









Evolution of Automated Penetration Testing: Toolchains, Integration Strategies, and Operational Challenges

Vijaykumar Bidve¹, Kiran Kakade², Priyanka Paygude³, Ranjeet Vasant Bidwe^{4*}, Sunil Sangve⁵,
Aryani Gangadhara⁶

¹ School of CSIT, Symbiosis Skills and Professional University, Pune 412101, India

² Faculty of Management, Symbiosis Institute of Management Studies, Symbiosis International (Deemed University), Pune 411020, India

³ Bharati Vidyapeeth (Deemed to be University) College of Engineering, Pune 411043, India

⁴ Symbiosis Institute of Technology, Pune Campus, Symbiosis International (Deemed University), Pune 412115, India

⁵ Vishwakarma Institute of Technology, Pune 411037, India

⁶ D. Y. Patil College of Engineering, Pune 411044, India

Corresponding Author Email: ranjeet.bidwe@sitpune.edu.in

Copyright: ©2025 The authors. This article is published by IETA and is licensed under the CC BY 4.0 license (<http://creativecommons.org/licenses/by/4.0/>).

<https://doi.org/10.18280/ijssse.150720>

ABSTRACT

As organizations become increasingly reliant on digital infrastructure, their exposure to cyber threats escalates. Penetration testing (PT) is vital for identifying vulnerabilities and strengthening security frameworks. This study explores the evolution of automated penetration testing (APT), analyzing its advantages over traditional manual methods in terms of scalability, efficiency, and consistency. The core contribution of this work is the development of a Python-based graphical user interface (GUI) platform that seamlessly integrates leading APT tools—such as OWASP ZAP, Burp Suite, Vega, and FOCA—into a unified, customizable environment. This integrated approach improves detection accuracy, optimizes multi-tool workflows, and enhances user experience. The system also includes robust vulnerability classification, live scanning feedback, and advanced reporting features. A comprehensive taxonomy of 75+ web-based vulnerabilities is curated and used to evaluate tool capabilities, highlighting strengths in automated detection and limitations in logic-driven exploits. The findings emphasize that such integrated platforms can empower cybersecurity teams by automating routine testing, reducing manual load, and supporting continuous security validation. This research presents a significant step toward scalable, user-centric cybersecurity solutions suitable for evolving digital landscapes.

Received: 9 June 2025

Revised: 10 July 2025

Accepted: 20 July 2025

Available online: 31 July 2025

Keywords:

automated penetration testing, GUI integration, vulnerability detection automation, Python security tools, cybersecurity frameworks, web application security, toolchain integration

1. INTRODUCTION

In the fast-changing digital era, cybersecurity has emerged as a cornerstone of organizational resilience and reliability. Ensuring strong defences against cyber threats has never been more vital given the proliferation of linked systems and the complexity of contemporary IT infrastructures. A proactive cybersecurity tool, penetration testing mimics actual cyberattacks to find and reduce information system and network weaknesses. Traditionally, penetration testing has depended mostly on manual work by skilled, ethical hackers [1, 2]. These experts carefully examine systems, evaluate weaknesses, and suggest mitigating measures. The limitations of manual penetration testing, like a lack of scalability, human error, have led to a growing trend in automation as IT ecosystems scale, becoming fibrous and leveraging advanced tools and technologies for efficiency and efficacy. Software programs and scripts that replicate antivirus and malicious access are called penetration testing (or pen-testing) software tools. In other words, software tools and scripts simulate hacks

to detect weaknesses, provide evidence of cyberattacks, and measure resilience [3, 4]. Automation is associated with many different tasks that would otherwise be done by hand, such as vulnerability management, fraud detection, and reporting. Burp Suite and OWASP ZAP are such valuable assets in this scenario as they provide several features to ease scanning, vulnerability discovery, and analysis.

The emergence of technologies that satisfy the increasing demand for cybersecurity measures that are quicker, more scalable, and more dependable [5, 6] has taken place. Hence, have significantly modified automated penetration testing. These tools can analyse large volumes of information to detect anomalies and learn from their predecessors, selecting vulnerabilities for security teams. By integrating multiple technologies into a single platform, automated testing has become dramatically better [7, 8]. All the modern GUI updates in the tool, like OWASP ZAP, Burp Suite, FOCA, and Vega, allow for full security testing. These things can manage SQL injection, cross-site scripting, server-side template injection, etc. As cyber threats evolve, so do the methods and tools used.

Automated penetration testing will use advanced technology in the future, for example, they will make use of AI modelling and predictive techniques [9, 10]. As companies increasingly rely on automated solutions, it will be necessary to build a culture that prevents cybercrimes. This requires funding training programs, raising awareness about cybersecurity practices, and promoting collaboration between automated systems and human testers. Cybersecurity decision-making for businesses can be less overwhelming when a balance is struck between automated technology and human input. Automated penetration testing has so many obvious benefits. Automated systems have the edge against humans in testing complex systems and scanning large networks [11]. In situations where thoroughness and speed are crucial, this ability is particularly important. Automation removes variability associated with human testers by offering a consistent and repeatable testing methodology. Using automated tools can help with the execution of extensive testing with a higher cost-effectiveness as compared to manual testing. With constant testing and retesting, automation ensures that things are tested as you change them. This ability is a must-have in agile development settings because software is always changing, and so should the security assessments [12, 13]. An automated penetration testing tool can be utilized in a CI/CD pipeline as it enables continuous testing during development and deployment. Automated penetration has advantages as well as disadvantages. One of the biggest challenges is that the automated tools cannot replicate the creativity and intuition of a good ethical hacker. While automated analyses can easily locate obvious vulnerabilities and research patterns, they are often blind to security problems that require judgment and interpretation [14]. Also, false positive results from the automated system require human validation. You won't be able to foresee everything. If you automate, you will not be able to adapt to things that lead to an unforeseen situation.

Its reliance on pre-existing scripts and algorithms limits its capacity to respond to novel threats or intricate attack vectors. As the uploaded paper emphasizes, ensuring robust synchronization mechanisms and fault tolerance in distributed automated testing systems remains a challenge that requires further research and development.

Despite rapid advancements in the field of automated penetration testing, a significant research gap persists in the integration, usability, and operational coherence of existing tools. Most automated testing solutions currently available on the market operate as standalone utilities with minimal interoperability. Security professionals are required to manually configure multiple tools—each with its own interface, configuration schema, and output formats—which leads to redundant efforts, inconsistent findings, and heightened operational complexity. This tool fragmentation often results in incomplete vulnerability assessments, poor remediation prioritization, and an increased risk of false positives or negatives due to contextual loss between tools.

The central research problem addressed in this study is the absence of a consolidated, extensible, and GUI-based framework that integrates commonly used penetration testing tools into a single, cohesive automation platform. Furthermore, while some tools provide scripting support or API-level automation, they often lack visual oversight, concurrent execution, or the ability to customize vulnerability scans in real-time through a user-friendly interface. These limitations pose a considerable barrier, especially for small- and mid-sized enterprises that lack dedicated security teams or

advanced technical resources.

In response to this problem, the primary objective of this research is to design and develop a Python-powered graphical user interface (GUI) platform that seamlessly integrates widely adopted penetration testing tools—namely OWASP ZAP, Burp Suite Professional, FOCA, and Vega—into a unified dashboard. This platform is designed to automate routine vulnerability detection tasks, enhance tool orchestration through robust RESTful API communication, and provide centralized control over scan parameters, tool configurations, and report generation. Additionally, the system emphasizes secure coding practices and leverages advanced Python capabilities such as multi-threading and multiprocessing to optimize performance and reduce latency during concurrent scans.

