Хмельницький національний університет
Факультет інформаційних технологій
Кафедра кібербезпеки

КВАЛІФІКАЦІЙНА РОБОТА

Сьомик Діани Ігорівни

на здобуття ступеня вищої освіти Бакалавра

Автоматизована система аналізу XSS та CSRF вразливостей вебдодатків на Python

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

Шифр КРБКБ.2102164.21.02.40 ПЗ

Виконала студентка 4 курсу група КБ-21-2 Діана СЬОМИК
Керівник канд. техн. наук, доцент Ігор МУЛЯР
Нормоконтролер старший викладач Сергій МОСТОВИЙ

До захисту допускаю:
Завідувач кафедри кібербезпеки Юрій КЛЬОЦ

9 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 02 2025 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Сьомик Діани Ігорівни

1 Тема роботи Автоматизована система аналізу XSS та CSRF вразливостей вебдодатків на Python

Керівник роботи канд. техн. наук, доцент Муляр І.В.

Затверджено наказом ректора університету від 7 лютого 2025 № 23

2 Строк подання студентом кваліфікаційної роботи на кафедру 2.06.2025

3 Вихідні дані до роботи Проаналізувати предметну область та існуючі рішення в галузі пошуку вразливостей. Сформулювати постановку задачі. Розробити архітектуру автоматизованої системи аналізу безпеки. Реалізувати модулі для сканування XSS та CSRF вразливостей. Реалізувати та налаштувати систему. Провести тестування автоматизованої системи.

4 Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Вступ. Огляд загроз інформаційної безпеки. Типи атак на вебдодатки. Огляд сканерів вразливостей. Потенційні наслідки атак на вебсервери. Постановка задачі. Обгрунтування обраного підходу. Архітектура автоматизованої системи аналізу безпеки. Інструментарій розробки та технології реалізації системи. Налаштування автоматизованої системи. Тестування системи.

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

Структура автоматизованої системи. Етапи виконання пошуку XSS вразливостей. Етапи виконання пошуку CSRF вразливостей.

6 Консультанти розділів кваліфікаційної роботи

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

7 Дата видачі завдання ___ _____ 2025 р.

КАЛЕНДАРНИЙ ПЛАН

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

Студентка _____ Діана СЬОМИК

Керівник кваліфікаційної роботи _____ Ігор МУЛЯР

АНОТАЦІЯ

Тема кваліфікаційної роботи: Автоматизована система аналізу XSS та CSRF вразливостей вебдодатків на Python.

Автор роботи: Сьомик Діана Ігорівна.

Керівник роботи: Муляр Ігор Володимирович

Пояснювальна записка: 61 с., 2 додатки, 19 рисунків, 1 таблиця, 36 джерел.

Графічна частина: 3 плакати, 12 презентаційних слайдів.

АВТОМАТИЗОВАНА СИСТЕМА, ВЕБДОДАТОК, СКАНЕР ВРАЗЛИВОСТЕЙ, ІНФОРМАЦІЙНА БЕЗПЕКА, БЕЗПЕКА ДАНИХ.

Кваліфікаційна робота бакалавра присвячена розробці автоматизованої системи аналізу XSS та CSRF вразливостей вебдодатків на Python.

В роботі проведено аналіз існуючих методів виявлення вразливостей у вебдодатках, розглянуто типи атак на вебдодатки, сканерів вразливостей та потенційні наслідки атак на вебсервери. Розроблено автоматизовану систему аналізу XSS та CSRF вразливостей, та розроблено модулі для пошуку цих вразливостей. Проведено налаштування та тестування системи.

09.06.2025 

ABSTRACT

Subject of qualification work: Automated system for analyzing XSS and CSRF vulnerabilities in web applications in Python.

Author: Somyk Diana Ihorivna

Head of work: Mulyar Ihor Volodymyrovych

Explanatory note: : 61 p., 2 appendices, 19 figures, 1 tables, 36 sources.

Graphic part: 3 poster, 12 presentation slides.

AUTOMATED SYSTEM, WEB APPLICATION, VULNERABILITY SCANNER, INFORMATION SECURITY, DATA SECURITY.

The bachelor's thesis is devoted to the development of an automated system for analyzing XSS and CSRF vulnerabilities of web applications in Python.

The paper analyzes existing methods for detecting vulnerabilities in web applications, considers types of attacks on web applications, vulnerability scanners, and potential consequences of attacks on web servers. An automated system for analyzing XSS and CSRF vulnerabilities has been developed, and modules for searching for these vulnerabilities have been developed. The system has been configured and tested.

09.06.2025 _____

ЗМІСТ

ВСТУП

Сучасні вебдодатки є важливим елементом цифрової інфраструктури, забезпечуючи зручний доступ до різноманітних сервісів та інформації в мережі. Відповідно, зростаюча популярність вебдодатків робить їх привабливими цілями для кібератак, головним чином через недоліки XSS і CSRF.

XSS є одним із переважаючих видів ризиків, який дозволяє зловмисникам впроваджувати шкідливий код у вебсторінки, які показують іншим особам. Це може призвести до крадіжки особистих даних, розповсюдження вірусів або повної компрометації облікових записів. CSRF, у свою чергу, дозволяє зловмисникам здійснювати несанкціоновані дії від імені користувачів, які вже мають доступ до вебсайту. Це може призвести до змін записів або виконання операцій без відома користувача.

Визначити ці проблемні області на великих і складних вебсайтах може бути дуже важко. Існує потреба в швидких і точних системах комп'ютерної безпеки, які можуть виявляти можливі небезпеки. Через це нам потрібні розумні системи, які можуть швидко знаходити можливі небезпеки. Системи роблять перевірку безпеки швидшою та простішою для людей.

З кожним роком кількість вебдодатків, які використовуються в різних сферах діяльності, зростає. Це стосується як комерційних проектів, так і державних установ. Безпека цих систем є критично важливою, оскільки незначні недоліки можуть призвести до серйозних наслідків. Крадіжка особистих даних або порушення роботи сервісу можуть завдати шкоди репутації та призвести до фінансових втрат.

Незважаючи на те що в сучасному світі є багато різних інструментів для виявлення вразливостей, вони часто вимагають практичних дій, що потребує значного часу та високого рівня знань у галузі кібербезпеки. Через це необхідно розробити автоматизовані системи, які можуть швидко й точно виявляти ризики та нейтралізувати їх.

Аби своєчасно виявити та усунути вразливості, необхідно використовувати ефективні методи автоматизованого тестування, які здатні швидко сканувати вебдодатки на предмет потенційного ризику. Серед таких методів є автоматизована система аналізу XSS та CSRF вразливостей, створена за допомогою Python.

Основна роль автоматизованої системи є забезпечення точності та швидкості у виявленні вразливостей в поєднанні із зменшенням хибних результатів. Завдяки інтеграції системи з відомими бібліотеками тестування безпеки, такими як requests, BeautifulSoup, Selenium та іншими, система ефективно аналізувати статичний та динамічний вебконтент.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Огляд загроз інформаційної безпеки

В сучасному світі безпека вебсайтів є важливою частиною забезпечення їх належної роботи. Загрози проти інформаційної безпеки зараз змінюються у міру зростання технології. Інформаційна безпека охоплює цілий ряд заходів, що використовуються для захисту інформації та активів від різноманітних ризиків. Конфіденційність, цілісність та доступність – це основні частини моделі CIA (Confidentiality, Integrity, Availability), яка формує ідею захисту даних.

Загроза інформаційній безпеці – це коли відбуваються певні речі, які можуть зашкодити захисту даних. Загрози інформаційній безпеці можна класифікувати відповідно до різних функцій: за аспектом інформаційної безпеки, на який спрямовані загрози; за розташуванням джерела загроз; за розміром нанесеного пошкодження; за ступенем впливу на інформаційну систему та за природою виникненням.

Розглянемо загрози інформаційної безпеки за аспектом інформаційної безпеки, на який спрямовані загрози.

Загрози конфіденційності (незаконний доступ до інформації) [1]. Ризик розголошення пов’язаний із наданням даних суб’єктам, які не мають відповідних дозволів. Це відбувається, коли надається вхід до деякої обмеженої інформації, що зберігається в комп’ютерній системі або передається з однієї системи до іншої. Через ризик порушення конфіденційності , використовується термін «витік». Загрози можуть виникати через недбалість користувача, як-от несанкціоноване призначення привілеїв, а також через збої програмного чи апаратного забезпечення.

Загрози цілісності (незаконна зміна даних). Загрози, які можуть змінити інформацію в інформаційній системі, відомі як загрози цілісності. Порушення цілісності, може бути викликано різними чинниками, від умисних дій до виходу з ладу обладнання [2].

Загрози доступності (здійснення дій, які перешкоджають або запобігають взаємодії з комп'ютерними мережевими спорудами). Порушення доступності – це налаштування обставин, коли доступ до інформації заблокований або обмежений, що перешкоджає досягненню ключових комерційних цілей [3].

Достовірність відомостей свідчить про те, що інформаційні дані належать довіреній особі або законному власнику, який також є основним джерелом інформації [4].

Невідмовність – це важливий принцип інформаційної безпеки, який гарантує, що жодна сторона не зможе відмовитися від здійснення певної дії або події.

Інформаційна безпека є сукупністю різноманітних ефективних заходів, які здатні запобігти, відстежити, позбутися від небажаних зловмисників ззовні. Крім того, інформаційна безпека має на меті захистити систему від шкоди, змін, блокування або копіювання даних.

Безпека інформації ще більш необхідна, якщо відомо, що проникнення в систему і крадіжка даних призведуть до вкрай негативних наслідків. Наприклад, до значних матеріальних збитків, масштабного удару по репутації людини або компанії і т.д.

Сьогодні ми стикаємось з сотнею різних видів загроз інформаційної системи. Тому, нам потрібно регулярно перевіряти усі наявні вразливості. Для цього використовуються різні методи діагностики.

Лише після ретельного аналізу, ми можемо вибрати найкращі методи, щоб зберегти нашу систему від небажаного доступу.

Безпека даних вебдодатків залежить від їх здатності протидіяти потенційним загрозам, які можуть поставити під загрозу конфіденційність, цілісність або доступність інформації.

У світі, що швидко розвивається, ідея захисту даних завжди змінюється для нових викликів. Класичні уявлення про небезпеку, що стосуються конфіденційності, цілісності та доступності, залишаються важливими, але

сьогоднішні хакери часто використовують більш складні та змішані стратегії. Це вимагає, щоб ми зрозуміли фундаментальні поняття захисту, а також регулярно вивчати нові стратегії та підходи.

Раніше небезпека часто спрямовувалась на те, щоб заподіяти шкоду одному з аспектів моделі CIA. Наприклад, традиційний напад DDoS просто перериває систему перешкоджаючи її доступність. Фішинг в свою чергу, націлений на крадіжку облікових даних, що призводить до порушення конфіденційності. Тим не менш, спостерігається зростання складних нападів, які об'єднують кілька методів для більшого впливу.

Наприклад, атаки програм-вимагачів (ransomware) [5], які стали надзвичайно поширеними, демонструють такий гібридний підхід. Вони не тільки порушують доступність даних, шифруючи їх, але й часто супроводжуються витоком конфіденційної інформації, якщо жертва відмовляється платити викуп. Зловмисники погрожують опублікувати викрадені дані, додаючи до фінансових втрат ще й значні репутаційні ризики. Це яскравий приклад того, як одна атака може одночасно впливати на доступність та конфіденційність, а також на невідмовність, оскільки довести початковий стан даних після шифрування стає вкрай складно.

Неетичні особи, користувачі інформації, суперники або експерти з укладеними контрактами часто можуть бути кіберзлочинцями.

Причиною може бути бажання отримати чужі кошти. Конкуренти періодично намагаються підірвати суперників шляхом крадіжки приватних даних. Співробітники компанії, які були звільнені за певні дії, можуть помститися колишньому роботодавцю.

Тому є численні причини незаконної поведінки. Робота даних про захист - зупинити порушників, зупинити їх на початку їхньої спроби порушити систему. Для найкращого результату шукати експертів, які успішно продемонстрували свою експертизу в цій галузі та дуже цінуються. Тоді системи даних будуть забезпечені безпечними.

Є кілька небезпек, основні з них:

– введення в мережу даних про пошкодження програм: віруси, троянські коні, мережеві хробаки, клавіатурні шпигуни, рекламні системи;

– хакерські атаки;

– BotNet використовуються в різних зловмисних цілях, таких як надсилання спаму, крадіжка даних або запуску атак розподіленої відмови від служби (DDOS);

– DDOS – атака на відмову, розподілену атаку на відмову (DOS-ATTACK (розподілена) атака відмови від обслуговування) -An атаки на комп'ютерну систему з наміром зробити комп'ютерні ресурси недоступними користувачами, для яких була призначена комп'ютерна система;

– фішинг – це оманлива практика, спрямована на передачу особистої інформації від користувачів онлайн-аукціону, послуг обміну валют та платформ електронної комерції.

1.2 Типи атак на вебдодатки

Вебдодатки обробляють важливі дані, такі як особиста інформація користувачів, банківські дані та корпоративні секрети. Недотримання правил безпеки може призвести до витоків даних, порушення конфіденційності і навіть втрати репутації компанії. Тому забезпечення безпеки вебзастосунків є критично важливим аспектом [6].

Захист інформації є найважливішим елементом сучасності. Щодня різноманітність методів нападу зростає, тому важливо оцінити потенційні слабкі місця та адаптувати сервер для запобігання атак.

Як кіберзлочинці виконують цифрові напади, пеeважно використовується чотири основні типи нападу:

– пасивні напади – це неактивні операції, коли злочинці приховують свої дії, щоб жертва залишалася невідомою про те, що переслідує. Пасивні атаки використовуються в основному для збору або крадіжки інформації в кібершпигунстві;

– активні напади – це, як правило, сильні напади, спрямовані на збиток або знесення особистих гаджетів, мереж або цілих систем. Може бути спрямований на людей, групи і навіть нації;

– інсайдерські атаки – як випливає з назви, ці типи атак роблять людину з внутрішнього кола, яке вже дозволило отримати доступ до системи, на яку спрямована.

– зовнішні напади – напади виконуються поза системою, націленою на напад. Вони виконуються незначними правопорушниками та окремими державами.

Кібератаки приймають багато форм: шкідливі програми, оманливі електронні листи, мережеві атаки перевантаження та численні інші. Розкриття таємних деталей, осушення заощаджень та крадіжки приватної інформації - лише деякі її жахливі результати.

Вебсервер – це комп'ютерна програма або пристрій, яка експлуатує програмне забезпечення для задоволення потреб користувачів в Інтернеті. Інтернет -сервер передбачає запити користувача, обробляє їх та виконує дії. Працює на задньому плані.

Переповнення буфера – ймовірно, є найвідомішим типом. Більшість програмістів розуміють концепцію переповнення буфера, але напади, що використовують цю вразливість, зберігаються як у старому, так і в новому програмному забезпеченні. Деякі питання випливають із широкого спектру методів, що спричиняють переповнення буфера, і деякі поширені методи профілактики сприйнятливі до помилок.

Ін'єкція SQL - це слабкість у безпеці веб -сайту, яка дозволяє кривднику зіпсувати запити, які програма надсилає на зберігання даних. Як стандарт, це

дозволяє зловмиснику бачити інформацію, до якої вони зазвичай не можуть отримати доступ. Це може включати інформацію, яку зберігають інші особи або будь -які дані, до яких може охопити програмне забезпечення. У багатьох випадках зловмисник може змінити або видалити ці дані, спричиняючи постійні зміни вмісту або поведінки програми.

Кібератака під назвою DOS (відмова в сервісі) - це коли хтось намагається зробити комп'ютер або пристрій недоступним для його передбачуваних користувачів, внаслідок чого він припиняє працювати нормально. DOS-атаки зазвичай працюють, затоплюючи або заповнюючи цільову машину запитами, поки вона не зможе впоратися з регулярним трафіком, внаслідок чого відмовляють у обслуговуванні більшій кількості користувачів. DOS-ATHAM-це тип кібер-атаки, який передбачає один комп'ютер, що ініціює напад.

Напад "людина посередині" - це випадки, коли агресор перехоплює зв'язок між двома сторонами, або приховано перегукується або змінює дані, що протікають між ними. Зловмисники можуть використовувати тактику перехоплення для подачі даних про вхід або персональних даних, шпигунства, саботажу, обміну повідомленнями або шкодою даних.

– XSS (міжсайтовий скриптинг) [7] – це напад, при якому зловмисник вбудовує шкідливий код у вебсторінку, що відображається у браузері користувача. Це може призвести до викрадення файлів комп’ютерного сеансу, перенаправляння на небезпечні Інтернет-сторінки або зміни вмісту вебсторінки .

– Атаки XSS дозволяють зловмисникам представити шкідливий сценарій (також називається експлуатацією) на сторінці програми, в наслідок чого користувачі, які відвідують цю сторінку, можуть вкрасти такі дані, як: файли cookie, сесійні токени, особисту інформацію та логіни з паролями.

– У різних методах зловмисники можуть вставити небезпечний код, наприклад, залишити коментар з скриптом під публікацією. І якщо розробники не перевірили точність даних, шкідливий сценарій розпочнеться в усіх користувачів, які переглянули коментарі на сторінці [8].

Історія створення походить з далекого 1995 року, коли дві американські IT-компанії представили LiveScript 1.0. це була скриптова мова програмування, на базі фреймворка Mocha та мови програмування Java. Вона дозволяла створювати вебсторінки, і далі була інтегрована в інтернет-браузер, де нова модифікація отримала назву – JavaScript [9].

– Типи міжсайтового скриптингу:

– Refected XSS – зловмисник може передати шкідливий код через URL, де він виконується на стороні браузера жертви;

– Stored XSS – шкідливий код впроваджується у вабсайт і зберігається на сервері у вигляді частини контенту, схему атаки можна побачити на рис. 1.1;

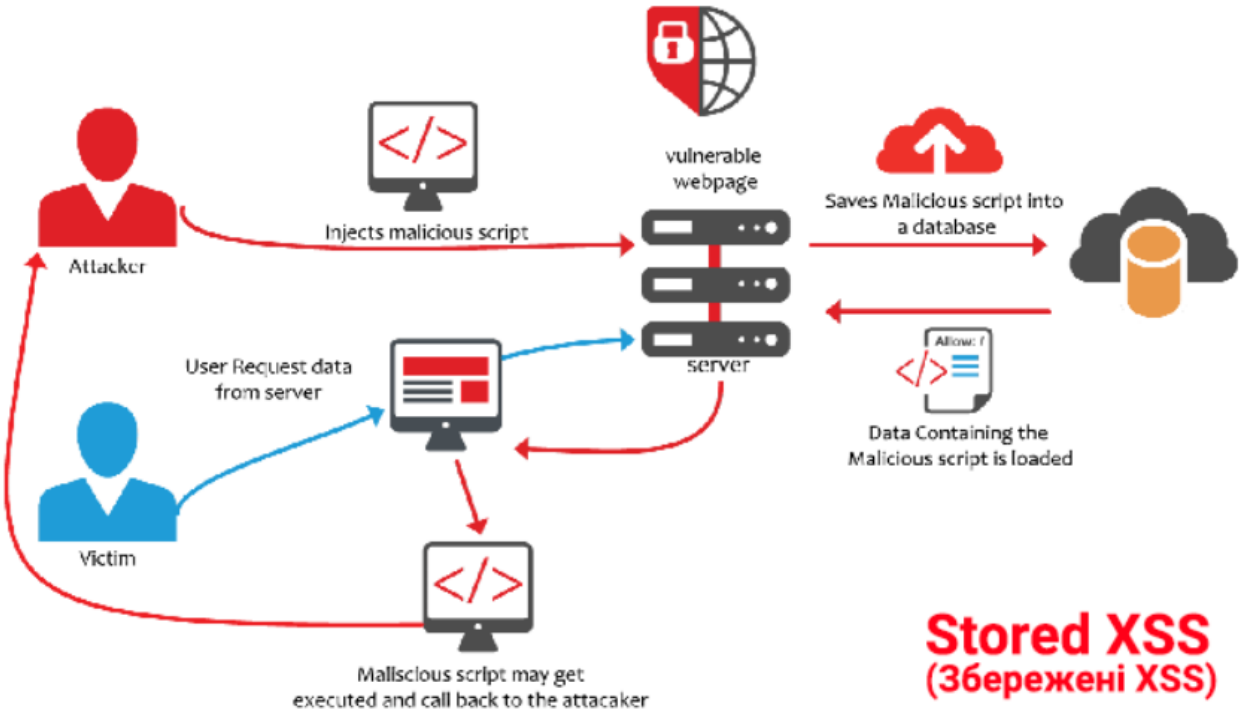


Рисунок 1.1 - Схема атаки

– DOM- based XSS – шкідливий код впроваджується в Document Object Model, DOM- структуру можна побачити на рис. 1.2.

