JTE HCM-UTE
ISSN: 2615-9740

JOURNAL OF TECHNICAL EDUCATION SCIENCE
Ho Chi Minh City University of Technology and Engineering
Website: https://jte.edu.vn
Email: jte@hcmute.edu.vn

A Framework for Automated and Visualized Penetration Testing

Thang Loi Nguyen, Thanh Van Nguyen*, Luu Gia Bao Nguyen
Ho Chi Minh City University of Technology and Engineering, Vietnam

*Corresponding author. *Email: vanntth@hcmute.edu.vn*

ARTICLE INFO

Received: 31/07/2025
Revised: 14/01/2026
Accepted: 03/02/2026
Published: 28/02/2026

KEYWORDS

Penetration Testing;
Security vulnerability;
Automation;
YAML workflow;
Visualization.

ABSTRACT

The fragmentation of command-line tools in penetration testing creates inefficient scenarios, additional manual use, and inconsistent results, all of which can make workflows extremely problematic for complex security testing scenarios. This paper presents EzPentest, a framework designed to automate and visualize penetration testing through a single web interface. EzPentest's novelty is its YAML-based workflows, which support conditional logic, looping, and parallelization to create flexible and repeatable testing processes. Key to the use of EzPentest, is the parser engine which will convert the output of different tools into a standardized JSON output, this transformation standardizes vulnerability analysis and reporting. Along with its parser, EzPentest has a modular approach to allow the community to enhance and share the workflows that will connect various tools to create holistic penetration testing scenarios. In experiments with benchmark applications, as in DVWA and bWAPP, EzPentest achieves the highest detection rate of 89.39%. As demonstrated, EzPentest is more than simply an solution to provide scalable, accessible, and collaborative penetration testing, it is an open community resource that is particularly beneficial in educational institutions as it makes easier to understand an advanced area of software vulnerability assessing and security testing and allows small-to-medium enterprises to undertake initiatives to automate pentesting.

Doi: https://doi.org/10.54644/jte.2026.1971

1. Introduction

Penetration Testing (PT) is a proactive method to console test the security of Identification, Authorization, and Authentication services, applications, and networks. Penetration testing is vital in security and is considered the cornerstone of cybersecurity methodologies. Traditional penetration testing workflows consist of several command-line interface (CLI) tools such as Nmap for network discovery, Nikto that has high-level web server vulnerability scanning function, SQLmap for SQL injection exploitation and others. Each of these CLI tools serves a particular, albeit separate purpose. These issues of being disparate regularly lead to inefficient workflow, a greater amount of manual intervention, different outputs for each scanner/tester, problems of reproducibility, and we particularly note here the negative effects in educational practices and small-to-medium enterprises (SMEs) organizations in third-world countries, such as many in Southeast Asia [1], where proprietary solutions like Nessus are often prohibitively expensive via commercial, licensing, and maintenance/hosting costs, and sadly often for safety-related outcomes as well.

Existing frameworks have developed solutions to the problems, but all have significant shortcomings. Collaborative tools (e.g., Faraday [2]) aggregate output of scanners but require manual configuration, and there is no workflow automation. OpenVAS [3] provides good network scanning, but suffers from false positive as well as scalability issues for a large environment as well as needs considerable post processing. There are research frameworks, for example WAVS [4] which introduced improvements in detection by combining other tools like OWASP ZAP and Arachni for better recall but does not provide exploitation or a user interface, only CLI aggregation. On the other hand, ADAPT [5] used a MAPE-K loop for adaptive automated penetration testing in large networks, but requires formal

modelling and computational resources, therefore limiting it to specialists rather than the layman or smaller team.

These limitations highlight an important need: a framework that aligns advanced automation with accessibility, scalability, and standardization. This paper presents EzPentest, an open-source framework that combines numerous PT tools in a single web-based interface developed entirely in the programming language Python. The novelty of EzPentest is its ability to provide YAML-defined workflows that include conditional logic, loops, and the ability to run certain stages of testing in parallel, resulting in dynamic automation of multi-stage tests. Along with our parser engine that normalizes and converts tool outputs (e.g., XML back from Nmap, text provided by SQLmap) into a consistent JSON schema for simplified analysis, reporting, and integration into subsequent systems. By expanding existing ideas of tools like Nuclei [6] which employs YAML templates for vulnerability scanning, we improved upon the abject aggregation of all tools to deliver tools with features of visual workflow editing, community-driven template sharing, and extensible tools and parsers for user-defined extensions. This design reduced hands-on work in experimental-based workflows while adding repeatability and collabortion consistent with a target of meeting the needs of learners, SMEs and security assessments.

