МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет
імені Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ
Завідувач кафедри інтелектуальних
інформаційних систем, д-р техн. наук, проф.
_____Ю. П. Кондратенко
«____»_____2024 р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

ІНФОРМАЦІЙНА СИСТЕМА АНАЛІЗУ БЕЗПЕКИ ВЕБСАЙТІВ

Спеціальність 122 «Комп'ютерні науки»

122 – КРБ – 401.2010301

Виконав студент 4-го курсу, групи 401
_____ ***В. В. Бардаков***
«18» червня 2024 р.

Керівник: д-р техн. наук, професор
_____***І. О. Калініна***
«18» червня 2024 р.

Миколаїв – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет ім. Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

Рівень вищої освіти **бакалавр**
Спеціальність **122 «Комп'ютерні науки»**
(шифр і назва)
Галузь знань **12 «Інформаційні технології»**
(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем, д-р техн. наук, проф.
_____ Ю. П. Кондратенко
«____» _____2024 р.

З А В Д А Н Н Я
на виконання кваліфікаційної роботи

Видано студенту групи 401 факультету комп'ютерних наук Бардакову Валерію Вікторовичу.

1. Тема кваліфікаційної роботи «Інформаційна система аналізу безпеки вебсайтів».

Керівник роботи Калініна Ірина Олександрівна, д-р техн. наук, професор.

Затв. наказом Ректора ЧНУ ім. Петра Могили від «28» грудня 2023 р. № 271

2. Строк представлення кваліфікаційної роботи студентом «18» червня 2024 р.

3. Вхідні (початкові) дані до роботи: експертні оцінки аналізу безпеки вебсайтів на основі OWASP, тестові випадки та сценарії атак, інструменти та звіти ручного сканування.

Очікуваний результат: інформаційна система аналізу безпеки вебсайтів.

4. Перелік питань, що підлягають розробці (зміст пояснювальної записки):

- аналіз літератури та існуючих рішень сучасного стану безпеки вебсайтів та можливих атак;
- огляд існуючих вразливостей для дослідження;
- експертне оцінювання методів виявлення вразливостей вебсайтів;

– реалізація інформаційної системи, тестування та верифікація.

5. Перелік графічного матеріалу: презентація.

6. Завдання до спеціальної частини: «Розробка заходів з поліпшення умов праці»

7. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис
Спеціальна частина з охорони праці	Алексєєва А. О., доцент кафедри екології	

Керівник роботи __д-р техн. наук, проф. Калініна І. О._____
(наук. ступінь, вчене звання, прізвище та ініціали)

_____
(підпис)

Завдання прийнято до виконання ___Бардаков В. В._____
(прізвище та ініціали)

_____
(підпис)

Дата видачі завдання «_14_» _____січня____ 2024 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: Інформаційна система аналізу безпеки вебсайтів _____

№	Найменування роботи	Початок	Закінчення	Примітки
1	Подання заяви на затвердження теми та керівників КРБ	10.11.2023	15.11.2023	Виконано
2	Отримання завдання на виконання КРБ	10.01.2024	15.01.2024	Виконано
3	Складання календарного плану роботи на весь період виконання КРБ	16.01.2024	30.01.2024	Виконано
4	Отримання завдання на переддипломну практику	15.04.2024	29.04.2024	Виконано
5	Проходження переддипломної практики, збір та аналіз матеріалів до КРБ	29.04.2024	11.05.2024	Виконано
6	Розробка звіту з переддипломної практики	12.05.2024	15.05.2024	Виконано
7	Виконання КРБ: аналіз сучасного стану задачі аналізу безпеки вебсайтів, огляд існуючих технологій, розробка ПЗ	13.05.2024	22.06.2024	Виконано
8	Перший попередній захист КРБ на засіданні комісії кафедри	27.05.2024	27.05.2024	Виконано
9	Доробка та остаточне оформлення КРБ	28.05.2024	09.06.2024	Виконано
10	Другий попередній захист КРБ на засіданні комісії кафедри	10.06.2024	10.06.2024	Виконано
11	Подання КРБ рецензенту	13.06.2024	13.06.2024	Виконано
11	Подання КРБ, її електронної копії та інших документів (відгуку, рецензії) до захисту	17.06.2024	21.06.2024	Виконано
12	Захист КРБ перед екзаменаційною комісією (ЕК)	24.06.2024	28.06.2024	Виконано

Розробив студент __Бардаков В. В._____ _____
(прізвище, ім'я, по батькові студента) (*підпис*)

Керівник роботи __ д-р техн. наук, проф. Калініна І. О. _____
(*посада, прізвище, ім'я, по батькові)* *(підпис)*

«_29_» _____01_____ 2024 р.

АНОТАЦІЯ

кваліфікаційної роботи студента групи 401 ЧНУ ім. Петра Могили
Бардакова Валерія Вікторовича
Тема: «Інформаційна система аналізу безпеки вебсайтів»

Зі зростанням кількості зловмисних атак та підвищенням складності методів злому, розробка ефективних систем для виявлення вразливостей стає все більш важливою.

Об'єкт роботи – процес аналізу безпеки вебсайтів.

Предмет роботи – інформаційні системи для виявлення вразливостей у вебсайтах.

Метою кваліфікаційної роботи є підвищення ефективності аналізу безпеки вебсайтів шляхом розробки інформаційної системи, яка використовує сучасні методи та технології. Одним із завдань було дослідження різних видів вразливостей вебсайтів та методів їхнього виявлення.

Пояснювальна записка складається зі вступу, трьох розділів, висновків та додатків.

У першому розділі розглядається актуальність та проблеми предметної області, а також основні поняття та визначення, пов'язані з безпекою вебсайтів.

У другому розділі досліджено основні вразливості вебсайтів, такі як SQL-ін'єкції, XSS, CSRF, XXE, Directory Traversal, LFI та Session Fixation, їх механізми дії та потенційні наслідки для безпеки вебдодатків.

У третьому розділі описано програмну реалізацію та тестування розробленої системи. Також розглянуто створення та налаштування конфігурацій проекту, програмну реалізацію пошуку вразливостей та тестування системи.

В результаті розроблено інформаційну систему аналізу безпеки вебсайтів, яка дозволяє виявляти основні вразливості з високою ефективністю.

Кваліфікаційна робота містить 81 сторінок, 12 рисунків, 1 таблицю, 25 використаних джерел та 2 додатка.

Ключові слова: безпека веб-ресурсів, аналіз вразливостей, інформаційна безпека, SQL-ін'єкції, XSS, CSRF.

ABSTRACT

Bachelor's qualification work of the student of 401 group of Petro Mohyla Black Sea National University

Bardakov Valerii Viktorovych

Title: “Web site security analysis information”

With the increase in the number of malicious attacks and the increasing sophistication of hacking methods, the development of effective systems to detect vulnerabilities is becoming increasingly important.

The object of the work is the process of website security analysis.

The subject of the work is information systems for detecting vulnerabilities in websites.

The purpose of the qualification work is to increase the effectiveness of website security analysis by developing an information system that uses modern methods and technologies.

The explanatory note consists of an introduction, three sections, conclusions and appendices.

The first chapter examines the relevance and problems of the subject area, as well as the main concepts and definitions related to website security.

The second chapter explores the main vulnerabilities in websites, such as SQL injection, XSS, CSRF, XXE, Directory Traversal, LFI and Session Fixation, their mechanisms of action and potential implications for web application security.

The third chapter describes the software implementation and testing of the developed system. Creation and configuration of project configurations, software implementation of vulnerability detection and system testing are also considered.

As a result, an information system for website security analysis has been developed, which allows detecting the main vulnerabilities with high efficiency.

The qualification paper contains 81 pages, 12 figures, 1 table, 25 used sources and 2 appendices.

Keywords: security of web resources, vulnerability analysis, information security, SQL injections, XSS, CSRF.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

БД – База даних

ІБ – Інформаційна безпека

ІС – Інформаційна система

CSRF – Cross-Site Request Forgery

DoS – Denial of Service

IDE – Integrated Development Environment

LFI – Local File Inclusion

ORM – Object-Relational Mapping

OWASP – Open Worldwide Application Security Project

XML – eXtensible Markup Language

XSS – Cross Site Scripting

XXE – XML External Entity

ВСТУП

У сучасному цифровому світі, де Інтернет відіграє ключову роль у функціонуванні бізнесів, урядових установ та особистого життя громадян, питання безпеки вебсайтів набуває особливої важливості. Кількість кібератак та різноманітних загроз постійно зростає, а зловмисники використовують все складніші методи для досягнення своїх цілей. Це робить захист ресурсів вебсайтів критично важливим завданням для розробників та адміністраторів.

Однією з найбільш ефективних стратегій у боротьбі з кібератаками є впровадження інформаційних систем, які здатні автоматично виявляти та аналізувати вразливості вебсайтів. Це дозволяє своєчасно реагувати на потенційні загрози і мінімізувати ризики компрометації даних.

Метою даної роботи є розробка інформаційної системи, яка забезпечує автоматичний аналіз безпеки вебсайтів, виявляючи вразливості, такі як Directory Traversal, XXE Test, Security Misconfiguration, Cross Site Scripting, Cross-Site Request Forgery, Session Fixation, та Local File Inclusion. Система спрямована на підвищення рівня захищеності вебресурсів шляхом надання інструментів для швидкого виявлення та усунення вразливостей.

Наукова новизна роботи полягає у поєднанні різних методів аналізу безпеки в рамках єдиної інформаційної системи, що дозволяє ефективно виявляти широкий спектр вразливостей. Це досягається завдяки використанню сучасних алгоритмів та технологій, які забезпечують високу точність та швидкість аналізу.

Практична значимість роботи полягає у створенні інструменту, який може бути використаний як розробниками, так і адміністраторами вебсайтів для забезпечення їх безпеки. Система надає можливість автоматичного виявлення вразливостей, що зменшує час на їх ідентифікацію та усунення, а також мінімізує ризики потенційних атак.

Об'єктом дослідження в даній курсовій роботі є вебсайти та їхні компоненти, що підлягають аналізу на предмет вразливостей безпеки. Це включає як загальні архітектурні елементи вебсайтів (серверна частина, клієнтська частина, бази

даних), так і конкретні програмні компоненти та налаштування, які можуть бути потенційними точками атаки. Вебсайти в даному контексті розглядаються як цілісні інформаційні системи.

Методологія дослідження в рамках даної роботи передбачає використання комплексного підходу до аналізу безпеки вебсайтів. Цей підхід включає кілька етапів, що забезпечують всебічний аналіз вразливостей та розробку ефективних методів їх виявлення та усунення.

Розробка інформаційної системи для аналізу безпеки вебсайтів є важливим кроком у забезпеченні захисту цифрових ресурсів від різноманітних загроз. Представлена робота не тільки підвищує розуміння актуальних проблем безпеки, але й пропонує ефективні рішення для їх вирішення. Це сприяє створенню більш надійного та безпечного Інтернет-простору для всіх користувачів.

1 ЗМІСТОВНА ЧАСТИНА

1.1 Актуальність та проблеми предметної області

З кожним днем кількість вебсайтів зростає, а разом з тим і загрози їхньої безпеки [1]. Відповідно, зростає і необхідність в ефективних засобах для виявлення та захисту від потенційних атак.

Система, яка розроблена в рамках даної теми, зосереджується на виявленні різноманітних вразливостей, які можуть бути використані для атак на вебсайти. Серед цих вразливостей перераховані такі типи, як Directory Traversal, XXE Test, Security Misconfiguration, Cross Site Scripting, CSRF, Session Fixation та Local File Inclusion [2].

Кожна з цих вразливостей може призвести до серйозних проблем для вебсайту та його користувачів. Наприклад, XSS може дозволити зловмиснику впровадити в шкідливий скрипт на сторінці вебсайту, що може використовуватися для крадіжки конфіденційної інформації користувачів або навіть для перехоплення сесій. CSRF може використовуватися для виконання небажаних дій в ім'я реєстрованого користувача без його на то згоди.

Розробка системи, спрямованої на аналіз та виявлення цих вразливостей, має велике значення для забезпечення безпеки і захисту користувачів від можливих атак. Така система може бути використана як власниками вебсайтів для перевірки безпеки своїх ресурсів, так і спеціалістами з безпеки інформаційних технологій для проведення аудиту та виявлення потенційних загроз.

Предметна область аналізу безпеки вебсайтів включає в себе ряд складних проблем, які можуть ускладнювати ефективність заходів забезпечення безпеки та розробки відповідних інструментів.

Загрози безпеки постійно еволюціонують і стають все більш винахідливими. Нові вразливості та методи атак появляються на щодень, тому системи безпеки повинні постійно оновлюватися, щоб відповідати сучасним вимогам.

Деякі вразливості можуть бути дуже складно виявити, особливо якщо вони

знаходяться у складних динамічних вебдодатках або якщо вони використовують нові, непопулярні техніки атаки. Одна вразливість може призвести до іншої, або їх поєднання може створити вектори атаки, які не враховані окремо.

Виявлення та виправлення вразливостей може бути дорогим та довготривалим процесом, особливо для малих організацій або проектів з обмеженими бюджетами та людськими ресурсами.

Деякі інструменти сканування на вразливості можуть видавати хибні результати, що призводить до витрат часу та ресурсів на перевірку недійсних проблем.

В процесі аналізу безпеки вебсайтів можуть збиратися конфіденційні дані або здійснюватися тестування, що може порушувати приватність користувачів та породжувати етичні проблеми. Відсутність єдиних стандартів для аналізу та забезпечення безпеки вебсайтів може ускладнювати співпрацю між різними інструментами та організаціями.

Загалом, забезпечення безпеки вебсайтів є складним завданням, яке потребує постійного вдосконалення та уваги до нових загроз і технологій.

1.2 Основні поняття та визначення

Кібербезпека – це галузь інформаційної безпеки, що займається захистом комп'ютерних систем, мереж, програмного забезпечення та даних від різноманітних загроз віртуального середовища. Основна мета кібербезпеки - запобігти доступу несанкціонованим особам до конфіденційної інформації, уникнути порушень інформаційної цілісності та доступності, а також захистити комп'ютерні системи від шкідливих програм та кібератак. Кібербезпека включає в себе різноманітні аспекти, такі як розробка та впровадження безпечних архітектур систем, захист мережевого трафіку, розробка та впровадження стратегій управління доступом, моніторинг та виявлення вразливостей, а також навчання персоналу щодо правил кібербезпеки та заходів безпеки.