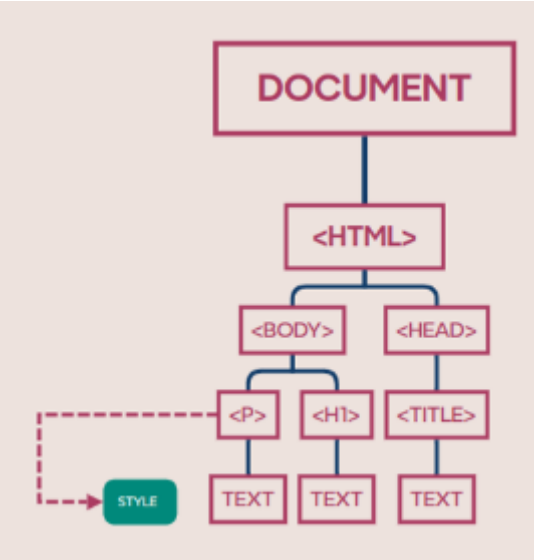


Рисунок 1.2 - DOM- структура

Практики захисту від XSS:

– екранування даних. Перед вставкою будь-яких даних в шаблон HTML-сторінки, потрібно переконатись що вони екрануються. Використовувати функції, наприклад encodeURIComponent або бібліотеки, що забезпечують екранування спеціальних символів;

– валідація введення. Перевіряти кожне введення повідомлення користувачем на всі небезпечні символи або коду. Використовувати регулярні вирази або спеціальні методи для валідації даних перед їх використанням;

– обмеження виконання скриптів. Content Security Policy застосування: CSP є механізмом безпеки для того щоб вказувати, звідки завантажувати вміст на сторінку. Виконувати обмеження по виконанню скриптів з ненадійного джерела та запобігання XSS-атакам за допомогою CSP.

CSRF (міжсайтова підробка запитів) – це атака, в якій зловмисник обманно змушує користувача виконати небажані дії в додатку, в якому він аутентифікований. Зловмисник може маніпулювати авторизацією користувача для виконання несанкціонованих дій під іменем користувача, таких як зміна його пароля, розміщення коментарів або вчинення небажаних дій [10].

Щоб зрозуміти CSRF, треба відрізнити його від інших недоліків безпеки вебдодатків. Однією з головних відмінностей є те, що CSRF не націлений на вебдодаток, а натомість використовує довіру між браузером користувача та вебсайтом, який від відвідує. Інша вразливість, така як міжсайтовий сценарій, орієнтована на маніпуляцію з кодом або вебпрограмою [11].

Практика захисту від CSRF:

– включити механізм використання токенів безпеки при використанні критичних дій, таких як зміна користувацького пароля або видалення облікового запису. Згенерувати токен на стороні сервера і передати його на клієнтську сторону, включаючи його у форму або як заголовок запиту. Приклад CSRF-токену можна побачити на рис.1.3 ;

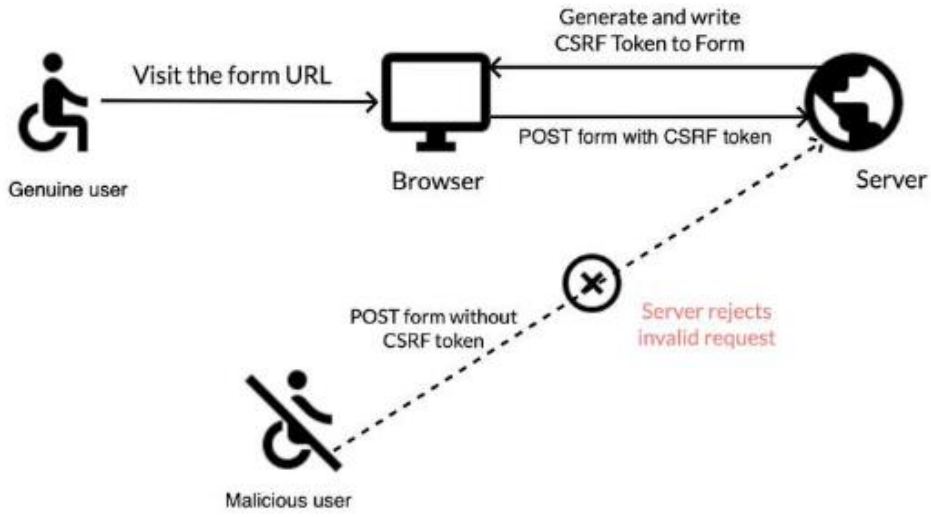


Рисунок 1.3 - CSRF-токени

– перевіряти HTTP-заголовки Referer або Origin, щоб переконатися, що вхідні запити надходять з очікуваних джерел. Це дозволяє відхилити запити, які надійшли зі сторонніх або небажаних сайтів;

– встановити обмеження, що використовуються для виконання критичних дій. Наприклад, видалення записів може бути дозволено лише для запитів DELETE або POST, а не для GET.

XSS і CSRF, націлені на довіру, зарезервоване між сайтом і користувачами. XSS у дії, коли зловмисники вводять зловмисні сценарії, вони виконуються під час відвідування сайту користувачем. Потім ці сценарії можуть залишити найрізноманітніші дії – від викрадення файлів cookie користувача (щоб зловмисники могли видати себе за користувача на сайті) і до більш складніших дій.

Коли CSRF у дії, з іншого боку, більше стосується шахрайства. Зловмисники створюють сценарії, коли користувачі виконують певні дії на сайтах, на який вони ввійшли. Якщо користувач попаде у сіті зловмисника, він може ненавмисно переказати гроші або змінити налаштування облікового запису через прихований запит, створений зловмисником.

Таблиця 1.1 Порівняння між XSS та CSRF.

XSS	CSRF
Пошкоджений сценарій вводиться в сценарій сайту користувача;	Користувача вебсайту обманюють так, що він в кінцевому результаті вводить пошкоджені HTTP-запити, не знаючи результату;
веб-сайт отримує пошкоджений код та обробляє його;	веб-сайт не зберігає код, після чого виникають проблеми, оскільки він зберігається на сайтах третіх сторін;
вразливість XSS прокладає шлях для CSRF-атаки;	CSRF-атака не означає, що сайт схильний до XSS-атаки;
результат може бути досить жахливим та шкідливим;	атака не надто руйнівна;
зловмисник після вдалої спроби взлому, здатний робити все що захоче.	зловмисники можуть завдати шкоди, яка підпадає під повноваження URL.

В таблиці 1.1 порівнюються 2 вразливості, XSS та CSRF, в якій можна побачити, як виконується атака і її результат.

Вебсайти розроблені так, щоб довіряти користувачам коли вони вводять дані або виконують будь-які дії. Важливий елемент, внутрішня довіра – це те, що використовують ці атаки.

Для багатьох користувачів до того моменту, коли вони усвідомили що стали жертвою зловмисника, і що щось не так, стались вже значні збитки. Ця небезпека підкреслює важливу необхідність надійної веб-безпеки.

Однак сучасний онлайн-світ не обмежується лише традиційними вебсторінками. Дедалі більшого значення набувають API (інтерфейси прикладного програмування) [12] та мобільні додатки**,** які часто взаємодіють з вебсерверами. Вони відкривають нові вектори для атак, які необхідно враховувати в стратегії веббезпеки.

Сьогодні більшість вебдодатків, особливо односторінкові (Single-Page Applications) та мобільні, взаємодіють із сервером через API**.** Це створює окремий набір загроз, які часто відрізняються від тих, що націлені на традиційні вебсторінки:

1.3 Огляд сканерів вразливостей

Продуктивність будь-якої вебсторінки визначається тим, як швидко вона адаптується до постійно розвиваються вебтехнологій. Сучасні інструменти для перегляду вебсторінок в режимі реального часу дали веб-майстру найкращу техніку для обробки інформації в Інтернеті. Очевидно, що такий ринок не помічав кіберзлочинці, спрямовані на пошкодження або викрадення приватних даних. У цих ситуаціях негайна робота полягає у зменшенні ризику вебсерверів та сайтів для запобігання порушенням даних. Сканери вразливостей – програмне забезпечення або обладнання для діагностики та моніторингу мережевих

комп'ютерів, що дозволяє сканувати мережі, комп'ютери та програми для виявлення можливих проблем у системі безпеки, оцінки та усунення вразливості. Сканери безпеки дозволяють вивчити різні програми в системі для буд-яких «прогалин», які можуть використовувати хакери.

Інші засоби можуть також використовуватися щодо визначення низькорівневих, такі як сканер портів для обробки і аналізу потенційних додаткових і протоколів, виконуваних за допомогою системи. Серед мережевої інфраструктики, комп’ютерів, апаратної системи та програмного забезпечення та інших, такі як метод розпізнавання, категоризації та характеристики дір в безпеці, відомий як аналіз вразливості. Може бути мало прикладів таких, неправильна конфігурація компонентів мережевої інфраструктури без доказу дефекту або помилки ОС.

Сканери вразливостей – це ключові інструменти для забезпечення безпеки вебзастосунків. Вони автоматизують процедуру визначення потенційних ризиків, що дозволяє розробникам та експертам з кібербезпеки виявити та усунути сприйнятливість до їх маніпуляцій зловмисниками. Особливо актуальними є сканери, що аналізують XSS та CSRF, враховуючи, що ці форми нападу є поширеними в онлайн-сфері [13].

Адміністратори повинні мати можливість визначити прогалини у своїй мережі на робочих місцях, серверах та брандмауерах. Програми повинні знайти якомога більше слабких сторін. Оцінка вразливості – не всі слабкі сторони однаково важливі. Інструменти сканування сканування можуть класифікувати вразливість, щоб допомогти адміністраторам визначити пріоритети найбільш тривожних проблем.

Хакери можуть порушити мережу та подати інформацію в різних методах. Але є загальні слабкі сторони, про які слід усвідомлювати. Не кожен інструмент мережевого сканування вирішить усі ці проблеми.

На жаль, іноді інсайдери зловживають своїм доступом, що призводить до навмисного або випадкового впливу приватних даних або не правильних

коригувань програмного забезпечення, створюючи додаткові вразливості в безпеці. Крім того, менеджери системи можуть дозволити стандартні деталі для входу, зберігати неактивних користувачів, груп або надавати неправильні права доступу, що загрожує безпеці [14].

Сканер працює за допомогою двох методів. Початковий метод відомий як зондування. Це не дуже швидко, але найкращий інструмент для активного обстеження. Суть полягає в тому, що він сам ініціює напади, і стежить, куди ці напади можуть пройти. У розслідуванні перевіряються потенційні переконання та шанси перешкоджати нападам на конкретні шляхи.

Інший метод – це сканування. У цьому сценарії пристрій працює швидко, але він проводить лише основну експертизу мережі, визначивши найпоширеніші та потенційні прогалини в безпеці. Другий підхід просто повідомляє менеджера системи про потенційні слабкі сторони, спираючись на непрямі ознаки, а не на перевірку їх існування. Наприклад, перевіряються порти, їх вказівки ідентифікуються, а потім вони узгоджуються зі стандартними рекомендаціями та критеріями. Якщо в вимірюванні є розбіжності, пристрій сигналізує про можливу слабкість безпеки, яку менеджер системи повинен перевірити за допомогою більш надійних методів.

Сканери вразливостей працюють шляхом методичного вивчення цільового комп'ютера або мережі для визнаних слабких місць у налаштуванні. Процес сканування зазвичай включає такі кроки:

- сканер налаштований на цільову систему (IP -адреса, діапазон IP -адрес, веб -дренд URL). На цьому момент цільова інформація може бути зібрана: відкриті порти, використані послуги, операційні системи та програмне забезпечення;
- сканер порівнює риси цілі з великою базою даних відомих вразливих місць (CVE). Якщо вразливість ідентифікується, ризик оцінюється;
- застосування методів та вказівок для виявлення нових або невизнаних недоліків безпеки шляхом спостереження за системними діями;

- сканер може імітувати визнані напади (такі як ін'єкція SQL, XSS та переповнення буфера) та вивчити реакцію системи на перевірку слабких місць;

- пропонування інструменту для перевірки облікових даних дозволяє йому отримати глибший вхід у систему, що призводить до більш точної оцінки внутрішніх слабкостей та помилок налаштування;

- після завершення сканування генерується звіт, який включає каталог слабких сторін, їх тяжкість (критична, висока, середня, низька), можливі наслідки, дії щодо виправлення та посилання на додаткову інформацію.

Основні принципи роботи сканерів вразливостей:

– збір усієї інформації в мережі, ідентифікація всіх служб, пристроїв та процесів;

– пошук потенційної вразливості;

– використання унікальних методик та моделювання загроз, щоб перевірити слабкі місця сканера мережі.

Сканери вразливостей можна розділити на певні категорії, такі як:

– статичні аналізатори – перевіряють код вебсайту на можливі слабкі сторони [15];

– динамічні аналізатори – оцінюють веб-програми під час роботи, відправляючи запити та ретельно вивчаючи відповіді [16];

– інтерактивні аналізатори безпеки – поєднують статичні та динамічні перевірки безпеки, пропонуючи розширений спосіб вивчення та покращення безпеки системи.

Серед багатьох інструментів для перевірки ризиків безпеки XSS та CSRF можна виділити декілька, такі як: ZAP, Skipfish, Wapiti.

ZAP - один із найпопулярніших сканерів безпеки для веб-додатків. Це безкоштовна програма експертизи доступу до коду. Також пропонує спеціалізовану підтримку для оцінки веб-програм. ZAP може працювати з кожним важливим системним комп’ютером та інструментом контейнера. Таким чином, можна вільно вибрати, який з них нам найбільше подобається. ZAP

підтримує розширення, які дозволяють користувачам додавати нові функції. Оскільки код ZAP відкритий для того щоб ми могли перевірити як працюють його функції. Будь хто може працювати над ZAP, виправляти помилки, посилювати функціональні можливості, створювати включення для сприяння унікальним сценаріям. Режими які підтримує OWASP ZAP: безпечний, захищений, стандартний, режим ATTACK [17-18].

Skipfish – це безкоштовний автоматичний інструмент тестування відкритого коду. Skipfish використовується для збору інформації та перевірки їх безпеки. Цей інструмент сканує цільовий сайт і створює карту, використовуючи повторні перевірки на основі словника. Також Skipfish створює звіт, який може бути використаний для оцінки безпеки. Це активний інструмент для розвідки безпеки веб-додатків. Це допомагає створити простий посібник із сайту, також допомагає глибоко сканувати на основі словника. Потім карта позначається результатами декількох перевірок безпеки [19].

Wapiti – це консольний інструмент для сканування веб-додатків, що шукає вразливості, як SQL-ін’єкції, XSS. Дозволяє контролювати безпеку онлайн-платформ. Він не вивчає вихідний код вебсайту, скануючи веб-сторінки розгорнутої веб-програми, шукаючи сценарії та форми, в які можна вставити дані. Коли Wapiti отримує список URL-адрес, форм та їх вхідних даних, він починає діяти як фазер, вставляючи корисні дані, щоб перевірити, чи є вразливим сценарій [20].

OpenVas – це повнофункціональний сканер вразливості. Його можливості включають тестування, що не мають автентифікації, та автентифіковані тестування, різні протоколи інтернет – та низькорівневі та промислові протоколи, налаштування продуктивності для масштабного сканування та потужної мови внутрішнього програмування для будь-якого типу тестування ulnera.

Сканер отримує тести, щоб знайти слабкі сторони з шляху даних, який має довгі минулі та щоденні зміни.

Nikto – безкоштовний (open source) сканер для пошуку вразливостей у веб-серверах. Утиліта належить до класу blackbox сканерів, тобто сканерів, що використовують стратегію сканування методом чорної скриньки. Це означає, що заздалегідь невідомо про внутрішній пристрій програми/сайту (доступ до вихідного коду відсутня) і наголос на функціональності. Програма може виявляти понад 6700 потенційно небезпечних файлів та вразливостей. Нові вразливості додаються до бази даних програми у міру їх виникнення.

1.4 Потенційні наслідки атак на вебсервери

У сучасному онлайн світі вебсервери є важливими елементами як для приватних організацій, так і для окремих людей. Вони допомагають вебдодаткам працювати, захищати приватну інформацію та спілкуватися з користувачами. Забезпечуючи їх важливість, вебсервери стають привабливими для зловмисників. Ефективні напади на вебсерверах можуть призвести до численних згубних результатів, що впливають на власників вебсайтів, так і їх користувачів.

Вдалі атаки впровадження команд на веб-сервер здатні мати тяжкі наслідки, порушуючи безпеку й цілісність системи. Ін'єкція команди – це різновид уразливості, що дає змогу зловмисникові виконувати довільні команди на сервері, вводячи шкідливий вхід у вразливу програму. Це може призвести до різних потенційних наслідків, включно з несанкціонованим доступом, витоком даних, підвищенням привілеїв та навіть повною компрометацією системи.

Одним з основних наслідків атак з впровадженням команд є несанкціонований доступ. Запроваджуючи зловмисні команди, зловмисник може обійти механізми аутентифікації та одержати несанкціонований доступ до конфіденційних даних чи функцій. Наприклад, якщо веб-додаток дозволяє користувачеві вводити дані для створення SQL-запитів без відповідної обробки, зловмисник може ввести команди SQL для отримання або зміни даних з головної

бази даних. Це може призвести до розголошення конфіденційної інформації, як-от облікових даних користувача, фінансових записів або особистих даних.

Іншим наслідком є ймовірність витоку даних. Вразливості вводу команд можуть дати зловмисникам змогу виконувати команди, які дозволяють їм викрадати інформацію з сервера. Приміром, зловмисник може ввести команди для пересилання конфіденційних файлів на зовнішній сервер або для відправлення їх електронною поштою. Це може призвести до розголошення секретної інформації, комерційної таємниці або інтелектуальної власності, що спричинить фінансові втрати чи репутаційну шкоду для потерпілої організації.