2. Background and Related Works

2.1. Penetration Testing Methodologies

Penetration testing is a process that systematically attempts to assess the security of an information system by emulating numerous aspects of a real-world cyber–attack. The goal of such testing is to discover exploitable vulnerabilities, preferably before attackers become awares of them and before they are exploited. To provide a consistent and thorough framework, various standard methodologies have been developed. One of the leading standardised methodologies universally accepted is the Penetration Testing Execution Standard (PTES) methodology [7]. PTES describes a systematic approach that can be broken down into a number of phases, as illustrated in Figure 1, including initial engagement, information gathering, threat modelling, vulnerability analysis, exploitation, and reporting. The process presents succinct guidelines and a repeatable method suffix which is essential components to assess security posture and plan remediation.

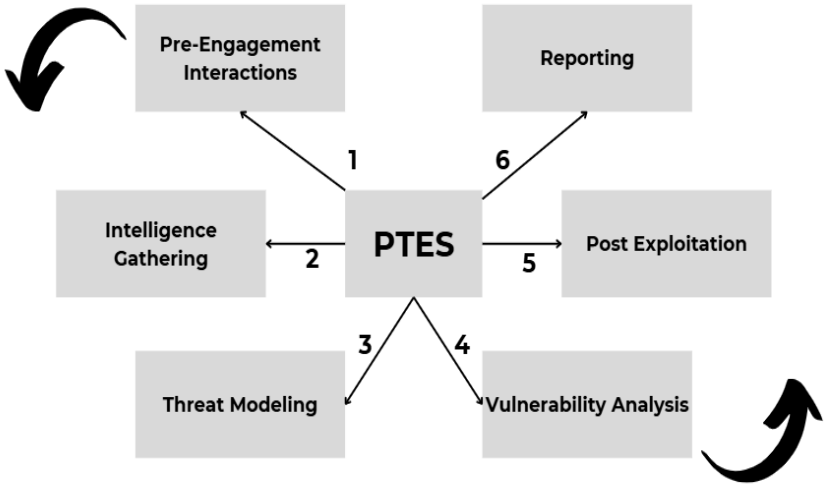

Figure 1. *PTES with six phases.*

Other frameworks include NIST SP 800-115 [8], OpenSource Security Testing Methodology Manual (OSSTMM) [9], and the Information Systems Security Assessment Framework (ISSAF) [10]. OSSTMM focuses much more on an audit driven methodology, and also contains a complete set of metrics for security assessment, while NIST SP 800-115 gives highly detailed steps to carry out in preparation for, and when conducting, security assessments, with an extremely heavy emphasis on vulnerability assessment and risk management. ISSAF is again flexible and is intended to be customized by the organization to fit its own special needs and infrastructures. While each of these methodologies has different levels of completeness and specificity, they all serve the same purpose: to provide an

organized and methodical way to test and improve the ability of an organization to defend itself against security threats. The adoption of these common methodologies into automated penetration testing tools has really matured the field. Whereas these methodologies provide good frameworks for manual or in-person tests, implementing the methodologies in a purely automatic context comes with challenges such as diffuse workflows, heightened false positives, and limited use of multi-phase tests. This indicates that we need integrated systems to be able to adopt the optimal processes from established frameworks and then add automation for skills and efficiencies for Security Tasks.

2.2. Automated Penetration Testing

Automated penetration testing refers to the use of tools and software to systematically and automatically detect security weaknesses and vulnerabilities in the computer system, and security holes in the application and network. Automated PT significantly reduces the time and effort involved in conducting security assessments, while also improving the accuracy of detection and giving organizations the ability to rapidly detect and mitigate potential threats. We provide a summary of some of the current automated PT solutions in Table 1.

Table 1. *Comparison of some existing automated PT solutions.*

Feature	Faraday [2]	OpenVAS [3]	WAVS [4]	ADAPT [5]	AIPenTool [12]
Primary Scope	Collaborative PT	Vulnerability Scanning	Web Scanner Combination	Full Autonomous APT (Network)	Web Scan
User Interface	GUI	GUI	CLI	CLI (with visualization tools)	CLI
Workflow Automation	Limited	Limited	Basic Scripting (Custom Logic)	Formalized Planning	Basic Scripting
Output Standardization	Custom Formats	XML	Combined (Custom Logic)	Formalized (Internal Model)	Inconsistent
Community Sharing	Limited	None	None	None	None

Established tools include Faraday [2], OpenVAS [3], and Nessus [11], which have all contributed to the practice of automated PT. Faraday [2] aggregates the output from multiple tools and certainly depends on the quality of the external scanner outputs. OpenVAS [3] performs active scans of the network, but again is susceptible to false positives, and certainly does not scale well in very large networks. Both Faraday and OpenVAS exemplify how research has expanded the possibilities of automation.