Вебсайт – це колекція вебсторінок та інших елементів, які знаходяться на

вебсервері та доступні через Інтернет. Кожен вебсайт має унікальний URL, який може бути використаний для доступу до нього через веббраузер. Вебсайти можуть містити різноманітний контент, такий як текст, зображення, відео, аудіо, а також інші інтерактивні елементи, такі як форми зв'язку або функції для взаємодії з користувачами. Вебсайти використовуються для різних цілей, включаючи представлення інформації про компанію або організацію, надання доступу до продуктів або послуг, спілкування з аудиторією через блоги або форуми, а також для розваг та освіти. Вони можуть бути статичними, зміст яких залишається незмінним протягом тривалого часу, або динамічними, зміст яких генерується або змінюється в залежності від дій користувачів або зовнішніх подій.

Оцінка ризику [3] – це процес визначення потенційних загроз, що можуть виникнути внаслідок конкретної діяльності або події, та визначення ймовірності їх виникнення та впливу на цю діяльність або подію. Цей процес включає аналіз різних факторів, таких як потенційні загрози, вразливості, наслідки та імовірність виникнення подій, з метою ідентифікації та пріоритету ризиків. Оцінка ризику є ключовим етапом у розробці стратегій управління ризиками та прийнятті рішень з метою запобігання або зменшення впливу можливих загроз на організацію або проект. Вона допомагає організаціям та індивідуумам ідентифікувати та розуміти ризики, що стоять перед ними, та розробляти ефективні стратегії для їх управління. Оцінка ризику може бути кількісною або якісною. У кількісному підході ризики вимірюються за допомогою числових показників, таких як ймовірність виникнення події та її вплив. У якісному підході ризики оцінюються з урахуванням їхньої сутності та контексту, не використовуючи точних числових значень.

Шифрування – це процес перетворення даних зрозумілої форми (звичайної текстової або даних) в незрозумілу форму (шифрований текст), що робить їх незрозумілими для неавторизованих осіб. Цей процес використовується для забезпечення конфіденційності даних під час їх передачі через мережі, зберігання на носіях або обробки в комп'ютерних системах. Шифрування базується на використанні спеціальних алгоритмів, які перетворюють вхідні дані за допомогою

ключа шифрування. Цей ключ є обов'язковим елементом процесу шифрування і визначає, як саме дані будуть зашифровані. Тільки особи, які мають правильний ключ, можуть розшифрувати зашифровані дані і перетворити їх назад у зрозумілу форму.

Вразливість – це слабке місце або дефект в системі, програмному забезпеченні, апаратурі або в процесі, яке може бути використане для здійснення атаки, порушення безпеки або несанкціонованого доступу. Вразливості можуть виникати внаслідок помилок в програмному коді, недоліків в дизайні системи, неправильної конфігурації, недостатнього контролю доступу або використання застарілих технологій. Вразливості можуть мати різний характер і включати в себе вразливості в безпеці мережі, вебдодатків, баз даних, операційних систем, криптографічних протоколах тощо. Потенційні наслідки вразливостей можуть варіюватися від втрати конфіденційності даних та порушення цілісності до втрати доступу до ресурсів або контролю над системою.

Exploit – це зразок коду, програмний скрипт або техніка, яка використовується для зловживання вразливістю в програмному забезпеченні, операційній системі або протоколі комунікації з метою отримання несанкціонованого доступу до системи або виконання певної дії.

Exploit використовуються зловмисниками для проведення атак на системи з метою отримання конфіденційної інформації, порушення цілісності даних, заволодіння контролем над системою або завдання інших шкідливих наслідків. Вони можуть бути написані для використання певних вразливостей, таких як SQL-ін'єкції, XSS атаки, переповнення буфера, недоліки в криптографічних алгоритмах, а також використовувати артефакти технічного стеку, такі як програмні ресурси, операційні системи, мережеві протоколи тощо.

1.3 Основні поняття та визначення

Забезпечення захищеності веб-ресурсів включає в себе декілька підходів і методів, кожен з яких має свої переваги та недоліки. Один з таких підходів —

інструментальний аналіз.

Інструментальний аналіз передбачає використання спеціалізованих програмних інструментів для автоматизації процесу виявлення вразливостей у веб-додатках. Цей метод дозволяє швидко та ефективно перевірити безпеку веб-ресурсів, мінімізуючи людський фактор. Розглянемо основні інструменти, які використовуються для такого аналізу.

Burp Suite [4] – це потужний інструмент для тестування безпеки веб-додатків, який пропонує широкий спектр функцій для аналізу та виявлення вразливостей. Проксі-сервер для перехоплення та модифікації HTTP-запитів між клієнтом і сервером. Сканер вразливостей для автоматичного виявлення широкого спектру вразливостей, таких як SQL-ін'єкції, XSS, CSRF тощо. Інструменти для тестування вручну: такі як Intruder, Repeater, для налаштування та проведення детальних тестів.

OWASP ZAP [5] – це безкоштовний інструмент з відкритим вихідним кодом, розроблений для автоматизованого сканування веб-додатків на наявність вразливостей. Пасивний сканер аналізує трафік без внесення змін, виявляючи вразливості, які не потребують активних дій. Активний сканер виконує запити до серверів з метою виявлення вразливостей, таких як SQL-ін'єкції та XSS.

Nikto [6] – це командний інструмент для сканування веб-серверів, який перевіряє наявність понад 6700 потенційно небезпечних файлів, налаштувань та інших вразливостей. Швидке сканування дозволяє швидко перевірити сервер на наявність відомих вразливостей. Підтримка різних протоколів: HTTP, HTTPS, та інші. Гнучкі налаштування дозволяють користувачам налаштовувати процес сканування відповідно до своїх потреб.

SQLMap [7] – це інструмент для автоматизації процесу виявлення та експлуатації SQL-ін'єкцій. Автоматичне виявлення SQLMap може автоматично виявляти різні типи SQL-ін'єкцій. Різноманітні атаки підтримка різних методів ін'єкцій, включаючи blind, error-based, time-based та інші. Експлуатація SQLMap може виконувати команди на базі даних, отримувати дані, маніпулювати

таблицями тощо.

Nessus [8] – це комплексний сканер вразливостей, який перевіряє не тільки веб-додатки, але й інші компоненти мережевої інфраструктури. Сканування мереж, перевірка мережевих пристроїв на наявність вразливостей. Аналіз конфігурацій, перевірка правильності налаштувань мережевих пристроїв та серверів. Генерування детальних звітів з рекомендаціями щодо усунення вразливостей.

Acunetix [9] – це комерційний інструмент для сканування вразливостей веб-додатків, який пропонує автоматичне виявлення широкого спектру вразливостей.

Автоматичне сканування виявлення вразливостей, таких як XSS, SQL-ін'єкції, CSRF тощо. Перевірка налаштувань серверів та інших компонентів інфраструктури. Звітування та управління виправленнями генерування звітів та управління процесом усунення вразливостей.

OpenVAS [10] – це відкрите рішення для управління вразливостями, яке включає в себе широкий спектр інструментів для сканування мережевих ресурсів. База даних вразливостей регулярно оновлюється. Широкий спектр перевірок, підтримка різних типів сканувань та перевірок. Детальні звіти з рекомендаціями щодо усунення знайдених проблем.

Аналіз вручну передбачає ручну перевірку веб-ресурсів на наявність вразливостей та помилок у їхньому коді та конфігураціях. Цей метод вимагає глибоких знань з безпеки, досвіду та уважності, але дозволяє виявити проблеми, які можуть бути пропущені автоматизованими інструментами.

Першим етапом ручного аналізу є детальне вивчення структури веб-додатку. Визначення структури URL-адрес, параметрів запитів та потенційно уразливих точок входу. Огляд файлової системи, вивчення директорій, файлів та їхніх прав доступу. Перевірка публічних ресурсів, визначення ресурсів, які доступні без аутентифікації.

Перевірка налаштувань безпеки. Цей етап включає перевірку конфігурацій серверів та додатків. Налаштування веб-сервера, конфігураційні файли, доступність адміністраторських панелей. Безпека передачі даних, перевірка

використання HTTPS, правильність налаштування SSL/TLS сертифікатів. Заголовки безпеки, перевірка наявності та коректності заголовків безпеки, таких як Content-Security-Policy, X-Frame-Options, X-Content-Type-Options та інших.

Верифікація аутентифікації та авторизації. Перевірка форми входу, аналіз захищеності форми входу, використання захисту від brute force атак. Управління сесіями, перевірка безпеки управління сесіями, використання secure та HttpOnly атрибутів для cookies, зміна ідентифікаторів сесій після аутентифікації. Ролі та права доступу, верифікація правильного розподілу прав доступу між різними ролями користувачів, перевірка захищеності адміністративних функцій.

Тестування на вразливості. Цей етап включає ручне тестування на наявність найбільш поширених вразливостей. Огляд логів серверів та додатків допомагає виявити підозрілі дії та можливі атаки. Аналіз логів серверів: перевірка логів веб-сервера на наявність підозрілих запитів, спроб експлойтів та інших аномалій. Логи додатків: аналіз журналів додатків на наявність помилок, некоректних дій користувачів та можливих спроб атак.

Оцінка механізмів обробки помилок. Правильна обробка помилок є важливим аспектом безпеки. Вивід повідомлень про помилки. Перевірка наявності детальних повідомлень про помилки, які можуть розкривати внутрішню структуру додатку або конфіденційну інформацію. Безпечна обробка винятків, оцінка механізмів обробки винятків та помилок для запобігання витоку інформації.

На основі проведеного аналізу вручну, формуються рекомендації щодо усунення виявлених вразливостей. Підготовка детальних звітів з описом виявлених вразливостей та рекомендацій щодо їх виправлення. Допомога в реалізації рекомендацій та впровадження заходів для покращення безпеки.

Комплексна оцінка захищеності веб-ресурсів являє собою систематичний підхід до виявлення та усунення вразливостей, який об'єднує різні методи аналізу, інструменти та техніки. Вона забезпечує всебічний огляд безпеки веб-додатку, включаючи автоматизований і ручний аналіз, а також оцінку архітектури та процесів. Основні елементи комплексної оцінки включають:

Комплексна оцінка починається з детального планування, яке включає визначення обсягу роботи, методів, що будуть використовуватись, та ресурсів, необхідних для проведення оцінки. Важливі кроки включають:

Визначення цілей, встановлення чітких цілей оцінки, наприклад, виявлення критичних вразливостей або перевірка дотримання стандартів безпеки. Збір інформації, збір даних про веб-додаток, включаючи архітектуру, технології, що використовуються, та процеси розробки. Вибір інструментів та методів, вибір підходящих інструментів для автоматизованого аналізу та методів для ручного тестування.

Автоматизовані інструменти грають ключову роль у комплексній оцінці, забезпечуючи швидке виявлення типових вразливостей. Сканери безпеки, використання сканерів, таких як OWASP ZAP, Burp Suite, Nikto, для виявлення широкого спектру вразливостей, включаючи SQL-ін'єкції, XSS, CSRF та інші.

Аналізатори коду, застосування статичних аналізаторів коду для перевірки вихідного коду на наявність вразливостей, стандартних помилок та проблем з безпекою. Динамічне тестування, використання інструментів для динамічного аналізу, які перевіряють веб-додаток під час його виконання для виявлення вразливостей, що виникають під час реальної взаємодії з користувачем.

Ручний аналіз доповнює автоматизовані інструменти, дозволяючи глибше дослідити специфічні аспекти безпеки веб-додатку.

Огляд коду, детальний перегляд критичних секцій коду для виявлення складних та неочевидних вразливостей, які можуть бути пропущені автоматичними інструментами. Аналіз логіки, перевірка бізнес-логіки додатку для виявлення помилок у реалізації процесів, які можуть призвести до порушення безпеки. Тестування на проникнення, проведення ручних тестів на проникнення, імітуючи атаки зловмисників для виявлення слабких місць у захисті додатку.

Комплексна оцінка також включає аналіз архітектури веб-додатку та процесів розробки. Оцінка архітектури, перевірка архітектури додатку на предмет відповідності кращим практикам безпеки, включаючи розподіл компонентів,

використання захищених протоколів, налаштування серверів та мережевих пристроїв.

Оцінка процесів розробки, аналіз процесів розробки, тестування та впровадження для виявлення можливих слабких місць та забезпечення дотримання стандартів безпеки.

Після завершення оцінки створюється детальний звіт. Опис виявлених вразливостей, повний опис всіх виявлених вразливостей, їх ступінь ризику та можливі наслідки для веб-додатку. Рекомендації щодо усунення, конкретні рекомендації щодо усунення виявлених проблем, включаючи технічні рішення, налаштування системи та зміни в процесах розробки. План дій, розробка плану дій для реалізації рекомендацій та забезпечення постійного моніторингу безпеки.

1.4 Огляд останніх публікацій та існуючих аналогічних систем

Сфера кібербезпеки постійно змінюється, і нові загрози виникають з часом. Аналіз останніх публікацій допомагає фахівцям залишатися в курсі останніх тенденцій та знаходити нові методи захисту.

Публікації та аналогічні системи часто включають опис нових типів кібератак, вразливостей та ефективних методів захисту. Аналіз цих матеріалів допомагає ідентифікувати потенційні ризики та розробляти стратегії їх запобігання.

Малкольм Макдональд [3] пропонує чіткий та практичний підхід до забезпечення безпеки вебсайтів. Він досліджує реальні загрози, з якими можуть зіткнутися розробники вебдодатків, та надає конкретні приклади та рекомендації щодо захисту. Книга звертається до розробників на будь-якому рівні, надаючи їм знання та інструменти, необхідні для ефективного захисту їхніх вебдодатків від потенційних загроз. Автор детально розглядає такі аспекти безпеки, як захист від SQL-ін'єкцій, XSS-атак, аутентифікація та авторизація, криптографічні методи, безпека мережі та багато іншого. Через призму практичних прикладів та реальних сценаріїв автор допомагає розробникам зрозуміти загрози та розробити ефективні

стратегії для їхнього запобігання.

З іншої сторони Майк Шейма[4] надає глибоке розуміння потенційних загроз для вебдодатків та методів їх виявлення та запобігання. Використання цього джерела дає зрозуміти, яким чином хакери можуть атакувати вебсайти та вебдодатки, і які саме уразливості можуть бути використані для цього. Зокрема, книга розглядає такі атаки, як SQL-ін'єкції, XSS, CSRF, а також атаки на аутентифікацію та авторизацію. Практичні поради та приклади в книзі стали основою для розроблення стратегій безпеки для власного проєкту. Ці поради включають в себе методи перевірки введених даних, захист від переповнення буфера, використання безпечних протоколів та криптографічних алгоритмів.