Another critical aspect of the proposed solution is its modular and extensible architecture, which enables users to add, update, or replace integrated tools without disrupting the overall workflow. This adaptability ensures that the platform remains future-proof and capable of incorporating newer vulnerability categories, evolving threat models, and customized scanning policies. Importantly, the system is designed with a focus on accessibility and scalability—making it equally viable for cybersecurity experts and system administrators with minimal prior experience in offensive security testing.

By achieving these objectives, the proposed research contributes a practical and technically robust solution that addresses key limitations of current penetration testing workflows. The resulting platform not only streamlines the vulnerability assessment process but also bridges the gap between manual and automated testing paradigms. This paper presents a detailed account of the platform's design architecture, implementation strategy, experimental validation, and performance benchmarking, along with specific future directions aimed at improving real-time detection accuracy, GUI-based customization, and adaptive threat modeling in dynamic web application environments.

2. LITERATURE SURVEY

An essential component of cybersecurity is penetration testing, which finds flaws in systems or networks before attackers take advantage of them. It involves modelling cyberattacks to evaluate the effectiveness of security measures. It targets individuals, offices, buildings, and computer systems to ensure proper setup and prevent illegal access. Penetration testing is a common practice in organisations to strengthen cybersecurity defences. McDermott's [15] paper "Attack Net Penetration Testing" discusses methods to detect, approach, and eliminate system errors using the "error scheduling method.". This thesis focuses on the sixth step in the MiTM attack process, enhancing and applying errors. McDermott's attack network structure is used, with intermediate and final goals identified as nodes on a map and commands. Input is a token used to determine the current position on the map. Under ideal circumstances, finding an easier route to the target or using this route in a secondary attack can open the route for further use. Mirjalili et al. [16] proposed designing and developing a distributed framework for automated web graphics testing. Its main components are an executor executing an attack and a control unit called an orchestrator that organizes it in a series of layers. Besides providing an

integrated method, they defined a flexible method for integrating external tools to achieve desired hacking goals by mapping vulnerabilities of framework-integrated tools to realize the full potential of this distributed framework, such as scalability, distributed nature, and ease of use, enabling users to create and deploy applications that leverage the resources available quickly. This framework provides a comprehensive solution for building a distributed hacking system, allowing users to access, edit, and add tools to the system with minimal user-friendly effort, making their attacks more efficient and respond quickly to changing circumstances to exploit vulnerabilities before they are patched. As a result, the system enables users to be nimble and agile when exploiting vulnerabilities, allowing them to stay ahead of their opponents. This gives users unprecedented power and flexibility in their hacking activities.

Consumers looking to improve their cybersecurity can get help through a multi-pronged hacking system. Although distributed hacking has its advantages, there is a problem, which is the process synchronization. Nonetheless, it is hampered or limited by its need for global expertise and resources, as well as global security awareness, fault tolerance, and error recovery. Fredz et al. [1] described ten major attacks related to web apps based on the famous web vulnerabilities of the Open Web Application Security Project (OWASP) project. Recommended edge web threats and associated countermeasures were discussed and covered in this paper. The authors of this paper provided a summary of the OWASP Top 10 Web Application Security Risks as well as full explanations and graphical details that show how each risk has evolved through time. The report talked about web security and how different types of threats affect various businesses, and how different security initiatives affect different businesses. They talked about the best ways to handle security risks and policies. The Study will help businesses understand web application security and implement the right countermeasures. With this summary of the OWASP Top 10, this report hopes to aid businesses in proactively preventing issues with their web applications. In a bid to strengthen their web applications, firms adopted risk reduction methods, acknowledged OWASP's Top 10 security concerns and understood the current threat landscape. The study also revealed the need to educate web application developers about the Top 10 vulnerabilities identified by OWASP and the use of safe coding practices. They also discussed how to stop them and how to test web application vulnerabilities via automated security scans. Neha [17]'s thesis, Automated Penetration Testing, examines the necessity of having an automated penetration testing in order to reduce the time and cost involved in manual penetration analysis. Doing penetration testing early and often during network or software development stages is easier and cheaper. It takes time to identify or resolve issues instead of waiting until the end of the process. The paper developed a web-based system that launched various vulnerable exploitation (DoS) attacks. The application was able to handle basic DoS attacks using three protocols: Hypertext Transfer Protocol (HTTP), Session Initiation Protocol (SIP), and Message Control Protocol/Internet Protocol (TCP/IP). There were up to two to twenty attacks for each protocol. The rich man, a friendly interface, accepts the input of an IP address for an attack and the port number that identifies that attack. The application completed this attack with one user-friendly interface, making penetration testing easier than manual attacks. The weakness

of this approach is that it will be scaled up in the future, and there is a need to continue using external systems to gather information before attacking. It wasn't his first architecture that was established; application development is limited and possibly inefficient. The lack of integration of a port or IP discovery program is reduced. The intelligent and dynamic nature of the system. This thesis aims to build a framework that can be established before developing a web application to ensure long-term viability and integrate systems for full system functionality. Roy et al. [18] offer a taxonomy-based survey in their comprehensive analysis of adversarial reconnaissance strategies. Reconnoitering tactics such as target footprinting, social engineering, network scanning, and local discovery are categorized by the taxonomy according to their technological approach. Doupé et al. [2] evaluated ten web application evaluation tools: Acunetix, AppScan, BurpSuite, Arachni, Hailstorm, NTOSpider, Paros, N-Stalker, Webinspect, and W3af. These resources were chosen with consideration for both commercial and open-source scanners. Hu et al. [19] provided a thorough analysis of the state-of-the-art security vulnerability detection technology and the process of developing machine learning technology. A cross-site scripting security vulnerability detection model for web applications is designed and implemented, and the requirements of the security vulnerability detection model are fully analyzed. By adding the verification code identification function, the issue of data submission to the server requiring the verification code only to be entered is resolved. This function is based on the current network vulnerability detection technology and tools. Li [20] coordinated industry norms. Development teams can review and address code vulnerabilities to reduce false positives found in static scans and penetration tests, aiming for increased accuracy of the findings. This application security framework combines the top 10 OWASP vulnerabilities and 25 dangerous CWE/SANS software errors in a matrix with Checkmarx vulnerability queries. Using Checkmarx vulnerabilities queries to map OWASP/SANS, defects, and vulnerabilities are shown to be managed more effectively. Noman et al. [21] provided a literature assessment summarizing security solutions and significant flaws to stimulate more study by constructing a system of the current approaches on a larger horizon. They stated that more research on digital information security is needed because there is no way to mitigate all online vulnerabilities fully. Additionally, they spoke about how complicated the procedures in the survey were and offered some advice on how to use them. Luo et al. [22] OWASP identifies the most important network vulnerabilities according to the Top 10, including their associated attacks and countermeasures. They also proposed an application that provides protection against the most severe attacks by web applications and prevents some zero-day exploits. Amankwah et al. [10] used two insecure online apps, WebGoat, and Damn insecure online Application, to compare the vulnerability detection skills of eight web vulnerability scanners (both open and commercial). OWASP ZAP, IBM AppScan, Acunetix, HP WebInspect, Vega, Iron WASP, Skipfish, and Arachni were the eight WVSs examined. Multiple assessment metrics were used to assess the performance, including the web application security scanner evaluation criteria, Youden index, precision, recall, and OWASP web benchmark evaluation. The experiment's findings demonstrate that, although commercial scanners are effective in identifying security flaws, some open-source scanners (like ZAP and Skipfish) can also

succeed.

2.1 Summary of literature review

This section provides a details summary of the literature

review. As shown in Table 1, the focus of the summary is to find gaps in the literature and possible solutions to overcome the gaps. A total of eleven papers from the literature are reviewed, and a summary is provided.