Щоби мінімізувати потенційні наслідки атак впровадженням команд, важливо дотримуватись методів безпечного програмування та здійснювати належну перевірку вхідних/вихідних даних. Розробники ніколи не повинні довіряти введеним користувачами даним і мають очищувати й перевіряти всі введені дані до їх використання у виконанні команд або запитах до бази даних. Крім того, використання брандмауерів вебзастосунків (WAF) і систем виявлення небезпек (IDS) можуть допомогти у виявленні та блокуванні спроб впровадження команд.

Більше того, успішні атаки з використанням командного впровадження можуть призвести до повної компрометації веб-сервера. Коли зловмисник отримує контроль над сервером, він може використовувати його як стартовий майданчик для подальших атак, таких як розповсюдження шкідливого програмного забезпечення, запуск розподілених атак типу «відмова в обслуговуванні» (DDoS) або перехід на інші системи в мережі. Це може мати серйозні наслідки для доступності, конфіденційності та цілісності всієї системи, що потенційно може призвести до значних фінансових та операційних збитків.

Потенційні наслідки атак на вебсервери включають:

– компроментація даних або незаконне введення даних - зловмисники можуть викрадати дані приватних користувачів (логіни, паролі, дані про кредитну карту, інформацію про проживання, фінансову звітність, медичні записи та іншу

дорогоцінну інформацію). Обмін приватною інформацією може спричинити великі грошові проблеми, шкоду іміджу людини чи компанії, юридичних проблем та порушення правил конфіденційності користувачів [21];

– використання ресурсів сервера для шкідливих цілей - зловмисники можуть використовувати вебсервери для шкідливих програм, спаму, більше атак або видобутку цифрових грошей без власника. Це може спричинити зростання витрат на обслуговування сервера, зменшуючи його ефективність та потенційні юридичні проблеми;

– репутаційні збитки - успішна атака може серйозно пошкодити довіру користувачі на вебсайті. Відновлення репутації після інциденту може бути складним і зайняти тривалий час;

– юридичні та регуляторні наслідки - якщо особиста інформація протікає, це може спричинити великі юридичні проблеми, штрафи та судові справи.

Розуміння потенційних наслідків атак на вебсервери є критично важливим для усвідомлення необхідності впровадження ефективних заходів безпеки, включаючи своєчасне виявлення та усунення вразливостей.

Окрім розуміння потенційних наслідків атак, не менш важливою є проактивна позиція у забезпеченні безпеки вебсерверів. Це означає не лише одноразове впровадження заходів захисту, а й їх постійний перегляд та оновлення.

Однією з найпоширеніших причин успішних атак є застаріле програмне забезпечення вебсерверів та встановлених на них додатків. Розробники регулярно випускають оновлення безпеки, які виправляють виявлені вразливості. Ігнорування цих оновлень створює "відкриті двері" для кіберзлочинців, які активно шукають відомі слабкі місця. Несвоєчасне оновлення може призвести до:

– експлуатації відомих вразливостей, де зловмисники можуть використовувати загальнодоступні відомості про недоліки в старому програмному забезпеченні для отримання несанкціонованого доступу.

– відсутності нових функцій безпеки - Сучасні версії програмного забезпечення часто включають покращені механізми захисту, які відсутні в попередніх версіях.

Впровадження автоматизованих систем захисту**,** таких як системи виявлення вторгнень (IDS) та системи запобігання вторгненням (IPS), є ще одним важливим кроком. Ці системи можуть в режимі реального часу відстежувати підозрілу активність і блокувати потенційні атаки, зменшуючи ризик компрометації [22].

1.5 Постановка задачі

У сучасному світі вебдодатки стали вирішальною складовою цифрової інфраструктури, пропонуючи величезний спектр послуг та перспектив для користувачів. Однак зростання їх складності та можливостей поєднується з підвищенням потенційної вразливості, яка може бути використана зловмисниками для завдань ураження. Серед найбільш поширених та небезпечних вразливостей вебдодатків є Cross-Site Scripting (XSS) та Cross-Site Request Forgery (CSRF).

Cross-Site Scripting (XSS) є типом вразливості, при якій шкідливий код дозволяє зловмиснику впроваджувати у зазначену вебсторінку, що переглядають інші користувачі. Це може привести до викрадення даних, перенаправлення на шкідливий сайт, зміни вмісту сторінки, розповсюдження шкідливого програмного забезпечення та інших зловмисних дій [23].

Cross-Site Request Forgery (CSRF) – атака, яка змушує авторизованого користувача виконувати небажані дії на вебдодатку без його відома. Зловмисник може відправляти підроблені запити від імені жертви, що може привести до зміни паролю, здійснення фінансових транзакцій, публікації небажаного контенту[20].

Зважаючи на потенційну серйозність наслідків цих вразливостей, розробка ефективних методів їхнього автоматизованого виявлення та аналізу є аутуальною та важливою задачою.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

– провести аналіз існуючих методів та інструментів для виявлення XSS та CSRF вразливостей вебдодатків;

– розробити алгоритми та програмні модулі для автоматизованого сканування вебдодатків на наявність XSS вразливостей, включаючи різні типи XSS (reflected, stored, DOM-based);

– розробити алгоритми та програмні модулі для автоматизованого виявлення CSRF вразливостей, враховуючи різні механізми захисту (токені CSRF, заголовки Referer та Origin);

– реалізувати розроблену систему у вигляді програмного застосунку;

– провести тестування розробленої системи на наборі тестових вебдодатків з відомими XSS та CSRF вразливостями для оцінки її ефективності та точності.

Вирішення поставлених задач дозволить створити ефективну та гнучку автоматизовану систему для аналізу XSS та CSRF вразливостей вебдодатків, що сприятиме підвищенню рівня їхньої безпеки.

2 ПРОЄКУВАННЯ СИСТЕМИ БЕЗПЕКИ

2.1 Обґрунтування обраного підходу

Головною метою автоматизованої системи є значне покращення безпеки вебдодатків шляхом автоматичного виявлення та звітування про дві критично важливі типи вразливостей. У контексті росту складності вебдодатків та збільшення кількості кіберзагроз, як атак типу Cross-Site Scripting або Cross-Site Request Forgery, є особливістю зростучості розробки ефективних методів їх виявлення.

XSS та CSRF є одними з найпоширеніших вразливостей вебдодатків. Успішна експлуатація цих вразливостей може передбачати викрадення конфіденційних даних користувачів, несанкціоновані дії та поширення шкідливого програмного забезпечення і інші серйозні результати.

Автоматизована система аналізу вразливостей вебдодатків розробляється з кількома ключовими цілями та перевагами. Основна мета полягає в тому, щоб автоматично виявляти потенційні вразливості типу XSS та CSRF. Це дозолить своєчасно виявляти та усувати недоліки до того, як ними зможуть скористуватися зловмисники.

Ручний аналіз коду та поведінки вебдодатків на предмет вразливостей є трудомістким, схильним до помилок та неефективним для великих і складних систем [24]. Автоматизовані системи аналізу пропонують значні переваги, включаючи:

– автоматизовані інструменти здатні сканувати великі обсяги коду та вебсторінок за значно коротший час порівняно з ручним аналізом;

– завдяки використанню формальних методів та алгоритмів, автоматизовані системи можуть виявляти вразливості з вищою точністю та консистентністю;

– автоматизовані рішення легше масштабуються для аналізу великої кількості вебдодатків та їхніх оновлень;

– інтеграція автоматизованих інструментів у процес розробки дозволяє виявляти вразливості на ранніх стадіях, що значно знижує вартість їхнього усунення.

Автоматизована система може проводити більш глибокий аналіз, генеруючи різноманітні тестові навантаження та аналізуючи реакцію вебдодатків на них. Це дозволить виявляти навіть складні та неочевидні вразливості. Також бонусом до всього стане, генерування звітів про знайдені вразливості, включаючи місцезнаходження, методи експлуатації та рекомендації щодо усунення.

В якості головної техніки для створення автоматизованої системи було вибрано мову програмування Python. Це рішення базується на ряді важливих переваг. Перше і важливе в моєму виборі було те, шо Python відрізняється простим і зрозумілим стилем, що значно швидше прискорює процес створення складних інструментів. Багато якісних бібліотек для роботи з мережами, а також для автоматизації браузерів, що робить Python ідеальним вибором для створення вебсистеми безпеки.

Для створення автоматизованої системи аналізу XSS та CSRF вразливостей вебдодатків, на мою думку підходить комбінований підхід, що поєднує використання існуючих бібліотек та розробку власних модулів для специфічних потреб. Перевагами комбінованого підходу, є:

– використання готових бібліотек, що значно прискорює розробку та надає перевірені інструменти для аналізу;

– розробка власних модулів, дозволить адаптувати систему під конкретні вимоги вебдодатків;

– створення власних модулів, допоможе краще зрозуміти механізми XSS та CSRF атак та способи їх виявлення.

Для ефективного виявлення XSS та CSRF вразливостей необхідний комплексний підхід, що включає парсинг вебсторінок, здійснення HTTP запитів, аналіз відповідей та автоматизацію процесу тестування.

Насамперед, що ж таке парсинг вебсторінок - це автоматизований процес збору та обробки інформації з вебресурсів. Він набагато кращий ніж «ручний» збір даних, і ось чому [25]:

– економія часу – тому що, цей автоматизований процес знизить навантаження команди та дозволить паралельно виконувати інші задачі;

– автономність – коли потрібно, дані будуть збиратися цілодобово, і це буде зроблено швидше за будь-яку людину;

– точність – програми чи будь-які скрипти сприймають задані параметри максимально точно, та шукають необхідний контент.

Нижче розглянуто ключові компоненти та підходи, які можуть бути використані при розробці інструментів для цієї мети.

Парсинг HTML та DOM. Для аналізу структури HTML-документів та навігації по Document Object Model (DOM) бібліотеки Beautiful Soup та lxml є незамінними інструментами.

- Beautiful Soup - забезпечує зручний інтерфейс для парсингу HTML та XML документів, дозволяючи легко знаходити елементи за тегами, атрибутами та їх значеннями. Її гнучкість робить її чудовим вибором для обробки навіть неідеально сформованого HTML [26];
- Lxml - є більш швидкою та потужною бібліотекою для парсингу XML та HTML, що особливо корисно при роботі з великими обсягами даних. Вона підтримує XPath та CSS селектори, що забезпечує більш витончені способи навігації по DOM [27].

Використання цих бібліотек дозволяє ефективно ідентифікувати потенційні точки впровадження XSS, аналізуючи структуру сторінки та виявляючи місця, де користувацький ввід може бути відображений у HTML.

Для виявлення XSS вразливостей потрібні спеціалізовані модулі, які включають генерацію payload (навантаження) [28], їх впровадження, відстеження та контекстний аналіз. Створення різноманітного набору XSS payload є ключовим

етапом [29]. Вони можуть включати теги для виконання коду, теги з обробниками подій або для виконання скриптів при помилці, різні способи обходу фільтрів.

Для кожної потенційної точки входу, такої як параметри URL та поля форм, необхідно автоматично вставляти згенеровані payload та відправляти відповіді HTTP запити. Важливо також відстежувати, як вони відображаються у відповіді сервера.

Після відправки запиту потрібно проаналізувати отриману відповідь. Ключовим є перевірка, чи відображаються введені payload в тілі відповіді без змін або належного екранування. Виявлення виконання коду є складнішим завданням для автоматизації і може вимагати використання певних інструментів.

Для виявлення CSRF вразливостей необхідні модулі, що здатні виявляти форми, шукати CSRF-токени та тестувати їхню наявність та валідність.

Використання Beautiful Soup [30] дозволяє легко знаходити всі HTML форми на вебсторінці, особливо ті, що використовують методи POST, PUT або DELETE, оскільки саме ці методи зазвичай використовуються для виконання дій, які можуть бути зловмисно використані через CSRF.

Після виявлення форм необхідно проаналізувати їхній HTML-код на наявність прихованих полів, які можуть містити CSRF-токени. Зазвичай ці поля мають імена на зразок csrf_token, authenticity_token або подібні, і містять випадкові, непередбачувані значення.

Для кожної знайденої форми, що виконує важливу дію, слід спробувати відтворити відповідний HTTP запит без включення CSRF-токена. Якщо сервер успішно обробляє такий запит, це свідчить про потенційну CSRF вразливість.

Якщо CSRF-токени присутні, необхідно перевірити їхню валідність на сервері. Це може включати спроби відтворити запит з:

- відсутнім токеном (як описано вище);
- неправильним або підробленим токеном;
- використаним токеном (якщо сервер не реалізує механізм одноразових токенів).

Для ефективного та систематичного тестування великої кількості вебсторінок та параметрів автоматизація є ключовою. Також важливим є надання чітких та інформативних звітів про знайдені вразливості.

Використання класів та функцій для модульної організації коду робить інструмент більш керованим, розширюваним та легким для підтримки.

Параметри цільового вебдодатку, такі як базовий URL, список сторінок для сканування, набір XSS payload, повинні бути легко конфігурованими, можливо, через файли конфігурації або аргументи командного рядка.

Згенеровані звіти повинні бути зрозумілими та містити всю необхідну інформацію про знайдені вразливості, включаючи:

- URL, на якому була виявлена вразливість;
- параметр або поле форми, яке було вразливим;
- використаний payload (для XSS);
- опис потенційної загрози та можливі наслідки експлуатації.

Звіти можуть бути форматовані для простоти читання або згенеровані як HTML-документи для більш розширеного форматування та можливості включення інтерактивних елементів.

2.2 Архітектура автоматизованої системи аналізу безпеки

В основу архітектури покладено комбінований підхід, що передбачає використання як готових, перевірених бібліотек мови програмування Python, так і розробку власних спеціалізованих модулів. Таке поєднання дозволяє досягти оптимального балансу між швидкістю розробки, ефективністю аналізу та гнучкістю налаштування системи під конкретні потреби досліджуваних вебдодатків.

Система розроблена як модульна програма, написана мовою Python. Її функціонування являє собою чітко визначену послідовність етапів, де кожен етап

виконується окремим, взаємодіючим компонентом. Основною метою є автоматизація процесу виявлення XSS (міжсайтового скриптингу) та CSRF (міжсайтової підробки запитів) вразливостей шляхом активного тестування вебдодатків. Загальну архітектуру системи можна представити у вигляді наступних послідовних кроків. Структуру проекту можна побачити на рис.2.1.

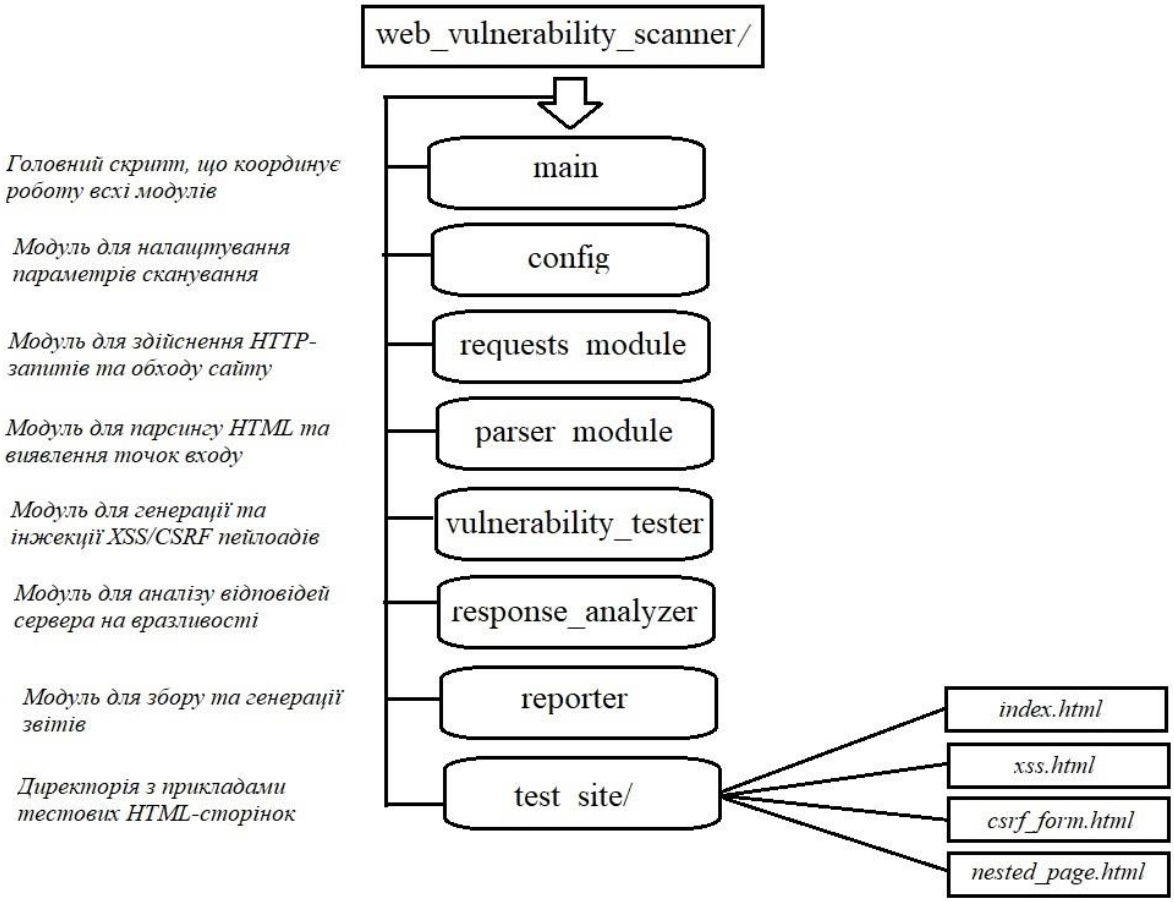


Рисунок 2.1 – Структура автоматизованої системи

На початковому етапі відбувається ініціація процесу аналізу. Користувач взаємодіє з системою через зручний веб-інтерфейс або програмний інтерфейс (API), надаючи URL-адресу цільового вебдодатку, який необхідно проаналізувати. Крім того, користувач має можливість налаштувати різноманітні параметри сканування, такі як:

– глибина рекурсивного обходу посилань, знайдених на вебсторінках, для розширення області аналізу;

– набір XSS-payload, які будуть використовуватись для тестування на вразливість до міжсайтового скриптингу;

– необхідність та особливості перевірки на вразливість до міжсайтової підробки запитів (CSRF) на різних типах HTML-форм. Модуль управління скануванням приймає введені користувачем налаштування та на їх основі формує внутрішній план сканування, визначаючи послідовність дій та параметри для наступних етапів аналізу.

На рисунку 2.2. можна побачити етапи виконання пошуку XSS вразливостей.