Крім того, через приклади підозрілої поведінки та реальних атак, які описані в книзі, дослідник може краще зрозуміти уразливості свого вебдодатка та розробити ефективні стратегії для їхнього запобігання. Такий підхід допомагає підтвердити або розширити розуміння теми та представити обґрунтовані рекомендації для покращення безпеки вебдодатка у роботі.

Висновки до розділу 1

У розділі "Змістовна частина" було всебічно досліджено та проаналізовано актуальність, основні поняття та визначення, а також сучасні публікації і існуючі аналогічні системи, що стосуються аналізу безпеки вебсайтів.

В підрозділі 1.1 було підкреслено важливість забезпечення безпеки вебсайтів у сучасному цифровому світі. З огляду на зростання кількості кіберзагроз, зокрема таких як SQL-ін'єкції та XSS-атаки, питання безпеки вебдодатків стало критично важливим для організацій, які прагнуть захистити свої дані та зберегти довіру користувачів. Були окреслені основні проблеми предметної області, серед яких недостатній рівень обізнаності розробників про сучасні методи атак, а також обмеженість ресурсів для їх ефективного виявлення та нейтралізації.

У підрозділі 1.2 розглянуто ключові поняття та терміни, які є основоположними для розуміння процесів аналізу безпеки вебсайтів. Було надано

визначення таких важливих термінів, як SQL-ін'єкція, XSS-атака, CSRF, шифрування, вразливість, експлойт, та інші. Це створило базову термінологічну основу, необхідну для подальшого детального аналізу та розробки інформаційної системи.

Підрозділ 1.3 присвячено огляду останніх публікацій та існуючих аналогічних систем. Було проаналізовано такі важливі джерела, як книги "Web Application Security: A Beginner's Guide" by Bryan Sullivan та "Hacking Web Apps: Detecting and Preventing Web Application Security Problems" by Mike Shema, що надали цінні інсайти та практичні рекомендації щодо захисту вебдодатків. Огляд існуючих систем, таких як Burp Suite та OWASP ZAP, продемонстрував їхні можливості та функціонал, що можуть бути використані для розробки ефективної інформаційної системи аналізу безпеки вебсайтів.

Узагальнюючи, змістовна частина роботи заклала теоретичну та практичну основу для подальшої розробки інформаційної системи аналізу безпеки вебсайтів. Розглянуті матеріали та аналіз сучасних технологій забезпечують глибоке розуміння предметної області та надають необхідні знання для створення ефективного інструменту, здатного виявляти та нейтралізувати загрози безпеці вебдодатків.

2 ОСНОВНІ ВРАЗЛИВОСТІ ВЕБСАЙТІВ

2.1 Найбільш розповсюджені ін’єкції

Ін'єкції є однією з найпоширеніших і найнебезпечніших категорій вразливостей у вебдодатках. Вони виникають, коли зловмисник має змогу вставити (ін'єктувати) шкідливий код або команди у вхідні дані додатку, які потім виконуються сервером. Це може призвести до несанкціонованого доступу до даних, компрометації системи або виконання небажаних операцій.

Механізм дії ін'єкцій простий. Зловмисник надсилає шкідливі дані до вебдодатку через форми введення, URL-запити або інші канали взаємодії. Вебдодаток обробляє ці дані без належної перевірки або екранування. Шкідливий код виконується сервером, що може призвести до витоку даних, змін у базі даних, компрометації серверу або інших небажаних наслідків. До ін’єкцій відносять: XSS, CSRF, XEE, SQL injection.

Cross-Site Scripting - це вразливість вебдодатків, яка дозволяє зловмисникам впроваджувати шкідливі скрипти у вебсторінки, які переглядають інші користувачі. Ці скрипти можуть виконуватися в контексті браузера користувача, що може призводити до крадіжки конфіденційних даних, виконання небажаних дій від імені користувача або інших шкідливих наслідків. XSS-атаки поділяються на три типи.

Перший, Stored XSS це шкідливий скрипт зберігається на сервері в базі даних і відображається на вебсторінках при кожному їхньому завантаженні користувачами за алгоритмом (рис. 2.1). Наприклад, зловмисник може вставити шкідливий скрипт у форму коментарів на вебсайті. Коли інші користувачі переглядають коментарі, скрипт виконується в їхніх браузерах.

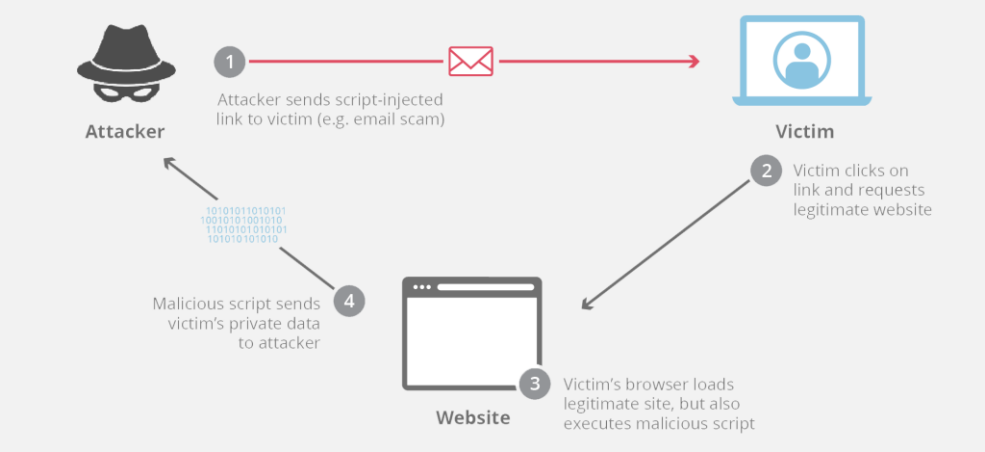

Рисунок 2.1 – Алгоритм stored XSS атаки

Другий, Reflected XSS, відрізняється тим, що відображається безпосередньо в відповідь на запит користувача. Це може статися, коли сервер обробляє вхідні дані і відображає їх у відповідях без належної валідації або екранування. Наприклад, зловмисник може надіслати жертві спеціально створений URL, який містить шкідливий скрипт.

Третій, DOM-based, впроваджується і виконується безпосередньо в браузері, маніпулюючи Document Object Model вебсторінки. Це відбувається, коли клієнтський JavaScript на вебсторінці обробляє небезпечні дані, введені зловмисником.

Наслідки XSS-атаки, можуть призвести то того, що зловмисник може викрасти сесійні файли cookie користувача, що дозволяє йому видавати себе за жертву. Підробляти частини вебсторінки, щоб виманити у користувача конфіденційну інформацію. Змінювати відображений контент сторінки, вводячи в оману користувачів. Виконувати небажані дії в контексті сесії користувача.

Основним захистом від XSS є використання функцій для екранування спеціальних символів у HTML, JavaScript, CSS та URL, щоб запобігти виконанню шкідливих скриптів. Перевірка та обмеження вхідних даних відповідно до очікуваного формату та вмісту. Використання Content Security Policy, що дозволяє визначати дозволені джерела виконуваних скриптів і значно знижує ризик XSS-атак.

Загалом, запобігання XSS вимагає належної обробки всіх вхідних даних і дотримання найкращих практик безпеки при розробці вебдодатків.

Cross-Site Request Forgery [11] - це тип атаки на вебдодатки, коли зловмисник змушує користувача виконувати небажані дії на вебсайті, на якому він аутентифікований. Атака використовує авторизацію жертви, змушуючи її веббраузер надсилати підроблені HTTP-запити, які зловмисник бажає виконати.

Механізм дії CSRF-атак ділиться на чотири кроки (рис. 2.2). Жертва реєструється на легітимному вебсайті, отримуючи сесію або токен авторизації, який зберігається у вигляді cookies або іншого методу збереження стану. Зловмисник створює підроблену вебсторінку або електронний лист із шкідливим кодом (зазвичай HTML-форма або JavaScript), який виконує запит до легітимного вебсайту, коли жертва його відкриває. Оскільки жертва вже зареєстрована на легітимному сайті, її браузер автоматично додає дані авторизації до підробленого запиту, дозволяючи зловмиснику виконати небажані дії.

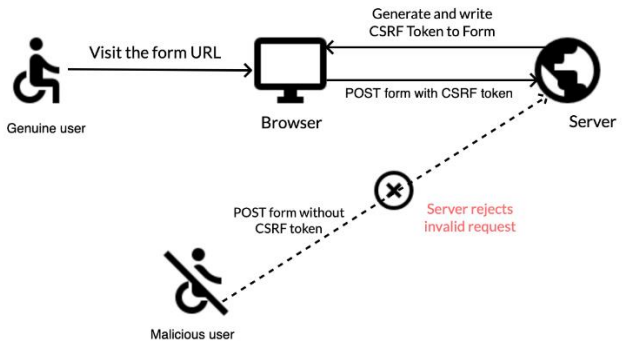

Рисунок 2.2 – Алгоритм CSRF-атаки

Наслідком CSRF-атаки зловмисник зможе змусити жертву виконати небажані дії, такі як зміна паролю, переказ коштів, або виконання адміністративних команд. Атака також призводить до витоку конфіденційної інформації або зміни важливих даних.

Щоб вберегтись від CSRF-атаки необхідне включення унікального, непередбачуваного токену в кожну форму або запит, що модифікує стан. Сервер перевіряє наявність і дійсність цього токену перед виконанням дії. Це дозволяє переконатися, що запит був ініційований з легітимної сторінки додатка. Перевірка заголовків Origin або Referer в запитах на сервері, щоб переконатися, що запит надходить з легітимного джерела. Використання POST-запитів для дій, що змінюють стан, оскільки GET-запити легше підробити. Використання атрибутів SameSite для cookies, щоб обмежити їх надсилання лише на запити, що походять з того ж сайту. Включення CAPTCHA для критичних дій може запобігти автоматичним запитам.

Захист від CSRF-атак потребує комплексного підходу[13], що включає використання CSRF-токенів, перевірку джерела запитів та правильну конфігурацію cookies. Це забезпечить додатковий рівень безпеки для вебдодатків та їхніх користувачів.

XML External Entity – це вразливість у вебдодатках, що обробляють XML-дані. Вона виникає, коли XML-парсер небезпечно обробляє зовнішні ресурси, що може дозволити зловмиснику отримати доступ до конфіденційних даних, здійснити DoS-атаку, або виконати інші шкідливі дії.

Механізм дії XXE-атак складається з декількох кроків (рис. 2.3). Зловмисник надсилає до вебдодатку спеціально створений XML-документ, який містить зовнішню сутність. XML-обробник обробляє зовнішню сутність, що може включати доступ до локальних файлів або зовнішніх ресурсів.

Внаслідок цього зловмисник отримує доступ до конфіденційних даних і може викликати завантаження великого об'єму даних або здійснити інші небажані дії.

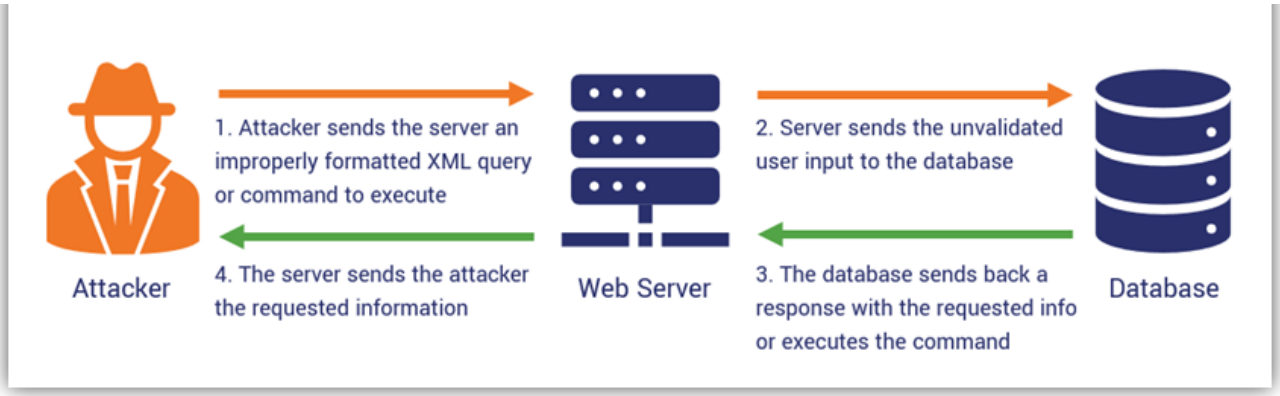

Рисунок 2.3 – Алгоритм XXE-атаки

Наслідком XXE-атак є розкриття конфіденційної інформації та виконання зовнішніх запитів. Крім цього зловмисник може здійснити DoS-атаку, примусивши сервер завантажувати великі або нескінченні ресурси.

Загалом, захист від XXE-атак вимагає належної конфігурації XML-парсерів, використання безпечних бібліотек та належної валідації вхідних даних. Це забезпечує додатковий рівень безпеки для вебдодатків та запобігає можливим атакам.

SQL-ін'єкції – це тип вразливості в вебдодатках, який дозволяє зловмиснику виконувати довільні SQL-запити в базі даних програми[15]. Це може призвести до витоку конфіденційних даних, видалення або зміни даних, а також до інших шкідливих дій.

Механізм дії SQL-ін'єкцій зображений на рисунку (рис. 2.4). Зловмисник надсилає спеціально сформовані вхідні дані до вебдодатку. Ці дані можуть бути введені через форми, URL, HTTP-заголовки або інші вхідні точки. Вебдодаток використовує ці вхідні дані для формування SQL-запиту до бази даних без належної обробки (валідації та екранування). База даних виконує згенерований запит, включаючи шкідливий SQL-код, наданий зловмисником.