In fact, frameworks like the WAVS Framework [4] have allowed researchers to expand the limits of web application security testing by improving the detection phase. Where it excels is by automating and aggregating the output of web application scanners such as OWASP ZAP and Arachni and has even yielded enhanced accuracy and better recall than any scanner individually. It certainly has limitations however, the biggest limitation being that the WAVS Framework itself is only a command-line tool that is intended to correlate the result of web scanners only, consequently it was not intended to be comprehensive or provide a useable workflow that incorporates multiple scanning and testing technologies.

AIPenTool [12] uses embedded processes that integrate network and web application scanning, increasing efficiency by reducing reliance on separate tools but resulting in limited data standardization from tools differing outputs. Scorpio [13] engages in simulated real world-like attacks in a cyber range, improving training and visualization of defensive strategies. Its effect is still constrained by the procedural complexity of maintaining stable integration with a dynamic environment; this will need ongoing procedural processes. Furthermore, new methods such as applying Reinforcement Learning

(RL) techniques to PT [14], have been attempted which allow intelligent, self-adaptation to network condition changes but present severe scaling issues due to require significant computational resources. Machine-learning based methods [15], [16] have potential for improving detection accuracy and auditability but they also introduce the concern of insufficient training data availability and model refresh frequency.

At the higher end of autonomy, ADAPT [5] is the current state-of-the-art, and while it is capable of can adapting to available cyber ranges, its key feature is its self adapting architecture (utilising a MAPE-K loop) that fully automates the whole penetration testing (PT) process, enabling online attack planning, while dynamically make decisions at runtime in a complex environment; the barrier to entry, however, may be the architectural complexity which may overwhelm teams requiring more straightforward, user-guided automation and not full autonomy.

3. Proposed Methodology

The fragmented nature of penetration testing tools and their inconsistent outputs present critical roadblocks, with inefficient workflows, manual effort, and scalability limitations in complicated environments being the primary challenges. EzPentest attempts to overcome some of these problems leveraging a modular automated framework which integrates multiple tools in a single web-based interface. EzPentest relies upon YAML-based condition-driven workflows and a JSON-standardized output. This section describes EzPentest's conceptual framework, system architecture, and methodology/tool integration approach, highlighting novel contributions and user-centered design.