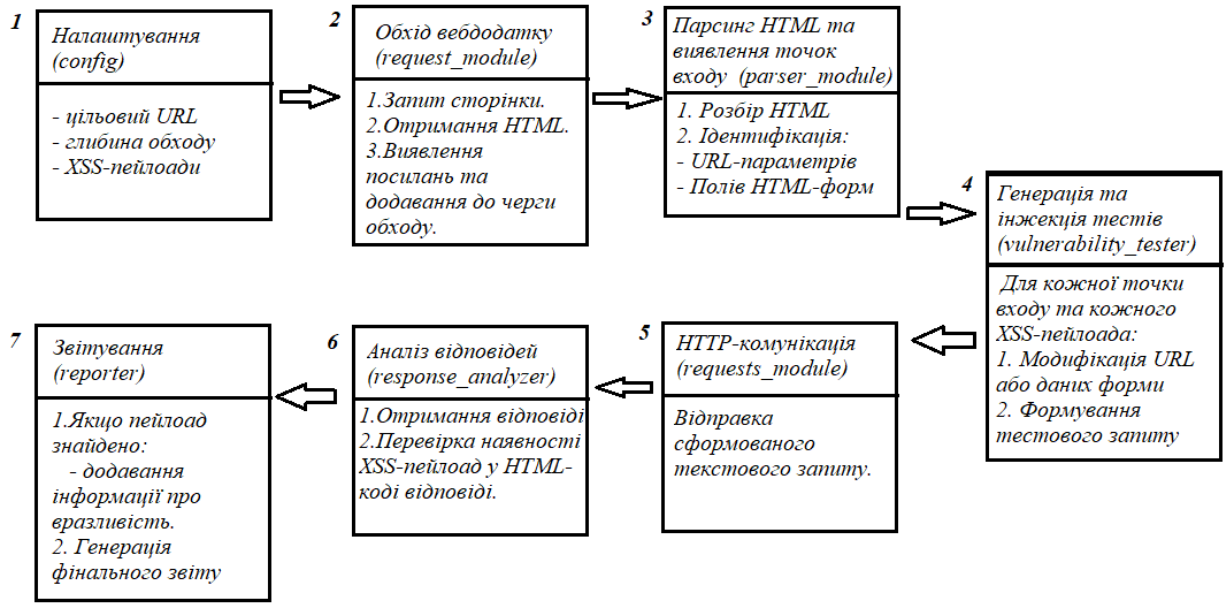


Рисунок 2.2 – Етапи виконання пошуку XSS вразливостей

У випадку із CSRF вразливістю, етап виконання пошуку схожий до XSS. Це можна побачити на рисунку 2.3.

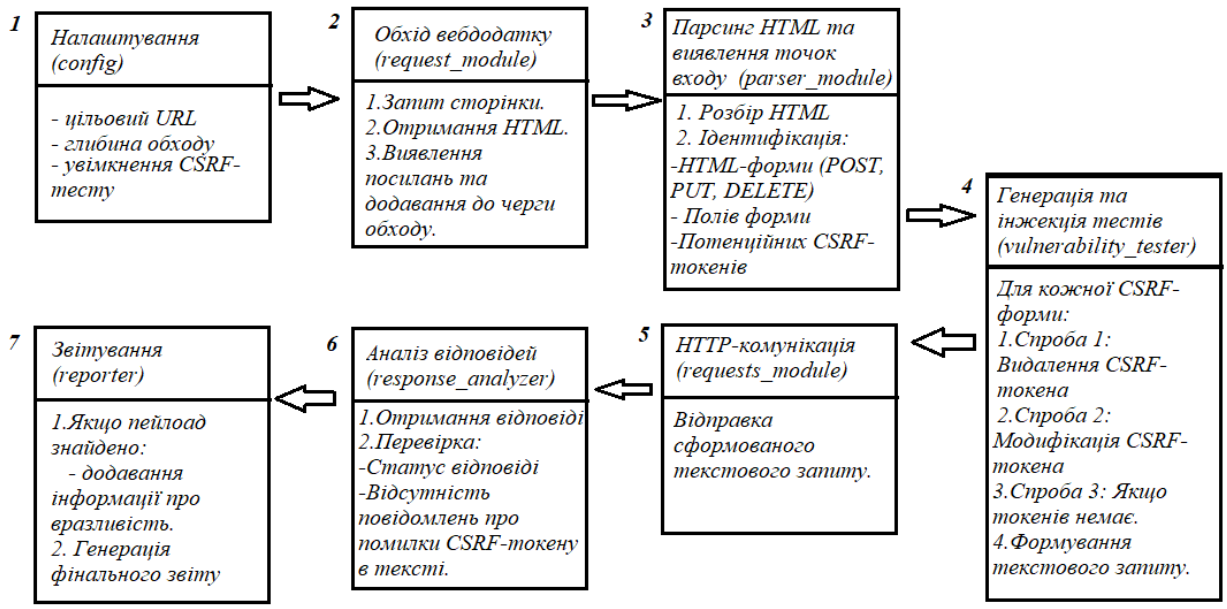


Рисунок 2.3 – Етапи виконання пошуку CSRF вразливостей

Після успішної ініціації сканування, в роботу вступає модуль HTTP-комунікації. Цей модуль, що використовує можливості бібліотеки requests мови Python, здійснює початковий HTTP-запит до вказаної URL-адреси цільового вебдодатку. Для більш глибокого аналізу, система може автоматично переходити за знайденими на сторінках посиланнями, здійснюючи рекурсивний обхід вебдодатку на задану користувачем глибину. Отриманий HTML-контент кожної відвіданої вебсторінки передається на наступний етап - до модуля парсингу HTML та DOM.

Модуль парсингу HTML та DOM, що базується на бібліотеках Beautiful Soup або lxml, здійснює обробку отриманого HTML-коду кожної вебсторінки. Основною метою цього етапу є виявлення потенційних точок входу для XSS-атак та ідентифікація елементів, важливих для аналізу CSRF-вразливостей:

Для XSS:

- модуль аналізує структуру HTML-документу з метою виявлення наступних потенційно вразливих місць [31]:

– параметри, що передаються у URL-адресі вебсторінок;

– значення полів введення, текстових областей та елементів вибору у HTML-формах;

– місця в HTML-коді, де динамічно відображаються дані, отримані з URL-адреси або полів форм.

Для CSRF:

- модуль здійснює пошук усіх HTML-форм на вебсторінці, особливу увагу приділяючи формам, що використовують HTTP-методи, які можуть призвести до зміни стану сервера (наприклад, POST, PUT, DELETE).

На наступному етапі система автоматично генерує та впроваджує спеціально розроблені тести для виявлення XSS та CSRF вразливостей у виявлених точках входу:

Для XSS:

– модуль генерації payload створює набір різноманітних XSS-пейлоадів. Ці шкідливі скрипти та HTML-конструкції розробляються з урахуванням різних сценаріїв впровадження та можливих методів фільтрації вхідних даних;

– модуль інжектора payload для кожної ідентифікованої потенційної точки входу формує нові HTTP-запити, замінюючи очікувані значення на згенеровані XSS- payload;

– модуль HTTP-комунікації надсилає ці навмисно сформовані запити до цільового вебсервера.

Для CSRF:

– модуль аналізу токенів здійснює спробу знайти CSRF-токени у виявлених HTML-формах. CSRF-токени зазвичай представлені прихованими полями з випадковими, непередбачуваними значеннями;

– модуль генерації тестових запитів CSRF створює копії оригінальних HTTP-запитів, що відправляються методом POST (або іншими модифікуючими методами), але при цьому видаляє або модифікує значення знайдених CSRF-токенів;

– модуль HTTP-комунікації надсилає ці сфальсифіковані запити до вебсервера для перевірки.

Після відправки тестових запитів, система аналізує отримані від вебсервера відповіді для виявлення ознак наявності XSS та CSRF вразливостей:

Для XSS:

– модуль аналізу відповідей XSS отримує та аналізує HTML-код відповідей на запити з впровадженими XSS-пейлоадами;

– основною метою є перевірка, чи відображається впроваджений шкідливий код у несанітизованому вигляді у відповіді сервера. Для цього може застосовуватися простий текстовий пошук впровадженого пейлоада в HTML-коді;

– модуль контекстного аналізу намагається визначити контекст, в якому відображається введений користувачем текст. Ця інформація є важливою для точної оцінки серйозності вразливості та визначення ефективних способів її експлуатації.

Для CSRF:

– модуль аналізу відповідей CSRF аналізує відповіді сервера на сфальсифіковані запити, в яких були відсутні або модифіковані CSRF-токени;

– якщо вебсервер приймає та обробляє такий запит так само, як і легітимний запит з дійсним CSRF-токеном, це є чіткою ознакою наявності потенційної CSRF-вразливості.

На завершальному етапі модуль звітування збирає всю інформацію про виявлені потенційні XSS та CSRF вразливості. Звіт може містити наступні ключові дані:

– URL-адреса вебсторінки, на якій була виявлена вразливість;

– назва параметра URL або поля форми, пов'язаного з вразливістю;

– використаний XSS-payload або опис атаки CSRF;

– детальний опис виявленої вразливості та оцінка її потенційного впливу на безпеку вебдодатку та його користувачів.

2.3 Висновок

У процесі розробки автоматизованої системи обґрунтування обраного підходу підкреслило нагальну потребу в автоматизації процесів аудиту безпеки з огляду на зростаючу складність вебінфраструктури та еволюцію кіберзагроз. Переваги автоматизованого аналізу, такі як швидкість, точність, масштабованість та можливість раннього виявлення вразливостей у циклі розробки, роблять його незамінним інструментом у забезпеченні безпеки вебдодатків.

Вибір мови програмування Python як основної платформи для розробки зумовлений її потужними можливостями, широким спектром бібліотек для роботи з мережею та веб-технологіями, а також зрозумілим синтаксисом, що сприяє прискоренню розробки. Ключовим елементом проєктування є комбінований підхід, який ефективно поєднує використання готових бібліотек для виконання стандартних завдань з розробкою власних спеціалізованих модулів, адаптованих під специфіку виявлення XSS та CSRF вразливостей.

Архітектура системи, що базується на модульному принципі, забезпечує чітку організацію та взаємодію між окремими компонентами. Кожен модуль відповідає за певний етап процесу аналізу, від ініціації сканування до формування кінцевого звіту. Така структура не лише полегшує розробку та підтримку системи, але й забезпечує гнучкість у налаштуванні та розширенні її функціональності в майбутньому.

Створена архітектура демонструє ефективний підхід до автоматизації аналізу безпеки вебдодатків. Поєднання надійності перевірених бібліотек з цілеспрямованою розробкою власних модулів дозволяє створити інструмент, здатний оперативно та якісно виявляти XSS та CSRF вразливості, надаючи розробникам та фахівцям з безпеки необхідну інформацію для їхнього своєчасного усунення та запобігання потенційним загрозам. Подальші дослідження можуть бути спрямовані на розширення спектру виявлюваних вразливостей та інтеграцію системи з існуючими інструментами безпеки.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ АВТОМАТИЗОВАНОЇ СИСТЕМИ АНАЛІЗУ ВРАЗЛИВОСТЕЙ

3.1 Інструментарій розробки та технології реалізації системи

У контексті даної роботи, представлено базовий сканер веб-вразливостей, де розроблено алгоритми та програмні модулі, необхідні для ефективного автоматизованого виявлення вразливостей. Виявлення XSS та CSRF вразливостей є ключовим завданням для забезпечення безпеки вебдодатків. Враховуючи основні типи вразливостей, дана система передбачає спеціалізовані підходи та розширення існуючих модулів.

Розроблений базовий сканер веб-вразливостей є важливою основою для забезпечення безпеки вебдодатків, адже він автоматизує виявлення критичних вразливостей XSS та CSRF. Його модульна архітектура забезпечує гнучкість та масштабованість, дозволяючи легку інтеграцію нових функцій та розширення можливостей.

Система побудована за модульним принципом, який забезпечує гнучкість, масштабованість та легкість підтримки. Кожен, з розробленим мною модулів відповідає за певну функціональність, що взаємодіє з іншими компонентами для досягнення поставленої мети.

Розроблений сканер має такі основні програмні модулі, які координуються головним скриптом main:

– модуль налаштування користувача (config) – відповідає за параметри сканування, такі як цільова URL-адреса, глибина обходу списки тестових пейлоадів та активація або деактивація певних типів тестів [32];

– модуль HTTP-комунікації (requests_module) - здійснює всі HTTP-запити до цільового вебдодатку, а також реалізує механізм рекурсивного обходу вебсайту для виявлення всіх доступних сторінок [33];

– модуль парсингу HTML та DOM (parser_module) - аналізує отриманий HTML-контент сторінок для ідентифікації потенційних точок входу, куди можуть бути ін'єктовані шкідливі дані (параметри URL, поля форм) [34];

– модуль генерації та інжекції тестів (vulnerability_tester) - Відповідає за формування та впровадження спеціалізованих XSS та CSRF пейлоадів у виявлені точки входу [35];

– модуль аналізу відповідей (response_analyzer) - оцінює відповіді вебсервера після тестових запитів, використовуючи логіку для виявлення ознак успішного спрацювання вразливостей;

– модуль звітування (reporter) - збирає інформацію про виявлені вразливості та генерує структурований звіт;

– головний скрипт (main) - координує послідовність виконання всіх модулів, забезпечуючи логічний потік роботи сканера [36].

Виявлення XSS-вразливостей вимагає специфічних алгоритмів для кожного з її основних типів: Reflected, (Stored) та DOM-based**.** Моя реалізація, зосереджується на Reflected XSS через URL-параметри та форми, а також закладає основи для подальшого розширення до Stored та DOM-based.

Алгоритм конфігурації цільової URL та глибини обходу, працює так що коли користувач задає початкову URL та максимальну глибину рекурсивного обходу, то це впливає на кількість сторінок які будуть просканові. Модуль містить попередньо визначений список XSS-пейлоадів. Цей список також можна розширювати для включення більш складних та обхідних варіантів.

Модуль HTTP-комунікації для XSS, реалізовано за допомогою алгоритму обходу в ширину за допомогою черги deque. Механізм даного модуля такий: початкова цільова URL додається до черги разом з поточною глибиною (0), на кожній ітерації з черги вилучається URL, здійснюється HTTP GET-запит за допомогою session.get(), отримана HTML відповідь передається до парсингу, з отриманого HTML витягуються всі посилання та фільтруються, щоб залишити лише ті, що належать цільовому домену, нові, ще не відвідані URL додаються до черги з інкрементованою глибиною, доки не досягнуто max_depth.

Модуль Парсингу HTML та DOM для XSS, реалізовано за допомогою алгоритму виявлення точок входу. Парсинг URL-параметрів, де функція

parse_page використовує urllib.parse для розбору URL (urlparse) та вилучення всіх запитових параметрів (parse_qs). Кожен виявлений параметр вважається потенційною точкою для ін'єкції XSS (xss_params).

Парсинг HTML-форм, де використовуються BeautifulSoup4 для ідентифікації всіх <form> елементів на сторінці (soup.find_all('form')). Для кожної форми:

- витягуються атрибути method (GET/POST) та action (URL для відправки форми);
- функція _extract_form_data рекурсивно проходить по всіх полях вводу (<input>, <textarea>, <select>), збираючи їх name та value;
- всі виявлені поля форми додаються до xss_forms.

Базовий "DOM-based" пошук (спрощено): Поточна реалізація включає дуже спрощений пошук відбиття значень URL-параметрів безпосередньо в тексті HTML-відповіді. Це не є повноцінним DOM-базованим аналізом, який вимагає виконання у безголовому браузері. Проте, цей підхід може виявити прості випадки, коли дані з URL-параметра відбиваються без належного кодування у тілі сторінки. Для повноцінного DOM-based XSS аналізу потрібна інтеграція з інструментами на кшталт Selenium або Playwright, що дозволяють:

– динамічно інструментувати JavaScript, тобто перехоплювати виклики функцій, що маніпулюють DOM (наприклад, innerHTML, document.write).

– відстежувати потік даних (Taint Analysis), визначати, чи дані, що походять з контрольованих зловмисником джерел (URL, localStorage, window.name), потрапляють до небезпечних "приймачів" (sinks).

Модуль Генерації та Інжекції Тестів для XSS побудований на алгоритмі інжекції. Для кожного параметра URL зі списку xss_params (отриманого від parser_module.py):

– створюється копія поточних параметрів запиту;

– значення цільового параметра замінюється на кожен xss_payload зі списку config.xss_payloads;

– формується новий URL, ініціюється GET-запит через http_module.fetch_page().

Якщо vulnerability_tester.py впроваджує пейлоад у форму, яка зберігає дані (наприклад, форму коментарів), то для виявлення Stored XSS після цього запиту, main.py повинен (у майбутньому розширенні) ініціювати повторний обхід або цільовий запит до сторінки, де ці дані відображаються, а потім response_analyzer.py мав би проаналізувати її.

Модуль Аналізу Відповідей для XSS поєднаний з алгоритмом аналізу Reflected XSS:

– модуль перевіряє, чи повний XSS-пейлоад (payload) присутній у тексті HTML-відповіді (response.text);

– пошук характерних частин пейлоаду або його кодованих/обфускованих версій;

– визначення, в якому HTML-контексті відображається пейлоад (наприклад, всередині script тегу, в атрибуті onerror), що вказує на його експлуатабельність;

– для виявлення DOM-based XSS, а також для більш надійного виявлення Reflected/Stored XSS, модуль повинен ініціювати завантаження сторінки у безголовому браузері;

– моніторинг викликів функцій JavaScript, таких як window.alert(), confirm(), prompt(), які є прямими індикаторами спрацювання XSS;

– відстеження змін у DOM-дереві, що можуть бути спричинені ін'єкцією шкідливого скрипту.

Виявлення CSRF-вразливостей зосереджується на ідентифікації форм та дій, які не мають належного захисту, такого як CSRF-токени, або які не перевіряють заголовки Referer та Origin.

Модуль налаштувань користувача (config.py) для CSRF це включення/виключення CSRF-тесту де параметр enable_csrf_test дозволяє контролювати виконання тестів на CSRF.

Модуль Парсингу HTML та DOM для CSRF. Алгоритми виявлення точок входу, працює наступним чином:

– модуль шукає <form> елементи, які використовують HTTP-методи POST, PUT або DELETE. Ці методи частіше використовуються для виконання дій, що змінюють стан на сервері (наприклад, зміна пароля, видалення облікового запису), і тому є основними цілями для CSRF-атак;

– для кожної такої форми, модуль _extract_form_data та основна логіка parse_page намагається ідентифікувати приховані поля (<input type="hidden">) або інші елементи, назви яких містять типові ключові слова для CSRF-токенів (наприклад, "token", "csrf", "nonce", "authenticity_token");

– поточна реалізація здійснює лише базовий пошук за назвою поля.

– Для більш надійного виявлення токенів потрібен:

– CSRF-токени можуть бути динамічно вбудовані JavaScript'ом або передаватися в HTTP-заголовках AJAX-запитів. Для цього потрібен складніший статичний/динамічний аналіз JavaScript;

– для пошуку токенів з нетиповими назвами, але характерною структурою.

Модуль генерації та інжекції тестів для CSRF, відповідає за створення та відправку спеціальних запитів, щоб перевірити, чи веб-додаток захищений від атак Cross-Site Request Forgery (CSRF). CSRF-атаки дозволяють зловмиснику змусити користувача виконати небажані дії на веб-сайті, на якому він вже автентифікований.

Усі модифіковані запити відправляються на цільову URL-адресу форми за допомогою модуля HTTPModule (http_module.fetch_page()). Зазвичай такі дії (наприклад, зміна пароля, видалення даних) виконуються методом POST.

Важливо зазначити, що при відправці цих запитів використовується параметр allow_redirects=False. Це означає, що забороняється автоматичне перенаправлення браузера після запиту.