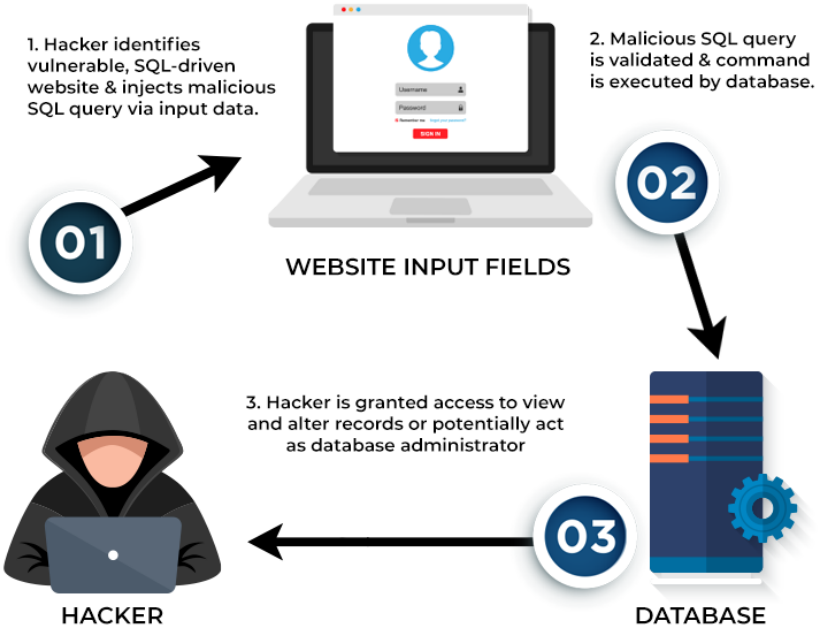

Рисунок 2.4 – Алгоритм виконання SQL-ін'єкції

Припустимо, що вебдодаток використовує наступний SQL-запит для аутентифікації користувача: «SELECT * FROM users WHERE username = 'user' AND password = 'pass'; ». Якщо користувач вводить user' OR '1'='1 як ім'я користувача і будь-який пароль, запит зміниться на: «SELECT * FROM users WHERE username = 'user' OR '1'='1' AND password = 'pass'; ». Це завжди буде правдивим, тому що '1'='1' завжди істинно, і зловмисник зможе обійти аутентифікацію [15].

Наслідком SQL-ін'єкцій зловмисник отримує доступ до даних, які йому не призначені. Він може змінити або видалити дані, що може призвести до втрати або спотворення інформації. У деяких випадках зловмисник може виконати адміністраторські команди на базі даних, що може призвести до повного компрометування системи.

Використання параметризованих запитів та екранування вхідних даних допомагають дозволяє відокремити SQL-код від даних, що запобігає впровадженню шкідливих SQL-команд.

SQL-ін'єкції залишаються однією з найсерйозніших загроз для вебдодатків, але правильне проектування і налаштування системи можуть значно знизити ризик їхнього успішного використання.

2.2 Обхід каталогу та включення локального файлу

Атака обходу каталогів дуже проста в реалізації, але наслідки можуть бути шкідливими, у зламі дані означають все, і якщо зловмисник отримає будь-які важливі або привілейовані дані, ця проста атака може призвести до багатьох інших.

Directory Traversal – це тип вразливості в вебдодатках, який дозволяє зловмиснику отримати доступ до файлів і директорій на сервері поза межами кореневого каталогу вебдодатку. Це досягається шляхом використання спеціально сформованих шляхів у запитах до сервера.

Механізм дії Directory Traversal складається з алгоритму (рис. 2.5). Зловмисник надсилає запит до вебдодатку з використанням спеціальних символів, таких як dot-dot-slash, щоб піднятися по файловій системі вище за кореневий каталог вебдодатку [18]. Якщо сервер не належним чином обробляє і не фільтрує ці запити, він може дозволити доступ до файлів і директорій, які знаходяться за межами дозволеного каталогу. Зловмисник може отримати доступ до конфіденційних файлів, таких як конфігураційні файли, файли з паролями або інші важливі дані, що можуть бути використані для подальших атак.

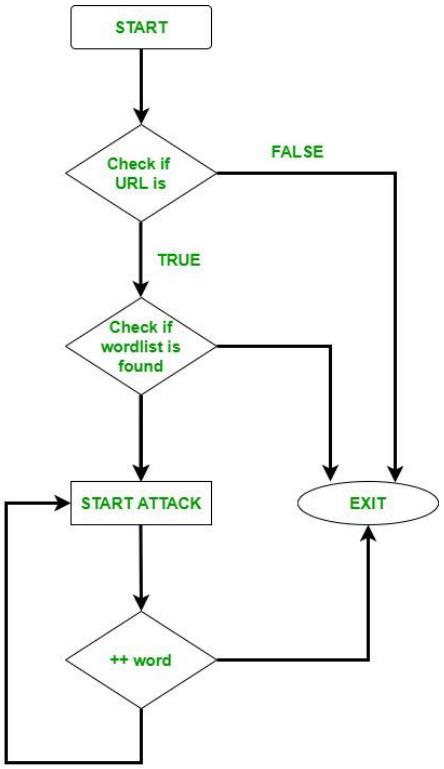

Рисунок 2.5 – Алгоритм обходу каталогів

Наслідки Directory Traversal атак:

- розкриття конфіденційної інформації;
- компрометація системи;
- виконання довільного коду.

Захист від Directory Traversal вимагає належної обробки вхідних даних, налаштування прав доступу та використання сучасних інструментів безпеки для зниження ризиків таких атак.

Local File Inclusion – це тип вразливості в вебдодатках, який дозволяє зловмиснику включати файли, що знаходяться на сервері, до виконання в контексті вебдодатку. Це може призвести до витоку конфіденційних даних, виконання довільного коду, або інших шкідливих дій.

Механізм дії LFI-атак [19] зображений на схемі (рис. 2.6). Зловмисник надсилає спеціально сформований запит до вебсайту, використовуючи параметри, що визначають шлях до файлу. Вебсайт обробляє цей запит та включає файл, зазначений у параметрах запиту. Якщо додаток не належним чином фільтрує та перевіряє ці параметри, зловмисник може вказати будь-який файл на сервері. Файл

включається в контекст виконання вебдодатку, що може призвести до виконання довільного коду, розкриття конфіденційної інформації або інших небажаних наслідків.

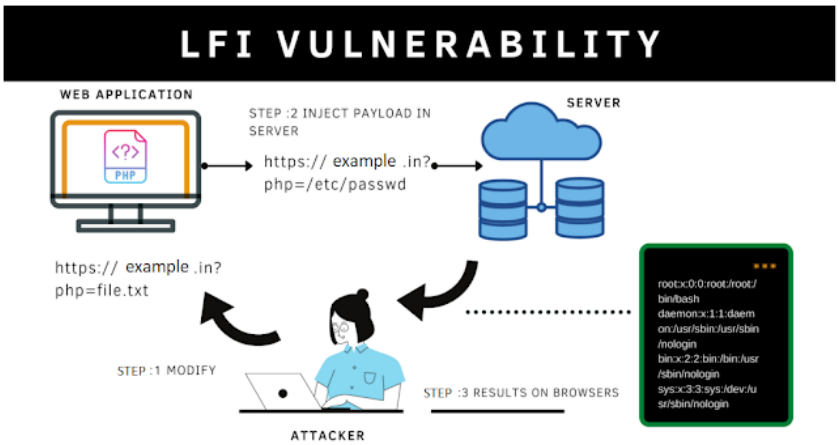

Рисунок 2.6 – Схема LFI-атаки

Наслідком LFI-атаки зловмисник може отримати доступ до файлів, які містять конфіденційні дані, такі як паролі, конфігураційні файли, ключі. Якщо зловмисник зможе включити файли з виконуваним кодом, він може виконати довільний код на сервері, що призведе до повного компрометування системи. Зловмисник зможе переміщатися по файловій системі сервера, використовуючи відносні шляхи і включати різні файли для збору інформації.

Загалом, захист від LFI-атак вимагає належної обробки вхідних даних, правильного налаштування файлової системи та використання сучасних інструментів безпеки для зниження ризиків таких атак.

2.3 Session Fixation

Session Fixation – це тип атаки на вебдодатки, при якій зловмисник змушує жертву використовувати відому йому сесію. Це дозволяє зловмиснику отримати доступ до облікового запису жертви після аутентифікації, використовуючи фіксовану сесію.

Механізм дії Session Fixation зображений на рисунку (рис. 2.7). Зловмисник отримує дійсну сесію від вебдодатку. Це може бути зроблено через легітимний вхід на сайт або через інші методи, такі як передбачувані ідентифікатори сесій. Зловмисник надсилає жертві сесію через URL, cookie або приховану форму. Жертва використовує цю сесію для доступу до вебдодатку. Жертва реєструється у вебдодатку, використовуючи фіксовану сесію. Після аутентифікації жертви, зловмисник використовує ту саму фіксовану сесію для доступу до облікового запису жертви.

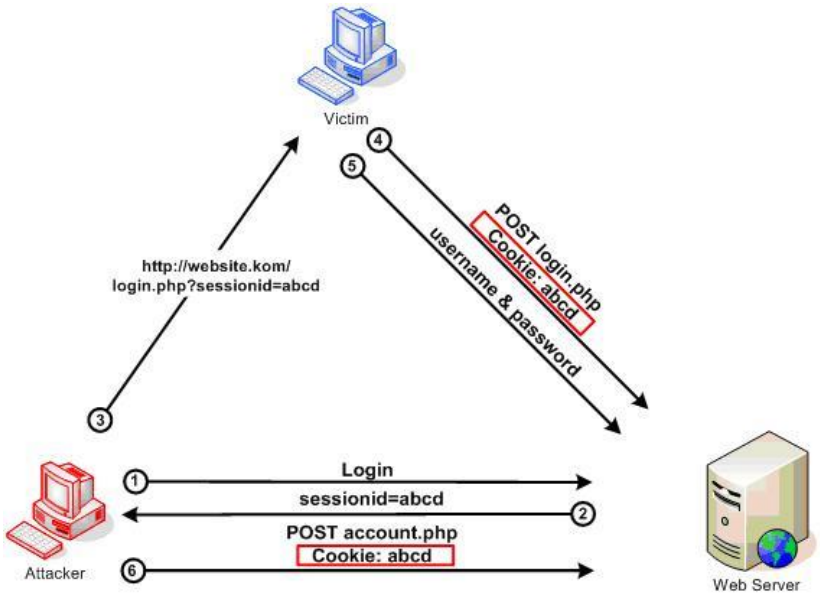

Рисунок 2.7 – Схема Session Fixation атаки

Наслідки Session Fixation атак:

- зловмисник може отримати повний доступ до облікового запису жертви, включаючи конфіденційну інформацію та можливість виконувати дії від імені жертви;
- зловмисник може змінювати або видаляти дані облікового запису жертви;
- зловмисник може використовувати обліковий запис жертви для проведення подальших атак на інших користувачів або системи.

Захист від Session Fixation:

- вебдодаток повинен генерувати новий ідентифікатор сесії після успішної аутентифікації користувача. Це забезпечує, що будь-яка попередня сесія стає недійсною;
- налаштування cookie так, щоб вони мали атрибути HttpOnly, Secure та SameSite, що зменшує можливість їх викрадення та підробки;
- впровадження механізму валідації сесій, щоб виявляти підозрілі дії або одночасні входи з різних IP-адрес або пристроїв;
- використання коротких термінів життя сесій.

Захист від Session Fixation атак вимагає належної обробки ідентифікаторів сесій та застосування сучасних засобів безпеки для забезпечення захищеності вебдодатків та їхніх користувачів.

Висновки до розділу 2

У розділі "Основні вразливості вебсайтів" були розглянуті ключові типи атак, що загрожують безпеці вебдодатків, а саме: SQL-ін'єкції, XSS, CSRF, XXE, Directory Traversal, LFI та Session Fixation. Кожна з цих вразливостей має свої унікальні механізми дії та потенційні наслідки для безпеки вебдодатків.

SQL-ін'єкції дозволяють зловмисникам виконувати довільні SQL-запити, що може призвести до розкриття конфіденційних даних або повної компрометації бази даних. XSS-атаки спрямовані на виконання шкідливого скрипту в браузері користувача, що може призвести до крадіжки сесій або виконання небажаних дій

від імені користувача. CSRF-атаки змушують користувача виконувати небажані дії на сайті, на якому він аутентифікований, без його відома.

XXE-атаки використовують небезпечну обробку XML-документів для отримання доступу до конфіденційних даних або виконання шкідливих дій на сервері. Directory Traversal та Local File Inclusion дозволяють зловмисникам отримувати доступ до файлів на сервері, що може призвести до розкриття конфіденційних даних або виконання довільного коду. Session Fixation атаки зосереджені на захопленні сесій користувачів, що дозволяє зловмисникам здійснювати несанкціонований доступ до облікових записів.

Для захисту від цих атак необхідно застосовувати комплексні заходи безпеки, включаючи валідацію та фільтрацію вхідних даних, використання параметризованих запитів, налаштування прав доступу, впровадження сучасних методів управління сесіями та використання захищених механізмів передачі даних. Тільки системний підхід до безпеки вебдодатків може забезпечити ефективний захист від різноманітних загроз та мінімізувати ризики компрометації.

Загалом, забезпечення безпеки вебдодатків є складним та багатогранним завданням, яке вимагає постійного моніторингу, оновлення та впровадження нових технологій і методології для протидії новим загрозам. Вивчення та розуміння основних вразливостей є першим кроком на шляху до створення безпечних та надійних вебдодатків.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ РОЗРОБЛЕНОЇ СИСТЕМИ

3.1 Програмні засоби реалізації інформаційної системи

Python – це потужна мова програмування, яка відома своєю простотою, читабельністю та універсальністю. Вона підтримує різні парадигми програмування, включаючи об'єктно-орієнтоване, процедурне і функціональне програмування. Python широко застосовується у веброзробці [20], аналізі даних, машинному навчанні, автоматизації та багатьох інших областях. Завдяки великій кількості бібліотек, Python дозволяє розробникам швидко створювати ефективні рішення для складних завдань.

Requests – це бібліотека для виконання HTTP-запитів, яка спрощує взаємодію з вебсервісами. Вона дозволяє легко відправляти запити GET, POST та інші типи запитів, обробляти відповіді та керувати сесіями.

BeautifulSoup – це бібліотека для парсингу HTML та XML документів. Вона надає інструменти для навігації, пошуку та модифікації HTML-документів, що робить її дуже корисною для вебскрапінгу та аналізу вмісту вебсторінок. BeautifulSoup дозволяє легко розбирати HTML і XML документи, створюючи дерево об'єктів, яке можна переглядати і змінювати. Бібліотека підтримує потужні методи пошуку елементів за тегами, класами, ідентифікаторами та іншими атрибутами. Це дозволяє швидко знаходити потрібні елементи в документі. BeautifulSoup дозволяє змінювати вміст HTML і XML документів. Можна додавати, видаляти або змінювати елементи та їх атрибути. Вона може обробляти неправильно сформований HTML, автоматично виправляючи помилки та заповнюючи пропущені теги.