3.1. Concept and Expected Efficiency

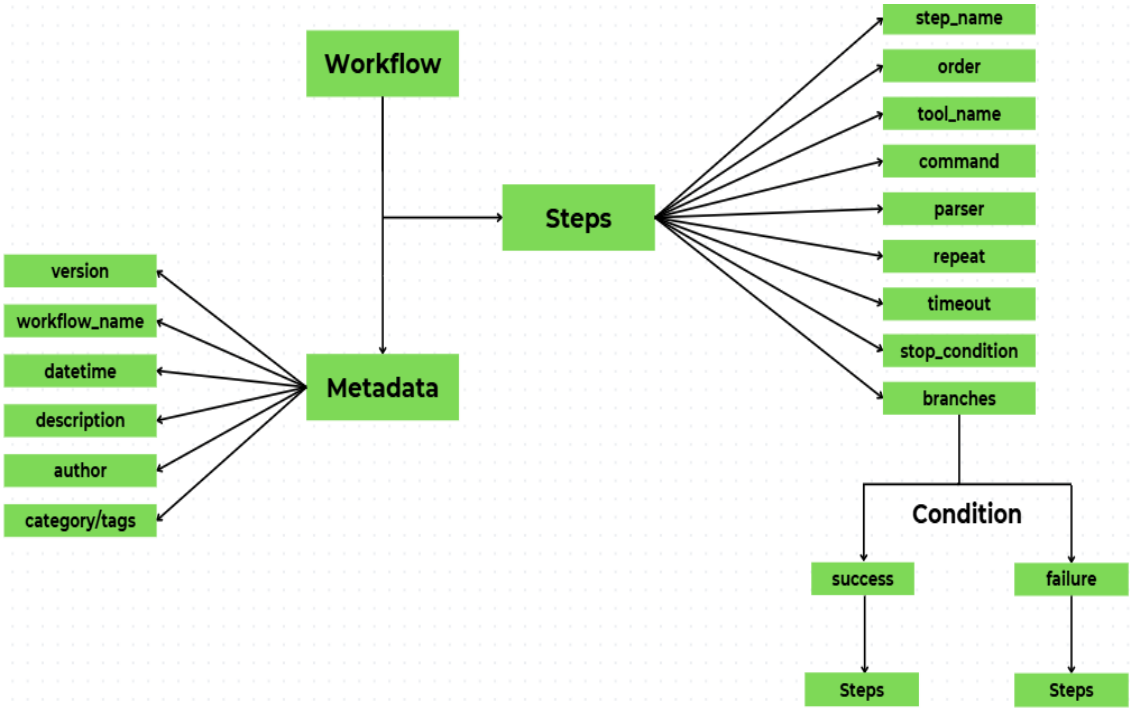

Figure 2. *Workflow structure.*

In EzPentest, the concept of workflow is derived from tools like Nuclei's [6] YAML-based templates,but with a significant advancement of logic for highly dynamic, multi-phase penetration testing. A workflow makes use of a YAML file that defines and organizes a list of steps - each step is a logical action (for example: scan or exploit). Figure 2 shows the workflow structure, steps can be nested by creating parent/child relationships. Branching, looping, and parallel execution are all possible to react to a real-time result of the test. As an example, below is a YAML example of a web vulnerability scan workflow utilizing branching and looping.

```yaml
workflow:
  name: WebVulnScan
  steps:
    - step_name: PortScan # A unique identifier for logging and reference.
      order: 1 # An integer dictating execution sequence.
      tool: nmap # The PT tool to invoke (e.g., "nmap", "sqlmap").
      command: nmap -sV -oX - {{target}} # The CLI command with placeholders (e.g., "{{target}}") for runtime substitution.
      parser: nmap_parser # The associated parser module for output standardization.
      timeout: 300 # Seconds for execution limit
      branches: # For conditional logic based on output matches
        success:
          condition: "open port 80 or 443"
          children:
            - step_name: WebScan
              order: 2
              tool: nikto
              command: nikto -h {{target}}
              parser: nikto_parser
              repeat: 2 # Loop twice for verification
            - step_name: SQLInject
              order: 3
              tool: sqlmap
              command: sqlmap -u {{target}}/search.php?searchfor=1 --batch
              parser: sqlmap_parser
              parallel: true # Execute in parallel with WebScan
        failure:
          condition: "no open ports"
          children: [] # No further steps
  stop_condition: "critical vulnerability found" # Criteria to halt the workflow
```

Figure 3. *Example for web vulnerability scan workflow with YAML.*

In Figure 3, the "PortScan" step branches to "WebScan"' and "SQLInject" in parallel if HTTP ports have been found open and returns back to Nikto for redundancy. The branches 'field' have conditions based on regex lookups on parsed outputs (for example, searching for keywords such as "open port"). This structure provides advantage over most structured methodologies because it has reusable and shareable templates that require less manual reconfiguration.

3.2. System Design and Architecture

3.2.1. System Architecture Overview

EzPentest is modular, web-based architecture focused on optimizing penetration testing workflows, thereby successfully prioritizing security, reliability and performance, which consists of three connected layers:

- **User Interface (UI) Layer:** An easy-to-use web interface workflow builder and executor where you can visually build workflows using drag-and-drop editors to generate YAML files. Includes command autocompletion, command-line template libraries and a Security Dashboard to view live progress (e.g., live logs, summary of vulnerabilities, etc..)
- **Core Processing Layer:** Parses YAML workflows and manages the execution of those steps while incorporating potential logic (i.e., conditions, loops). Any tools/commands are invoked using Python's subprocess module, making sure to manage failure with try-except for subprocess errors and to log exceptions (e.g., timeout, non-zero exit) or allow for graceful fallback (e.g., retry of some transient error). Performance is also an important consideration, so the Core Processing Layer manages parallelism using Python's “concurrent.futures.ThreadPoolExecutor” and limits the number of threads running at any time (the default is set to a maximum of 4 concurrent steps) to prevent exhaustion of resources. Everything that is generated by the workflow is managed through an output that is sent to the parser engine, also for consistency.
- **Data Management Layer:** Uses SQLite database for analytics such as workflows, results, logs and metadata, while keeping JSON outputs for indexed querying to fast querying, thereby allowing for features like historical comparisons and export to CSV or PDF.

Regarding scalability management, the Core Processing Layer implements a dynamic resource throttle. While Python's ThreadPoolExecutor manages concurrency, the system actively monitors

system saturation via psutil. To prevent “resource starvation” where the OS kills processes due to Out-Of-Memory (OOM) errors, the scheduler halts new thread spawning if CPU usage exceeds 90% or RAM availability drops below 500MB. Furthermore, since SQLite relies on file-level locking, the Database Manager Layer implements a “retry-with-backoff” mechanism to handle concurrent write attempts during high-load scenarios.