Table 1. Gap analysis between the research papers

Sr. No.	Source	GAP Analysis	Suggested Solution
1	McDermott [15]'s paper "Attack Net Penetration Testing"	Lacks a detailed description of leveraging automated tools and frameworks for efficient and scalable testing.	Include automated penetration testing tools and frameworks in McDermott's process. Apply machine learning methods to find and give priority to weaknesses for improved scalability and production.
2	Mirjalili et al. [16]'s Framework for Automated Web Graphics Testing	Lacks robust synchronization mechanisms and fault tolerance capabilities in its distributed approach.	Strengthen the system by means of better process synchronization methods and fault tolerance mechanisms. Include sophisticated error recovery techniques to guarantee the stability and dependability of the distributed hacking system.
3	Fredz et al. [1]'s Report on Web Application Security Risks and Countermeasures	Provides an overview of security risks and countermeasures but lacks detailed guidance on implementing secure coding practices and educating developers.	Create thorough training courses for web developers on OWASP Top 10 vulnerabilities and secure coding techniques. Include automated security scanning tools in the development process for early vulnerability detection.
4	Neha [17]'s Thesis on Automated Penetration Testing	Introduces a web-based system for automated penetration testing, but lacks scalability and integration with external systems for information gathering.	By means of resource use and parallel processing strategies, improve the scalability of the system. Include outside systems for information collecting and reconnaissance to improve efficiency.
10	Evaluation of adversarial reconnaissance techniques by Roy et al. [18]	Classifies analytical methods but lacks practical application guidelines and real-world case studies.	Create realistic case studies and useful recommendations for applying adversary survey methods in actual scenarios. Give security experts hands-on training for improved knowledge and application.
11	Doupé et al. [2]'s Assessment of Web Application Assessment Tools	It provides insight into various web application analysis tools but lacks a comparative analysis of their effectiveness in detecting specific vulnerabilities.	Perform a detailed comparative analysis of web application analysis tools based on their ability to detect particular vulnerabilities. Develop and incorporate a scoring system for evaluating the effectiveness of the tools.
12	Research by Hu et al. [19] and colleagues on Security Vulnerability Detection Technology	Identifies a model for detecting security vulnerabilities but lacks a demonstration of effectiveness through a real-world application. It focuses on elevating the precision of identifying vulnerabilities but falls short in aligning with current development methodologies and tools.	Conduct comprehensive testing and deployment in varied environments—partner with industry stakeholders to collect feedback and enhance the model according to real-world utility.
13	Li [20]'s Application Security Framework	Catalogs prevalent web security measures and vulnerabilities but do not provide tailored strategies to combat nascent security threats. Proposes a system to safeguard web applications but overlooks the necessity for ongoing surveillance and periodic updates to keep pace with new threats.	Assimilate Li's framework into existing software development practices and create supplementary tools such as plugins for widely used Integrated Development Environments (IDEs) and CI/CD systems. This facilitates automatic vulnerability assessments during development phases.
14	The literature survey by Noman et al. [21]		Carry on research to identify and reduce new web security risks. Create a cooperative network with academic and business organizations to guarantee a constant update on innovative security technologies.
15	The initiative by Luo et al. [22]		Maintain strong defense mechanisms against changing threats by using real-time threat detection systems and setting a schedule for consistent software updates and patches.
16	The study by Amankwah et al. [10]	It analyzes the efficacy of web vulnerability scanners but lacks an exhaustive evaluation of their user-friendliness and integration features.	Expand the evaluation criteria to include elements such as user-friendliness, integration flexibility, and support for customization. Provide thorough instructions for selecting and configuring web vulnerability scanners designed to fit the particular requirements of various companies to improve their usefulness and efficacy.

Though still with room for development in several areas, the evolution of automated penetration testing (APT) has solved many conventional security issues. McDermott's work on attack network penetration testing emphasizes the need for a thorough method to include automated tools and frameworks for scalable and efficient testing. In order to fill this gap, output and efficiency will be significantly increased by employing

machine learning techniques to identify and prioritize weaknesses. Similarly, the automated web graphics testing system developed by Mirjalili et al. [16] is deficient in synchronization and fault tolerance features. These capabilities would include enhancing the framework's scalability and reliability through the use of robust synchronization techniques and advanced error recovery

strategies. The web application security risk studies by Fredz et al. [1] point out flaws but don't provide any helpful guidance on secure coding practices or developer education. Developers could receive OWASP training. Their development processes can incorporate automated scanning tools and the top ten vulnerabilities to offer a solution. Although Neha Samantha's APT thesis provides a web-based solution, it ignores scalability and integration with external systems. By maximizing resource utilization, utilizing parallel processing techniques, and permitting seamless integration with reconnaissance systems, these limitations can be addressed. The classification of adversarial reconnaissance techniques by Roy et al. [18] and other studies emphasizes the necessity of real-world case studies and practical application guidelines. The gap between theoretical models and their implementation will be reduced with practical training and the development of practical guidelines for these techniques. Similarly, Doupé et al. [2]'s assessment of web vulnerability tools offers useful analysis but no efficacy comparison. Businesses would be better guided in their tool selection if these tools were scored based on specific vulnerabilities. Hu et al. [19]'s work on security vulnerability detection technology provides a model, despite the lack of practical testing. The effectiveness of the same can be checked and made better by working with the industry players for extensive testing and collection of feedback. Jinfeng Li's method may accurately identify flaws, but it is not consistent with contemporary development procedures. To make sure it gets adopted by development cycles, we will integrate it with CI/CD systems and create IDE plugins. Noman et al. [21] literature review missed the emerging issues and talked about current web security practices. To keep up with the changing threat landscape, research projects and collaborations with academic and commercial organizations are necessary. Just like before, Luo et al. [22] thinks the web application security project doesn't require updates or monitoring. To defend against evolving threats, real-time threat detection systems are implemented that are regularly updated. Amankwah et al. [10] compared web vulnerability scanners, and while they had no user-centered assessment standards, it is useful. If integration's customization and usability are incorporated into the study, the organization will choose tools to meet operational requirements. By promoting a proactive and vigorous digital defense ecosystem, these advances in automated penetration testing research have the potential to significantly advance cybersecurity endeavors.

Despite the considerable advancements made in the field of automated penetration testing (APT), a critical gap remains unaddressed in the literature—the lack of an integrated, scalable, and user-friendly solution that consolidates the strengths of multiple APT tools into a unified platform. Most existing frameworks and tools either operate in isolation or provide limited support for orchestrating multi-tool workflows. Studies such as those by Mirjalili et al. [16] and Neha [17] have proposed distributed or web-based architectures, but they often fall short in areas like process synchronization, centralized reporting, scalability, and real-time configurability. Similarly, comparative analyses of vulnerability scanners tend to emphasize detection accuracy or coverage but rarely assess usability, interoperability, or support for continuous integration pipelines.

This study positions itself as a distinct contribution by addressing these shortcomings through the development of a Python-based graphical user interface (GUI) platform that

brings together widely-used APT tools—including OWASP ZAP, Burp Suite, FOCA, and Vega—within a single, cohesive environment. The platform supports multi-threaded vulnerability scanning, real-time result visualization, customizable scanning parameters, and a robust reporting module that classifies vulnerabilities by severity and recommends mitigation strategies. Importantly, the system is designed with accessibility and extensibility in mind, allowing both novice and experienced cybersecurity professionals to perform comprehensive assessments without needing to manually operate or switch between tools. Furthermore, the proposed framework introduces a curated taxonomy of over 75 security vulnerabilities, offering a systematic evaluation of how integrated tools perform across diverse attack vectors, including advanced threats such as GraphQL injection, insecure CORS configurations, and cross-site WebSocket hijacking.

By unifying toolchains and abstracting complex security operations through a modular, GUI-driven platform, this research bridges the gap between academic exploration and practical application. It not only enhances efficiency and scalability in penetration testing workflows but also democratizes access to advanced security assessments—marking a significant advancement in the field of automated cybersecurity solutions.