Urllib.parse – це модуль для роботи з URL-адресами. Він дозволяє розбирати, створювати, змінювати та кодувати URL-адреси, що є важливим для роботи з вебзапитами та аналізу вебадрес.

Ці бібліотеки значно розширюють можливості Python, роблячи його потужним інструментом для розробки вебдодатків, автоматизації завдань та обробки даних.

Django – це високорівневий фреймворк для веброзробки на Python, створений для побудови надійних та масштабованих вебдодатків [21]. Він розроблений з акцентом на швидкий розвиток та чистий, практичний дизайн. Django надає все необхідне для створення вебдодатків, від обробки запитів до управління базами даних. Він включає в себе ORM для взаємодії з базами даних, що дозволяє працювати з даними як з об'єктами Python.

Однією з визначних рис є автоматично створюваний адміністративний інтерфейс. Це зручний вебінтерфейс для управління даними, який автоматично генерується на основі моделей даних, що дозволяє адміністраторам сайту легко виконувати CRUD-операції. Django використовує потужну систему маршрутизації, яка дозволяє визначати URL-шляхи та пов’язувати їх з відповідними функціями або класами представлень. Це дозволяє чітко та гнучко визначати логіку обробки запитів. Фреймворк містить власну систему шаблонів, яка дозволяє динамічно генерувати HTML, забезпечуючи чистий поділ між логікою програми і її поданням. Це допомагає легко створювати і підтримувати вебсторінки.

Django включає низку вбудованих механізмів безпеки, таких як захист від SQL-ін'єкцій, XSS, CSRF та Clickjacking. Ці функції допомагають створювати безпечні вебдодатки з мінімальними зусиллями. Він підтримує використання різних баз даних і може бути легко масштабований для обробки великих обсягів трафіку. Це досягається завдяки підтримці кешування, можливості горизонтального масштабування і використанню сторонніх служб, таких як Memcached або Redis.

3.2 Середовище розробки Pycharm

PyCharm – це інтегроване середовище розробки для Python, розроблене компанією JetBrains. Це потужний інструмент, який надає розробникам безліч

функцій для створення, налагодження та тестування коду. Він надає інтелектуальний редактор коду з підтримкою автодоповнення, синтаксичного підсвічування та аналізу коду в режимі реального часу. Це допомагає швидко писати код і виявляти помилки ще до запуску програми. IDE включає потужний відлагоджувач з підтримкою точок зупинки, покрокового виконання коду та моніторингу змінних. Він також підтримує написання та виконання тестів, забезпечуючи інтеграцію з фреймворками для тестування, такими як unittest, pytest і nosetests.

PyCharm підтримує інтеграцію з популярними системами контролю версій, такими як Git, Mercurial, Subversion і Perforce. Це дозволяє легко керувати версіями коду, виконувати коміти, створювати гілки та вирішувати конфлікти. PyCharm надає інструменти для роботи з базами даних, включаючи перегляд структур баз даних, виконання SQL-запитів і редагування даних. Це спрощує розробку додатків, що використовують бази даних. IDE підтримує розробку вебдодатків з використанням популярних фреймворків, таких як Django, Flask і Pyramid. Середовище розробки надає спеціалізовані інструменти для роботи з шаблонами, маршрутизацією та налаштуванням серверів.

PyCharm пропонує інтеграцію з бібліотеками для наукових обчислень, такими як NumPy, SciPy і Matplotlib. Це робить його зручним інструментом для розробників, які займаються аналізом даних та машинним навчанням. Широкий набір інструментів та автоматизованих функцій дозволяє значно підвищити продуктивність розробки, скорочуючи час, необхідний для виконання рутинних завдань.

3.3 Мова програмування JavaScript та framework Bootstrap

JavaScript – це високорівнева, динамічна мова програмування, яка використовується для створення інтерактивних вебсторінок. Вона є однією з основних технологій веброзробки разом з HTML та CSS. Спочатку створена для роботи на стороні клієнта, тепер також активно використовується на стороні

сервера завдяки середовищу Node.js. JavaScript дозволяє створювати інтерактивні елементи на вебсторінках, такі як анімації, форми, галереї зображень та інші інтерактивні компоненти. DOM представляє структуру HTML-документу у вигляді дерева. JavaScript дозволяє програмно маніпулювати цим деревом, змінюючи вміст, структуру та стиль вебсторінок у режимі реального часу. Також дана мова програмування надає можливість обробляти події користувача, такі як кліки, наведення миші, введення даних, що дозволяє створювати динамічні та інтерактивні інтерфейси. JavaScript підтримується всіма сучасними веббраузерами, що забезпечує високу сумісність та широке застосування в веброзробці. Вона підтримує як синхронне, так і асинхронне програмування. Асинхронні операції, такі як робота з мережею або анімації, виконуються без блокування головного потоку виконання коду.

JavaScript широко використовується для створення динамічних і взаємодійних вебсторінок. З його допомогою можна створювати розширені функціональні можливості на стороні клієнта, забезпечуючи покращений користувацький досвід. На стороні сервера використовується разом з Node.js для створення масштабованих і високопродуктивних вебдодатків.

Bootstrap – це фреймворк для розробки frontend-інтерфейсів, створений компанією Twitter. Він містить набір інструментів для швидкого створення адаптивних вебдизайнів, забезпечуючи сумісність з різними браузерами та пристроями. Основні можливості Bootstrap надає широкий спектр готових компонентів інтерфейсу, таких як кнопки, форми, навігаційні панелі, модальні вікна, каруселі та багато інших. Це значно скорочує час розробки, дозволяючи використовувати готові рішення замість створення з нуля.

Фреймворк підтримує адаптивний дизайн завдяки використанню сіткової системи. Це дозволяє створювати вебсторінки, які автоматично адаптуються до різних розмірів екранів та пристроїв, забезпечуючи зручне користування як на десктопах, так і на мобільних пристроях.

Хоча Bootstrap надає багато готових компонентів, він також дозволяє налаштовувати дизайн за допомогою змінних SASS/LESS, що дає можливість змінювати кольори, відступи, розміри та інші стилістичні параметри для відповідності вашим вимогам. Він забезпечує сумісність з сучасними браузерами, що дозволяє розробникам створювати стабільні та однаково виглядаючі інтерфейси на різних платформах. Також фреймворк підтримується великою спільнотою розробників, що дозволяє легко знаходити приклади використання та отримувати допомогу.

Фреймворк використовується для швидкої розробки адаптивних вебдизайнів. Він підходить як для створення простих вебсайтів, так і для розробки складних вебдодатків. Завдяки готовим компонентам та адаптивній сітці, розробники можуть значно скоротити час на створення інтерфейсів, забезпечуючи при цьому високу якість та сумісність своїх рішень.

JavaScript і Bootstrap є потужними інструментами для веброзробки. JavaScript забезпечує динамічність і інтерактивність вебсторінок, дозволяючи створювати багатофункціональні вебдодатки. Bootstrap, у свою чергу, спрощує процес створення адаптивних і стильних інтерфейсів, забезпечуючи швидкість розробки та високу якість дизайну. Разом ці технології дозволяють створювати сучасні, ефективні та зручні для користувача вебпродукти.

3.4 Створення та налаштування конфігурацій проекту

Новий Django проект створений у середовищі розробки PyCharm. Після створення віртуального середовища Python та встановлення Django, команда django-admin startproject project_name використовується для створення нового проекту. Назва проекту обирається відповідно до теми.

Для стилізації та поліпшення зовнішнього вигляду вебзастосунку, Bootstrap використаний за допомогою CDN-посилання, файл CSS Bootstrap буде підключений до HTML-коду. Крім того, класи Bootstrap використані для внесення відповідних стилів до елементів. Після налаштування конфігурації проекту і

підключення необхідних інструментів, наступним кроком буде встановлення двох важливих бібліотек Python: BeautifulSoup і Requests.

Було створено зовнішній вигляд головної сторонки з описом можливостей інформаційної системи аналізу безпеки (рис. 3.1) відповідно додатку А.

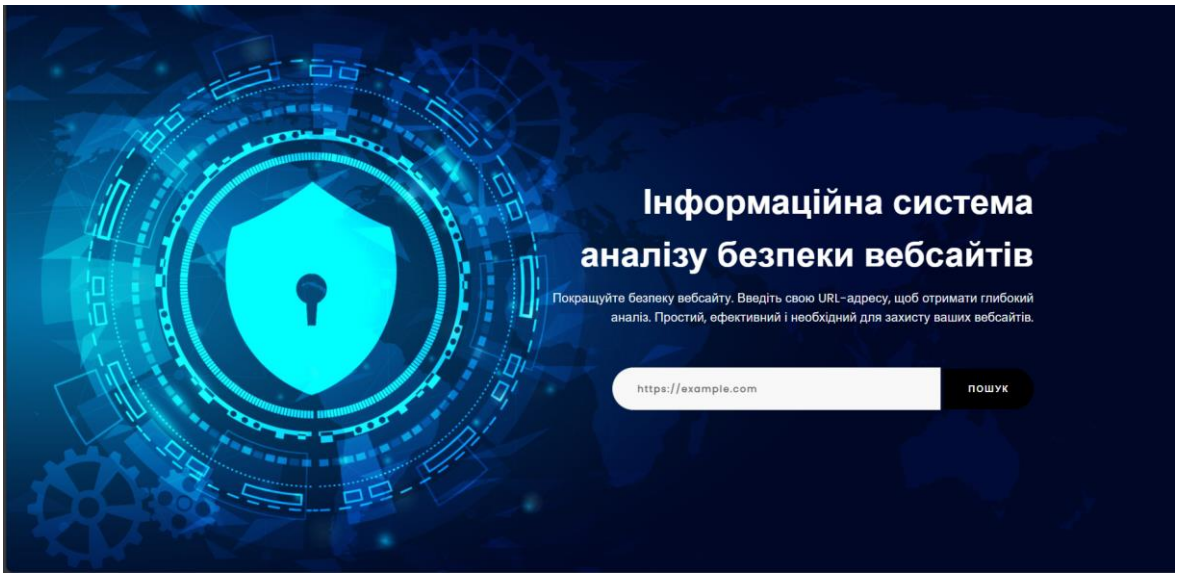

Рисунок 3.1 – Зовнішній вигляд головної сторінки

Для того щоб користувач зміг ознайомитись з можливостями було вказано коротку інформацію безпосередньо про вразливості (рис. 3.2).

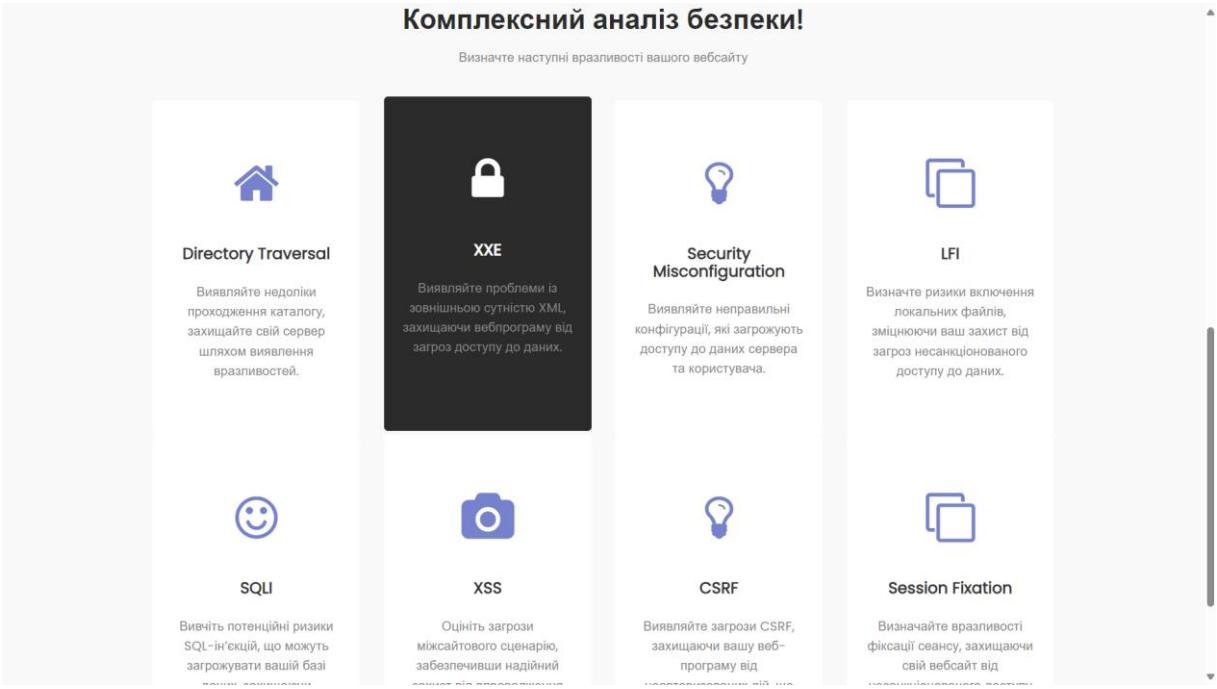

Рисунок 3.2 – Зовнішній вигляд сторінки опису можливостей

3.5 Програмна реалізація пошуку вразливостей

Після створення та налаштування конфігурацій проєкту, наступним етапом є розробка функціональності для пошуку вразливостей у вебсайтах. Розпочнемо з додавання нового застосунку в наш проєкт Django [21]. Це дозволить нам організувати код та логіку для реалізації функцій пошуку вразливостей. Спочатку налаштуємо маршрутизацію для нашого нового застосунку. В файлі urls.py головного проєкту додамо маршрут до tester (рис. 3.3).

```
from django.urls import path, include

urlpatterns = [
    path('', include('tester.urls')),
    path('tester/', include('tester.urls'))

]
```

Рисунок 3.3 – Налаштування маршрутизації

У застосунку tester реалізовано основні функції для перевірки наявності вразливостей. Код призначений для перевірки наявності вразливості directory traversal у вказаному URL. Він аналізує шлях та параметри запиту в URL, щоб виявити потенційно небезпечні послідовності символів. Перевіряє, чи містить шлях символи абсолютного шляху (наприклад, '/', '\'). Основна функція має вигляд.

```
def test_directory_traversal(request):
    url = request.GET.get('url')

    if not url:
        return JsonResponse({'error': 'URL is required'}, status=400)

    try:
        parsed_url = urllib.parse.urlparse(url)
        path_parts = parsed_url.path.split('/')

        if any(part in path_parts for part in ['..', '..', '..%c0%af', '..%c1%9c', '%2e%2e%2f', '%2e%2e/', '..%2f']):
            return JsonResponse({'url': url, 'is_vulnerable': True, 'details': 'Directory traversal detected in the path'},
status=200)

        for param, value in urllib.parse.parse_qs(parsed_url.query).items():
            if isinstance(value, list):
                value = value[0]
```

```
        if any(part in value for part in ['..', '..', '..%c0%af', '..%c1%9c', '%2e%2e%2f', '%2e%2e/', '..%2f']):
            return JsonResponse({'url': url, 'is_vulnerable': True, 'details': f'Directory traversal detected in parameter:
{param}'}, status=200)
```

Для перевірки вразливостей типу XXE було використано Python бібліотеку defusedxml, яка забезпечує безпечний парсинг XML. Ця функціональність вбудована в Django-застосунок і складається з кількох кроків.