3.2.2. Functional Workflow and Execution Process

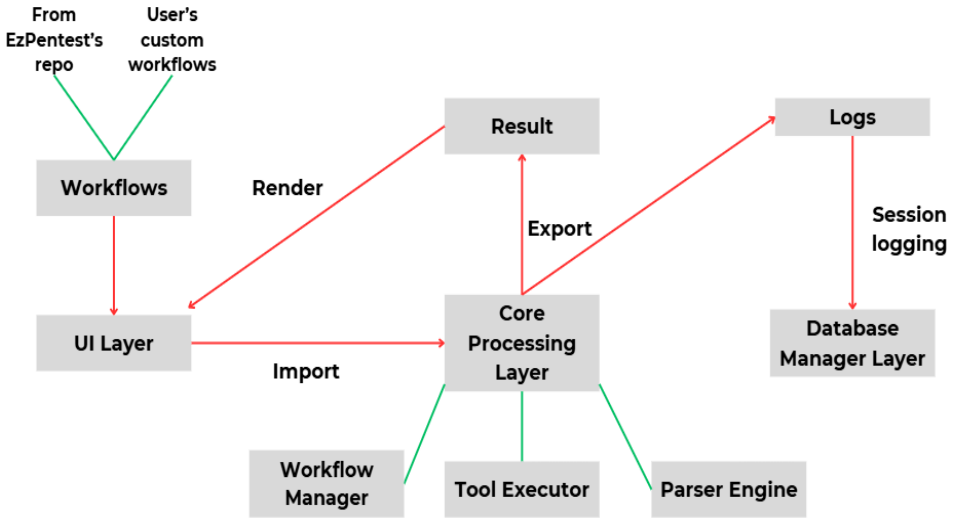

Figure 4. *Execution process in EzPentest.*

Figure 4 outlines the core execution process of EzPentest:

Workflow Definition and Initialization: Users can easily upload, create, or edit YAML-based workflows directly through the interface. The system automatically checks for syntax errors (using PyYAML) and securely fills in placeholders, such as target IPs or domain names.

- **Tools Execution**: Tasks are queued in order, with any branching logic evaluated only after the tools finish their work. We use a secure executor for subprocess calls. To speed things up, parallel steps run via a ThreadPoolExecutor, while psutil monitors the system to ensure we don't overload the CPU or memory. Importantly, running tools in parallel doesn't change how their underlying scanning algorithms work. This means our efficiency gains don't hurt the accuracy, the true positive and false positive rates remain exactly the same as if the tools were run individually.
- **Output Standardization**: Our parser engine uses specific modules to handle different types of tool outputs, whether that’s using regex for plain text or dedicated libraries for XML and JSON. In this architecture, the YAML logic and the JSON parser are two sides of the same coin. The parser isn't just an "add-on"; it’s what makes conditional branching possible. You can't really test the branching logic without the parser, because the system needs that normalized data to decide what to do next.
- **Storage and Reporting**: Once the data is parsed into JSON, it’s stored in a SQLite database. From there, we use Jinja2 templates to turn that data into interactive HTML reports, which you can view directly from the dashboard.

By combining structured YAML workflows with automated logic, we’ve made it possible to streamline complex security operations. The real value of this framework comes from the synergy between parallel execution, logic-heavy workflows, and standardized reporting. This combined approach is what allows our "automated and visualized" solution to achieve the high efficiency we saw in our tests.

3.3. Integration of Penetration Testing Tools

Every penetration test tool is called via custom Python-based wrapper scripts, hiding the complexity of the command line syntax and execution logic. The wrappers are managed by the internal workflow executor module that parses the user-provided YAML workflows, builds the corresponding command sequences dynamically, and calls them via Python's subprocess interface.

One of the fundamental strengths of EzPentest is its normalized output parsing. It parses raw command-line output to normalized JSON, allowing consistent reporting and reducing downstream processing and visualization. The parser engine's algorithm is tool-agnostic but customizable:

- **Input:** Raw stdout/stderr.
- **Processing:**
  - Detect format (e.g., XML, JSON, text via heuristics) and normalize line endings.
  - Apply tool-specific extractors (e.g., for SQLmap: use regex to parse logs for target, DBMS, vulnerabilities, databases, tables, columns; for Nmap: traverse XML tree).
  - Normalize to JSON: Map findings to common fields (type, severity, details), handling errors like "no vulnerabilities" or WAF detection.
  - Error handling: If parsing fails (>10% unmatched lines), embed raw output and flag for manual review.
- **Output:** Consistent JSON, enabling uniform analysis.

The normalized integration greatly enhances ease of use, especially with non-experts, by eliminating the need to memorize arcane flags or manually interpret raw output. For experts, repetitive work is minimized and productivity increased by allowing repeat and sharable workflows.

