
            @foreach (var dict in Model)
            {
                <li style="margin-bottom: 10px;">
                    <button class="dict-button" data-dict="@dict"
                            style="width: 100%; padding: 10px; background: #7c5cff; color: white; border: none; border-radius:
15px; cursor: pointer;">
                        @dict
                    </button>
                </li>
            }
        </ul>
    </div>

    <div style="width: 60%;">
        <h3>Вміст вибраного словника</h3>
        <div id="dictContainer" style="height: 400px; background: #e0e0e0; border: 1px solid #444; border-radius: 8px;
padding: 15px; overflow-y: auto; color: #2d2d44 ; font-family: monospace; font-size: 14px;">
            <p style="text-align: center; color: #888;">Оберіть словник, щоб переглянути його вміст.</p>
        </div>
    </div>
</div>

@section Scripts {
    <script>
        document.addEventListener("DOMContentLoaded", function () {
            const buttons = document.querySelectorAll('.dict-button');
            const dictContainer = document.getElementById('dictContainer');
```

```
        buttons.forEach(button => {
            button.addEventListener('click', function () {
                const dictName = this.getAttribute('data-dict');

                dictContainer.innerHTML = '<p style="text-align: center; color: #888;">Завантаження...</p>';

                fetch(`/Settings/GetDictionaryContent?dictName=${encodeURIComponent(dictName)}`)
                    .then(response => response.json())
                    .then(data => {
                        if (data.success) {
                            dictContainer.innerHTML = '';

                            const pre = document.createElement('pre');
                            pre.style.whiteSpace = 'pre-wrap';
                            pre.style.wordBreak = 'break-word';

                            pre.textContent = data.entries
                                .map((entry, index) => `${index + 1}. ${entry}`)
                                .join('\n');

                            dictContainer.appendChild(pre);
                        } else {
                            dictContainer.innerHTML = `<p style="color: red; text-align: center;">Помилка: ${data.error}</p>`;
                        }
                    })
                    .catch(err => {
                        console.error('Error fetching dictionary content:', err);
                        dictContainer.innerHTML = `<p style="color: red; text-align: center;">Помилка завантаження даних.</p>`;
                    });
            });
        });
    });
    </script>
}

Program.cs
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();

var basePath = AppContext.BaseDirectory;
var configuration = new ConfigurationBuilder()
    .SetBasePath(basePath)
    .AddJsonFile("Infrastructure/Config/config.json", optional: false, reloadOnChange: true)
    .Build();

builder.Services.AddSingleton<IConfiguration>(configuration);

builder.Services.AddLogging(logging =>
{
    logging.ClearProviders();
    logging.AddConsole();
});

builder.Logging.ClearProviders();
builder.Logging.AddConsole();
builder.Logging.AddProvider(new InMemoryLogProvider());

builder.Services.AddSingleton<HtmlParser>();
builder.Services.AddSingleton<IFormScanner, FormScanner>();
```

```
builder.Services.AddSingleton<IThrottlingService>(provider =>
{
    return new ThrottlingService();
});

builder.Services.AddScoped<IFormScanner, FormScanner>();

builder.Services.AddSingleton<IDictionaryProvider>(provider =>
{
    var config = provider.GetRequiredService<IConfiguration>();
    var logger = provider.GetRequiredService<ILogger<FileDictionaryProvider>>();
    return new FileDictionaryProvider(config, logger);
});

builder.Services.AddSingleton<IUserAgentProvider>(provider =>
{
    var config = provider.GetRequiredService<IConfiguration>();
    var userAgentFilePath = config.GetValue<string>("UserAgent:FilePath");
    var logger = provider.GetRequiredService<ILogger<UserAgentProvider>>();
    return new UserAgentProvider(userAgentFilePath, logger);
});

builder.Services.AddHttpClient<IHttpRequester, HttpRequester>()
    .ConfigureHttpClient(client =>
    {
        client.Timeout = TimeSpan.FromSeconds(30);
    })
    .ConfigurePrimaryHttpMessageHandler(() =>
    {
        var handler = new SocketsHttpHandler
        {
            PooledConnectionLifetime = TimeSpan.FromSeconds(2),
            PooledConnectionIdleTimeout = TimeSpan.FromSeconds(30),
            MaxConnectionsPerServer = 10
        };
        return handler;
    });

builder.Services.AddSingleton<IDirectoryScanner, DirectoryScanner>();

var app = builder.Build();

if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
}
app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();
app.MapControllers();
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Fuzzing}/{action=Index}/{id?}");