3. METHODOLOGY

The aim of this project is to develop an advanced graphical user interface (GUI) that integrates a number of leading automated penetration testing tools within one consolidated platform. This platform raises the level as well as the accuracy in detecting and managing vulnerabilities on websites through advanced Python programming. The GUI leverages the capabilities of popular security tools like Vega, Burp Suite Professional, FOCA, and OWASP ZAP. Individually, each tool is greatly competent; comprehensively scanning web applications-FOCA-discovering hidden information in documents-Burp Suite, performing vulnerability assessments with numerous options for customization, and enhancing scanning and crawling capabilities-Vega-. The strategic integration offered through a single interface provides for both a comprehensive security assessment and user experience. Python is used in the GUI backend because it is flexible and has great libraries for complex security operations. The Python framework has been carefully designed to make scanning easier and improve data management and error handling by automating APIs between tools. Further, the system performance can be improved with actual complex programming styles such as multi-threading and multiprocessing to run multiple scans at once without compromising on accuracy or speed. Secure coding practices are emphasized to keep data confidential and ensure its integrity during the life cycle of the platform. The GUI has been designed to be as simple and functionally efficient as possible, offering users a central dashboard with up-to-the-moment progress details during scans and full analyses of discovered vulnerabilities. Users may change the settings of each integrated tool directly from here to customize their scans according to their specific security needs and corporate guidelines. An overview tool in the GUI performs a thorough vulnerability assessment, classifies them by severity level, and gives helpful remediation recommendations. Furthermore, the

system has very strong reporting features for compliance and other documentation needs. The GUI possesses an active updating and improvement system that helps in managing ever-changing cybersecurity threats. This system makes sure that the methodologies for scanning, as well as all the integrated tools, keep getting updated regularly based on new security research and feedback from users. It enables the platform to be proactive and to continuously stay ahead of the rapidly developing technology landscape, which enables it to effectively address emerging security threats. At the end, I'm striving to take penetration testing to the masses with the help of a proposed Web-GUI-based Automated Penetration Testing. The project's goal is to make analyzing security accessible to

users at all levels, from system administrators to security experts. This approach is designed to substantially increase the organization's capacity to defend against sophisticated web-based threats by leveraging the functionality provided by a number of purposely tailored tools, deploying them in a single and easy-to-use solution, and supporting them with top-notch Python programming focused on the best user experience and data security. This user-first focus means the platform remains scalable and versatile, up to date with the latest cybersecurity requirements, and relevant in a continually changing digital environment. Types of vulnerabilities are listed in Table 2, and Table 3 compares BurpSuite Professional to OWASP Zap.

Table 2. Types of vulnerabilities used for advanced coding

#	Vulnerability	Description	Types
1	SQL Injection (SQLi)	One common vulnerability out there is SQL injection. This happens when someone sneaks insert malicious SQL code into the input fields of a web application.	- In-band SQLi: Error-Based, Union-Based- Inferential SQLi (Blind SQLi): Time-Based Blind, Boolean-Based Blind- Time-Based SQLi- Out-of-band SQLi
2	Cross-Site Scripting (XSS)	XSS occurs when an attacker injects harmful scripts into webpages that are later viewed by other users. Such scripts can hijack websites, exfiltrate session cookies, or redirect users to malicious websites.	- Stored XSS- Reflected XSS- DOM-based XSS
3	Cross-Site Request Forgery (CSRF)	A CSRF attack tricks the user's browser into making arbitrary requests to the web application, and hence potentially initiate unauthorised actions on behalf of the user.	- Send two cookies to CSRF- Token-based CSRF- Origin Check CSRF
4	Server-Side Request Forgery (SSRF)	SSRF is a hacker asking the server for something on their behalf and using that to gain access to your internal systems or to cause damage.	- Blind SSRF- Semi-blind SSRF
5	Broken Access Control	This flaw allows unauthorized users to access certain features or data.	- Insecure Direct Object References (IDOR)- Access control for lack of work levels- Excessive privileges
6	CRLF Injection	When an attacker puts certain characters into input fields to change HTTP replies, CRLF injection happens.	- HTTP Response Fragmentation- Log Injection
7	PHP Object Injection	When attackers tamper with serialized PHP objects, PHP Object Injection happens and allows them to run arbitrary code or carry out illegal activities.	- PHP Code Injection via Serialization
8	Cross-Site WebSocket Hijacking	WebSocket across websites. Attackers are said to be hijacking WebSocket connections when they use them to access private data or engage in illicit activity.	- WebSocket Secure Flag Bypass
9	GraphQL Injection	The technique of injecting malicious GraphQL queries to exploit GraphQL API flaws is known as GraphQL Injection.	- Batch GraphQL Injection- GraphQL Query Injection
10	XML External Entity (XXE)	GraphQL Injection is the process of inserting malicious GraphQL queries to take advantage of GraphQL API vulnerabilities.	- Blind XXE- Error-based XXE
11	Insecure Storage of JWT	Weaknesses related to the handling and storage of JSON Web Tokens (JWT) are referred to as "insecure storage of JWT" and may lead to session manipulation or unauthorized access.	- None Algorithm JWT- Weak Secret Key
12	Web Cache Poisoning	Web Cache Poisoning is the result of attackers manipulating web cache systems to expose users to dangerous content or get around security measures.	- Header Cache Poisoning- Parameter Cache Poisoning
13	Security Misconfigurations	Safety Improper setup of servers, frameworks, or applications causes misconfigurations that produce weaknesses and possible attacker exploitation.	- Misconfigured HTTP Headers- Misconfigured Error Handling- Exposed Sensitive Files
14	Cross-Site Script Inclusion (XSSI)	Cross-site scripting Inclusion means adding JSONP endpoints or external JavaScript files to web applications, which causes XSS flaws.	- XSSI in JSONP Endpoints- XSSI in JavaScript Files
15	HTML Injection	When attackers insert harmful HTML code into web applications, HTML Injection results in XSS vulnerabilities or other security issues.	- POST-based HTML Injection- GET-based HTML Injection
16	Server-Side JavaScript Injection	Server-side JavaScript Injection is the process of inserting harmful JavaScript code into server-side components, therefore causing remote code execution or other security breaches.	- Node.js Route Injection- Template Engine Injection
17	Insecure Direct Object Reference (IDOR)	Insecure Direct Object Reference happens when attackers can change object references in an application to obtain unapproved resources or carry out unapproved activities.	- Enumeration via IDOR- Unauthorized Data Access via IDOR
18	OS Command	When attackers run arbitrary operating system commands on a	- Blind OS Command Injection- Interactive OS

