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

Sariha Sanjeena
Dept. of Computer Science and Engineering
Brac University
Dhaka, Bangladesh
sariha.sanjeena@g.bracu.ac.bd

Dip Gourab Isaac Gomes
Dept. of Computer Science and Engineering
Brac University
Dhaka, Bangladesh
dip.gourab.isaacgomes@g.bracu.ac.bd

Sanjida Rahman
Dept. of Computer Science and Engineering
Brac University
Dhaka, Bangladesh
sanjida.rahman@g.bracu.ac.bd

Mahdi Tazwar
Dept. of Computer Science and Engineering
Brac University
Dhaka, Bangladesh
mahdi.tazwar@g.bracu.ac.bd

Asif Arman Rafsan
Dept. of Computer Science and Engineering
Brac University
Dhaka, Bangladesh
asif.arman.rafsan@g.bracu.ac.bd

Dr. Muhammad Iqbal Hossain
Dept. of Computer Science and Engineering
Brac University
Dhaka, Bangladesh
iqbal.hossain@bracu.ac.bd

Md Faisal Ahmed
Dept. of Computer Science and Engineering
Brac University
Dhaka, Bangladesh
faisal.ahmed@bracu.ac.bd

***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. 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. To address these challenges, 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**

I. Introduction

In the current digital world, the use of web applications have become essential to the operations of governments, organizations and individuals. However, there is a higher chance of cyberattacks because of this increased dependence on web applications. Despite these widely recognized security concerns, many web applications lack the required security, thereby making them an easy target for black hat attacks [6]. 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 [7]. However, traditional penetration testing requires a huge amount of time [8]. 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.

A. Research Problem

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 [8]. Besides, hiring human testers is quite a costly process [9], which may be a hindering factor for many low funded organizations operating in cyberspace, eventually leading to the existence of untested insecure web applications. 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.

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?***

B. Research Objective

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. The objectives of this research are:

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.

II. Literature Review

The primary idea proposed by Happe and Cito [1] was to utilize Large Language Models (LLMs) to elevate and empower human penetration testers. The paper presents a compelling concept of integrating and augmenting LLMs in the exploitation process of vulnerable systems. Similarly, Shahid et al. [2] conducted an extensive comparative analysis of web application scanners, while Albahar et al. focused on showcasing the applied and observed comparison of different web application vulnerability detection penetration testing tools, based on proper standards and techniques. This comparison is performed based on an evaluation score generated by an enhanced benchmarking framework augmented with the latest research. In another paper, Abdulghaffar et al. [3] proposed an integrated approach that combines multiple web application vulnerability scanners to enhance vulnerability detection capabilities and mitigate the insufficiency of traditional manual testing methods. This study [3] highlights that their integrated approach improved the precision and recall scores compared to individual WAVS, providing more comprehensive and accurate vulnerability scan results. For CVSS metric determination, Costa et al. aimed to address existing challenges by leveraging Natural Language Processing techniques to predict CVSS metrics from vulnerability descriptions. The paper concludes that DistilBERT, when combined with lemmatization and a 5000-word vocabulary expansion, provides an improved model with extensive accuracy. Similarly, Shi et al. [4] proposed a method based on a pre-trained XLNet model to reduce the time required for assessing vulnerability severity using the CVSS metric. The accuracy comparison with three other pre-trained models demonstrates that the fine-tuned XLNet model significantly enhances the prediction of vulnerability metric values and manages data scarcity challenges more effectively than other deep learning and machine learning models. Lastly, for effective reporting, Alghamdi [5] assessed the methodology used in creating penetration testing reports, focusing on generating detailed, well-rounded, and effective test documentation. The study is impactful in helping identify the strengths and weaknesses of existing reports, uncovering issues such as low-quality reporting, lack of critical information, and missing components necessary to produce a comprehensive and well-structured penetration test report.

III. Methodology

The proposed framework in Figure 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 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.