EzPentest combines a wide array of established tools that together provide in-depth web application and network penetration testing. The tools are organized by their main functionalities as follows:

- **Subdomain Analysis:** Subfinder – Advanced subdomain enumeration tool; Subzy – Comprehensive subdomain takeover detection
- **Information Gathering:** Katana – High-performance web crawling engine; Wayback Machine – Historical data analysis and archiving
- **Vulnerability Assessment Suite:** Nuclei – Template-based vulnerability scanner; Nikto – Web server security scanner
- **Specialized Exploitation Tools:** SQLmap – Automated SQL injection detection and exploitation; XSSStrike – Advanced XSS detection framework; SSRFmap – Server-Side Request Forgery automation; Dirsearch – Web path enumeration utility; LFImap – Automated file inclusion vulnerability scanner; SSTImap – Server-Side Template Injection analyzer; Hydra – Parallelized login brute-forcing tool
- **Network Reconnaissance:** Nmap – Network discovery and security auditing
- **Network Protocol Security:** Yersinia – Network protocol vulnerability testing; Ettercap – Comprehensive MITM attack suite

The toolset enables EzPentest to deliver broad-spectrum coverage across different stages of a penetration test, from reconnaissance and enumeration to exploitation and reporting, while maintaining a high level of automation and flexibility.

4. Experiments and Results

4.1. Experimental Setup

Our methodology aligns with related works like AIPenTool [12], integrating tools such as Nmap and OWASP ZAP, and the WAVS Framework [4], which combines Arachni and OWASP ZAP for precision and recall analysis on benchmarks like NodeGoat and Juice Shop. We adopted a black-box testing approach, where scanners lack prior knowledge of the application’s internal structure, simulating real-world adversarial conditions, ensuring reproducibility and comparability with established standards. The

test targets were benchmark web applications: testphp.vulnweb.com [17] (an Acunetix framework with many vulnerability instances across OWASP categories), Damn Vulnerable Web Application [18] (DVWA, a PHP/MySQL app with vulnerabilities in three security levels) and bWAPP [19] (a buggy web application with over 100 vulnerabilities).

EzPentest's workflow began with an Nmap port scan to identify open ports and services, setting the stage for subsequent vulnerability detection. This was followed by parallel vulnerability scans using Nikto for server misconfigurations (e.g., detecting missing security headers/config issues), Dirsearch for directory enumeration (e.g., uncovering local/remote file inclusion), SQLmap for Injection exploitation (e.g., targeting SQL Injection variants), XSSStrike for XSS detection (e.g., identifying cross-site scripting flaws), Nuclei for template-based scanning (e.g., spotting CSRF and other general vulnerabilities), Hydra for brute-forcing (e.g., probing sensitive data exposure) and others well-known security tools. This aligns with sequential and parallel strategies in ADAPT [5] and AIPenTool [12].

4.2. Results and Discussion

Table 2. *EzPentest's scanning results in three vulnerable web applications.*

Vulnerability Type	**Vulnerability Detection Rate**		
	testphp.vulnweb.com	DVWA (medium level)	bWAPP
SQL Injection	83.33%	75.00%	77.78%
Cross-Site Scripting	83.33%	91.67%	84.62%
File Inclusion	100%	100%	87.50%
Command Injection	N/A	100%	75.00%
Sensitive Data Exposure	100%	66.67%	88.89%
Missing Security Headers/Config	75.00%	N/A	80.00%
Cross-Site Request Forgery	N/A	100%	83.33%
Detection Rate Overall	88.89%	89.39%	82.52%

Table 3. *Detection Performance of EzPentest Across Vulnerability Types.*

Vulnerability Type	**TPR (Recall) [%]**	**FPR [%]**	**Precision [%]**	**F1-Score [%]**
SQL Injection	78.70	7.39	76.06	77.36
Cross-Site Scripting	86.54	6.08	84.03	85.26
File Inclusion	95.83	3.11	94.59	95.20
Command Injection	87.50	5.07	84.95	86.20
Sensitive Data Exposure	85.18	9.85	77.68	81.26
CSRF	91.66	4.49	89.49	90.56
Overall Average	87.57	6.00	84.47	85.97

The experimental results highlight just how capable EzPentest is at identifying vulnerabilities across various web application environments. As shown in Table 2 and Table 3, the framework maintained strong detection rates across all our benchmark targets: 88.89% on testphp.vulnweb.com, 89.39% on DVWA, and 82.52% on bWAPP. EzPentest performed exceptionally well in several critical categories. For instance, it achieved a perfect 100% detection rate for File Inclusion and Sensitive Data Exposure on testphp.vulnweb.com, as well as 100% for Command Injection and CSRF on DVWA. Our statistical analysis confirms these results are reliable. The overall F1-Score, which balances precision and recall,

reached 85.97%. Notably, the File Inclusion category earned the highest individual F1-Score at 95.20%. These numbers suggest that by running specialized tools like SQLmap and XSSStrike in parallel, the system effectively reduces missed vulnerabilities (false negatives) without sacrificing accuracy.