Модуль аналізу відповідей відіграє ключову роль у визначенні, чи був успішним тестовий CSRF-запит, і відповідно, чи вебдодаток є вразливим до цього

типу атаки. Він аналізує відповіді сервера після того, як ми відправили запити з видаленим або зміненим токеном.

Модуль parser_module.py шукає <form> елементи, що використовують HTTP-методи POST, PUT або DELETE, оскільки вони найчастіше застосовуються для зміни стану на сервері. Для цих форм здійснюється спроба ідентифікувати приховані поля (<input type="hidden">) або інші елементи, назви яких містять типові ключові слова для CSRF-токенів (наприклад, "token", "csrf", "nonce", "authenticity_token").

Vulnerability_tester.py відповідає за створення та відправку модифікованих запитів (зазвичай POST) на цільову URL-адресу форми через http_module.fetch_page(). При цьому використовується параметр allow_redirects=False для запобігання автоматичному перенаправленню браузера, що є важливою умовою для тестування CSRF. Запити відправляються без або зі зміненими CSRF-токенами.

Response_analyzer.py аналізує відповіді сервера після таких тестових запитів. Його логіка спрямована на визначення, чи вебдодаток виконав дію, незважаючи на відсутність або некоректність CSRF-токенів, що вказуватиме на вразливість.

Майбутні вдосконалення системи включатимуть:

- поглиблення аналізу CSRF-токенів, тобто розробка складніших алгоритмів для виявлення токенів, які динамічно генеруються JavaScript або передаються в HTTP-заголовках AJAX-запитів, а також пошук токенів з нетиповими назвами, але характерною структурою;
- інтеграція фаззингу - для підвищення ефективності виявлення XSS планується додавання механізмів фаззингу, що дозволить генерувати більш різноманітні та складні обфусковані пейлоади;
- оптимізація продуктивності - впровадження асинхронних запитів та паралельної обробки для прискорення сканування великих вебдодатків;

– розширення функціоналу - можливе розширення сканера для виявлення інших критичних вразливостей, таких як SQL Injection, Broken Authentication та Sensitive Data Exposure;

– покращення звітування - надання більш детальних рекомендацій щодо усунення виявлених вразливостей, включаючи приклади патчів та конфігурацій.

3.2 Налаштування автоматизованої системи

Для ефективної та масштабованої розробки сканера вразливостей я застосувала модульну структуру. Кожен компонент відповідає за свою унікальну функціональність, що спрощує підтримку, тестування та подальше розширення проєкту. Основні компоненти автоматизованої системи були описані раніше.

Автоматизована система розроблялася в PyCharm, як на мене ця програма є дуже зручною та багатофункціональною, та підходить для мого проєкту.

Перед тим як почати роботу, потрібно створити на своєму комп'ютері папку (web_vulnerability_scanner), з якою далі будемо працювати. В цій паці знаходяться файли .py та папка test_site з HTML-файлами. Для того щоб перевірити як працює сканер, потрібен простий вебсервер, для цього і були створені HTML-файли. Приклад створених файлів можна побачити на рис.3.1.

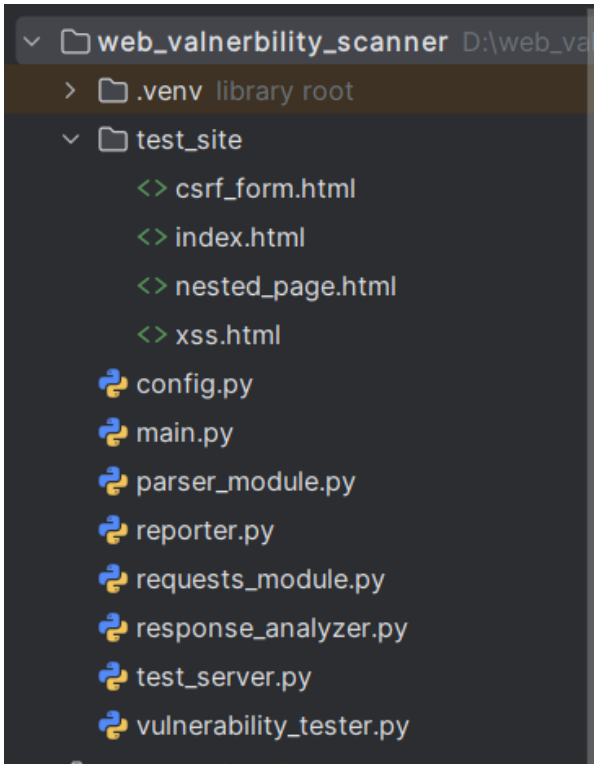


Рисунок 3.1 – Структура проекту

Наступне, що важливо зробити, це налаштувати інтерпретатор Python. В налаштуваннях програми, з якою працюємо, переходимо до Python Interpreter, адже потрібно створити віртуальне середовище. Деталі для створення віртуального середовища, можна побачити на рис.3.2.

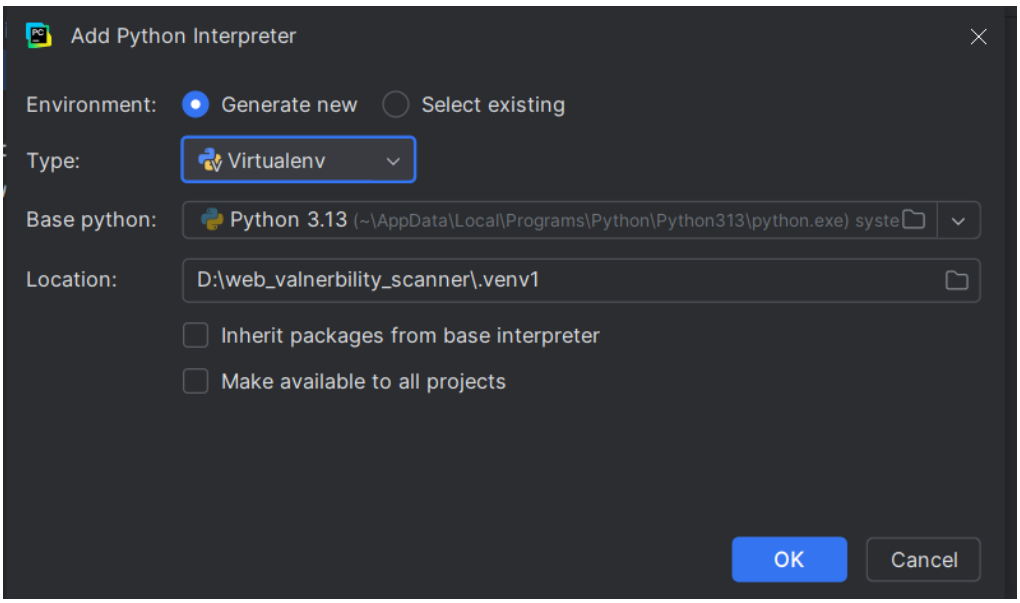


Рисунок 3.2 – Створення віртуального середовища

Після створення віртуального середовища, потрібно встановити необхідні бібліотеки (requests, beautifulsoup4, lxml), для подальшої роботи. Потрібно відрити Terminal та ввести команду, яка необхідна для встановлення потрібних бібліотек. Потрібну команду можна побачити на рис. 3.3. Головне, потрібно переконатися що віртуальне середовище активоване.

```
(.venv) PS D:\web_valnerbility_scanner> pip install requests beautifulsoup4 lxml
```

Рисунок 3.3 -Команда для встановлення бібліотек

Для того щоб зручно запускати програму, можна налаштувати Run/Debug Configurations. Перше що обрала - це Python. Дала конфігурації назва яка буде мені зрозуміла, в полі script, обрала файл який веде до проекту. Деталі заповнення можна побачити на рис.3.4.

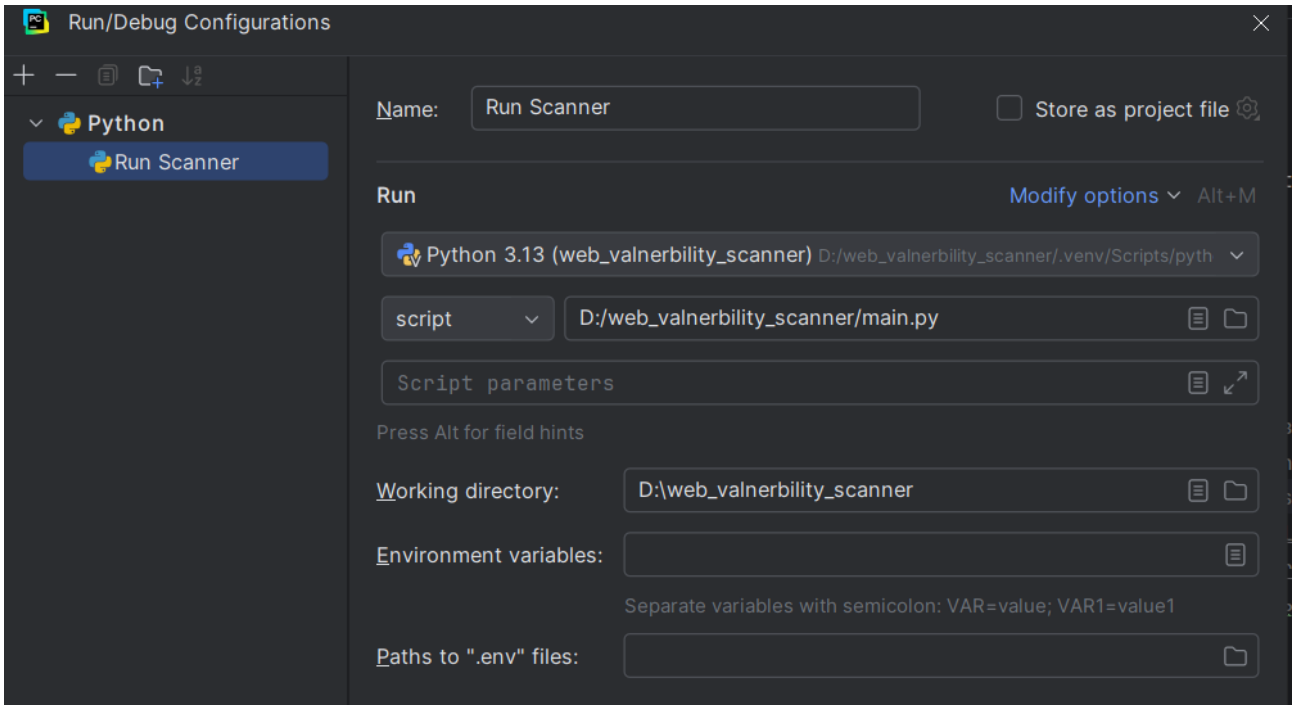


Рисунок 3.4 - Налаштування Run/Debug Configuration

Коли налаштування готове, можна розпочинати підготовку до тестування системи. Перш ніж запускати сканер, потрібен тестовий вебсайт. Будемо

використовувати простий вбудований HTTP-сервер Python. Для цього в терміналі ввожу команду, за допомогою якої, переходжу до папки test_site. Потрібну команду можна побачити на рис.3.5.

```
(.venv) PS D:\web_valnerbility_scanner> cd test_site
```

Рисунок 3.5 – Підготовка до тестування

Щоб сканер міг повноцінно працювати і виявляти вразливості, потрібно запустити більш складний вебсервер, який зможе:

– обробляти POST запити;

– імітувати "уразливі" ендпоінти, які динамічно відображають вхідні дані (для XSS) або обробляють форми без належної перевірки (для CSRF).

Найпростіший спосіб це зробити – використати мікро-фреймворк, наприклад, Flask. Його потрібно встановити через термінал за допомогою команди (рис.3.6). Обов’язково з відкритим віртуальним середовищем venv.

```
(.venv) PS D:\web_valnerbility_scanner> pip install Flask
```

Рисунок 3.6 – Встановлення мікро-фреймворку

Після успішного виконання команди, запускаю наступна команду (рис.3.7), для того щоб запустити HTTP-сервер.

```
(.venv) PS D:\web_valnerbility_scanner\test_site> python -m http.server 8000
```

Рисунок 3.7 – Запуск HTTP-сервера

Тепер коли сервер працює, і PyCharm налаштований можна запускати програму. Вивід програми буде відображатися у вікні «Run». Повинен

відображатися процес обходу, виявлення точок входу, тестування та фінальний звіт.

3.3 Тестування системи

Розроблена автоматизована система проходила тестування з метою перевірки її функціональності, точності виявлення XSS та CSRF вразливостей. Автоматизовану систему побудовану на модульній архітектурі, що забезпечує гнучкість та розширюваність. Опис кожного з модулів було наведено в тексті раніше.

Перед використанням системи необхідно врахувати певні аспекти, такі як:

– програма генерує та відправляє потенційно шкідливі запити, потрібно використовувати її виключно на власних тестових середовищах або з явного дозволу власника ресурсу;

– виявлення реальних вразливостей XSS та CSRF є складним завданням;

– для стислості коду, обробка помилок є мінімальною. У реальних умовах необхідно впровадити більш надійний механізм.

Для того щоб програма почала працювати, потрібно, запустити HTTP-сервер, команда якого зображена на рис.3.8, після цього запустити main.py. І тільки тоді почнеться сканування.

```
(.venv) PS D:\web_valnerbility_scanner> python test_server.py
```

Рисунок 3.8 - Запуск тестового Flask-сервера

Після сканування можна розглянути, що знайшла програма. Отже, якщо детальніше подивитись на рис.3.9, то можна побачити наступне:

– програма почала перевіряти сайт за адресою http://localhost:8000/;

– вона заглянула на всі сторінки, які були на відстані одного кліка від головної сторінки (глибина обходу:1);

– вона знайшла кілька місць, де можуть бути проблеми з безпекою.

```
[Етап 1: Обхід та Парсинг Вебдодатку]
Початок обходу сайту http://localhost:8000/ до глибини 1...
Обхід: http://localhost:8000/ (Глибина: 0)
  Запит: GET http://localhost:8000/
  Знайдено 0 URL-параметрів для XSS.
  Знайдено 1 форм для XSS.
  Знайдено 1 форм для CSRF.
Обхід: http://localhost:8000/xss_vulnerable?name=John (Глибина: 1)
  Запит: GET http://localhost:8000/xss_vulnerable?name=John
  Знайдено 2 URL-параметрів для XSS.
  Знайдено 2 форм для XSS.
  Знайдено 1 форм для CSRF.
Обхід: http://localhost:8000/csrf_test (Глибина: 1)
  Запит: GET http://localhost:8000/csrf_test
  Знайдено 0 URL-параметрів для XSS.
  Знайдено 2 форм для XSS.
  Знайдено 2 форм для CSRF.
Обхід: http://localhost:8000/nested_page (Глибина: 1)
  Запит: GET http://localhost:8000/nested_page
  Знайдено 0 URL-параметрів для XSS.
  Знайдено 0 форм для XSS.
  Знайдено 0 форм для CSRF.

Загалом знайдено точок входу:
  XSS URL-параметри: 2
  XSS форми: 5
  CSRF форми: 4
```

Рисунок 3.9 – Обхід та парсинг вебдодатку

Другий етап який зображено на рис.3.10, це робота сканера, зосередженого на XSS. На цьому етапі сканер активно "нападає" на знайдені точки входу, використовуючи різні XSS-payloads. Мета — спровокувати виконання шкідливого коду на стороні клієнта (браузера). Якщо будь-який з цих запитів призведе до успішного виконання скрипта, це означатиме, що знайдено XSS-вразливість. Лог показує, які конкретні запити та з якими payload'ами

відправляються. Далі очікується результат кожного тесту (наприклад, "XSS-вразливість знайдена" або "Немає вразливостей").

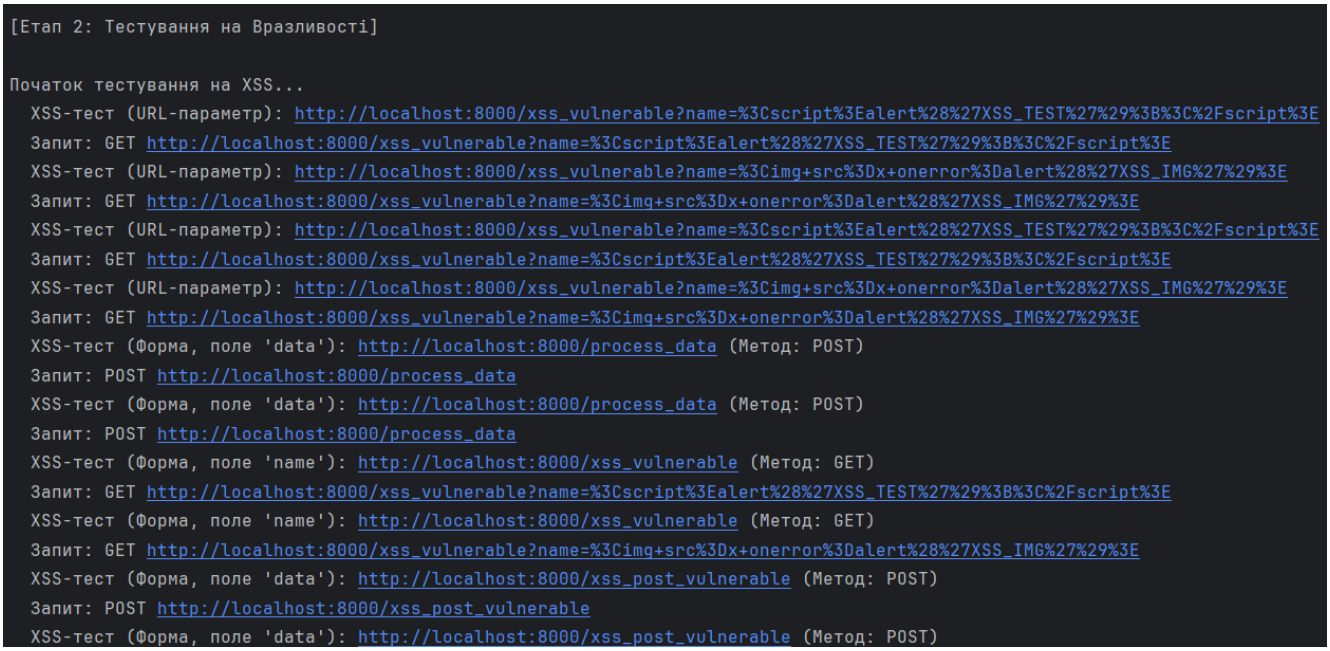


Рисунок 3.10 – Тестування на вразливості

На рис.3.11. можна побачити, що лог чітко вказує на те, що веб-додаток на http://localhost:8000/ є дуже вразливим до XSS-атак**.** Шкідливий код, відправлений через URL-параметри та форми (як GET, так і POST запитами), відображається назад у відповіді сервера, що дозволяє його виконання в браузері користувача. Це серйозна проблема безпеки, яка може бути використана для крадіжки сесійних кукі, перенаправлення користувачів на шкідливі сайти, дефейсу сторінок або виконання інших довільних дій в браузері користувача.