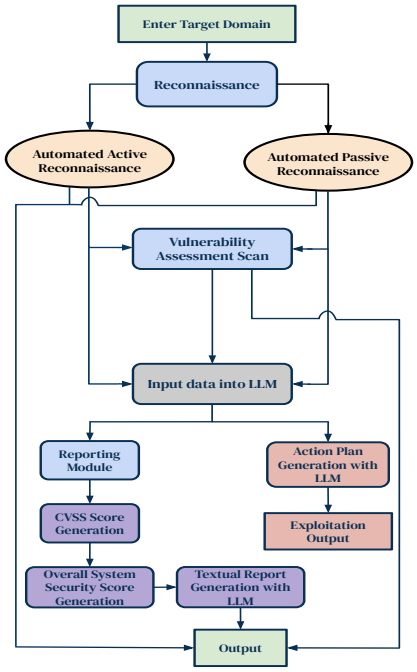


Fig. 1. The flowchart of the proposed Penetration Testing framework

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.

A. Flow of the Penetration Testing Process

1) Reconnaissance: Reconnaissance is the initial step of penetration testing and it 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 which help understand the web application's security status, as well as provides substantial help for the next phase of vulnerability assessment.

The diagram illustrated in Figure 2 provides a clear, step-by-step overview of the reconnaissance module. Firstly, to search for SSL/TLS certificates issued under Certificate Transparency (CT) logs [10], Crt.sh web interface can be utilized. It allows one to discover the subdomains associated with a given domain since these certificates are publicly logged for transparency. Similarly, subdomains can also be discovered and verified 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 [11]. Afterwards, 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.

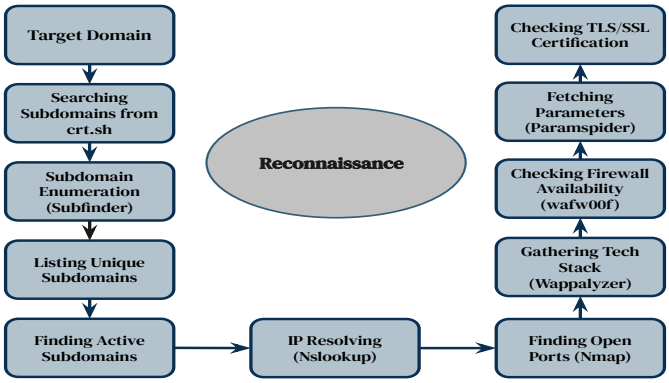


Fig. 2. Reconnaissance module steps, mechanism and tools

In addition, to find IP addresses, tools such as NSLookup can be used to query the Domain Name System (DNS) and retrieve information about a domain, such as IP address [12]. 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. [13]. 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 [14]. 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. 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 [15]. 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, which collects and identifies parameters used in URLs [16].

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.

In this paper, the focus remains largely on the OWASP top 10 vulnerabilities [17], which represent the most critical security risks to web applications. Table I 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.

SQL injection is a common vulnerability, where the attacker injects malicious SQL queries and commands to retrieve, modify, or delete data [18]. Tools such as Wapiti automate the detection of SQL injection vulnerabilities. [19] Similarly, another common injection vulnerability is cross-site scripting or (XSS) that implants malicious content, allowing the attacker

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 Failures	Weak credentials.
Server-Side Request Forgery	Server-Side Request Forgery.

TABLE I
COMMON VULNERABILITIES MAPPED TO OWASP TOP 10 (2021).

to access cookies, user sessions, and other sensitive data [18]. Tools such as PwnXSS automate the detection of XSS vulnerabilities in web applications [20]. In addition, the issue of security misconfiguration may arise due to unprotected files and directories being left publicly accessible on a web server [21]. By scanning the target domain, Gobuster automates the retrieval of unprotected directories and files [22]. Moreover, cryptographic failures such as weak cipher suites, expired certificates and insecure SSL/TLS protocol versions can compromise data confidentiality and integrity during transmission [23]. Tools such as Testssl.sh automate the detection of such cryptographic weaknesses [24].