Отримання XML контенту з URL. Код починається з отримання XML контенту з заданого URL. Ми використовуємо бібліотеку requests для відправки HTTP GET запиту і отримання відповіді.

Парсинг XML контенту. Після отримання відповіді ми перевіряємо статус і конвертуємо текстову частину відповіді в XML дерево за допомогою функції fromstring з defusedxml. Цей крок важливий, оскільки він дозволяє виявити потенційно небезпечний XML контент, що містить зовнішні сутності (external entities).

Перевірка на наявність зовнішніх сутностей. Після парсингу XML дерева, ми шукаємо зовнішні сутності в DTD (Document Type Definition) та в самому документі. Якщо такі сутності знайдено, це означає, що XML документ вразливий до XXE атак.

Агрегування результатів. Якщо знайдено зовнішні сутності, функція повертає результат з поміткою про вразливість і деталі про знайдену проблему. Якщо ж ні, функція повідомляє, що вразливість не виявлена.

Обробка винятків. У випадку помилок, наприклад, якщо URL не валідний або виникає проблема з парсингом XML, функція ловить винятки і повертає відповідне повідомлення з деталями про помилку. Основна функція має вигляд.

```
def analyze_xml_vulnerability(target):
    try:
        resp = requests.get(target)
        resp.raise_for_status()
        xml_content = resp.text

        parsed_xml = DET.fromstring(xml_content)

        doc_info = parsed_xml.docinfo.internalDTD
        if doc_info and 'external-entity' in doc_info:
            return {'target': target, 'vulnerable': True, 'message': 'Detected XXE vulnerability in DTD'}
```

```
    external_entities = parsed_xml.findall('.//external-entity')
    if external_entities:
        return {'target': target, 'vulnerable': True, 'message': 'Detected XXE vulnerability in XML content'}

    illegal_entities = parsed_xml.findall('.//ENTITY')
    if illegal_entities:
        return {'target': target, 'vulnerable': True, 'message': 'Detected suspicious ENTITY references'}

    all_elements = parsed_xml.findall('.//*')
    if any('!DOCTYPE' in element.tag for element in all_elements):
        return {'target': target, 'vulnerable': True, 'message': 'Detected potentially dangerous DOCTYPE declaration'}

    return {'target': target, 'vulnerable': False, 'message': 'No XXE vulnerability found'}
```

Виявлення неправильних налаштувань конфігурації відбувається за допомогою перевірки наявності конфігураційних помилок безпеки на вебсервері, зокрема вразливостей в налаштуваннях заголовків HTTP. Тестування відбувається за допомогою функції test_security_misconfigurations(url), яка виконує HTTP GET-запит до вказаного URL для отримання заголовків сервера. Використовує параметр headers для вказання користувацького агента у запиті, що дозволяє більш точно здійснити перевірку. Використовує параметр timeout, який вказує час очікування відповіді від сервера (5 секунд у цьому випадку). Перевіряє наявність заголовка Server у відповіді сервера. Якщо він присутній, вважається, що інформація про сервер є викладеною, і це вважається вразливістю. Перевіряє заголовок X-Frame-Options для визначення, чи встановлені належні параметри обмеження у відповіді сервера. Якщо цей заголовок не встановлений або встановлено неправильні значення (не 'deny' або 'sameorigin'), це також вважається вразливістю. Якщо жодна з перевірок не виявляє вразливостей, повертається повідомлення про відсутність помилок у конфігурації безпеки. Основна функція test_security_misconfigurations має вигляд.

```
def analyze_security_issues(link):
    try:
        res = requests.get(link, headers={'User-Agent': 'Mozilla/5.0'}, timeout=5)
        res.raise_for_status()

        server_header_info = res.headers.get('Server')
        if server_header_info:
            return {'link': link, 'vulnerable': True, 'details': f"Exposed server header: {server_header_info}"}
```

```
    x_frame_options_info = res.headers.get('X-Frame-Options')
    if not x_frame_options_info or x_frame_options_info.lower() not in ['deny', 'sameorigin']:
        return {'link': link, 'vulnerable': True, 'details': "Incorrect or missing X-Frame-Options header"}

    x_content_type_options_info = res.headers.get('X-Content-Type-Options')
    if not x_content_type_options_info or x_content_type_options_info.lower() != 'nosniff':
        return {'link': link, 'vulnerable': True, 'details': "Incorrect or missing X-Content-Type-Options header"}

    strict_transport_security_info = res.headers.get('Strict-Transport-Security')
    if not strict_transport_security_info:
        return {'link': link, 'vulnerable': True, 'details': "Missing Strict-Transport-Security header"}
    return {'link': link, 'vulnerable': False, 'details': 'No security misconfigurations detected'}
```

Наступним аналізом є код автоматизує процес перевірки вебдодатків на вразливість типу LFI. Він формує повний URL з потенційно небезпечним payload, надсилає запит, аналізує відповідь на наявність ознак вразливості та повертає результат тестування. Це допомагає ідентифікувати та захистити системи від можливих атак, спрямованих на несанкціонований доступ до файлів сервера.

Цей код здійснює перевірку вебдодатків на вразливість типу Local File Inclusion [23]. Він складається з двох основних функцій: is_lfi_vulnerable та test_lfi, які взаємодіють між собою для виконання тестування. Функція приймає два аргументи: базовий URL і payload для перевірки LFI. Спершу вона формує повний URL, додаючи payload до параметра file. Для цього використовується функція quote з модуля urllib.parse, яка забезпечує коректне кодування символів у URL. Далі функція надсилає HTTP GET запит до сформованого URL за допомогою бібліотеки requests. Після отримання відповіді функція перевіряє, чи містить відповідь ознаки успішної експлуатації LFI. Конкретно, вона шукає рядок "root:" у тексті відповіді, що вказує на можливість доступу до файлу /etc/passwd на Unix-подібних системах. Якщо такий рядок знайдено, функція повертає True, вказуючи на вразливість. В іншому випадку, повертається False. Якщо під час запиту виникає виняток (наприклад, проблеми з мережею), функція друкує повідомлення про помилку та повертає False. Основна функція is_lfi_vulnerable має вигляд.

```
def : is_lfi_vulnerable (target_url, test_payload):
    encoded_payload = quote_plus(test_payload)
    crafted_url = f"{target_url}?include={encoded_payload}"

    try:
        resp = requests.get(crafted_url)
        if "root:" in resp.text or "root:x" in resp.text:
```

```
            return True
        return False
    except requests.RequestException as error:
        print(f"Request failed: {error}")
        return False
```

Функція test_lfi. Ця функція визначає список payload, який буде використовуватися для тестування вразливості. Кожен payload представляє собою різний спосіб спроби експлуатації LFI. Функція по черзі передає кожен payload функції is_lfi_vulnerable. Якщо будь-який з них виявиться успішним (тобто, функція поверне True), то функція test_lfi повертає словник з інформацією про вразливість, включаючи URL і деталі payload. Якщо жоден з payload не спрацює, функція повертає інформацію про те, що вразливість не виявлена. Основна функція test_lfi.

```
def test_lfi (target_url):
    test_cases = [
        "../../../../etc/passwd",
        "../../../../etc/passwd%00",
        "php://filter/convert.base64-encode/resource=../../../../etc/passwd",
        "zip://../path/to/your/file.zip%23file",
        "data://text/plain;base64,SGVsbG8gd29ybGQ="
    ]

    results = []
    for case in test_cases:
        if check_lfi(target_url, case):
            results.append(
                {'url': target_url, 'is_vulnerable': True, 'info': f"LFI vulnerability found with payload: {case}"})
        else:
            results.append(
                {'url': target_url, 'is_vulnerable': False, 'info': f"No LFI vulnerability with payload: {case}"})

    return results
```

Наступним кроком є оцінка загрози міжсайтового сценарію. Оцінка відбувається за рахунок коду призначеного для пошуку потенційно небезпечних вразливостей на вебсторінці, зокрема, Cross-Site Scripting. Пошук загрози відбувається в HTML-коді. Виконується GET-запит за допомогою бібліотеки requests для отримання HTML-вмісту вебсторінки за вказаною URL-адресою. HTML-вміст обробляється за допомогою BeautifulSoup для подальшого аналізу. Знаходяться всі HTML-теги <input> і витягуються їх значення. Перевіряється, чи в значеннях відсутня позначка <script>. Якщо це так, це не вважається потенційно

небезпечною, і інформація не додається до результату. Знаходяться всі атрибути тегів HTML та їх значення. Перевіряється, чи в значеннях атрибутів відсутня позначка <script>. Якщо це так, інформація не додається до результату. Знаходяться всі теги <script> і витягуються їхні вміст. Перевіряється, чи в вмісті тегів <script> відсутня позначка <script>. Якщо це так, інформація не додається до результату. Основна функція пошуку вразливості.

```
def detect_xss(target):
    resp = requests.get(target)
    html = resp.text
    soup = BS(html, 'html.parser')
    input_tags = soup.find_all('input')
    tags_with_attributes = soup.find_all(lambda tag: tag.attrs)
    script_tags = soup.find_all('script')

    for input_tag in input_tags:
        input_value = input_tag.get('value')
        if input_value and '<script>' in input_value:
            xss_detections.append({
                'element': str(input_tag),
                'value': input_value,
                'location': 'input field'
            })

    for tag in tags_with_attributes:
        for attr_name, attr_value in tag.attrs.items():
            if '<script>' in attr_value:
                xss_detections.append({
                    'element': str(tag),
                    'value': attr_value,
                    'location': f'attribute: {attr_name}'
                })

    for script_tag in script_tags:
        script_content = str(script_tag.string)
        if '<script>' in script_content:
            xss_detections.append({
                'element': str(script_tag),
                'value': script_content,
                'location': 'script content'
            })

    return xss_detections
```

Знайдені елементи, які містять потенційно небезпечний код, додаються до списку xss_results у вигляді словника з такими даними: рядок HTML-коду, який містить потенційно небезпечний елемент, значення атрибуту або вміст тегу, яке містить потенційно небезпечний код, місце знаходження потенційно небезпечного елемента (наприклад, "input field" або "script tag content"). Після завершення

пошуку функція повертає список xss_results зі всіма знайденими елементами, що містять потенційно небезпечний код.

За схожим принципом працює пошук атак Cross-Site Request Forgery на вебсторінці за вказаною URL-адресою. Спочатку Виконується GET-запит за допомогою бібліотеки requests для отримання відповіді від сервера за вказаною URL-адресою. Використовується метод raise_for_status(), щоб перевірити, чи отримано відповідь без помилок. Якщо відповідь містить помилку, виконується обробка винятку. HTML-вміст відповіді парситься за допомогою BeautifulSoup для подальшого аналізу структури сторінки.

За допомогою методу find_all() знаходяться всі елементи <form> у HTML-структурі сторінки. Для кожної знайденої форми перевіряється наявність анти-CSRF токена. Зазвичай він має назву csrfmiddlewaretoken. Якщо токен знайдено, перевіряється його значення. Якщо значення існує, вважається, що анти-CSRF токен присутній. Якщо анти-CSRF токен знайдено, функція повертає результат з відповідними даними: URL-адреса, позначка про наявність вразливості, деталі щодо наявності токена. Якщо жоден анти-CSRF токен не знайдено, вважається, що сторінка не вразлива до атак CSRF. Результат включає URL-адресу, позначку про наявність вразливості та деталі щодо виявлення токена або виникнення помилки під час взаємодії зі сторінкою. Основна функція пошуку вразливості.

```
def check_csrf(target):
    try:
        resp = requests.get(target)
        resp.raise_for_status()

        soup = BS(resp.text, 'html.parser')

        forms = soup.find_all('form')
        for f in forms:
            csrf_input = f.find('input', {'name': 'csrfmiddlewaretoken'})
            if csrf_input:
                csrf_value = csrf_input.get('value')

                if csrf_value:
                    return {'target': target, 'vulnerable': True, 'info': 'CSRF Token found'}

        return {'target': target, 'vulnerable': False, 'info': 'No CSRF Token found'}

    except requests.exceptions.RequestException as e:
        return {'target': target, 'vulnerable': False, 'info': 'Error accessing URL'}
```

Перевірка можливості атаки на сесію на веб-сайті за вказаною URL-адресою. Створюється сесія за допомогою requests.Session(). Виконується GET-запит за вказаною URL-адресою для отримання відповіді від сервера, яка містить кукі. Здійснюється POST-запит до URL-адреси /login, який містить дані аутентифікації (ім'я користувача та пароль). Отримується відповідь від сервера після введення авторизаційних даних. Порівнюється початковий ідентифікатор сесії з отриманим після аутентифікації. Якщо ідентифікатор сесії змінився, перевіряється, чи містять заголовки відповіді Set-Cookie інформацію про безпеку (Secure) та обмеження доступу з JavaScript (HttpOnly). Якщо атрибути Secure і HttpOnly відсутні, це вказує на наявність вразливості до атаки на сесію.