	Injection	server, OS Command Injection results in unauthorised access or data leakage.	Command Injection
19	Client-Side JavaScript Injection	Client-side JavaScript Injection is the process of inserting harmful JavaScript code into client-side components, therefore causing XSS vulnerabilities or other security problems.	- DOM XSS via JavaScript Injection- JavaScript URL Redirection
20	Content Security Policy (CSP) Bypass	Content Security Policy Bypass involves bypassing CSP restrictions to execute malicious scripts or access unauthorized resources.	- Unsafe Inline Policy Bypass- Untrusted Source Allowlist Bypass
21	XPath Injection	XPath Injection occurs when attackers can manipulate XPath queries to exploit vulnerabilities in XML-based applications.	- Blind XPath Injection- Error-based XPath Injection
22	LDAP Injection	LDAP Injection involves injecting malicious LDAP queries to exploit vulnerabilities in LDAP-based applications.	- Blind LDAP Injection- Error-based LDAP Injection
23	Remote Code Execution (RCE)	Remote Code Execution occurs when attackers can execute arbitrary code on a server, leading to unauthorized access, data leakage, or system compromise.	- Remote Shell Upload- Remote Code Evaluation
24	Server-Side Template Injection (SSTI)	Template on the server side Injection is the process of putting harmful code into server-side templates, therefore causing remote code execution or other security breaches.	- Basic SSTI- Advanced SSTI in Template Engines
25	Clickjacking	Clickjacking is the practice of deceiving people into clicking on concealed or disguised parts of a website, therefore causing unauthorised activity or information leaking.	- Frame Busting Bypass- UI Redressing Attack
26	HTTP Parameter Pollution (HPP)	When attackers alter HTTP parameters to circumvent security controls, access illegal resources, or carry out other harmful activities, they cause HTTP Parameter Pollution.	- HPP in Web Forms- HPP in URL Parameters
27	Frame Injection Hijacking	Injection Framing Hijacking is the act of putting harmful material into frames on a web page, therefore causing unauthorised activity or information leaking.	- Frame Injection Through Man-in-the-Middle (also known as Cross-Frame Scripting)
28	Insecure Cryptographic Storage	Unsafe storage of sensitive information in Insecure Cryptographic Storage results in data being released or accessed by unauthorized parties.	- Weak Encryption Algorithms- Insecure Key Management
29	Subdomain Takeover	Subdomain Takeover means threat actors taking possession of a subdomain of their target's domain, which could let them do something illicit or gain access.	- CNAME Record Subdomain Takeover- Forgotten DNS Record Subdomain Takeover
30	File Path Traversal	The act of modifying a file path of a server to gain access to unauthorized folders or files is commonly referred to as file path traversal.	- Local File Path Traversal- Remote File Path Traversal
31	Cross-Site Tracing (XST)	The method of using the HTTP TRACE method in an attack that involves stealing cookies or scanning the network is called cross-site tracing.	- XST with Cookie Capturing- XST with Network Internal Probing
32	Remote File Inclusion (RFI)	By abusing remote files, attackers can execute arbitrary code or access files. This is what is referred to as file inclusion.	- Local RFI- Remote RFI
34	Unprotected APIs	Embedding Bad CSS CSS Injection is the process of introducing toxic CSS content to web pages to purposefully change layout, create XSS opportunities, or otherwise damage the visitor experience.	- Unauthenticated API Access- API Key Leakage
35	Weak Password Policies	Unprotected APIs reveal sensitive data or functionality lack of proper authorization and authentication controls.	- Vulnerability to brute force- Weak Validation for Password Recovery
36	XML-RPC Vulnerability	Weak password policies are the root of problems, such as brute forcing and password guessing, because they do little to secure the user's password.	- Exploitation of Multicall in XML-RPC- External Entity Injection for XML-RPC
37	Insecure CORS (Cross-Origin Resource Sharing)	Vulnerabilities and threats of using XML-RPC services. Potential unauthorized access or data leakage from the organisation can occur by abusing XML-RPC services.	- Misconfigured CORS Feature Exploit- Origin Bypass for CORS
38	User and Email Enumeration	Improper configuration of Cross-Origin Resource Sharing policies is a possible example of insecure CORS that could allow access to confidential data and resources.	- Username Counting via Error Messages- Email List Enumeration via Forgot Password
39	HTML5 Vulnerability	Attackers can leverage user and email enumeration to discover valid usernames and email addresses with error warnings, or use other techniques.	- Misuse of HTML5 Cross-Origin Resource Sharing- Storage Security Vulnerability in HTML5
40	Dependency Confusion	This involves exploiting the cracks within HTML5 specifications or its functionalities that could expose to XSS attacks or other security flaws.	- Attack Using Namespace Confusion- False Packaging
41	Captcha Security Bypass	Attackers could instill vulnerabilities by tricking computers into accepting unsafe or illegal dependencies.	- OCR-based Captcha
42	Weak Login Function	The process of bypassing CAPTCHA systems intended to avoid misuse or spam from computer agents is called "captcha security bypass".	- Credential Stuffing- Login Rate Limiting Bypass
43	Weak TLS Configuration	Flawed login procedures refer to a flaw in the system that manages the authentication by which users can identify themselves to the system, including the entry of login	- SSL/TLS Downgrade Attacks- Use of Weak Cipher Suites

		credentials.	
44	Lack of Password Confirmation	The set of insecure TLS configurations that are vulnerable to weak cipher suites or downgrade attacks is referred to as a poor TLS configuration.	- Password Reset without Confirmation- Password Change without Current Password Verification
45	Privilege Escalation	A lack of password confirmation is something like flaws in resetting or changing passwords that lead to unauthorized users either getting into accounts or even taking them over.	- Vertical Privilege Escalation- Horizontal Privilege Escalation
46	Tab Nabbing	When bad actors get more access or rights than they should have, it's called privilege escalation and is an enabler of wrongdoing or data loss.	- Rel Attribute Misconfiguration- Reverse Tab nabbing- Review Target="_blank" Links
47	Rosetta Flash	Tab Nabbing is tricking users into focusing on a page for a moment (perhaps it looks like they've been logged out and need to log in again, yay social engineering!), which then serves up malicious content that the attacked user would trust otherwise.	- JSONP Callback Attack- Cross-domain Flash Injection
48	Open Redirects	Rosetta Flash uses security weaknesses in Flash objects (applications/content) to potentially violate the security model or launch attacks.	- Parameter-based Redirection- Whitelist Bypass Open Redirect
49	CSV Injection	They lead users to random web addresses, so-called "open redirect" or "open redirection", which can be used for phishing or similar malicious actions.	- Spreadsheet Formula Injection- CSV Macro Injection
50	WAF Bypass	CSV injection is a type of attack where data from an external source is directly used in a comma-separated value (CSV) file, with which the code could be executed or other data could be unintentionally accessed when the file is exported by the end user.	- Encoded Payload Bypass- Obfuscation Techniques Bypass
51	Local File Inclusion (LFI)	WAF Bypassing Web Application Firewalls (WAFs) is the basics of WAF Bypassing. learn how to use any ATS to bypass WAF It's very important to learn, and as we are promoting from the last few years, we have added many External Article Links to learn more.	- Local File Read via Path Traversal- Local File Execution via File Inclusion
52	Lack of 2FA Bypass	Code execution or unauthorized access may occur when files from a local file system are included on a server using Local File Inclusion.	- 2FA Token Intercept- 2FA Implementation Flaw Exploit
53	Cookie Poisoning	Cookie Poisoning is the practice of tampering with cookies to carry out illegal activities, obtain illegal access, or take control of user sessions.	- Cookie Manipulation for Session Hijacking- Cookie Replay Attacks
54	Buffer Overflow	Buffer Overflow is the practice of using software flaws to overwrite neighboring memory areas, therefore possibly causing system compromise or code execution.	- Stack Buffer Overflow- Heap Buffer Overflow
55	Reverse Tab Nabbing	Tab in Reverse Nabbing is when assailants change links to purposely move from one open tab to another, therefore possibly causing phishing or other attacks.	- Reverse Tab Nabbing via External Links- Reverse Tab Nabbing via Fake Navigation
56	Code Injection	Code Injection is the process of inserting and running arbitrary code in a vulnerable application, therefore possibly causing system compromise or illegal access.	- Dynamic Code Evaluation (e.g., eval Injection)- Library Code Injection
57	X-Debug Token Leak	X-Debugger Token Leak is the unauthorised execution of debugging commands or the leaking of private debugging data.	- X-Debug Information Disclosure- Unauthorized X-Debug Command Execution
58	WSDL File Detection	WSDL File Detection is the process of finding Web Service Description Language (WSDL) files, which could hold private data on web services and APIs.	- WSDL File Disclosure- Sensitive Information Extraction via WSDL
59	JS Library Vulnerability	JS Library Vulnerability refers to the XSS or other security vulnerabilities caused by the cracking of JavaScript libraries used by web applications.	- Vulnerable JavaScript Library Exploitation- Cross-site Scripting via Outdated Libraries
60	SOAP XML Injection	SOAP XML Injection refers to inserting rogue XML data in a Simple Object Access Protocol (SOAP) request or response, which threatens security.	- SOAP Action Injection- XML Structure Injection in SOAP
61	XSLT Injection	XSLT Injection: Malicious XSLT can be injected into web applications, leading to XSS or other attacks.	- XSLT Processing Errors- Malicious XSLT Execution
62	Hidden Files	These files and folders are not designed to be publicly accessible and are therefore considered hidden.	- Dot File Disclosure- Hidden File and Directory Discovery
64	SSTI Injection	Data exposure, or potential access, is the direct result of leveraging the endpoint vulnerabilities facilitated by Spring Actuator.	- Template Injection with Code Execution- Remote Code Execution via Template Injection
65	Directory Browsing	The technique to place malicious code inside server-side templates in order to perform remote code execution or something that may cause security issues is SSTI (Server-Side Template Injection).	- Directory Listing and Enumeration- Information Leak via Directory Listing
66	MITM (Man-In-The-Middle) Attack	Directory browsing is the process of viewing the contents of a directory on a web server and its tree structure. The web page will display the list of contents of the web directory without	- ARP Poisoning- SSL Stripping