Besides, Nikto addresses security risks raised by web servers due to poor configuration, such as outdated software versions and insecure http headers [25]. CRLF (Carriage Return Line Feed) injection occurs when an attacker injects newline characters into the web application. Similarly, attackers may manipulate URLs and redirect users to untrusted external sites using open redirection vulnerability [26]. In addition, weak credentials leave applications highly vulnerable to brute-force attacks or unauthorized access [27]. The tool Wapiti automates the detection of these vulnerabilities by analyzing input handling in HTTP header, identifying redirection flaws and flagging weak credential usage [19]. 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 [28]. In contrast, CSRF occurs when an attacker targets the user by tricking them into performing unintended actions such as changing credentials or initiating transactions [29]. 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 [29]. Wapiti detects both vulnerabilities by analyzing server responses for unauthorized requests and verifying the presence or absence of anti-CSRF tokens [19]. Command Injection occurs when an attacker injects malicious input into a vulnerable web application, enabling the execution of arbitrary system commands with the same privileges as the application [30]. 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 [31]. Wapiti, a black-box vulnerability scanner, automates the detection of both command injection and XXE injection [19]. The absence of the HttpOnly flag in a Set-Cookie response header increases the risk of client side script access to sensitive cookies [32]. 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 [19]. Exposure of backup files provide the attackers access to the source code and configuration details of a server which increases the attack surface. Wapiti finds the publicly accessible backup scripts archives by performing a vulnerability scan against the target URL [19]. 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 [33]. Wapiti detects the presence of CSP vulnerability by finding lack of csp or weak CSP configurations in urls [19].

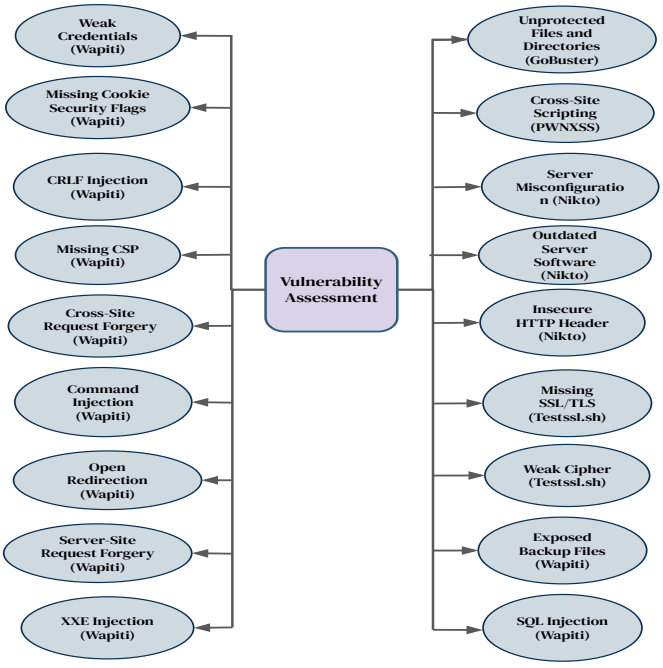


Fig. 3. Vulnerability Assessment Scan flowchart

The diagram illustrated in Figure 3 provides a sequence flow of the vulnerability assessment scan along with the tools used in each step.

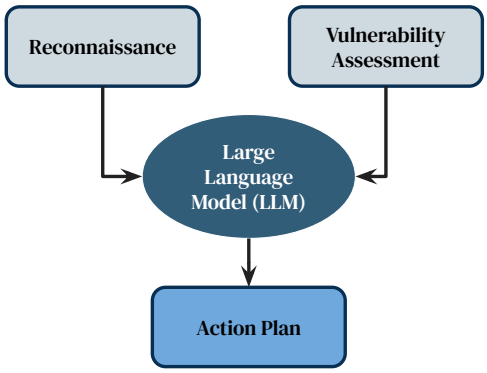


Fig. 4. LLM Action Plan Generation Flowchart

3) Action Plan Generation with LLM: Figure 4 illustrates the generation of the action plan utilizing the Large Language Model (LLM). The proposed framework stores the outputs generated from automated reconnaissance and vulnerability assessment, 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 [1].

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.

Despite recent advancements in AI-assisted penetration testing, as reported in recent research [34], LLMs still face limitations when it comes to autonomously performing exploitation techniques and human expertise continues to outperform these systems in constructing functional exploit code within a limited number of iterations [34]. Given these limitations, the proposed framework deliberately refrains from automating the exploitation step. 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.