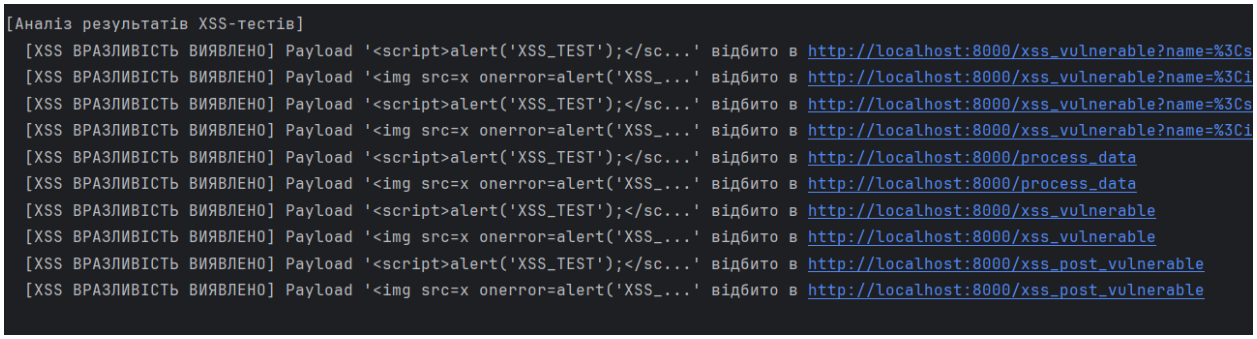


Рисунок 3.11 – Аналіз результатів XSS вразливостей

На рис.3.12, можна побачити що сканер успішно виявив три CSRF-вразливості у веб-додатку за адресою http://localhost:8000/. Ці вразливості знаходяться у формах, які обробляють запити на /process_data, /xss_post_vulnerable та /change_password_no_token. У всіх цих випадках сервер приймає запити, навіть якщо вони не містять або містять неправильний CSRF-токен, що робить додаток вразливим до атак, де зловмисник може змусити користувача виконати небажану дію.

На противагу цьому, форми на /change_password_with_token (які сканер тестував з видаленим або зміненим токеном) виявилися захищеними, оскільки сервер правильно відхилив такі запити з помилкою 403 FORBIDDEN.

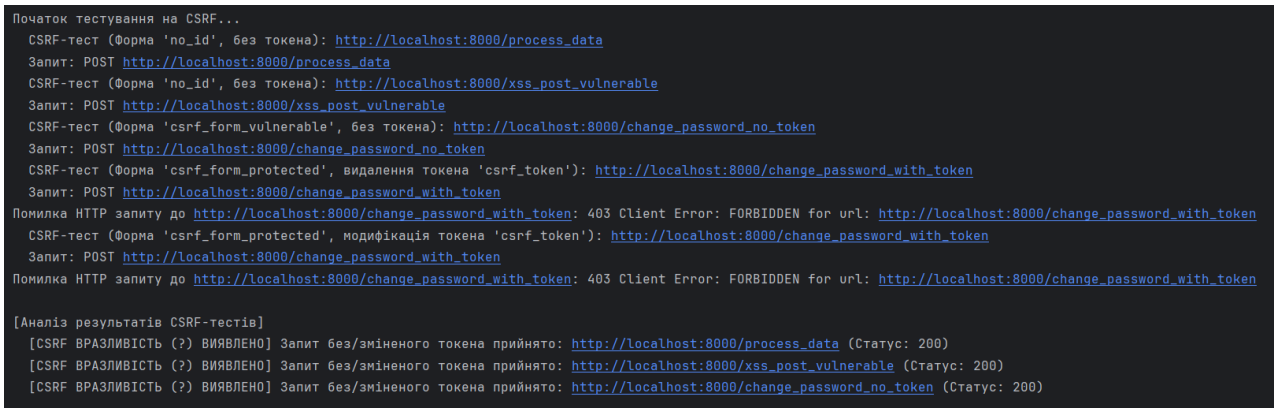


Рисунок 3.12 – Тестування на CSRF

Звіт чітко документує дві окремі XSS-вразливості, обидві виявлені на сторінці http://localhost:8000/xss_vulnerable через параметр name. Сканер успішно підтвердив, що два різні XSS-пейлоади ("звичайний" скрипт та скрипт через помилку зображення) були "відбиті" сервером, що свідчить про наявність Reflected XSS (відбитого XSS). Це дозволить зловмиснику вставляти шкідливий JavaScript у відповідь сервера, який потім буде виконаний у браузері жертви. Генерацію звіту можна побачити на рис.3.13.

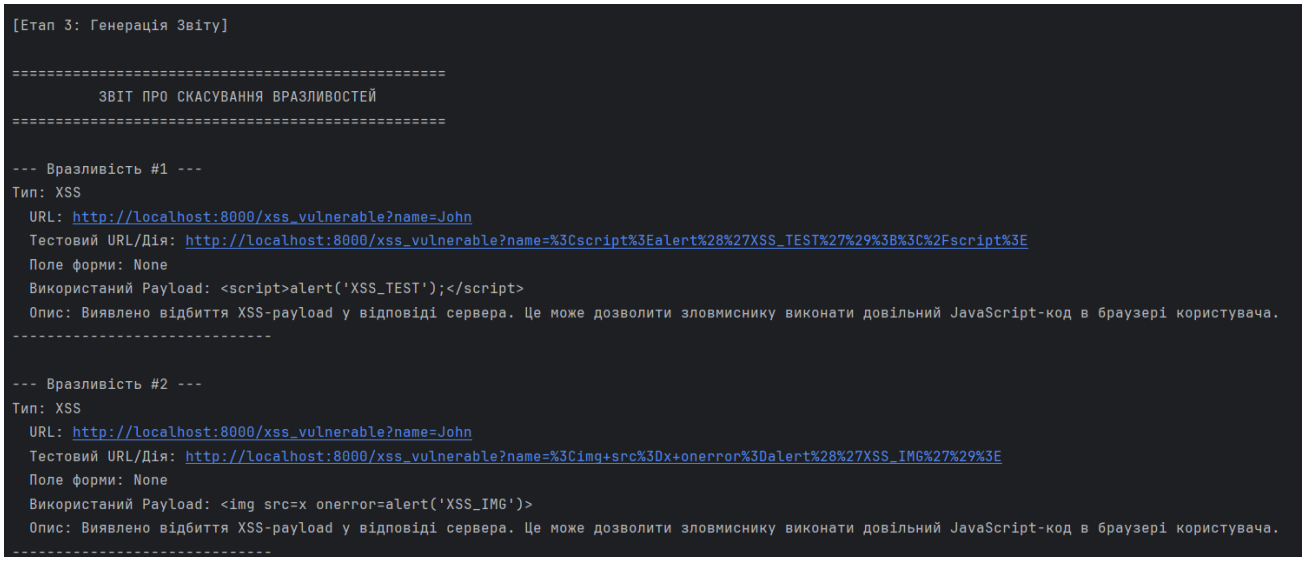


Рисунок 3.13 – Генерація звіту

Після успішного сканування, можна передивитись дані які висвітились , та детальніше все розглянути.

3.4 Висновок

У цій роботі представлено базовий модульний сканер веб-вразливостей, що автоматично виявляє XSS (Cross-Site Scripting) та CSRF (Cross-Site Request Forgery) вразливості. Система складається з ключових модулів:

– Config -керує налаштуваннями сканування (URL, глибина, пейлоади, активація тестів);

– Requests_module - здійснює HTTP-запити та обходить вебсайт (алгоритм обходу в ширину);

– Parser_module - аналізує HTML для пошуку потенційних точок ін'єкції (URL-параметри, поля форм);

– Vulnerability_tester - генерує та впроваджує XSS/CSRF пейлоади у виявлені точки;

– Response_analyzer - оцінює відповіді сервера для визначення спрацювання вразливостей.

- Reporter - формує структурований звіт про виявлені вразливості.
- Main - координує роботу всіх модулів.

Сканер зосереджений на виявленні Reflected XSS (через URL-параметри та форми) та CSRF-вразливостей, перевіряючи, чи форми обробляють запити без належних токенів або з підробленими токенами. Хоча поточна реалізація DOM-based XSS є спрощеною, архітектура системи дозволяє подальше розширення для повного аналізу.

Модульний дизайн забезпечує гнучкість та масштабованість для майбутнього розвитку та вдосконалення функціоналу сканера.

Також описано практичні аспекти розробки та налаштування базового сканера веб-вразливостей. Застосовано модульну структуру для зручності розробки та підтримки.

Проєкт розроблявся в PyCharm. Була створена локальна папка web_vulnerability_scanner з файлами .py та папкою test_site з HTML-файлами. Детально описано кроки налаштування робочого середовища:

– створення віртуального середовища Python для ізоляції залежностей;

– встановлення необхідних бібліотек (requests, beautifulsoup4, lxml, Flask) через термінал з активованим віртуальним середовищем;

– налаштування PyCharm для зручного запуску головного скрипта main.py через Run/Debug Configurations;

– підготовка тестового середовища: для повноцінного тестування, що включає обробку POST-запитів та імітацію вразливих ендпоінтів, було рекомендовано використовувати мікрофреймворк Flask для запуску складнішого вебсервера замість простого вбудованого HTTP-сервера Python.

Загалом, система продемонструвала ефективність у виявленні базових веб-вразливостей, підтверджуючи необхідність посилення безпеки на тестованому веб-додатку.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було розроблено автоматизовану система — це не просто інструмент, а стратегічний крок до посилення кібербезпеки вебдодатків. Вона дозволяє нам перейти від реактивного реагування на інциденти до проактивного виявлення та запобігання потенційним загрозам.

Без автоматизованих систем ми просто не встигаємо за темпами розвитку загроз. Розроблений сканер вирішує цю проблему, забезпечуючи:

– швидкість та ефективність - він може просканувати тисячі рядків коду та десятки сторінок за лічені хвилини, тоді як ручний аналіз тривав би дні або тижні. Це означає, що вразливості виявляються набагато раніше, що суттєво зменшує вартість їх виправлення;

– точність та послідовність - людський фактор завжди несе ризик помилок. Автоматизовані алгоритми виконують перевірки з високою точністю, дотримуючись визначених правил, що гарантує стабільні результати;

– масштабованість - незалежно від розміру вебдодатку чи кількості проєктів, які потрібно перевірити, система легко масштабується, дозволяючи охопити весь інфраструктурний ландшафт;

– інтеграція в DevOps - можливість інтегрувати сканер безпосередньо в процеси безперервної інтеграції та доставки (CI/CD) дозволяє автоматично перевіряти код щоразу, коли вносяться зміни. Це створює надійний "захисний екран" на кожному етапі розробки;

Хоча XSS та CSRF є одними з найстаріших загроз у світі вебу, вони залишаються надзвичайно актуальними. Наш сканер акцентує увагу на них, оскільки вони часто стають першою ланкою в ланцюгу складніших атак:

– XSS - це як підкинути шкідливу записку в публічний чат. Якщо вебсайт не перевіряє вміст, який ви публікуєте, хтось може додати JavaScript-код. Кожен, хто прочитає це повідомлення, несвідомо запустить шкідливий скрипт у своєму браузері. Це може призвести до крадіжки сесійних кукі, перенаправлення на

фішингові сайти, зміни вмісту сторінки або навіть виконання довільних дій від імені користувача. Наш сканер виявляє, де такий "шкідливий код" може бути впроваджений і виконаний;

– CSRF - сканер ідентифікує форми, які не перевіряють, чи запит справді надійшов від легітимного користувача, а не від підробленої сторінки.

Автоматизована система аналізу вразливостей є не просто інструментом, а важливим елементом культури безпечної розробки. Вона дозволяє розробникам та фахівцям з безпеки зосередитися на більш складних завданнях, довіряючи рутинні перевірки автоматизації, тим самим створюючи більш захищене цифрове майбутнє.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Інформаційна безпека: види загроз і методи усунення. Datami. URL:https://datami.ee/ua/blog/informatsijna-bezpeka-vidi-zagroz-i-metodi-yih-usunennya/ (дата звернення: 18.01.2025)

2. Основні загрози цілісності. StudFiles. URL: https://studfile.net/preview/6012701/page:10/ (дата звернення: 18.01.2025)

3. Загрози інформаційної безпеки, їх класифікація. Класифікація джерел загроз інформаційної безпеки. Всеосвіта. URL: https://vseosvita.ua/lesson/zahrozy-informatsiinoi-bezpeky-ikh-klasyfikatsiia-klasyfikatsiia-dzherel-zahroz-informatsiinoi-bezpeky-323171.html (дата звернення: 18.01.2025)

4. Достовірність інформації та надійність джерел. ITosvita. URL: https://it-osvita.diia.gov.ua/task/item/8edcb2bf-8fde-4159-a57c-d0509272ba93 (дата звернення: 18.01.2025)

5. Програма-вимагач: виклик сучасній інформаційній безпеці. Datami. URL: https://datami.ee/ua/blog/ransomware-5-strategies-to-protect-against-cyberattacks/ (дата звернення: 19.01.2025)

6. Безпека вебдодатків: найкращі практики та вразливості. ItProger. URL: https://itproger.com/ua/news/bezopasnost-veb-prilozheniy-luchshie-praktiki-i-uyazvimosti (дата звернення:19.01.2025)

7. Захист від XSS, CSRF та інших типових атак. IT Блог. URL: https://blog-it.com.ua/zahyst-vid-xss-csrf-ta-inshyh-typovyh-atak/ (дата звернення: 19.01.2025)

8. Що таке XSS вразливості. IT-notes. URL: https://www.it-notes.wiki/other/what-is-xss/ (дата звернення:19.01.2025)

9. Великий гайд по XSS. Міжсайтовий скриптинг: атаки і захист. UKR.Laboratories. URL: https://kr-labs.com.ua/blog/velykyj-gajd-po-xss-mizhsajtovyj-skryptyng-typy-atak-ta-zahyst/#elementor-toc__heading-anchor-2 (Дата звернення: 19.01.2025)

10. CSRF vs XSS: What is the difference? Escape. URL: https://escape.tech/blog/csrf-vs-xss/ (дата звернення: 19.01.2025)

11. Як працює CSRF-атака та які можливі наслідки для вебпрограми та її користувачів? UK.Eitca. URL: https://uk.eitca.org/cybersecurity/eitc-is-wapt-web-applications-penetration-testing/web-attacks-practice/csrf-cross-site-request-forgery/examination-review-csrf-cross-site-request-forgery/how-does-a-csrf-attack-work-and-what-are-the-potential-consequences-for-a-web-application-and-its-users/ (дата звернення: 20.01.2025)

12. Що таке API і навіщо потрібна ця технологія. ProSeo. URL: https://proseo.kiev.ua/seo/shcho-take-api-i-navishcho-potribna-tsia-tekhnolohiia/ (дата звернення: 21.01.2025)

13. Сканер вразливостей. VPN Unlimited. URL: https://www.vpnunlimited.com/ua/help/cybersecurity/vulnerability-scanner?srsltid=AfmBOoo4W0bYOsUubFV-gjhF38RFM7Fae1adQ6zZJReW1A2C9KiETgTm (дата звернення: 21.01.2025)

14. Топ-15 платних і безплатних інструментів для сканування вразливостей. Itest. ULRL: https://itest.com.ua/instrumenty/top-15-platnyh-i-bezplatnyh-instrumentiv-dlya-skanuvannya-vrazlyvostey/ (дата звернення:22.01.2025)

15. Інструменти з відкритим вихідним кодом для аналізу коду. HackYourMom. URL: https://hackyourmom.com/servisy/instrumenty-z-vidkrytym-vyhidnym-kodom-dlya-analizu-kodu/ (дата звернення: 22.01.2025)

16. Динамічний аналіз коду. Wikipedia. URL: https://uk.wikipedia.org/wiki/%D0%94%D0%B8%D0%BD%D0%B0%D0%BC%D1%96%D1%87%D0%BD%D0%B8%D0%B9_%D0%B0%D0%BD%D0%B0%D0%BB%D1%96%D0%B7_%D0%BA%D0%BE%D0%B4%D1%83 (дата звернення: 22.01.2025)

17. Тестування XSS та інших вразливостей за допомогою OWAST ZAP. Security QA. URL: https://svyat.tech/testing-for-xss-and-other-vulnerabilities-with-owasp-zap (дата звернення: 25.01.2025)

18. ZAP. ZAP by Checkmarkx. URL: https://www.zaproxy.org/getting-started/ (дата звернення: 27.01.2025)

19. Skipfish – інструмент тестування на проникнення. Medium. URL: https://medium.com/@kidnapshadow/skipfish-penetration-testing-tool-in-kali-linux-kidnapshadow-eceac38ba243 (дата звернення: 27.01.2025)

20. Сканер вразливостей веб-додатків Wapiti. Wapiti. URL: https://wapiti-scanner.github.io/ (дата звернення: 30.01.2025)

21. Як убезпечити свою компанію від компроментованих даних. Softlist. URL:https://softlist.ua/cases/data-compromise (дата звернення:01.02.2025)

22. Що таке IPS/IDS і де застосовується. Hostzealot. URL: https://www.hostzealot.com.ua/blog/about-solutions/shho-take-ipsids-i-de-zastosovujetsya (дата звернення:01.02.2025)

23. Міжсайтовий скриптинг. PortSwigger. URL: https://portswigger.net/web-security/cross-site-scripting (дата звернення: 01.02.2025)

24. Підробка міжсайтових запитів. PortSwigger. URL: https://portswigger.net/web-security/csrf (дата звернення: 02.02.2025)

25. Ручне тестування – що це таке, типи, процеси, підходи, інструменти та інше. Zaptest. URL: https://surl.li/iuipcb (дата звернення: 02.02.2025)

26. Парсинг сайтів. Webpromo. URL: https://web-promo.ua/ua/blog/parsing-sajtov-chto-eto-i-zachem-nuzhen/#sho-oznachaye-parsiti-sajt (дата звернення: 10.02.2025)

27. Beautiful Soup. Pypi. URL: https://pypi.org/project/beautifulsoup4/ (дата звернення: 20.04.2025)

28. Lxml – XML та HTML з Python. Lxml. URL: https://lxml.de/ (дата звернення: 23.04.2025)

29. Корисне навантаження. Vpnunlimited. URL: https://www.vpnunlimited.com/ua/help/cybersecurity/payload?srsltid=AfmBOoqvL_6Bek1Ij4y7svUFCFQMHDrvtaoM6kFZwE6FZC_8Ei0WcbWX (дата звернення: 24.04.2025)

30. Документація Beautiful Soup Crummy. URL: https://www.crummy.com/software/BeautifulSoup/bs4/doc/ (дата звернення: 25.04.2025)

31. Як запобігти атакам XSS. Cloudflare. URL: https://www.cloudflare.com/learning/security/how-to-prevent-xss-attacks/?_gl=1*1kozr1o*_gcl_au*OTM2ODIzMjY5LjE3NDU0MTA5NjE.*_ga*OTcyOTEwNjA1LjE3NDU0MTA5NjU.*_ga_SQCRB0TXZW*MTc0NTQxMDk2NS4xLjEuMTc0NTQxMDk3NS41MC4wLjA. (дата звернення: 25.04.2025)

32. Logging configuration. DocsPython. URL: https://docs.python.org/uk/3.15/library/logging.config.html (дата звернення: 26.04.2025)

33. Request Python – приклади використання. IT-notes. URL: https://www.it-notes.wiki/python/requests-python-usage-examples/ (дата звернення: 27.04.2025)

34. PHP парсинг за допомогою Simple HTML DOM. Kovelpost. URL: https://kovelpost.com/blogs/9280 (дата звернення: 27.04.2025)