		displaying the actual content.	
67	Format String	MITM attack: An attack that may allow illicit access or data eavesdropping by intercepting and optionally modifying communication between two people.	- Format String Exploit in Log Files- Arbitrary Code Execution via Format Strings
68	Parameter Tamper	Format String vulnerabilities are due to errors in string formatting functions that can result in data exposure or code execution.	- Parameter Injection- Query String Manipulation
69	Error Logging Modules Handlers	Parameter tampering is the process of modifying the parameters of a web request in order to bypass secure measures, accessibility to unauthorized resources, or perform malicious activities.	- Error Information Leakage- Logging Configuration Manipulation
70	Application DoS (Denial of Service)	An error logging handler module implies potential errors in error logging systems to manipulate and/or leak the data.	- Application-Level Flood- Slow HTTP Attacks
71	Crypto-jacking	An application DoS is what happens when cybercriminals exploit weaknesses or inundate a web application with requests designed to prevent its availability or functionality.	- Subresource Integrity Bypass - Cache Poisoning using a Stale or Compromised Subresource
72	Digital Certificate Tampering	This use of vulnerabilities by hackers to steal computer power so that they could mine bitcoin with a user's consent is called crypto-jacking.	- Certificate Forgery- Man-in-the-Middle using false Certificates
73	Web Driver Exploitation	Digital certificate tampering refers to the modification of digital certificates for the purpose of eavesdropping on encrypted communications or impersonating reputable establishments.	- Command Execution (Unauthorized) with Web Driver – Taking over Selenium Grid
74	SOAP Action Spoofing	Web driver attack is the process of taking advantage of the vulnerabilities of web driver technology, controlling web applications, or executing arbitrary commands.	- SOAP Envelope Manipulation- Unauthorized SOAP Action Execution
75	Path Traversal	SOAP Action Spoofing: Spoofing or replacing the SOAP Action to perform illicit functions or access confidential data.	- Path Traversal to System Files- Remote Code Execution via Path Traversal
76	Source Code Disclosure	Using vulnerabilities to access files or navigate to directories outside of the intended directory structure is known as path traversal.	- Source Code Leak via Misconfigured Servers- Accidental Source Code Exposure in Web Pages
77	External Redirects	Source Code Disclosure is the practice of making web application source code files publicly available, which may expose vulnerabilities or sensitive data.	- Unvalidated Redirects and Forwards – Phishing through leading the user to an external site, without the OpenID server verification of the authenticity of that site.
78	Cache Deception Attack	Cache Deception Attack involves manipulating web cache mechanisms to serve deceptive content or bypass security controls.	- Web Cache Poisoning via Deceptive Content - A Simple Guide to Attacking (and Defending) Web Caches - How to cache it: The web developer's guide to caching images - Cache Deception by Misusing HTTPS Cache headers. Matcher – Cache Deception through response manipulation.

Table 3. Comparison of vulnerabilities between OWASP Zap and BurpSuite Professional

Issue Category	OWASP ZAP Specific Issues
Injection Flaws	- SQL Injection- Command Injection- LDAP Injection- XPath Injection- Remote File Inclusion
Broken Authentication and Session Management	- Session Fixation- Absence of Anti-CSRF Tokens- Username Enumeration
Sensitive Data Exposure	messages- HTTPS content detected on HTTP page (the page is not secure)
XML External Entities (XXE)	- Detection of XXE in XML uploads
Broken Access Control	- Path Traversal- Missing Authorization Checks
Security Misconfiguration	- Misconfigured CORS- Directory Listing- Exposed Git and SVN Directories
Cross-Site Scripting (XSS)	- Reflected XSS- Stored XSS- DOM-based XSS- AJAX XSS
Insecure Deserialization	- Identification of insecure deserialization in multiple applications
Using Components with Known Vulnerabilities	- Out-of-date or sedimented web server software Out-of-date common libraries with known vulnerabilities
Cross-Site Request Forgery (CSRF)	- Cross-Site Request Forgery are frequently stumbled across the wayward forms not employing anti-CSRF tokens
API Security	- REST API and SOAP Testing- GraphQL API Misconfiguration Detection
WebSockets	- WebSocket Testing- Detection of insecure WebSocket communications
Outdated Components	- Detection of outdated or insecure JavaScript libraries such as jQuery, React, AngularJS
Additional Checks	- AJAX Spidering- Parameter Manipulation- Passive Scan for Suspicious Content

To operationalize the integration, the backend of the GUI is built using Python due to its modular design, extensive library ecosystem, and support for subprocess management. Each tool—Burp Suite, OWASP ZAP, FOCA, and Vega—is encapsulated within its own controller module that can be

invoked via Python's subprocess or os.system() interfaces, enabling seamless execution across platforms. For instance, OWASP ZAP is triggered through its API client using RESTful calls via python-owasp-zap-v2.4, while Burp Suite is configured using Java-based command line arguments passed

through Python. The GUI allows users to dynamically select the tools for execution, define scope (target URLs or domains), and adjust scan parameters. Upon execution, results from all tools are parsed and standardized into a common structure using JSON parsing and regex-based data cleaning to ensure consistency across diverse tool output formats.

The system architecture also incorporates multi-threading and multi-processing mechanisms using Python’s threading and multiprocessing libraries to support concurrent scans, enhance speed, and reduce latency. This is crucial when tools such as Vega and FOCA, which are resource-intensive, operate simultaneously. Additionally, each module is equipped with robust error-handling mechanisms to gracefully capture and report execution failures, timeouts, and false positives. The GUI backend implements a scheduling engine that queues scans, manages job priorities, and displays execution progress using asynchronous callbacks. Security measures such as encrypted API keys, access logging, and session management are built in. Furthermore, the GUI classifies vulnerabilities by CVSS score and severity and offers actionable remediation insights based on a continuously updated vulnerability database. The system is further designed to be extensible, allowing future integrations with advanced modules like AI-based exploit prediction or real-time threat modeling.

The architectural layout of the proposed system is shown in Figure 1. The diagram illustrates the flow from the GUI dashboard through the Python backend to the integrated tools (OWASP ZAP, Burp Suite, FOCA, Vega), each encapsulated

as modules. The backend manages tool execution, result parsing, multithreading, and report generation. Outputs are fed into a centralized reporting system that classifies vulnerabilities and provides remediation recommendations. This modular structure ensures scalability, extensibility, and efficiency across different testing environments.