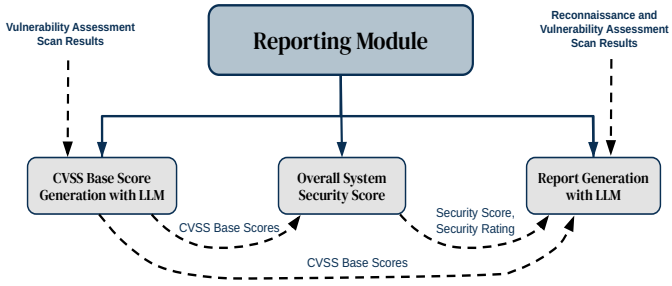


Fig. 5. Phases of the Reporting Module

5) Reporting: The proposed framework entirely automates the reporting step and divides it into three phases as presented in figure 5. The three phases are: i) CVSS base score generation for each found vulnerability, ii) Overall Web Application Security Scoring and iii) Textual Report Generation.

i) 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 [35]. Typically, penetration testers assess each vulnerability manually and select the appropriate CVSS base score metrics, which are then used to calculate the final score using the base score equation. The proposed framework 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 [35], 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. 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 6 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) [36]. The approach of CVSS base score calculation helps standardize severity assessment and ensures that reported vulnerabilities include meaningful impact indicators.

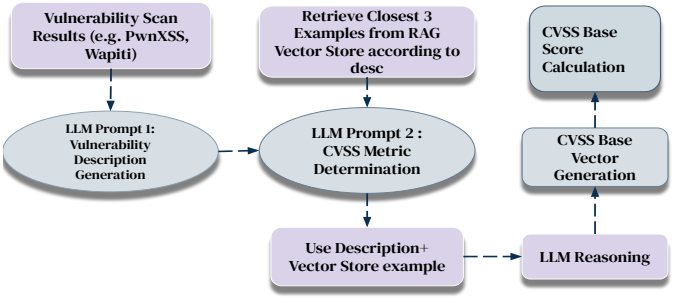


Fig. 6. The flow of CVSS Base score calculation

ii) 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 calculated following 4 steps: (i) Normalization, (ii) Power Transformation, (iii) Aggregation, (iv) Mapping to the range.
The inputs for this scoring system are the CVSS base scores for every detected vulnerability, ranging from 0 to 10. As the overall security score is calculated for a bound ranging from 0 to 100, normalization is done for appropriate scaling and by scaling 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 each vulnerability by the highest possible CVSS base score (usually 10). Then power transformation is performed on each of the normalized values in order to have an exponential weight distribution and effect of vulnerabilities for different levels of severity on the overall score. Meaning a lower rated vulnerability should have less impact on the overall score compared to a higher rated vulnerability. The power law transformation formula, $y = x^{\lambda}$, ensures such conversion when $\lambda > 1$. Here, x is the normalized value. The power law transformation formula penalizes the large values less and the smaller values more [37]. 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.
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.
Lastly, the aggregated value is then mapped on to the range of 0 to 100.
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.

iii) 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.

B. Implementation and Result Analysis

This subsection 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 [38] 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 [38]. 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.

1) Reconnaissance: The python code for the reconnaissance step is presented as a comprehensive tool to perform reconnaissance on a given domain, with a primary goal to gather detailed information about a domain. The process is divided into multiple functions, each handling different aspects.

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. 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. 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, 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. The get_ip(subdomains) function resolves the IP addresses for each of the active subdomains using the NSLookup tool. 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. Moreover, the techstack(domain) function uses Wappalyzer to analyze the target domain's website, by sending a request to the domain and detect technologies based on response patterns. In the FireWall Check(domain) function, the wafw00f tool detects web application firewalls that may be protecting the target domain. 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 concluded that SSL/TLS is not supported on the target domain. 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 7 and Figure 8.

```
[#] 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
```

Fig. 7. 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.
```

Fig. 8. Reconnaissance Results (ii)

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 as illustrated in Figure 9 and Figure 10

```
[#] 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.
```

Fig. 9. Vulnerability Assessment Scan Results [Gobuster]

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. 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. Afterwards, 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 9. Also, 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 10. 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. 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 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 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).

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 structured manual exploitation roadmap composed of human-readable outputs that help testers in systemically conducting exploitation and post-validation.

The figures 11 and 12 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.

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. Afterwards,