35. Що таке тестування на проникнення, або як не потрапити в пастку хакерів? Datami. URL: https://datami.ee/ua/blog/what-is-penetration-testing/ (дата звернення: 28.04.2025)

36. Python приклад основної функції та методу: def Main(). Guru99. URL: https://www.guru99.com/uk/learn-python-main-function-with-examples-understand-main.html (дата звернення: 28.04.2025)

ДОДАТОК А

Код програми

```
# main.py
from config import ScannerConfig
from parser_module import HTMLParser
from reporter import Reporter
from requests_module import HTTPModule
from response_analyzer import ResponseAnalyzer
from vulnerability_tester import VulnerabilityTester


def main():
    target_url = "http://localhost:8000/"


    config = ScannerConfig(
        target_url=target_url,
        max_depth=1,
        xss_payloads=["<script>alert('XSS_TEST');</script>", "<img src=x
onerror=alert('XSS_IMG')>"],
        enable_csrf_test=True
    )
    http_module = HTTPModule(config.target_url)
    html_parser = HTMLParser()
    vulnerability_tester = VulnerabilityTester(http_module)
    response_analyzer = ResponseAnalyzer()
    reporter = Reporter()
    all_entry_points = {
        'xss_params': [],
        'xss_forms': [],
        'csrf_forms': []
    }
```

```
print("\n[Етап 1: Обхід та Парсинг Вебдодатку]")
for url, html_content in http_module.crawl_site(config.max_depth):
    if html_content:
        page_entry_points = html_parser.parse_page(url, html_content)
        all_entry_points['xss_params'].extend(page_entry_points['xss_params'])
        all_entry_points['xss_forms'].extend(page_entry_points['xss_forms'])
        all_entry_points['csrf_forms'].extend(page_entry_points['csrf_forms'])
print(f"\nЗагалом знайдено точок входу:")
print(f"  XSS URL-параметри: {len(all_entry_points['xss_params'])}")
print(f"  XSS форми: {len(all_entry_points['xss_forms'])}")
print(f"  CSRF форми: {len(all_entry_points['csrf_forms'])}")
print("\n[Етап 2: Тестування на Вразливості]")
xss_test_results = vulnerability_tester.test_xss(all_entry_points, config.xss_payloads)
print("\n[Аналіз результатів XSS-тестів]")
for result in xss_test_results:
    if response_analyzer.analyze_xss_response(result):
        reporter.add_vulnerability('xss', {
            'original_url': result['original_url'],
            'test_url': result['test_url'],
            'payload': result['payload'],
            'form_field': result.get('form_field')
        })
if config.enable_csrf_test:
    csrf_test_results = vulnerability_tester.test_csrf(all_entry_points)
    print("\n[Аналіз результатів CSRF-тестів]")
    for result in csrf_test_results:
        if response_analyzer.analyze_csrf_response(result):
            reporter.add_vulnerability('csrf', {
                'original_url': result['original_url'],
                'test_url': result['test_url'],
                'method': result['method'],
```

```
                'csrf_token_removed': result.get('csrf_token_removed'),
                'csrf_token_modified': result.get('csrf_token_modified'),
                'no_token_found': result.get('no_token_found')
            })
    print("\n[Етап 3: Генерація Звіту]")
    reporter.generate_report()
if __name__ == "__main__":
    main()

# config.py
class ScannerConfig:
    def __init__(self, target_url: str,
                 max_depth: int = 1,
                 xss_payloads: list = None,
                 enable_csrf_test: bool = True):
        self.target_url = target_url
        self.max_depth = max_depth
        self.xss_payloads = xss_payloads if xss_payloads else self._default_xss_payloads()
        self.enable_csrf_test = enable_csrf_test
        print(f"Конфігурація сканування для {self.target_url}:")
        print(f"  Глибина обходу: {self.max_depth}")
        print(f"  Кількість XSS-payloads: {len(self.xss_payloads)}")
        print(f"  CSRF-тест: {'увімкнено' if self.enable_csrf_test else 'вимкнено'}\n")
    def _default_xss_payloads(self):
        return [
            "<script>alert('XSS');</script>",
            "<img src=x onerror=alert('XSS')>",
            "<body onload=alert('XSS')>",
            "\"><script>alert(document.domain)</script>",
            "';alert(String.fromCharCode(88,83,83))//",
            "<svg onload=alert('XSS')>",
```

```
        "<details open ontoggle=alert('XSS')>" # HTML5 XSS
    ]


# parser_module.py
from bs4 import BeautifulSoup
from urllib.parse import urlparse, parse_qs, urlencode, urlunparse, urljoin
class HTMLParser:
    def __init__(self):
        pass
    def parse_page(self, url: str, html_content: str) -> dict:
        soup = BeautifulSoup(html_content, 'lxml')
        entry_points = {
            'xss_params': [],  # Список словників: {'url': ..., 'param': ..., 'value': ...}
            'xss_forms': [],   # Список словників: {'url': ..., 'form_id': ..., 'method': ..., 'action': ..., 'inputs': {'name': 'value', ...}}
            'csrf_forms': []   # Список словників: {'url': ..., 'form_id': ..., 'method': ..., 'action': ..., 'inputs': {'name': 'value', ...}}
        }
        parsed_url = urlparse(url)
        query_params = parse_qs(parsed_url.query)
        for param, values in query_params.items():
            for value in values:
                entry_points['xss_params'].append({
                    'url': url,
                    'param': param,
                    'value': value,
                    'type': 'url_param'
                })
        forms = soup.find_all('form')
        for form in forms:
            form_data = self._extract_form_data(form, url)
            if form_data:
```

```
            entry_points['xss_forms'].append({
                'url': url,
                'form_id': form.get('id', 'no_id'),
                'method': form_data['method'],
                'action': form_data['action'],
                'inputs': form_data['inputs'],
                'type': 'form'
            })
            if form_data['method'] in ['POST', 'PUT', 'DELETE']: # Можуть бути і інші, якщо вебсервер їх обробляє
                entry_points['csrf_forms'].append({
                    'url': url,
                    'form_id': form.get('id', 'no_id'),
                    'method': form_data['method'],
                    'action': form_data['action'],
                    'inputs': form_data['inputs'],
                    'type': 'form'
                })
        for param, values in query_params.items():
            for value in values:
                if html_content.find(value) != -1:
                    entry_points['xss_params'].append({
                        'url': url,
                        'param': param,
                        'value': value,
                        'type': 'reflected_in_body' # Можливе відбиття в тілі сторінки
                    })
        print(f"  Знайдено {len(entry_points['xss_params'])} URL-параметрів для XSS.")
        print(f"  Знайдено {len(entry_points['xss_forms'])} форм для XSS.")
        print(f"  Знайдено {len(entry_points['csrf_forms'])} форм для CSRF.")
        return entry_points
```

```
    def _extract_form_data(self, form_soup, base_url):
        method = form_soup.get('method', 'get').upper()
        action = form_soup.get('action', '')
        absolute_action = urljoin(base_url, action)
        inputs = {}
        for input_tag in form_soup.find_all(['input', 'textarea', 'select']):
            name = input_tag.get('name')
            value = input_tag.get('value', '')
            input_type = input_tag.get('type')
            if name:
                if input_type == 'checkbox':
                    if 'checked' in input_tag.attrs:
                        inputs[name] = value if value else 'on'
                    else:
                        inputs[name] = '' # або пропустити, якщо не встановлено
                elif input_type == 'radio':
                    if 'checked' in input_tag.attrs:
                        inputs[name] = value
                elif input_tag.name == 'select':
                    selected_option = input_tag.find('option', selected=True)
                    if selected_option:
                        inputs[name] = selected_option.get('value', '')
                    elif input_tag.find('option'):
                        inputs[name] = input_tag.find('option').get('value', '')
                else:
                    inputs[name] = value
        return {'method': method, 'action': absolute_action, 'inputs': inputs}



# reporter.py
```

```
class Reporter:
    def __init__(self):
        self.vulnerabilities = []
    def add_vulnerability(self, vul_type: str, details: dict):
        self.vulnerabilities.append({
            'type': vul_type,
            'details': details
        })
    def generate_report(self):
        print("\n" + "="*50)
        print("          ЗВІТ ПРО СКАСУВАННЯ ВРАЗЛИВОСТЕЙ")
        print("="*50)
        if not self.vulnerabilities:
            print("Вразливостей не виявлено. (Або не вдалося виявити за допомогою поточних тестів)")
            print("="*50)
            return
        for i, vul in enumerate(self.vulnerabilities):
            print(f"\n--- Вразливість #{i+1} ---")
            print(f"Тип: {vul['type'].upper()}")
            details = vul['details']
            if vul['type'] == 'xss':
                print(f"  URL: {details.get('original_url', 'N/A')}")
                print(f"  Тестовий URL/Дія: {details.get('test_url', 'N/A')}")
                if 'form_field' in details:
                    print(f"  Поле форми: {details['form_field']}")
                print(f"  Використаний Payload: {details.get('payload', 'N/A')}")
                print(f"  Опис: Виявлено відбиття XSS-payload у відповіді сервера. Це може дозволити зловмиснику виконати довільний JavaScript-код в браузері користувача.")
            elif vul['type'] == 'csrf':
                print(f"  URL форми: {details.get('original_url', 'N/A')}")
                print(f"  URL дії: {details.get('test_url', 'N/A')}")
```

```
            print(f"  Метод: {details.get('method', 'N/A')}")
            if 'csrf_token_removed' in details:
                print(f"  Спроба: Видалення токена '{details['csrf_token_removed']}'")
            elif 'csrf_token_modified' in details:
                print(f"  Спроба: Модифікація токена '{details['csrf_token_modified']}'")
            elif 'no_token_found' in details:
                print("  Спроба: Форма не має видимих CSRF-токенів.")
            print(f"  Опис: Вебсервер прийняв запит без або з недійсним CSRF-токеном, що може свідчити про вразливість до міжсайтової підробки запитів. Це дозволяє зловмиснику виконати дії від імені авторизованого користувача.")
            print("-" * 30)
        print("="*50)



# requests_module.py
import requests
from urllib.parse import urljoin, urlparse
from collections import deque
class HTTPModule:
    def __init__(self, base_url: str):
        self.session = requests.Session()
        self.base_url = base_url
        self.visited_urls = set()
        self.urls_to_visit = deque([(base_url, 0)]) # (url, depth)

    def fetch_page(self, url: str, method='GET', data=None, headers=None, allow_redirects=True):
        try:
            print(f"  Запит: {method} {url}")
            if method.upper() == 'GET':
                response = self.session.get(url, headers=headers, allow_redirects=allow_redirects)
            elif method.upper() == 'POST':
```

```
            response = self.session.post(url, data=data, headers=headers,
allow_redirects=allow_redirects)
            response.raise_for_status()
            return response
        except requests.exceptions.RequestException as e:
            print(f"Помилка HTTP запиту до {url}: {e}")
            return None
    def _get_absolute_url(self, base_url, relative_url):
        return urljoin(base_url, relative_url)
    def discover_links(self, html_content: str, current_url: str, max_depth: int, current_depth: int):
        if current_depth >= max_depth:
            return []
        from bs4 import BeautifulSoup
        soup = BeautifulSoup(html_content, 'lxml')
        found_links = []
        for link in soup.find_all('a', href=True):
            href = link.get('href')
            absolute_url = self._get_absolute_url(current_url, href)
            if urlparse(absolute_url).netloc == urlparse(self.base_url).netloc:
                if absolute_url not in self.visited_urls and (absolute_url, current_depth + 1) not in
self.urls_to_visit:
                    self.urls_to_visit.append((absolute_url, current_depth + 1))
                    found_links.append(absolute_url)
        return found_links
    def crawl_site(self, max_depth: int):
        print(f"Початок обходу сайту {self.base_url} до глибини {max_depth}...")
        while self.urls_to_visit:
            current_url, current_depth = self.urls_to_visit.popleft()
            if current_url in self.visited_urls:
                continue
            print(f"Обхід: {current_url} (Глибина: {current_depth})")
            self.visited_urls.add(current_url)
```

```
            response = self.fetch_page(current_url)
            if response and response.status_code == 200:
                html_content = response.text
                yield current_url, html_content
                if current_depth < max_depth:
                    self.discover_links(html_content, current_url, max_depth, current_depth)
            else:
                print(f"  Не вдалося отримати контент з {current_url}")



# response_analyzer.py
class ResponseAnalyzer:
    def __init__(self):
        pass
    def analyze_xss_response(self, test_result: dict) -> bool:
        response = test_result['response']
        payload = test_result['payload']
        if response and response.text and payload in response.text:
            print(f"  [XSS ВРАЗЛИВІСТЬ ВИЯВЛЕНО] Payload '{payload[:30]}...' відбито в {test_result['test_url']}")
            return True
        return False
    def analyze_csrf_response(self, test_result: dict) -> bool:
        response = test_result['response']
        if response:
            if 200 <= response.status_code < 300 or response.status_code in [302, 303]:
                if "csrf token mismatch" not in response.text.lower() and \
                   "invalid token" not in response.text.lower() and \
                   "403 forbidden" not in response.text.lower(): # 403 часто використовується для CSRF захисту
                    print(f"  [CSRF ВРАЗЛИВІСТЬ (?) ВИЯВЛЕНО] Запит без/зміненого токена прийнято: {test_result['test_url']} (Статус: {response.status_code})")
```

```
                return True
        return False


#test_server.py
from flask import Flask, request, render_template_string, redirect, url_for, session
import secrets
app = Flask(__name__)
app.secret_key = 'a_very_secret_key_for_testing'
def generate_csrf_token():
    if 'csrf_token' not in session:
        session['csrf_token'] = secrets.token_hex(16)
    return session['csrf_token']
@app.route('/')
def index():
    return render_template_string(
  """
        <!DOCTYPE html>
        <html lang="en">
        <head>
            <meta charset="UTF-8">
            <meta name="viewport" content="width=device-width, initial-scale=1.0">
            <title>Головна сторінка Flask</title>
        </head>
        <body>
            <h1>Вітаємо на тестовому сайті Flask!</h1>
            <p>Це головна сторінка. Перейдіть за посиланнями, щоб протестувати сканер.</p>
            <ul>
                <li><a href="/xss_vulnerable?name=John">Сторінка для тестування XSS (вразлива)</a></li>
                <li><a href="/csrf_test">Сторінка для тестування CSRF</a></li>
                <li><a href="/nested_page">Вкладена сторінка</a></li>
```

```
        </ul>

        <h2>Проста POST форма (для обходу)</h2>
        <form action="/process_data" method="POST">
            <label for="data">Дані:</label><br>
            <input type="text" id="data" name="data" value="some_data"><br><br>
            <input type="submit" value="Відправити">
        </form>
    </body>
    </html>
    """
  )
@app.route('/xss_vulnerable')
def xss_vulnerable():
  name = request.args.get('name', 'Гість')
  return render_template_string(
    """
    <!DOCTYPE html>
    <html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>XSS Тест</title>
    </head>
    <body>
        <h1>Тест XSS (Вразливий)</h1>
        <p>Ваші дані: <span id="output">{{ name | safe }}</span></p> {# '| safe' тут імітує відсутність санітизації #}

        <h2>GET форма (XSS вразлива)</h2>
        <form action="/xss_vulnerable" method="GET">
```

```
            <label for="param_get">Введіть щось (GET):</label><br>
            <input type="text" id="param_get" name="name" value=""><br><br> {# 'name' - той самий параметр #}
            <input type="submit" value="Відправити GET">
        </form>

        <h2>POST форма (XSS вразлива)</h2>
        <form action="/xss_post_vulnerable" method="POST">
            <label for="param_post">Введіть щось (POST):</label><br>
            <input type="text" id="param_post" name="data" value=""><br><br>
            <input type="submit" value="Відправити POST">
        </form>
    </body>
    </html>
    """, name=name
  )
@app.route('/xss_post_vulnerable', methods=['POST'])
def xss_post_vulnerable():
  # Вразливе місце: дані з POST без санітизації відображаються
  post_data = request.form.get('data', 'Немає даних')
  return render_template_string(
    """
    <!DOCTYPE html>
    <html lang="en">
    <head>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
      <title>XSS POST Результат</title>
    </head>
    <body>
      <h1>Результат POST-тесту</h1>
```

```
        <p>Ви відправили: <span id="output">{{ post_data | safe }}</span></p> {# '| safe' імітує відсутність санітизації #}
        <p><a href="/xss_vulnerable">Назад</a></p>
    </body>
    </html>
    """, post_data=post_data
  )
@app.route('/csrf_test')
def csrf_test():
  token = generate_csrf_token()
  return render_template_string(
    """
    <!DOCTYPE html>
    <html lang="en">
    <head>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
      <title>CSRF Тест</title>
    </head>
<body>
      <h1>Тест CSRF</h1>
      <p>Ця форма імітує зміну пароля або інші критичні дії.</p>

      <h2>Форма без CSRF-токена (Вразлива)</h2>
      <form id="csrf_form_vulnerable" action="/change_password_no_token" method="POST">
        <label for="new_password_no_token">Новий пароль:</label><br>
        <input type="password" id="new_password_no_token" name="new_password" value="test1234"><br><br>
        <input type="submit" value="Змінити пароль (без токена)">
      </form>

      <h2>Форма з CSRF-токеном (Захищена, якщо правильно обробляється)</h2>
```

```
        <form id="csrf_form_protected" action="/change_password_with_token" method="POST">
            <label for="new_password_with_token">Новий пароль:</label><br>
            <input type="password" id="new_password_with_token" name="new_password" value="test1234"><br>
            {# CSRF-токен - його значення буде генеруватися сервером #}
            <input type="hidden" name="csrf_token" value="{{ csrf_token() }}">
            <input type="submit" value="Змінити пароль (з токеном)">
        </form>
    </body>
    </html>
    """, csrf_token=generate_csrf_token
  )
@app.route('/change_password_no_token', methods=['POST'])
def change_password_no_token():
  # Цей ендпоінт ВРАЗЛИВИЙ до CSRF, бо не перевіряє токен
  new_password = request.form.get('new_password')
  print(f"!!! CSRF-ВРАЗЛИВІСТЬ: Пароль змінено без токена на: {new_password} !!!")
  return "Пароль змінено (без перевірки токена). Це CSRF-вразливість!", 200
@app.route('/change_password_with_token', methods=['POST'])
def change_password_with_token():
  # Цей ендпоінт захищений (імітація перевірки токена)
  received_token = request.form.get('csrf_token')
  if received_token == session.get('csrf_token'):
    new_password = request.form.get('new_password')
    print(f"Пароль змінено (з перевіркою токена): {new_password}")
    return "Пароль змінено (токен перевірено).", 200
  else:
    print("!!! CSRF-ЗАХИСТ СПРАЦЮВАВ: Недійсний токен або відсутній !!!")
    return "Помилка: Недійсний CSRF-токен.", 403
@app.route('/process_data', methods=['GET', 'POST'])
def process_data():
```

```
    # Заглушка для POST-форми на головній
    if request.method == 'POST':
        data = request.form.get('data')
        print(f"Отримано POST-дані: {data}")
        return f"Дані отримано: {data}", 200
    return "Очікується POST-запит."
@app.route('/nested_page')
def nested_page():
    return render_template_string(
        """
        <!DOCTYPE html>
        <html lang="en">
        <head>
            <meta charset="UTF-8">
            <meta name="viewport" content="width=device-width, initial-scale=1.0">
            <title>Вкладена Сторінка Flask</title>
        </head>
        <body>
            <h1>Це вкладена сторінка Flask</h1>
            <p>Повернутися на <a href="/">головну</a>.</p>
        </body>
        </html>
        """
    )
if __name__ == '__main__':
    app.run(debug=True, port=8000) # 'debug=True' дозволяє перезавантаження при змінах


# vulnerability_tester.py
from urllib.parse import urlparse, parse_qs, urlencode, urlunparse
import copy
```

```
class VulnerabilityTester:
    def __init__(self, http_module):
        self.http_module = http_module
    def test_xss(self, entry_points: dict, xss_payloads: list):
        print("\nПочаток тестування на XSS...")
        tested_requests = []
        for ep in entry_points['xss_params']:
            original_url = ep['url']
            param_name = ep['param']
            original_value = ep['value']
            parsed_url = urlparse(original_url)
            query_params = parse_qs(parsed_url.query, keep_blank_values=True) # Важливо: keep_blank_values для коректної обробки
            for payload in xss_payloads:
                test_query_params = copy.deepcopy(query_params)
                test_query_params[param_name] = [payload] # Замінюємо значення параметра на payload
                new_query_string = urlencode(test_query_params, doseq=True)
                test_url = urlunparse(parsed_url._replace(query=new_query_string))
                print(f"  XSS-тест (URL-параметр): {test_url}")
                response = self.http_module.fetch_page(test_url, method='GET')
                if response:
                    tested_requests.append({
                        'type': 'xss',
                        'subtype': 'url_param',
                        'original_url': original_url,
                        'test_url': test_url,
                        'payload': payload,
                        'response': response
                    })
        for form_ep in entry_points['xss_forms']:
            original_url = form_ep['url']
```