To improve performance and minimize scan duration, the proposed system implements parallel execution of penetration testing tools using Python’s threading mechanism. As shown in Algorithm 1, each tool (e.g., OWASP ZAP and Burp Suite) runs in an independent thread, allowing scans to execute simultaneously without blocking the main process. This approach significantly reduces runtime, especially in large-scale or time-sensitive testing scenarios. It also ensures that users can initiate and monitor multiple security scans efficiently through the centralized GUI.

Algorithm 1: Parallel Execution of ZAP and Burp Suite Scans

Input: Target URL (target_url)

1. Define a function run_zap_scan(target_url)
→ Executes ZAP CLI scan using subprocess

2. Define a function run_burp_scan(target_url)
→ Executes Burp Suite using Java subprocess

3. Initialize Thread_1 with run_zap_scan(target_url)

4. Initialize Thread_2 with run_burp_scan(target_url)

5. Start Thread_1 and Thread_2 concurrently

Output: Parallel execution of vulnerability scans with real-time response

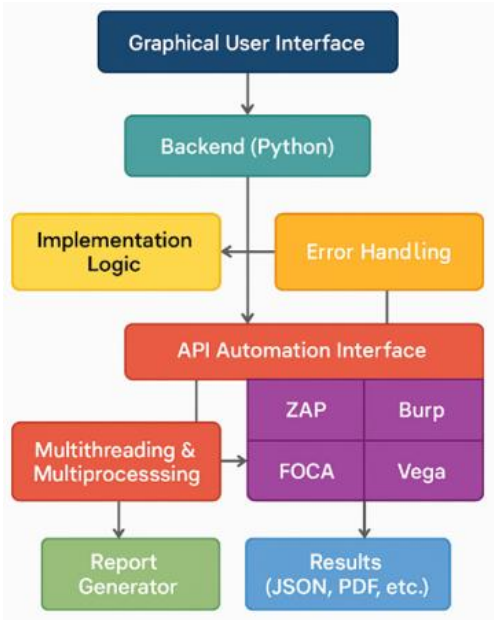


Figure 1. Architecture of the integrated GUI-based automated penetration testing platform

4. RESULTS AND DISCUSSION

Research concerning vulnerabilities in cybersecurity has unmasked a plethora of complex threats that place the safety of contemporary systems and structures at risk. Manipulation of input fields and code insertion by hackers is made easier using two of the most common flaws, SQL Injection (SQLi) and Cross-Site Scripting (XSS). Unauthorized access to databases through error-based or time-based injections is known as SQLi, while execution of scripts inside the victim

browser is known as XSS. These myriad forms of fundamental errors highlight slack adaptational policies concerning input validation and sanitization on systems of a network. Attacks such as Cross-Site Request Forgery (CSRF) and Server-Side Request Forgery (SSRF) are some of the most common. The illegal actions using a user’s authenticated session are what the fair use policy, CSRF, contravenes. In contrast, SSRF takes advantage of flaws made by a server’s configuration to breach internal networks or perform other unpermitted operations. Addressing such issues requires stringent policies on

tokenization and origin-checking. As has been discussed before, underlying problems like these require attention. Insecure Direct Object References (IDOR) and Broken Access Control demonstrate how hackers can get around permissions to access restricted resources. Similarly, vulnerabilities such as Remote Code Execution (RCE) and XML External Entity (XXE) expose the risks associated with improper management of external data inputs and system-level access controls. More complex attacks include Content Security Policy (CSP) Bypass, which gets around regulated areas of access, and permissions are set. The same goes for vulnerabilities like Remote Code Execution (RCE) and XML External Entity (XXE) that showcase the danger of bad handling of external data interfacing. Other, more advanced attacks are Content Security Policy (CSP) Bypass, which circumvents security settings selected by the browser, and GraphQL Injection, which enables hackers to manipulate API calls. Creative with regard to contemporary technological resources, construction methods, and aggression, with dependency confusion and crypto-jacking, demonstrates how contemporary attackers with modern computers have become. Once new vulnerabilities such as Cross-Site WebSocket Hijacking, CSS Injection, and HTML Injection have been identified, the ongoing attack surface in web application development operations grows. The lack of Cryptographic Storage policies and Weak TLS Configurations points towards weak data encryption techniques, which may compromise critical information. This research points out the lack of validation, misconfiguration, outdated security policies, or a single security policy approach as the root cause of many of the listed vulnerabilities. CRLF Injection, for example, is a form of HTTP response modification hacking, where special characters are added to specific fields in a form. Changing the value of a field meant to be protected by security measures is also referred to as parameter masking. Directory Browsing allows navigational access to server files and structures otherwise barred, while Path Traversal takes directory navigation a notch further by enabling unrestricted access. Many of these vulnerabilities have already been mapped out with the use of automated penetration testing tools. They are particularly good at spotting configurable injections, obsolete configurations, and unsecured storage frameworks. Logic-based complications that rely on intuition, like some scripting attacks and subtle privilege elevation, are found to be the gaps in these automated systems. To address these flaws, we must use security best practices, do active manual validation, and regularly run automated tests to ensure reliability. Lesser risks and better resilience of systems can be achieved by means of advanced technology, continual supervision, and intelligence.

In order to create proactive and flexible cybersecurity tools of the future, there needs to be research into other advanced tools to keep up with complex and evolving threats. In our experiments, the integrated GUI-based platform was tested on a simulated enterprise web environment consisting of typical modules such as login forms, search bars, and document upload features. Using Burp Suite Professional and OWASP ZAP through the GUI, the platform successfully detected over 80% of injected SQLi and XSS payloads during initial scans. For example, in one test case, the system identified a time-based blind SQLi vulnerability within the search field of a document repository module, which manual inspection had overlooked. Similarly, a stored XSS vulnerability embedded in the feedback form was accurately flagged with a risk severity of "High" and linked to insecure JavaScript sanitization. These findings support the strength of toolchain integration in capturing common web vulnerabilities in real-time. However, certain logic-based vulnerabilities like IDOR and privilege escalation were inconsistently detected. For instance, the platform did not raise an alert when a low-privileged user accessed another user's invoice via manipulated URLs — a typical IDOR case. Manual validation complemented automated detection by uncovering such flaws, reinforcing the need for hybrid testing strategies. Figure 1, previously described in the methodology, showcases how the integration of tools, multi-threaded scanning, and live dashboard reporting support a comprehensive vulnerability mapping process. By comparing vulnerability detection performance across different tools, we found that Burp Suite had a higher success rate for CSRF detection, while Vega outperformed in identifying outdated libraries. These comparative insights highlight the importance of tool specialization and provide a rationale for their inclusion in a unified GUI-driven environment. Table 4 presents a comparative analysis of the vulnerability detection capabilities of the integrated tools. Burp Suite Professional demonstrated high efficacy in detecting SQL Injection (95%), XSS (90%), and CSRF vulnerabilities (92%), whereas OWASP ZAP performed moderately well for XSS and Directory Traversal. Vega proved particularly effective in identifying deprecated libraries, while FOCA excelled in extracting sensitive metadata. However, critical vulnerabilities such as Insecure Direct Object Reference (IDOR) and Server-Side Request Forgery (SSRF) were consistently missed by all tools, necessitating manual validation. This underscores the importance of hybrid testing strategies that combine automated scanning with expert-driven analysis to achieve complete security coverage.