```
[#] 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>', 'uname': '<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.
```

Fig. 10. Vulnerability Assessment Scan Results [PwnXSS]

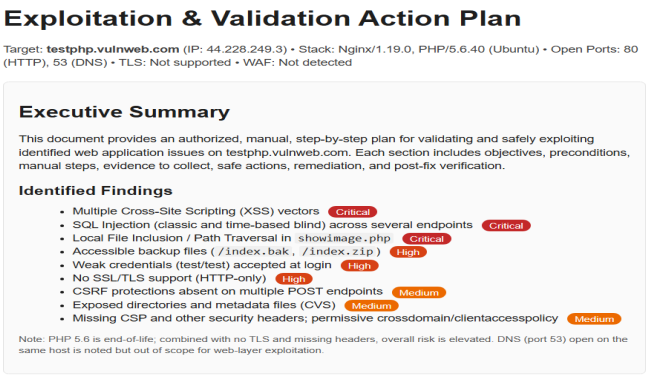
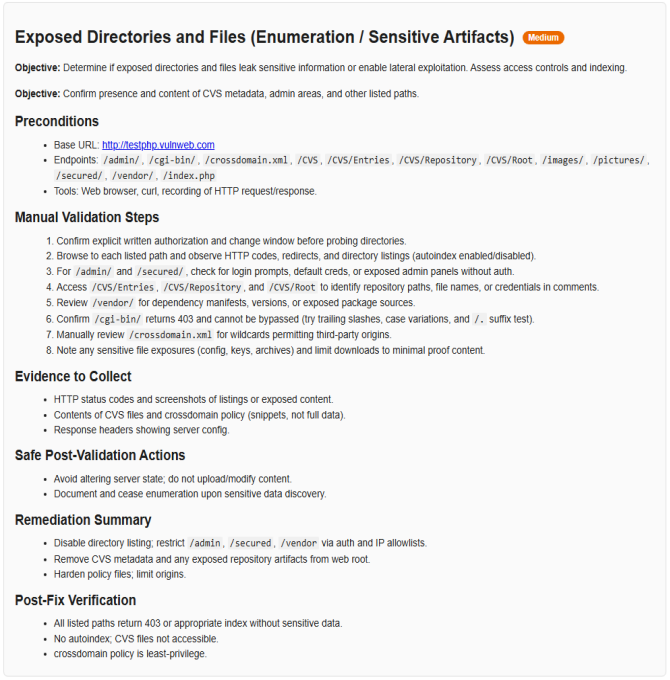


Fig. 11. Action Plan Generated by LLM (i)



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.

Fig. 12. Action Plan Generated by LLM (ii)

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 12.

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. At the beginning of the document, Executive Summary outlines the identified vulnerabilities and their severity level which is shown in figure 11. At the end section of the report, it provides a concise Retest and Closure Checklist. This structured and standardized format that ensures readability and offers significant assistance to the manual penetration testers

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.

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

ii) CVSS base score calculation:
The function calculate_cvss_base_score computes CVSS base scores by applying the official CVSS version 3.1 formula [39] 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).

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).

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, detected vulnerabilities, their severity levels, corresponding CVSS score, pre-calculated security score and security rating are passed to the LLM through prompt engineering. 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. 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

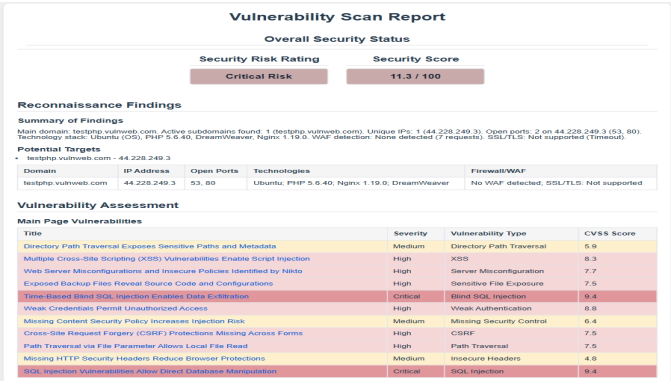


Fig. 13. Report Generated by LLM (i)

Fig. 14. Reports Generated by the LLM.(ii)

generation process.

The HTML report is generated in a specific format consisting of two distinct sections. The first section, as portrayed in Figure 13, contains overall Security Status of the target website along with its security rating and security score that were pre-calculated, 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 13 and Figure 14.

IV. Evaluation

The proposed framework is evaluated through a combination of quantitative and comparative analyses. The framework's performance is assessed using key metrics, as well as a scoring system and subtask completion analysis, adapted from previous research. 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.

A. 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. In this paper, the ground truth data were collected for the deliberately vulnerable web application testphp.vulnweb.com [38].

The deliberately vulnerable application testphp.vulnweb.com [38] 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 [41], the official Acunetix scan report published for target website [42], publicly available technical articles [43], [44] and GitHub repositories [45], [46], [47] 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.

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

B. 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 [3] 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.

Precision is 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 [3]. Similarly, 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 [3]. 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 [3].

In Table II, 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	testphp.vulnweb.com		
	Precision	Recall	F-measure
SQL Injection	100%	70%	82.35%
Cross-Site Scripting (XSS)	88.89%	66.67%	76.21%
Unprotected Files and Directories	100%	55.55%	70.96%
Insecure HTTP Headers	100%	100%	100%
Missing SSL/TLS	100%	100%	100%
Outdated Server Software	100%	100%	100%
Weak Credentials	100%	100%	100%
Server-Side Request Forgery	0%	0%	0%
CSRF	20%	50%	28.57%
CSP	100%	100%	100%
Missing Cookie Security Flags	0%	0%	0%
Weak Cipher	0%	0%	0%

TABLE II
PERFORMANCE EVALUATION OF THE PROPOSED FRAMEWORK ON TESTPHP.VULNWEB.COM

C. 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 [8]. The structured scoring system in [8] evaluates a framework based on different criteria, which includes test coverage, efficiency, vulnerability detection and other features.

Based on the evaluation mechanism described in [8], this paper assigns scores to the proposed framework, Acunetix Web Vulnerability Scanner [48] and OWASP ZAP scanner [49], 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. 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 III.

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

TABLE III
TOTAL SCORES FOR THE 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.