app.Run();
```

Завідувачу кафедри кібербезпеки
к.т.н., доц. Кльоцу Ю.П.
Дейчука Ростислава Руслановича
ПІБ здобувача вищої освіти

Студента ФІТ, 3 курсу, групи КБс-22-1

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності у Хмельницькому національному університеті, згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщений та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії Хмельницького національного університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-обчислювального комплексу StrikePlagiarism та/або програмнотехнічного засобу Anti-Plagiarism і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення текстових збігів в роботах.

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

27.05.2025
дата

підпис

Anti-Plagiarism v-15.274 Educational

The maximum coincidence with one document 1.0%

Dictionaries check: en_US, ru_RU, ua_UA. **Errors in the documents: 11%**

	Document		Sum coincidence on the DB	
ID: 242333 Title: Система проактивного сканування вразливостей хостинг-платформ Added in a DB: 2025-05-28 Authors: Дейчук Ростислав Русланович Heads: Стецюк В.М. Consultants: Opponents:	Symbols	Lexemes	Symbols	Lexemes
	71129	588	508 (1%)	5 (1%)

Plagiarism sources

ID	Description	Plagiarism presence in the document	
		Symbols	Lexemes

Протокол аналізу звіту подібності науковим керівником

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

Автор: Дейчук Ростислав Русланович

Співавтор:

Назва: Система проактивного сканування вразливостей хостинг-платформ

Науковий керівник:

Підрозділ: Кафедра кібербезпеки

Коефіцієнт подібності 1:1%

Коефіцієнт подібності 2:0.3%

Мікропробіли: 0

Заміна букв: 0

Інтервали: 0

Білі знаки: 0

Дата створення звіту: 2025-05-29 03:39:53.0

Після аналізу Звіту подібності констатую наступне:

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

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

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

Обгрунтування:

29.05.2025р.

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ КІБЕРБЕЗПЕКИ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

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

Назва: Система проактивного сканування вразливостей хостинг-платформ

Автор: Дейчук Ростислав Русланович

Спеціальність: 125 – Кібербезпека та захист інформації

Освітня програма:Кібербезпека та захист інформації

Науковий керівник: Микола СТЕЦЮК др. філософії, ст. викладач

Після аналізу звіту подібності зроблено такий висновок:

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

Підтвердження:

Оригінальність тексту роботи за результатами перевірки системою StrikePlagiarism складає 99.0%, оригінальність тексту роботи за результатами перевірки системою Anti-Plagiarism складає 99.0%.

Згідно з правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 24.09.2024, авторська робота, обсяг оригінального тексту у відсотках до загального обсягу матеріалу в якій складає 90-100%, визначається роботою з високою унікальністю тексту і допускається до захисту.

Виявлені модифікації стосуються математичних формул і не є порушенням академічної доброчесності.

Керівник роботи _____ Микола СТЕЦЮК

Гарант ОП _____ Віктор ЧЕШУН

Завідувач кафедри кібербезпеки _____ Юрій КЛЬОЦ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

освітнього ступеня «бакалавр»

Студент Дейчук Ростислав Русланович

Тема Система проактивного сканування вразливостей хостинг-платформ

Галузь знань 12 – Інформаційні технології

Спеціальність 125 – Кібербезпека та захист інформації

Обсяг кваліфікаційної роботи освітнього ступеня «бакалавр»:

кількість листів креслень 3; кількість сторінок записки 62

1. Кваліфікаційна робота розробці прикладного інструменту для автоматизованого виявлення потенційно вразливих директорій на веб-хостингах шляхом сканування URL-шляхів із використанням методів фаззингу. Запропоноване рішення дозволяє підвищити ефективність аудитів безпеки вебресурсів та може застосовуватись у практиці пентестингу, інформаційного аналізу й внутрішнього контролю.

2. Висновок про відповідність кваліфікаційної роботи завданню Кваліфікаційна робота відповідає поставленому завданню як в теоретичній, так і в практичній частині.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У роботі використано сучасні практики в області інформаційної безпеки, веб-фаззингу та програмної інженерії. У першому розділі детально проаналізовано актуальні вектори атак на вебсайти, а також надано огляд існуючим інструментам. У другому розділі описано архітектуру та реалізацію проекту, що містить модуль багатопоточного сканування, підтримку кастомізованих словників та можливість заміни User-Agent заголовків У третьому розділі виконано порівняльне тестування розробленої системи з існуючими аналогами, що підтвердило її конкурентоспроможність і функціональну придатність для практичного використання.

4. Позитивні сторони роботи Позитивні сторони проекту Робота добре структурована, чітко висвітлені всі етапи дослідження. Використані сучасні інструменти та підходи, що свідчить про якість проведеного дослідження.

5. Негативні сторони роботи До недоліків можна віднести обмежене опрацювання тематики обробки нетипових відповідей сервера та недостатню увагу до автоматизації побудови звітності після сканування. Також перспективною для майбутньої розробки виглядає реалізація системи адаптації словників залежно від типу цілі.

6. Оцінка графічного оформлення та пояснювальної записки роботи Пояснювальна записка оформлена належним чином, має чітку структуру та логічну послідовність викладення матеріалу. Ілюстративний та графічний матеріал якісно доповнює зміст, полегшуючи розуміння функціоналу системи. Усі таблиці, рисунки відповідають змісту розділів і відображають хід розробки.

7. Відгук про роботу в цілому Кваліфікаційна робота справляє позитивне враження. Вона демонструє як глибоке розуміння теми так і здатність до самостійної розробки повноцінного прикладного програмного рішення. Розроблений інструмент відрізняється рівнем функціональності, а також може бути основою для подальшого розширення. Презентаційна частина логічно ілюструє хід реалізації та ключові особливості програмного забезпечення.

8. Інші зауваження

9. Оцінка дипломної роботи Розглянувши позитивні та негативні сторони представленої кваліфікаційної роботи, можна зробити висновок, що вона заслуговує оцінку «відмінно».

РЕЦЕНЗЕНТ (прізвище, ім'я, по батькові, посада, місце роботи)
Підченко Сергій Констянтинович, завідувач кафедри ТМІТ

« _____ » _____ 2025. _____ (підпис)