Table 4. *Summarizes EzPentest's performance against AIPenTool, the WAVS Framework and ADAPT.*

Feature/Tool	EzPentest (Web)	AIPenTool [12] (CLI)	WAVS [4] Framework (CLI)	ADAPT [5] (CLI/Hybrid)
Primary Scope	Integrated APT (Web & Network Scan + Exploit)	Web Scan (Limited Exploitation)	Web Scan (Scanner Combination)	Full APT (Network Planning + Exploit)
User Interface	Web-based (Intuitive)	CLI	CLI	CLI (with visualization tools)
Workflow Automation	YAML (Conditional, Looping, Parallel)	Basic Scripting	Basic Scripting	Formalized Planning (MAPE-K)
Output Standardization	JSON (Consistent)	Inconsistent	Combined (Custom Logic)	Formalized (Internal Model)
Execution Time	9.8 mins	5.2 mins	10.5 mins	N/A
FPR	6%	15%	8%	N/A
Manual Interactions	Low (2-3 clicks)	High (~8-10 commands)	High (~8-10 commands)	Moderate (~15-20 commands for setup)

To provide context for EzPentest's capabilities, we compared its features and performance against established automated solutions like AIPenTool, the WAVS Framework, and ADAPT. A detailed breakdown of this comparison is available in Table 4.

- Accuracy and Reliability: EzPentest stands out in performance, particularly against the WAVS Framework, which has previously reported an F-measure of around 73% using multi-scanner methods. By utilizing a standardized JSON parsing engine, EzPentest achieved a significantly lower False Positive Rate (FPR) of 6%. This is a notable improvement over the 8% seen in WAVS and 15% in AIPenTool. Reducing this "noise" is vital, as it directly lightens the manual validation workload for security professionals.
- Operational Efficiency: When it comes to speed, AIPenTool is faster (5.2 minutes) because of its lightweight, CLI-based design. However, it doesn't offer the deep exploitation capabilities or the standardized reporting that EzPentest provides. On the other end of the spectrum, ADAPT offers high autonomy through its MAPE-K loop, but it demands complex formal modeling and a time-consuming setup. EzPentest bridges this gap with its "low-code" YAML workflow. This simplifies the process down to just 2-3 clicks, a sharp contrast to the 8-10 complex commands required by traditional CLI-based alternatives.
- Usability: EzPentest features a web-based GUI, this makes the tool much more accessible for non-experts and encourages community collaboration through shareable workflow templates, a feature that is largely missing from other comparable tools.

In short, EzPentest strikes an ideal balance between deep automated exploitation and user-friendliness. Our framework offers better detection accuracy and standardization than aggregative frameworks like WAVS, all while avoiding the steep technical hurdles found in fully autonomous systems like ADAPT [5].

5. Conclusions & Future Work

EzPentest automates security tools through modular YAML workflows and an user-friendly web interface, streamlining vulnerability analysis with interactive reports. In testing, the framework achieved

a high 89.39% detection rate with only a 6% false positive rate. The current setup uses S*QLite* and Python's *ThreadPoolExecutor* for a lightweight experience, perfect for educational use and small businesses. The system is architected for scalability, allowing for a seamless transition to PostgreSQL to manage high-frequency enterprise data and concurrent users without bottlenecks. Furthermore, it is evolving toward a horizontal scaling model using Celery and Redis, enabling task distribution across a cluster of nodes to handle massive network ranges efficiently while overcoming main server's hardware limitations.

6. Availability and Ethical Considerations

To facilitate reproducibility and community adoption, the core source code for the EzPentest framework is made publicly available: https://github.com/nlgbao1340/EzPentest_1.0. To minimize any potential risks, we recommend running EzPentest within an isolated sandbox environment and using low-privilege accounts whenever you can. We have also included a "Safe Mode" which prevents the kind of aggressive scans that might disrupt services, ensuring the tool remains a safe, defensive resource.

Conflict of Interest

The authors declare no conflict of interest.