D. Subtask Completion Evaluation

In order to support a systematic evaluation method, the concept of Task Decomposition was adopted from [34]. 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 [34], 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. 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 [48] and OWASP ZAP [49], to analyze their performance across all defined subtasks mapped under defined categories. The evaluation is conducted based on each scanner's test results obtained from testing on the deliberately vulnerable web application http://testphp.vulnweb.com [14].

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 IV.

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 IV
COMPLETION RATE OF EACH CATEGORY FOR THE PROPOSED FRAMEWORK, ACUNETIX, AND OWASP ZAP.

From the table IV, it can be determined that 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 [34], 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.

E. Analytical Comparison Between LLM Models for CVSS Scoring

Different LLM models such as GPT4.0, GPT 4o mini and GPT5 are used to perform the comparison of the Common Vulnerability Scoring System (CVSS) calculation. 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 [50]. 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.

Among eleven vulnerabilities, GPT-5 determines eight severity levels accurately. GPT-4.1 determines six vulnerability severity correctly out of eleven, which demonstrates moderate consistency. GPT-4o-mini, conversely, produces less consistent results than the NVD baseline as it only produces four accurate vulnerability severities out of eleven. Therefore, 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.

F. 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.

One such tool is Zed Attack Proxy (ZAP) by Checkmarx, which is an open source web application scanner that provides an automated scan against the target URL for vulnerability assessment to undergo the known vulnerabilities and hidden content of a web application [49]. However, OWASP ZAP is mostly feasible towards experienced security experts who have a strong grasp on predefined scanning rules, and 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.

Similarly, Acunetix Web Vulnerability Scanner is a commercial web application security scanner which efficiently identifies common vulnerabilities [25]. However, it offers as a paid service and has a tiered licensing structure that restricts access to several advanced features, thus making it suitable for only enterprise clients [25]. Another such tool is Burp Suite, of which the available Community Edition offers limited services that are needed for manual penetration testing [51]. Though the Pro version provides automation in vulnerability scanners, it remains an enterprise product with limited accessibility [51]. Nessus is another enterprise based vulnerability assessment tool, which has a free version called Nessus Essentials. However, the free version of Nessus operates with a limited plugin library, utilizing only a subset of Tenable's vulnerability database [52]. The proposed framework, on the other hand, introduces a modular, open source architecture that emphasizes accessibility due to being cost effective. 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.

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 [34]. 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 [34]. Compared to PentestGPT, the proposed framework demonstrates better automation by integrating the Large Language Model (LLM) directly into penetration testing workflow.

Overall, the evaluation demonstrates that the proposed framework performs competitively against established vulnerability scanners while maintaining the advantages of being modular, open-source and cost-effective.

V. Strengths and Limitations

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. 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 without performing intrusive actions. Additionally, the framework automates 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 [9], 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.

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, increasing configuration difficulty and preventing from being plug-and-play. Further, framework's overall performance may fluctuate depending on the target web applicationthe, since each tool operates with distinct detection logic and vulnerability coverage scope. Additionally, the interpretation of vulnerability severity via the Large Language Model (LLM) may varydue to LLM remaining susceptible to biases or inaccuracies. The automated nature of this process may also raise ethical concerns regarding potential misuse by non-ethical actors. 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.