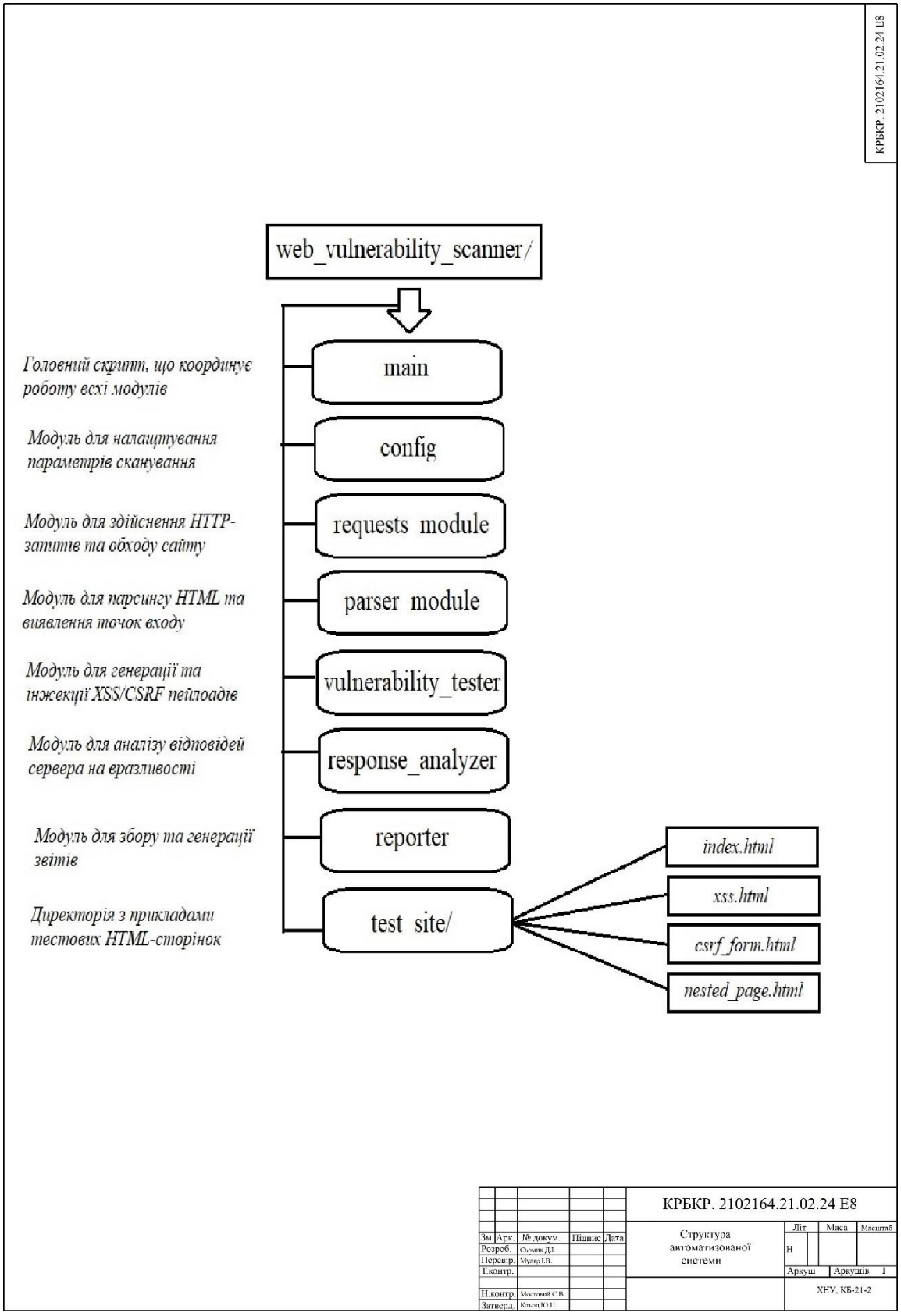
```
            action_url = form_ep['action']
            method = form_ep['method']
            original_inputs = form_ep['inputs']
            for payload in xss_payloads:
                for input_name, input_value in original_inputs.items():
                    test_data = copy.deepcopy(original_inputs)
                    test_data[input_name] = payload
                    print(f"  XSS-тест (Форма, поле '{input_name}'): {action_url} (Метод: {method})")
                    if method == 'GET':
                        parsed_action = urlparse(action_url)
                        query_params = parse_qs(parsed_action.query, keep_blank_values=True)
                        query_params.update(test_data)
                        test_url = urlunparse(parsed_action._replace(query=urlencode(query_params,
doseq=True)))
                        response = self.http_module.fetch_page(test_url, method='GET')
                    else: # POST
                        response = self.http_module.fetch_page(action_url, method='POST', data=test_data)
                    if response:
                        tested_requests.append({
                            'type': 'xss',
                            'subtype': 'form',
                            'original_url': original_url,
                            'test_url': action_url,
                            'payload': payload,
                            'form_field': input_name,
                            'method': method,
                            'data': test_data, # Дані, які були відправлені
                            'response': response
                        })
        return tested_requests
    def test_csrf(self, entry_points: dict):
```

```
        print("\nПочаток тестування на CSRF...")
        tested_requests = []
        for form_ep in entry_points['csrf_forms']:
            original_url = form_ep['url']
            action_url = form_ep['action']
            method = form_ep['method']
            original_inputs = form_ep['inputs']
            csrf_token_candidates = [
                name for name in original_inputs.keys()
                if 'token' in name.lower() or 'csrf' in name.lower() or 'nonce' in name.lower()
            ]
            test_data = copy.deepcopy(original_inputs)
            if csrf_token_candidates:
                for token_name in csrf_token_candidates:
                    print(f"  CSRF-тест (Форма '{form_ep['form_id']}', видалення токена '{token_name}'): {action_url}")
                    modified_data_no_token = copy.deepcopy(test_data)
                    modified_data_no_token.pop(token_name, None) # Видалити токен
                    response = self.http_module.fetch_page(action_url, method='POST', data=modified_data_no_token, allow_redirects=False)
                    if response:
                        tested_requests.append({
                            'type': 'csrf',
                            'subtype': 'token_removed',
                            'original_url': original_url,
                            'test_url': action_url,
                            'method': method,
                            'data': modified_data_no_token,
                            'csrf_token_removed': token_name,
                            'response': response
                        })
                for token_name in csrf_token_candidates:
```

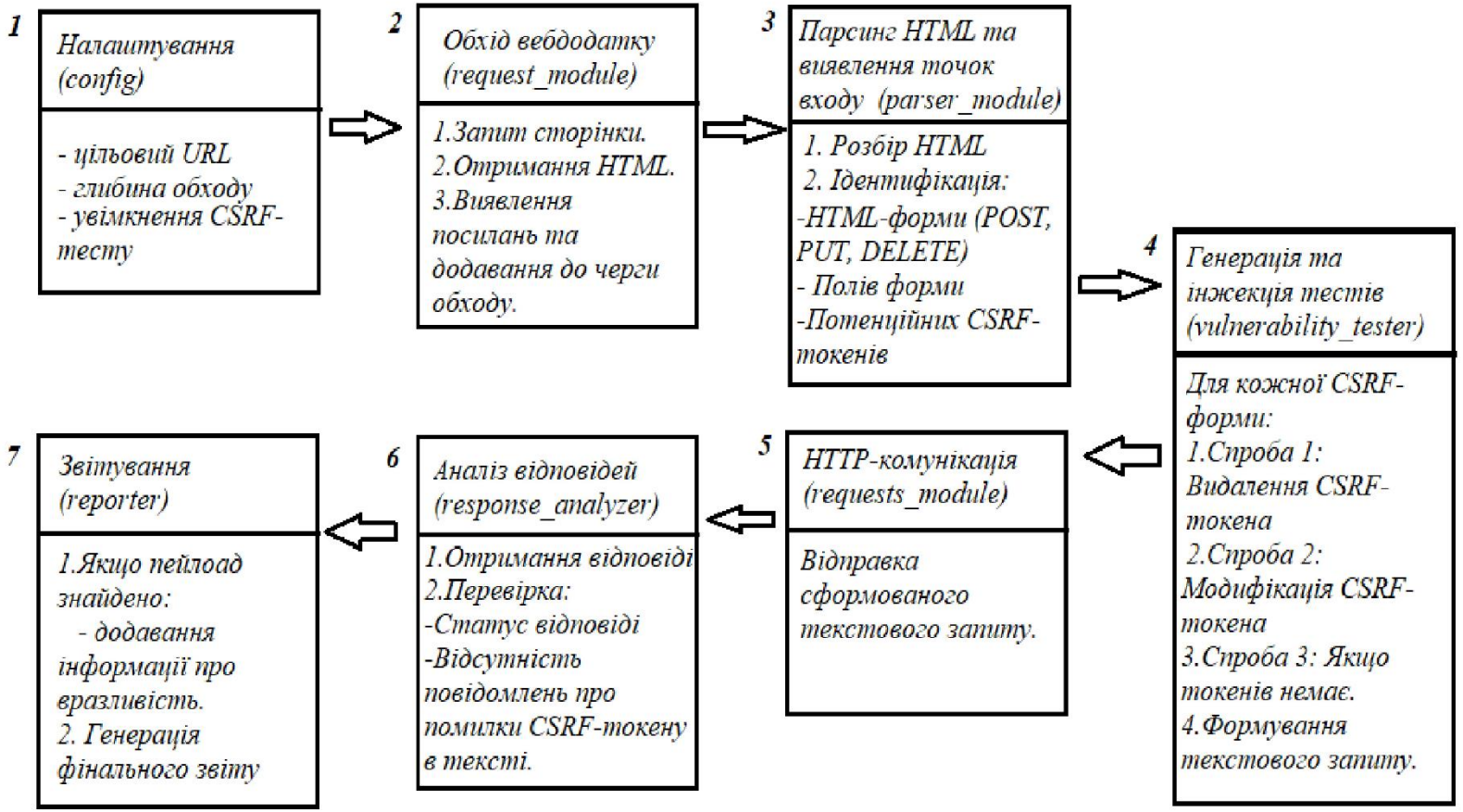
```
                print(f"  CSRF-тест (Форма '{form_ep['form_id']}', модифікація токена '{token_name}'): {action_url}")
                modified_data_invalid_token = copy.deepcopy(test_data)
                modified_data_invalid_token[token_name] = 'INVALID_CSRF_TOKEN_TEST' # Змінити токен
                response = self.http_module.fetch_page(action_url, method='POST', data=modified_data_invalid_token, allow_redirects=False)
                if response:
                    tested_requests.append({
                        'type': 'csrf',
                        'subtype': 'token_modified',
                        'original_url': original_url,
                        'test_url': action_url,
                        'method': method,
                        'data': modified_data_invalid_token,
                        'csrf_token_modified': token_name,
                        'response': response
                    })
            else:
                print(f"  CSRF-тест (Форма '{form_ep['form_id']}', без токена): {action_url}")
                response = self.http_module.fetch_page(action_url, method='POST', data=test_data, allow_redirects=False)
                if response:
                    tested_requests.append({
                        'type': 'csrf',
                        'subtype': 'no_token_found',
                        'original_url': original_url,
                        'test_url': action_url,
                        'method': method,
                        'data': test_data,
                        'response': response
                    })
        return tested_requests
```

ДОДАТОК Б

Копія графічної частини



					КРБКР. 2102164.21.02.24 Е8		
Зм.	Арк.	№ докум.	Підпис	Дата	Етапи виконання пошуку XSS вразливостей	Літ	Маса / Масштаб
Розроб.		Сьомик Д.І.				Н	
Перевір.		Музюр І.В.				Аркуш	Аркушів 1
Т.контр.							
Н.контр.		Мостовий С.В.				ХНУ, КБ-21-2	
Затверд.		Ковиш Ю.П.					



Зм	Арк.	№ докум.	Підпис	Дата	КРБКР. 2102164.21.02.24 Е8	
Розроб.	Сеник Д.І.				Етапи виконання пошуку CSRF вразливостей	Літ. Маса Масштаб
Перевір.	Мустер І.В.					Н
Т. контр.						
Н. контр.	Мостовий С.В.					Аркуш 1
Затверд.	Ковін Ю.П.					ХНУ, КБ-21-2

Завідувачу кафедри кібербезпеки
к.т.н., доц. Кльоцу Ю.П.
Сьомик Діани Ігорівни
ПІБ здобувача вищої освіти

Студентки ФІТ, 4 курсу, групи КБ-21-2

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 31.08.2023, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлена. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщена та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення 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 плагіату в текстах робіт.

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

09.06.2025
дата

підпис

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: Сьомик Діана Ігорівна

Співавтор:

Назва: Автоматизована система аналізу XXS та CSRF вразливостей вебдодатків на Python

Науковий керівник:

Підрозділ: Кафедра кібербезпеки

Коефіцієнт подібності 1:3.6%

Коефіцієнт подібності 2:0.3%

Мікропробіли: 0

Заміна букв: 0

Інтервали: 0

Білі знаки: 0

Дата створення звіту: 2025-06-09 19:18:24.0

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:

10.06.2025р.

Anti-Plagiarism (UA) v-15.281 Educational

The maximum coincidence with one document 1.0%

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

ID: 244284 Title: Автоматизована система аналізу XXS та CSRF вразливостей вебдодатків на Python Added in a DB: 2025-06-09 Authors: Сьомик Діана Ігорівна Heads: Муляр І.В. Consultants: Opponents:	Document		Sum coincidence on the DB	
	Symbols	Lexemes	Symbols	Lexemes
	71335	600	864 (1%)	9 (2%)

Plagiarism sources

ID	Description	Plagiarism presence in the document	
		Symbols	Lexemes

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: Автоматизована система аналізу XSS та CSRF вразливостей вебдодатків на Python

Автор: Сьомик Діана Ігорівна

Спеціальність: 125 – Кібербезпека

Освітня програма: Кібербезпека

Науковий керівник: Ігор МУЛЯР, канд. техн. наук, доцент

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 96,7%, оригінальність тексту роботи за результатами перевірки системою Anti-Plagiarism складає 99%.

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.

Керівник роботи _____ Ігор МУЛЯР

Гарант ОП _____ Віктор ЧЕШУН

Завідувач кафедри кібербезпеки _____ Юрій КЛЬОЦ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

освітнього ступеня «бакалавр»

Студентка Сьомик Діана Ігорівна

Тема Автоматизована система аналізу XSS та CSRF вразливостей вебдодатків на Python

Спеціальність 125 – Кібербезпека

Обсяг кваліфікаційної роботи освітньо-кваліфікаційного рівня «бакалавр»:

кількість листів креслень 3; кількість сторінок записки 60.

1. Короткий зміст роботи та прийнятих рішень У кваліфікаційній роботі, відповідно до поставленого завдання, проведене дослідження предметної області, та існуючі рішення в галузі пошуку враливостей. Також розроблено архітектуру автоматизованої системи аналізу безпеки, та реалізовано модулі для сканування XSS та CSRF вразливостей. Реалізовано та налаштовано систему, та проведено тестування автоматизованої системи.

2. Висновок про відповідність кваліфікаційної роботи завданню У кваліфікаційній роботі повністю виконано поставлене завдання як у теоретичній, так і в практичній частині

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У розділі 1 розглянуто загрози інформаційної безпеки, типи атак на вебдодатки, також розглянуто існуючі сканери вразливостей та потенційні наслідки атак на вебсервер. У розділі 2 обгрунтовано обраний підхід для реалізації системи, та описано архітектуру автоматизованої системи аналізу безпеки. У розділі 3 розроблено автоматизовану систему аналізу вразливостей, описано інструментарій розробки та технологію реалізації системи. Також, проведено налаштування та тестування системи

4. Позитивні сторони роботи Робота базується на детальному аналізі загроз інформаційної безпеки, розглядаються також можливі типи атак на вебдодатки. Робота над такою системою є надзвичайно актуальною та практично значущою. Можна отримати глибокі знання у сфері веббезпеки та вдосконалите навички програмування на Python. Це дозволить зробити внесок у спільноту кібербезпеки, підвищити свою цінність на ринку праці та зіткнутися з цікавими технічними викликами.

5. Негативні сторони роботи Постійна еволюція загроз та необхідність оновлення і складність імітації реальної поведінки користувача та браузера.

6. Оцінка графічного оформлення та пояснювальної записки роботи Графічне оформлення кваліфікаційної роботи відповідає темі роботи та виконане з дотриманням стандартів. В цілому, графічне оформлення є якісним, а пояснювальна записка відповідає нормам оформлення.

7. Відгук про роботу в цілому Кваліфікаційна робота заслуговує позитивної оцінки, оскільки весь матеріал роботи є структурованим, чітким та послідовним. Усі розділи роботи мають логічну послідовність, що сприяє зрозумінню викладеного матеріалу в рамках теми роботи. Графічний матеріал допомагає наочно продемонструвати доцільність та ефективність прийнятих рішень у проєктуванні та супроводі розробленої автоматизованої системи аналізу XSS та CSRF вразливостей вебдодатків.

8. Інші зауваження

9. Оцінка кваліфікаційної роботи Враховуючи всі позитивні сторони кваліфікаційної роботи, а також негативні сторони, які не зменшують практичну цінність отриманих результатів і загальну якість роботи, рекомендованою оцінкою є «відмінно»

РЕЦЕНЗЕНТ (прізвище, ім'я, по батькові, посада, місце роботи)
Пивовар Олег Сергійович, к.т.н., доцент кафедри ТМІТ

«9» січня 2025. Пивовар О.С. (підпис)