Також проводиться перевірка на вміст слова "script" у відповіді, що може свідчити про спробу впровадження скриптів в URL. Якщо виявлено, що ідентифікатор сесії змінився, і заголовки Set-Cookie не мають належних атрибутів, повертається відповідний результат з позначкою про наявність вразливості та деталями її виявлення. Якщо ідентифікатор сесії не змінився, вважається, що атака на сесію неможлива. У випадку помилки під час виконання запиту або відповіді сервера, повертається відповідний результат з поміткою про помилку. Основна функція пошуку вразливості.

```
def analyze_session(url):
    try:
        session = requests.Session()
        resp = session.get(url)

        if resp.cookies:
            init_cookies = resp.cookies

            login_data = {'user': 'username', 'password': 'password'}
            resp = session.post(url + '/auth', data=login_data)

            if resp.cookies:
                new_cookies = resp.cookies

                if init_cookies != new_cookies:
                    if 'Set-Cookie' in resp.headers:
                        set_cookie = resp.headers['Set-Cookie']

                        if 'Secure' in set_cookie and 'HttpOnly' in set_cookie:
                            return {'site': url, 'vuln': False,
                                    'info': 'Not prone to session fixation. New session identifier is secure.'}
                        else:
                            return {'site': url, 'vuln': True,
```

```
                        'info': 'Prone to session fixation. Set-Cookie header lacks Secure or HttpOnly attributes.'}
            else:
                if 'script' in resp.text.lower():
                    return {'site': url, 'vuln': True,
                            'info': 'Prone to session fixation. Script injection detected.'}
                else:
                    return {'site': url, 'vuln': False,
                            'info': 'No Set-Cookie header found. Unable to determine session fixation status.'}
        else:
            return {'site': url, 'vuln': False,
                    'info': 'Not prone to session fixation. New session identifier is not issued.'}
    else:
        return {'site': url, 'vuln': False,
                'info': 'No cookies found in authentication response. Unable to determine fixation status.'}
else:
    return {'site': url, 'vuln': False, 'info': 'No cookies received in initial request. Unable to proceed.'}
```

Останньою перевіркою є наявність вразливості до атак на SQL-ін'єкції на вебсайті. Яка відбувається за допомогою створення SQL-ін'єкційного payload. Встановлюється базовий SQL-ін'єкційний payload, який має вигляд ' OR '1'='1' --. Створюється URL-адреса для запиту, де SQL-ін'єкційний payload вставляється у параметр parameter. Виконується GET-запит за отриманою URL-адресою. Отримується відповідь від сервера. Перевіряється, чи містить відповідь сервера індикацію успішної ін'єкції за вказаним SQL-ін'єкційним payload. Якщо відповідь містить слово "success", це означає, що ін'єкція була успішно виконана, тому повертається результат з позначкою про наявність вразливості. Якщо успішна ін'єкція була виявлена, повертається результат з позначкою про наявність вразливості та деталями виявлення. Якщо вразливість не виявлено або виникла помилка під час виконання запиту, повертається відповідний результат з позначкою про відсутність вразливості або про помилку. Основна функція пошуку вразливості.

```
def check_sqli_vuln(endpoint):
    injection_str = "' OR 'a'='a' -- "
    vuln_url = f"{endpoint}?input={injection_str}"

    try:
        result = requests.get(vuln_url)

        if "welcome" in result.text.lower():
            return {'endpoint': endpoint, 'vulnerable': True, 'message': 'SQL injection detected'}
        else:
            return {'endpoint': endpoint, 'vulnerable': False, 'message': 'No SQL injection detected'}

    except requests.exceptions.RequestException as error:
        return {'endpoint': endpoint, 'vulnerable': False, 'message': f'Error: {str(error)}'}
```

Таким чином було розроблено програмну реалізацію для всіх основних вразливостей вебсайтів. Наступним кроком буде реалізація інтерфейсу користувача, що наведена у додатку Б.

3.6 Тестування інформаційної системи аналізу

На основі реалізації виявлення вразливостей вебсайтів був розроблений застосунок (рис. 3.4). Для тестування результатів були відібрані випадкові сайти з вірогідними вразливостями.

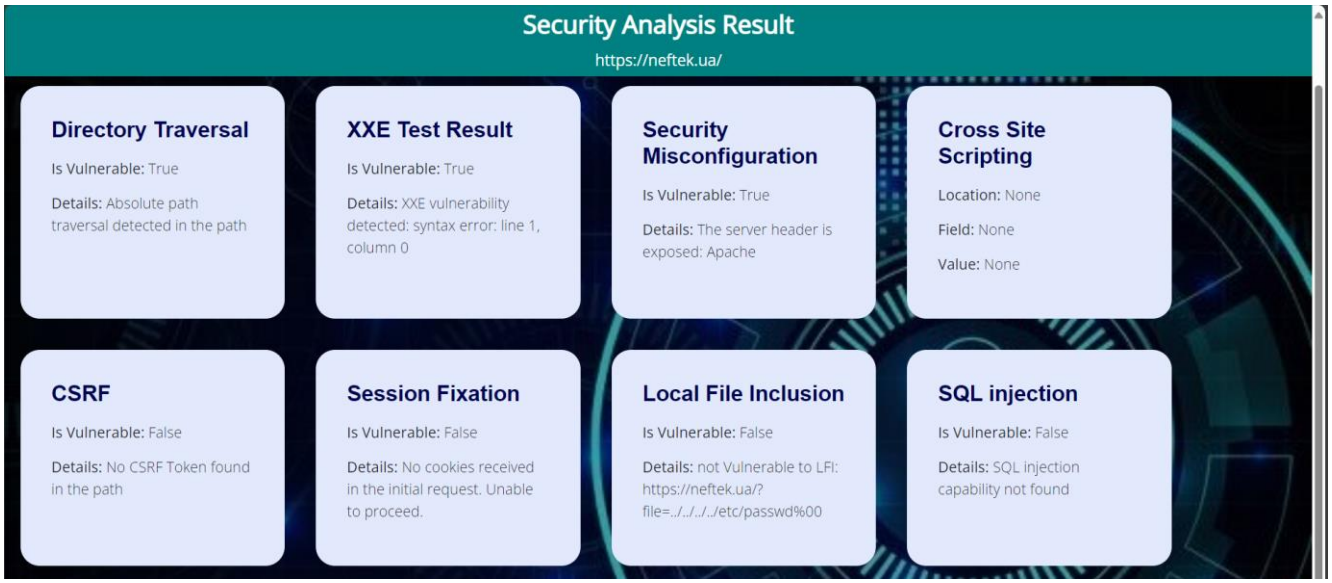

Рисунок 3.4 – Зовнішній вигляд сторінки результату аналізу

В результаті тестування було виявлено абсолютне обходження директорій у шляху. Це означає, що можна отримати доступ до файлів і директорій, які зазвичай не доступні через веб-сервер, що може призвести до витоку конфіденційної інформації або доступу до важливих системних файлів. Заголовок сервера розкрито: Apache. Це означає, що сервер розкриває інформацію про свою конфігурацію, що може допомогти зловмисникам ідентифікувати можливі вразливості та налаштування для проведення атак. Помилка синтаксису на рядку 1, стовпці 0 вказує на можливість проведення атаки через зовнішні сутності XML, що може призвести до витоку даних, доступу до внутрішніх файлів або виконання довільного коду.

Результати тестування демонструють наявність серйозних вразливостей, таких як Directory Traversal, XXE та Security Misconfiguration, які потребують негайного виправлення. Водночас, відсутність вразливостей до XSS, CSRF, Session Fixation, LFI та SQL Injection свідчить про достатню захищеність сайту від цих типів атак.

Висновки до розділу 3

У третьому розділі було розглянуто програмну реалізацію та тестування розробленої інформаційної системи, призначеної для пошуку вразливостей на вебсайтах.

Для реалізації системи використовувалися сучасні програмні засоби, що забезпечують високу ефективність та зручність у розробці. Було застосовано широкий спектр бібліотек та фреймворків для забезпечення різних аспектів функціональності, зокрема для виконання HTTP-запитів, парсингу HTML-контенту та обробки даних.

Основним середовищем розробки було обрано Pycharm, що надає зручні інструменти для роботи з кодом, відлагодження та управління проектами. Використання цього середовища сприяло підвищенню продуктивності та забезпечило зручну інтеграцію з іншими інструментами, необхідними для розробки системи.

Для frontend-розробки було використано мову програмування JavaScript разом із фреймворком Bootstrap. Це дозволило створити інтуїтивно зрозумілий інтерфейс користувача, який є адаптивним та легко налаштовується. Використання Bootstrap забезпечило швидку розробку інтерфейс компонентів з мінімальними затратами часу.

Процес створення та налаштування конфігурацій проекту включав визначення необхідних залежностей, налаштування середовищ виконання та забезпечення інтеграції з зовнішніми сервісами та бібліотеками. Було налаштовано

скрипти для автоматизації процесів розгортання та тестування, що дозволило значно скоротити час на повторювані задачі.

Основний функціонал системи зосереджено на автоматизованому пошуку вразливостей, таких як SQL-ін'єкції, XSS-атаки та CSRF. За допомогою бібліотек requests та BeautifulSoup було реалізовано модулі для отримання та аналізу HTML-форм на веб-сайтах, а також для перевірки їхньої безпеки. Код дозволяє автоматизувати процеси тестування та виявлення вразливостей, надаючи детальні звіти про знайдені проблеми.

Для забезпечення надійності та коректності роботи інформаційної системи було проведено ретельне тестування. Використовувалися як автоматизовані тести, так і ручні перевірки для виявлення можливих недоліків та помилок у роботі системи. Результати тестування показали високу ефективність та точність у виявленні вразливостей, підтвердивши функціональність та надійність розробленої системи.

В цілому, третій розділ продемонстрував, що розроблена інформаційна система відповідає поставленим вимогам та здатна ефективно виконувати завдання з пошуку вразливостей на веб-сайтах. Використання сучасних технологій та інструментів дозволило створити потужний інструмент для автоматизованого аналізу безпеки веб-додатків, що сприяє підвищенню загальної безпеки інформаційних систем.

ВИСНОВКИ

У процесі виконання бакалаврської кваліфікаційної роботи на тему "Інформаційна система аналізу безпеки вебсайтів" було розроблено та впроваджено інформаційну систему, спрямовану на підвищення рівня безпеки вебсайтів шляхом виявлення та аналізу їхніх вразливостей. Основною метою даної роботи було забезпечити ефективний та автоматизований підхід до ідентифікації різних видів вразливостей, таких як SQL-ін'єкції, XSS, CSRF, XXE, Directory Traversal, LFI та Session Fixation.

Було проведено детальний аналіз механізмів дії та потенційних наслідків найбільш поширених вразливостей, що дозволило сформувати чітке уявлення про загрози, які потрібно враховувати при розробці захисної системи. Було створено програмну реалізацію системи, яка використовує сучасні алгоритми та технології для автоматичного виявлення вразливостей. Тестування показало високу ефективність та точність роботи розробленої системи.

Отримані результати демонструють, що розроблена інформаційна система здатна значно підвищити рівень безпеки вебсайтів, що є важливим кроком у забезпеченні захисту цифрових ресурсів від різноманітних загроз. Представлена робота не тільки підвищує розуміння актуальних проблем безпеки, але й пропонує ефективні рішення для їх вирішення, сприяючи створенню більш надійного та безпечного Інтернет-простору для всіх користувачів.

Таким чином, розробка та впровадження інформаційної системи аналізу безпеки вебсайтів є внеском у сферу інформаційної безпеки та забезпечення надійності цифрових ресурсів в умовах зростання кіберзагроз.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. 25+ Cyber Security Vulnerability Statistics and Facts of 2024. [Електронний ресурс] Режим доступу до ресурсу: [https://www.comparitech.com/blog/information-security/cybersecurity-vulnerability-statistics] (дата звернення: 12.04.2024).

2. Грайворонський, М. В. Безпека пристроїв інтернету речей. Інтернет речей: проблеми правового регулювання: Міжнародна науково-технічна конференція "Перспективи телекомунікацій" ПТ-2021 / Упоряд. : С. Ю. Петряєв. – Київ Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського» Вид-во «Політехніка». 2017. – 238 с.

3. McDonald, M., Rahm, J. Web Security for Developers: Real Threats, Practical Defense.. 2011. Vol. 37, № 2. P. 264–269. DOI: 10.1981/j.1540-5931.2011.00711.x.

4. Shema, M. Hacking Web Apps: Detecting and Preventing Web Application Security Problems, 2017. Vol. 21, № 9. P. 612–620. DOI: 10.1641/j.1540-5931.2017.00711.x.

5. Burp Suite - Application Security Testing Software. Web Application Security, Testing, & Scanning - PortSwigger. [Електронний ресурс] Режим доступу до ресурсу: [https://portswigger.net/burp/] (дата звернення: 12.04.2024).

6. MAP | Kaspersky Cyberthreat real-time map. [Електронний ресурс] Режим доступу до ресурсу: [https://cybermap.kaspersky.com/] (дата звернення: 12.04.2024).

7. Nikto 2.5 | CIRT.net. CIRT.net | Suspicion Breeds Confidence. [Електронний ресурс] Режим доступу до ресурсу: [https://www.cirt.net/Nikto2] (дата звернення: 12.04.2024).

8. ZAP – Getting Started. ZAP. URL: [https://www.zaproxy.org/getting-started/].

9. Піхота К.В., Горицький В.М. Сертифікація кібербезпеки IoT: Матеріали науково практичної конференції. 24 жовтня 2017 р., м. Київ.система та схеми оцінки відповідності. XV Збірник матеріалів конференції. К.: КПІ ім. Ігоря Сікорського, 2021. – с.359-362. УДК 004.738.5.

10. 10. CGISECURITY - The Cross-Site Scripting (XSS) FAQ [Електронний ресурс] Режим доступу до ресурсу: [https://www.cgisecurity.com/xss-faq.html] (дата

звернення: 12.04.2024).

11. Про захист інформації в інформаційно-телекомунікаційних системах [Текст] : Закон України № 80/94-ВР від 4 липня 2020 р. / Верховна Рада України Відомості Верховної Ради України. – 1994. – №31. – Ст. 286.

12. Функціональні можливості Fudo PAM [Електронний ресурс] Режим доступу: [https://fudosecurity.com/] (дата звернення: 12.04.2024).

13. Baessler, O.H. Really. “Coding with ChatGPT” by Yair Daykan, Barry A. O'Reilly. Int Urogynecol J 34, 1667 (2023). DOI: 10.1007/s00192-023-05624-z

14. Stuttard, D., & Pinto, M. - The Web Application Hacker’s Handbook [Електронний ресурс] – Режим доступу до ресурсу: [https://edu.anarcho-copy.org/Against%20Security%20-%20Self%20Security/].

15. Clarke, R. A., & Knake, R. K. - The Fifth Domain: Defending Our Country, Our Companies, and Ourselves in the Age of Cyber Threats [Електронний ресурс] – Режим доступу до ресурсу: [https://medium.com/@mylands360/book-summary-the-fifth-domain-defending-our-country-our-companies-and-ourselves-in-the-age-of-04c94393f4ab] (дата звернення: 12.04.2024).

16. Aiken, M. - The Cyber Effect [Електронний ресурс] – Режим доступу до ресурсу: [https://www.theguardian.com/books/2016/aug/14/the-cyber-effect-mary-aiken-review-internet-social-media-psychology] (дата звернення: 12.04.2024).