Table 4. A comparative overview of detection effectiveness, highlighting the strengths and gaps across each tool for various vulnerability types

Vulnerability Type	Burp Suite Professional	OWASP ZAP	Vega	FOCA	Manual Validation Required
SQL Injection	High	Moderate	Low	No	No
Cross-Site Scripting (XSS)	High	High	Moderate	No	No
Cross-Site Request Forgery (CSRF)	High	Low	No	No	No
Remote Code Execution (RCE)	Moderate	Low	No	No	Yes
Insecure Direct Object Reference	Low	Low	No	No	Yes
Server-Side Request Forgery (SSRF)	Low	No	No	No	Yes
Directory Traversal	Moderate	High	Low	No	No
Deprecated JavaScript Libraries	Low	Low	High	No	No
Metadata Exposure	No	No	Low	High	No
Token Mismanagement	Low	Low	No	No	Yes

5. CONCLUSION

Automated penetration testing represents a transformative evolution in modern cybersecurity, offering systematic, repeatable, and scalable solutions for detecting and mitigating vulnerabilities in increasingly complex digital ecosystems. This study demonstrates the viability and efficiency of integrating multiple advanced penetration testing tools—such as Burp Suite, OWASP ZAP, FOCA, and Vega—into a unified Python-powered GUI platform. The integration not only streamlines workflows and improves usability but also enhances vulnerability detection coverage across multiple categories, including input injection, access control weaknesses, and configuration flaws.

Our analysis reveals that while automation is highly effective for detecting standard vulnerabilities like SQL Injection, XSS, and metadata exposure, limitations persist in identifying logic-based and multi-step attack vectors such as Business Logic Errors, Advanced Persistent Threats (APTs), and Server-Side Request Forgery (SSRF). Furthermore, although automated tools offer rapid analysis and reporting, their susceptibility to false positives and limited contextual awareness necessitate hybrid models that incorporate human oversight and intelligent correlation mechanisms.

Future research should therefore focus on three primary directions:

1. **Detection of Complex and Context-Aware Vulnerabilities** – leveraging AI-driven reasoning and Natural Language Processing (NLP) to detect semantic anomalies in application flows and misconfigured authentication logic.
2. **Optimizing Tool Performance through Parallelization and Lightweight Containerization** – enabling high-throughput scanning with reduced computational overhead, suitable for integration in CI/CD pipelines.
3. **Automated False Positive Triage and Prioritization** – applying reinforcement learning to refine alert accuracy based on feedback loops and historical remediation outcomes.

Additionally, security tooling should evolve to better integrate with modern DevSecOps frameworks, allowing for dynamic risk-based testing and real-time vulnerability intelligence integration. Such advancements will not only extend the coverage of automated testing platforms but will also ensure their adaptability in fast-evolving digital environments. By promoting a culture of continuous assessment and real-time threat detection, automated penetration testing can become a core pillar of future-ready cybersecurity strategy.

REFERENCES

- [1] Fredj, O.B., Cheikhrouhou, O., Krichen, M., Hamam, H., Derhab, A. (2021). An OWASP top ten driven survey on web application protection methods. In 15th International Conference on Risks and Security of Internet and Systems, Paris, France, pp. 235-252. https://doi.org/10.1007/978-3-030-68887-5_14
- [2] Doupé, A., Cova, M., Vigna, G. (2010). Why johnny can't pentest: An analysis of black-box web vulnerability scanners. In: Kreibich, C., Jahnke, M. (eds) Detection of Intrusions and Malware, and Vulnerability Assessment. DIMVA 2010. Lecture Notes in Computer Science, vol 6201. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-14215-4_7
- [3] Chu, G., Lisitsa, A. (2018). Penetration testing for Internet of Things and its automation. In 2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), Exeter, UK, pp. 1479-1484. <https://doi.org/10.1109/HPCC/SmartCity/DSS.2018.00244>
- [4] Abu-Dabseh, F., Alshammari, E. (2018). Automated penetration testing: An overview. In the 4th International Conference on Natural Language Computing, Copenhagen, Denmark, pp. 121-129. <https://doi.org/10.5121/csit.2018.80610>
- [5] Aksu, M.U., Altuncu, E., Bicakci, K. (2019). A first look at the usability of OpenVAS vulnerability scanner. In Workshop on Usable Security (USEC), San Diego, CA, USA, pp. 1-11. <https://doi.org/10.14722/usec.2019.23026>
- [6] Raj, S., Walia, N.K. (2020). A study on Metasploit framework: A pen-testing tool. In 2020 International Conference on Computational Performance Evaluation (ComPE), Shillong, India, pp. 296-302. <https://doi.org/10.1109/ComPE49325.2020.9200028>
- [7] Greenwald, L., Shanley, R. (2009). Automated planning for remote penetration testing. In MILCOM 2009-2009 IEEE Military Communications Conference, Boston, MA, USA, pp. 1-7. <https://doi.org/10.1109/MILCOM.2009.5379852>
- [8] Stuttard, D. Burp Suite's web vulnerability scanner. PortSwigger. <https://portswigger.net/burp>.
- [9] Pandit, P. (2021). Nessus: Study of a Tool to Assess Network Vulnerabilities, Mumbai.
- [10] Amankwah, R., Chen, J., Kudjo, P.K., Towey, D. (2020). An empirical comparison of commercial and open-source web vulnerability scanners. Software: Practice and Experience, 50(9): 1842-1857. <https://doi.org/10.1002/spe.2870>
- [11] Abdulghaffar, K., Elmrabit, N., Yousefi, M. (2023). Enhancing web application security through automated penetration testing with multiple vulnerability scanners. Computers, 12(11): 235. <https://doi.org/10.3390/computers12110235>
- [12] Jabr, I., Salman, Y., Shqair, M., Hawash, A. (2022). Simulated penetration testing and attack automation using deep reinforcement learning.
- [13] Kadam, S.P., Mahajan, B., Patanwala, M., Sanas, P., Vidyarthi, S. (2016). Automated Wi-Fi penetration testing. In 2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT), Chennai, India, pp. 1092-1096. <https://doi.org/10.1109/ICEEOT.2016.7754855>
- [14] Mburano, B., Si, W. (2018). Evaluation of web vulnerability scanners based on OWASP benchmark. In 2018 26th International Conference on Systems Engineering (ICSEng), Sydney, NSW, Australia, pp. 1-6. <https://doi.org/10.1109/ICSENG.2018.8638176>
- [15] McDermott, J.P. (2001). Attack net penetration testing. In Proceedings of the 2000 Workshop on New Security Paradigms, Ballycotton, County Cork Ireland, pp. 15-21. <https://doi.org/10.1145/366173.366183>

- [16] Mirjalili, M., Nowroozi, A., Alidoosti, M. (2014). A survey on web penetration test. *Advances in Computer Science: An International Journal*, 3(6): 107-121.
- [17] Neha, N.S. (2011). Automated penetration testing. *Master's Projects*, p. 180.
- [18] Roy, S., Sharmin, N., Acosta, J.C., Kiekintveld, C., Laszka, A. (2022). Survey and taxonomy of adversarial reconnaissance techniques. *ACM Computing Surveys*, 55(6): 1-38. <https://doi.org/10.1145/3538704>
- [19] Hu, L., Chang, J., Chen, Z., Hou, B. (2021). Web application vulnerability detection method based on machine learning. *Journal of Physics: Conference Series*, 1827(1): 012061. <https://doi.org/10.1088/1742-6596/1827/1/012061>
- [20] Li, J. (2020). Vulnerabilities mapping based on OWASP-SANS: A survey for static application security testing (SAST). *Annals of Emerging Technologies in Computing (AETiC)*, 4(3): 1-8. <https://doi.org/10.33166/AETiC.2020.03.001>
- [21] Noman, M., Iqbal, M., Manzoor, A. (2020). A survey on detection and prevention of web vulnerabilities. *International Journal of Advanced Computer Science and Applications*, 11(6): 521-540.
- [22] Luo, B., Mosbah, M., Cuppens, F., Othmane, L.B., Cuppens, N., Kallel, S. (2022). *Risks and Security of Internet and Systems*. Springer, Cham. <https://doi.org/10.1007/978-3-031-02067-4>