VI. Conclusion and Future Work

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.

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. 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. 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.

REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] 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.

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

[6] 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.

[7] 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 / jwe15409589.2178.

[8] 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.

[9] 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.

[10] G. J. Kathrine, R. T. Baby, and V. Ebenzer, "Comparative analysis of subdomain enumeration tools and static code analysis," ISSN (Online), pp. 24547190, 2020.

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

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

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

[14] 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.

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

[16] 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.

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

[18] 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.

[19] 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.

[20] 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.

[21] 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/.

[22] 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.

[23] 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/.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.
[29] 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.
[30] OWASP Foundation, Command Injection, [Accessed 07-10-2025], n.d. [Online]. Available: https : / /owasp.org/www - community/attacks/Command_Injection.
[31] 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.
[32] OWASP Foundation, HTTPOnly, [Accessed 07-10-2025], n.d. [Online]. Available: https://owasp.org/www-community/HttpOnly.
[33] PortSwigger Web Security Academy, Content security policy, https://portswigger.net/web-security/cross-site-scripting/content-security-policy, [Accessed 07-10-2025], n.d.
[34] G. Deng et al., "Pentestgpt: An llm-empowered automatic penetration testing tool," arXiv preprint arXiv:2308.06782, 2023.
[35] J. Cabral Costa, T. Roxo, B. Sequeiros, H. Proen¸ca, and P. In´acio, "Predicting cvss metric via description interpretation," IEEE Access, vol. 10, pp. 59 125–59 134, Jan. 2022. doi: 10.1109/ACCESS.2022.3179692.
[36] N. I. of Standards and Technology, Cvss v3 calculator, Available online, NVD, 2025. [Online]. Available: https : / / nvd . nist . gov / vuln - metrics / cvss / v3 - calculator.
[37] J. Maindonald and W. J. Braun, Data analysis and graphics using R: an example-based approach. Cambridge University Press, 2010, vol. 10.
[38] TestPHP VulnWeb, Disclaimer, [Accessed 07-10-2025], n.d. [Online]. Available: http://testphp.vulnweb.com/disclaimer.php.
[39] 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.
[40] Digininja, GitHub - digininja/DVWA: Damn Vulnerable Web Application (DVWA), [Accessed 07-10-2025], n.d. [Online]. Available: https://github.com/digininja/DVWA.
[41] 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.
[42] Sullo, GitHub - sullo/nikto: Nikto web server scanner, [Accessed 07-10-2025], n.d. [Online]. Available: https://github.com/sullo/nikto.
[43] 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.
[44] 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.
[45] Tutorial, GitHub - tutorial0/testphpvulns : Collectal-lvulnsinhttp : //testphp.vulnweb.com/.[Online]. Available: https://github.com/tutorial0/testphp vulns.
[46] 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.
[47] 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.
[48] Acunetix, Web Vulnerability Scanner - Website vulnerability scanning, [Accessed 07-10-2025], Mar. 2025. [Online]. Available: https : / / www . acunetix . com/vulnerability-scanner/.
[49] OWASP ZAP Project, ZAP – Getting Started, [Accessed 07-10-2025], n.d. [Online]. Available: https://www.zaproxy.org/getting-started/.
[50] 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.
[51] A. Mahajan, Burp Suite Essentials. Packt Publishing Ltd, 2014.
[52] OpenAI, Openai platform: Gpt-5 documentation, https : / / platform . openai . com/docs/models/gpt-5, Accessed: 2025-10-07, 2025.