REFERENCES

[1] H. M. Adam, W. Widyawan, and G. D. Putra, "A review of penetration testing frameworks, tools, and application areas," in *Proc. 2023 IEEE 7th Int. Conf. on Information Technology, Information Systems and Electrical Engineering (ICITISEE)*, Nov. 2023, pp. 416–421, doi: 10.1109/ICITISEE58992.2023.10404397.
[2] Faraday Security, "Faraday: Collaborative penetration testing platform," 2025. [Online]. Available: https://faradaysec.com
[3] A. Muharrom and R. Saktiansyah, "Analysis of vulnerability assessment technique implementation on network using OpenVAS," *Int. J. Eng. Comput. Sci. Appl. (IJECSA)*, 2023.
[4] K. Abdulghaffar, N. Elmrabit, and M. Yousefi, "Enhancing web application security through automated penetration testing with multiple vulnerability scanners," *Computers*, vol. 12, no. 11, art. no. 235, Nov. 2023, doi: 10.3390/computers12110235.
[5] C. Skandylas and M. Asplund, "Automated penetration testing: Formalization and realization," *Comput. Security*, vol. 155, art. no. 104454, 2025, doi: 10.1016/j.cose.2025.104454.
[6] ProjectDiscovery, "Nuclei: Fast and customizable vulnerability scanner based on templates." [Online]. Available: https://nuclei.projectdiscovery.io/
[7] PTES, "The penetration testing execution standard (PTES) technical guidelines." [Online]. Available: http://www.pentest-standard.org/index.php/PTES_Technical_Guidelines
[8] National Institute of Standards and Technology, "Technical guide to information security testing and assessment," NIST Special Publication 800-115, 2008. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-115.pdf
[9] ISECOM, "The open source security testing methodology manual (OSSTMM) 3.0," 2015. [Online]. Available: http://www.osstmm.org/OSSTMM.3.pdf
[10] ISSAF, "The information systems security assessment framework (ISSAF) overview." [Online]. Available: http://www.issaftesting.org/ISSAF_Overview.pdf
[11] Tenable, "Nessus Professional." [Online]. Available: https://www.tenable.com/products/nessus
[12] N. P. Kumar, "AIPenTool: A unified approach to automated penetration testing for enhanced network and web application security," in *Proc. 2025 Int. Conf. on Intelligent and Innovative Technologies in Computing, Electrical and Electronics (IITCEE)*, Jan. 2025, doi: 10.1109/IITCEE64140.2025.10915305.
[13] W. Pan, J. Han, and M. Y. Yin, "Scorpio: An automated penetration testing tool and its integration with a cyber range," in *Proc. 2021 2nd Int. Conf. on Electronics, Communications and Information Technology (CECIT)*, 2021, doi: 10.1109/CECIT53797.2021.00197.
[14] M. C. Ghanem and T. M. Chen, "Reinforcement learning for intelligent penetration testing," in *Proc. 2018 2nd World Conf. on Smart Trends in Systems, Security and Sustainability (WorldS4)*, Oct. 2018, pp. 185–192, doi: 10.1109/WorldS4.2018.8611595.
[15] T. Huizinga, "Using machine learning in network traffic analysis for penetration testing auditability," Nov. 2019. [Online]. Available: https://rp.os3.nl/2018-2019/p39/report.pdf
[16] D. Suhartono, "The usage of machine learning on penetration testing automation," Aug. 2023, doi: 10.1109/ICE3IS59323.2023.10335188.
[17] Acunetix, "Acunetix web vulnerability scanner." [Online]. Available: https://www.acunetix.com/
[18] Digininja, "Damn Vulnerable Web Application (DVWA)." [Online]. Available: https://github.com/digininja/DVWA
[19] MME, "bWAPP (buggy web application)." [Online]. Available: http://www.itsecgames.com/

Thang Loi Nguyen is currently an undergraduate student majoring in Information Security at the Faculty of Information Technology, Ho Chi Minh City University of Technology and Engineering (HCM-UTE) (formerly Ho Chi Minh City University of Technology and Education), Vietnam. His research interests include cybersecurity and penetration testing.
Email: 22162023@student.hcmute.edu.vn. ORCID: https://orcid.org/0009-0004-5823-443X

Thanh Van Nguyen graduated from university in Infomatics in 1998 at Hue University's College of Education, Hue University, and received a master's degree in computer science in 2005 at Da Nang University. She is currently working at the Faculty of Information Technology, Ho Chi Minh City University of Technology and Engineering (HCM-UTE) (formerly Ho Chi Minh City University of Technology and Education). Her research interests include Information and Network security, machine learning and deep learning technologies.
Email: vanntth@hcmute.edu.vn. ORCID: https://orcid.org/0009-0003-9686-606X

Luu Gia Bao Nguyen is currently an undergraduate student majoring in Information Security at the Faculty of Information Technology, Ho Chi Minh City University of Technology and Engineering (HCM-UTE) (formerly Ho Chi Minh City University of Technology and Education), Vietnam. His research interests include penetration testing and AI technology.

Email: 22162005@student.hcmute.edu.vn. ORCID: https://orcid.org/0009-0009-9593-444X