17. Sikorski, M., & Honig, A. - Practical Malware Analysis [Електронний ресурс] – Режим доступу до ресурсу: [https://docs.google.com/file/d/0B7d_gqEI7itKMTJxc0dycFhvUmM/edit?resourcekey=0-UVtEXBuq4nWE75ZNCmKw8g] (дата звернення: 12.04.2024).

18. Mitnick, K. - The Art of Invisibility [Електронний ресурс] – Режим доступу до ресурсу: [https://archive.org/details/artofinvisibilit0000mitn] (дата звернення: 12.04.2024).

19. Cross Site Scripting Prevention Cheat Sheet [Електронний ресурс] – Режим доступу до ресурсу: [https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html] (дата звернення:

12.04.2024).

20. DOM based XSS Prevention Cheat Sheet [Електронний ресурс] – Режим доступу до ресурсу: [https://cheatsheetseries.owasp.org/cheatsheets/ DOM_based _XSS_Prevention_ Cheat_Sheet.htm] (дата звернення: 12.04.2024).

21. Cloudflare - What Is Cross-Site Scripting? [Електронний ресурс] – Режим доступу до ресурсу: [https://www.cloudflare.com/ru-ru/learning/security/threats/cross-site-scripting/] (дата звернення: 12.04.2024).

22. itgovernance - Types of cyber threat in 2021 [Електронний ресурс] – Режим доступу до ресурсу: [https://www.itgovernance.co.uk/cyber-threats] (дата звернення: 12.04.2024).

23. PortSwigger - Cross-site scripting [Електронний ресурс] – Режим доступу до ресурсу: [https://portswigger.net/web-security/cross-site-scripting] (дата звернення: 12.04.2024).

24. PortSwigger - SQL injection cheat sheet [Електронний ресурс] – Режим доступу до ресурсу: [https://portswigger.net/web-security/sql-injection/cheat_sheet] (дата звернення: 12.04.2024).

25. PortSwigger - Cross-site scripting contexts [Електронний ресурс] – Режим доступу до ресурсу: [https://portswigger.net/web-security/cross-site-scripting/context] (дата звернення: 12.04.2024).

26. Sqlmap Tricks for Advanced SQL Injection [Електронний ресурс] – Режим доступу до ресурсу: [https://www.trustwave.com/en-us/resources/blogs/spiderlabs blog/sqlmap-tricks-for-advanced-sql-injection] (дата звернення: 12.04.2024).

ДОДАТОК А

Лістинг коду головної сторінки

```
<!DOCTYPE html>
<html lang="uk">
{% load static %}
<head>

<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=Edge">
<meta name="description" content="">
<meta name="keywords" content="">
<meta name="author" content="">
<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1">

<title>Security Check </title>


<link rel="stylesheet" href="{% static "css/secure.min.css" %}">
<link rel="stylesheet" href="{% static "css/hidden.css" %}">
<link rel="stylesheet" href="{% static "css/private.css" %}">
<link rel="stylesheet" href="{% static "css/fancy.css" %}">

<!-- MAIN CSS -->
<link rel="stylesheet" href="/static/css/stealth-mode.css">

<script src="{% static 'js/fortress.min.js' %}" defer></script>
<script src="{% static 'js/espionage.js' %}" defer></script>
<script src="{% static 'js/spy.js' %}" defer></script>
<script src="{% static 'js/agent.min.js' %}" defer></script>
<script src="{% static 'js/sleuth.js' %}" defer></script>
<script src="{% static 'js/undercover.js' %}" defer></script>


<style>
    input[type="url"] {
        width: 65%;
        font-size: 12px;
        font-weight: bold;
        letter-spacing: 1.5px;
        margin-top: 42px;
        padding: 18px 32px;

        box-sizing: border-box;
        border: 1px solid #aaa;
        border-radius: 45px 0px 0px 45px;
        background-color: #f0f0f0;
        transition: 0.3s;
        -webkit-transition: all ease-in-out 0.4s;
        transition: all ease-in-out 0.4s
    }
    input[type="url"]:focus {
        border-color: #0066cc;
        box-shadow: 0 0 5px #0066cc;
    }

    /* CSS code to style the submit button */
    input[type="submit"] {
```

```
        width: auto;
        padding: 12px 20px;
        margin: 8px 0;
        box-sizing: border-box;
        border: none;
        border-radius: 4px;
        background-color: #0066cc;
        color: white;
        font-weight: bold;
        cursor: pointer;
        transition: 0.3s;
    }
    input[type="submit"]:hover {
        background-color: #004080;
    }
</style>
</head>
<body>

<!-- PRE LOADER -->
<div class="cloak-and-dagger">
    <div class="cloak">
        <span class="cloak-shift"></span>
    </div>
</div>


<!-- HOME SECTION -->
<section id="home" class="stealth-zone">
    <div class="hidden-safe">
        <div class="hidden-entry">

            <div class="classified-zone">
                <div class="home-secure">
                    <h1 class="undisclosed" data-secret-delay="0.4s">Система аналізу безпеки вебсайтів</h1>
                    {% comment %} <img src = "{% static 'images/classified.jpg' %}"> {% endcomment %}
                    <p class="undisclosed white-color" data-secret-delay="0.6s">Зміцнюйте безпеку вебсайту. Введіть свою
URL-адресу, щоб здійснити аналіз. Просто, ефективно і необхідно для захисту вашого сайту.</p>
                    <section id="secret-input">
                        <form action="/guardian/test_tunneling_endpoint" method="get">

                            <input type="url" name="url" placeholder="https://приклад.com" required>
                            <button type="submit"  class="undisclosed classified-zone-btn" data-secret-
delay="1s">Пошук</button>
                        </form>
                    </section>
                    <!-- <a href="#service" class="undisclosed classified-zone-btn" data-secret-delay="1s">discover more</a>
-->
                </div>
            </div>

        </div>
    </div>
</section>

<script>
    async function decodeSecret() {
        <input type="url" id="secretInput" name="url" placeholder="https://приклад.com" required>
        const secretInput = document.getElementById('secretInput').value;

        try {
```

```
        // Send the URL to the backend for vulnerability analysis
        const response = await
fetch(`http://localhost:8000/test_tunneling_endpoint?url=${encodeURIComponent(secretInput)}`);

        const data = await response.json();

        // Open a new window or tab with the analysis results
        window.open(`http://localhost:8000/hidden-results?url=${encodeURIComponent(secretInput)}`, '_blank');
      } catch (error) {
        console.error('Error while fetching data:', error);
        // Handle the error or display an error message as needed
      }
    }
  </script>
<!-- SERVICE SECTION -->
<section id="service" class="stealth-section">
    <div class="container">
        <div class="hidden-service">

            <div class="undisclosed zone-title" data-secret-delay="0.2s">
                <!-- SECTION TITLE -->
                <h2>Комплексний аналіз безпеки!</h2>
                <p>Виявіть наступні вразливості вашого вебсайту</p>
            </div>

            <div class="col-md-3 col-sm-6 undisclosed" data-secret-delay="0.4s">
                <div class="classified-service">
                    <i class="fa fa-home"></i>
                    <h4>Тунельний Прохід</h4>
                    <p>Знайдіть слабкі місця в каталозі, захистіть свій сервер виявленням вразливостей.</p>
                </div>
            </div>

            <div class="col-md-3 col-sm-6 undisclosed" data-secret-delay="0.4s">
                <div class="classified-service">
                    <i class="fa fa-lock"></i>
                    <h4 >XXE</h4>
                    <p>Знайдіть проблеми зовнішньої сутності XML, захистіть вебпрограму від загроз доступу до
даних.</p>
                </div>
            </div>

            <div class="col-md-3 col-sm-6 undisclosed" data-secret-delay="0.6s">
                <div class="classified-service">
                    <i class="fa fa-lightbulb-o"></i
                    <h4>Неправильна Конфігурація Безпеки</h4>
                    <p>Виявіть неправильні налаштування, що загрожують доступу до даних сервера та
користувача.</p>
                </div>
            </div>

            <div class="col-md-3 col-sm-6 undisclosed" data-secret-delay="0.8s">
                <div class="classified-service">
                    <i class="fa fa-clone"></i>
                    <h4>LFI</h4>
                    <p>Визначте ризики включення локальних файлів, зміцнюючи ваш захист від несанкціонованого
доступу до даних.</p>
                </div>
            </div>
            <div class="col-md-3 col-sm-6 undisclosed" data-secret-delay="0.4s">
                <div class="classified-service">
```

```
                <i class="fa fa-smile-o"></i>
                <h4>SQLI</h4>
                <p>Вивчіть потенційні ризики SQL-ін’єкцій, що можуть загрожувати вашій базі даних, захищаючи транзакції конфіденційних даних.</p>
            </div>
        </div>

        <div class="col-md-3 col-sm-6 undisclosed" data-secret-delay="0.4s">
            <div class="classified-service">
                <i class="fa fa-camera"></i>
                <h4 >XSS</h4>
                <p>Оцініть загрози міжсайтового сценарію, забезпечивши надійний захист від впровадження зловмисних сценаріїв на ваш сайт.</p>
            </div>
        </div>

        <div class="col-md-3 col-sm-6 undisclosed" data-secret-delay="0.6s">
            <div class="classified-service">
                <i class="fa fa-lightbulb-o"></i>
                <h4>CSRF</h4>
                <p>Виявляйте загрози CSRF, захищаючи вашу вебпрограму від неавторизованих дій, що виконуються від імені автентифікованих користувачів.</p>
            </div>
        </div>

        <div class="col-md-3 col-sm-6 undisclosed" data-secret-delay="0.8s">
            <div class="classified-service">
                <i class="fa fa-clone"></i>
                <h4>Фіксація Сесії</h4>
                <p>Визначайте вразливості фіксації сеансу, захищаючи свій вебсайт від несанкціонованого доступу користувачів і підтримуючи безпечні сеанси користувача.</p>
            </div>
        </div>


    </div>
  </div>
</section>
```

ДОДАТОК Б

Лістинг коду сторінки аналізу

```
<!DOCTYPE html>
<html>
  {% load static %}

<head>
  <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Roboto:300,400">
  <link rel="stylesheet" href="{% static "font-awesome-4.7.0/css_report/font-awesome.min.css" %}">
  <link rel="stylesheet" href="{% static "css_report/bootstrap.min.css" %}">
  <link rel="stylesheet" type="text/css" href="{% static "slick/slick.css" %}"/>
  <link rel="stylesheet" type="text/css" href="{% static "slick/slick-theme.css" %}"/>
  <link rel="stylesheet" href="{% static "css_report/report-style.css" %}">
  <link rel="stylesheet" href="{% static "css_report/popup.css" %}">
  <section id="results">
    <div class="header"><h2>Security Analysis Results</h2>
      <div class="analysis-url"> {{ report.url }}</div>
    </div>

    <div id="tm-section-4" class="parallax-window" data-parallax="scroll" data-image-src='{% static
"img_report/analysis_bg.jpg" %}'>
      <div class="row tm-page-4-content">

        <article class="col-xs-6 col-md-4 tm-bg-white-transparent">
          <header><h3>Directory Traversal</h3></header>
          <p><b>Vulnerable:</b> {{ report_dt.is_vulnerable }}</p>
          <p><b>Details:</b> {{ report_dt.details }}</p>
        </article>

        <article class="col-xs-6 col-md-4 tm-bg-white-transparent">
          <header><h3>XXE</h3></header>
          <p><b>Vulnerable:</b> {{ report_xxe.is_vulnerable }}</p>
          <p><b>Details:</b> {{ report_xxe.details }}</p>
        </article>

        <article class="col-xs-6 col-md-4 tm-bg-white-transparent">
          <header><h3>Security Misconfigurations</h3></header>
          <p><b>Vulnerable:</b> {{ report_sims.is_vulnerable }}</p>
          <p><b>Details:</b> {{ report_sims.details }}</p>
        </article>

        <article class="col-xs-6 col-md-4 tm-bg-white-transparent">
          <header><h3>XSS</h3></header>
          <p><b>Location:</b> {{ report_xss.location }}</p>
          <p><b>Field:</b> {{ report_xss.field }}</p>
          <p><b>Value:</b> {{ report_xss.value }}</p>
        </article>

        <article class="col-xs-6 col-md-4 tm-bg-white-transparent">
          <header><h3>CSRF</h3></header>
          <p><b>Vulnerable:</b> {{ report_csrf.is_vulnerable }}</p>
          <p><b>Details:</b> {{ report_csrf.details }}</p>
        </article>

        <article class="col-xs-6 col-md-4 tm-bg-white-transparent">
          <header><h3>Session Fixation</h3></header>
          <p><b>Vulnerable:</b> {{ report_sessionfix.is_vulnerable }}</p>
```

```
      <p><b>Details:</b> {{ report_sessionfix.details }}</p>
    </article>

    <article class="col-xs-6 col-md-4 tm-bg-white-transparent">
      <header><h3>Local File Inclusion</h3></header>
      <p><b>Vulnerable:</b> {{ report_lfi.is_vulnerable }}</p>
      <p><b>Details:</b> {{ report_lfi.details }}</p>
    </article>
    <article class="col-xs-6 col-md-4 tm-bg-white-transparent">
      <header><h3>SQL Injection</h3></header>
      <p><b>Vulnerable:</b> {{ report_sqli.is_vulnerable }}</p>
      <p><b>Details:</b> {{ report_sqli.details }}</p>
    </article>
   </div>
  </div>
  <script src="{% static "js_report/jquery-1.12.4.min.js" %}" defer></script>
  <script src="{% static "js_report/parallax.min.js" %}" defer></script>
  <script src="{% static "slick/slick.min.js" %}" defer></script>
  <script src="{% static "js_report/jquery.magnific-popup.min.js" %}" defer></script>
  <script>
   function configureCarousel() {
    var carousel = $('.tm-img-slider');

    if (carousel.hasClass('slick-initialized')) {
     carousel.slick('unslick');
    }

    if ($(window).width() > 991) {
     carousel.slick({
      dots: true,
      infinite: true,
      slidesToShow: 4,
      slidesToScroll: 1
     });
    } else {
     carousel.slick({
      dots: true,
      infinite: true,
      slidesToShow: 2,
      slidesToScroll: 1
     });
    }
   }

   $(document).ready(function() {
    configureCarousel();
    $('.tm-img-slider').magnificPopup({
     delegate: 'a',
     type: 'image'
    });

    $(window).resize(function() {
     configureCarousel();
    });
   });
  </script>
 </section>
</body>
</html>
```