The subject of this study is a tool for automating vulnerability detection using large language models, developed to reduce the time spent on conventional penetration testing. In addition, a detailed analysis has been conducted comparing the effectiveness of the automated approach with that of conventional manual security testing. The tool utilizes application programming interface access to LLMs, enabling the analysis of large volumes of data, the identification of complex relationships between system components, and the provision of interactive support to specialists during the testing process. By conducting experiments under actual conditions, the tool demonstrated the ability to integrate with popular penetration test tools and deal with real cyber threats, particularly in scenarios involving active attacks on networks and web applications. By automating routine tasks, such as configuration checks, analysis of tool outputs, and generating recommendations, the tool significantly reduces the workload on specialists. On average, the tool shortened the testing time by 54.4 % compared to a manual approach. Recall reached 94.7 % in network analysis scenarios but dropped to 66.7 % in web application testing, while the automated approach's precision ranged from 80 % to 90 %. The study results confirmed that the application of large language models in the penetration testing process significantly reduces the time required to complete tasks and improves the accuracy of vulnerability detection. The tool could be used both independently and in combination with other automation tools, making it a versatile solution for organizations of various sizes. Thus, the proposed solution is a substantial contribution to the development of modern cybersecurity technologies and demonstrates the prospects of integrating artificial intelligence into automation processes

Keywords: large language models, vulnerability detection automation, artificial intelligence, multi-vector testing

UDC 004.056
DOI: 10.15587/1729-4061.2025.325251

DESIGN AND DEVELOPMENT OF A LARGE LANGUAGE MODEL-BASED TOOL FOR VULNERABILITY DETECTION

Anastasiia Zhuravchak
PhD Student*
Andrian Piskozub
PhD, Associate Professor*
Bohdan Skorynovych
PhD Student*
Yuriy Lakh
PhD, Professor
Department of Financial Markets and Technologies
University of the State Fiscal Service of Ukraine
Universytetska str., 31, Irpin, Kyiv region, Ukraine, 08205
Danyil Zhuravchak
Corresponding author
PhD**
E-mail: danyil.zhuravchak@lnu.edu.ua
Pavlo Hlushchenko
PhD Student*
Petro Venherskyi
Doctor of Technical Sciences, Professor, Head of Department**
Igor Beliaiev
PhD Student**
Maksym Vorokhob
PhD, Senior Lecturer
Department of Information and Cyber Security named after Professor Volodymyr Buriachok
Borys Grinchenko Kyiv Metropolitan University
Bulvarno-Kudriavska str., 18/2, Kyiv, Ukraine, 04053
Ivan Kolbasynskyi
PhD Student
Department of Theoretical Physics
Uzhhorod National University
Narodna sq., 3, Uzhhorod, Ukraine, 88000
*Department of Information Security
Lviv Polytechnic National University
S. Bandery str., 12, Lviv, Ukraine, 79013
**Department of Cybersecurity
Ivan Franko National University of Lviv
Universytetska str., 1, Lviv, Ukraine, 79000

Received 14.01.2025
Received in revised form 26.02.2025
Accepted date 17.03.2025
Published date 22.04.2025

How to Cite: *Zhuravchak, A., Piskozub, A., Skorynovych, B., Lakh, Y., Zhuravchak, D., Hlushchenko, P., Venherskyi, P., Beliaiev, I., Vorokhob, M., Kolbasynskyi, I. (2025). Design and Development of a Large Language Model-Based Tool for Vulnerability Detection. Eastern-European Journal of Enterprise Technologies, 2 (2 (134)), 75–83. https://doi.org/10.15587/1729-4061.2025.325251*

1. Introduction

In the modern world, the importance of new methods for detecting and eliminating vulnerabilities is becoming increasingly relevant. In particular, the use of large language models opens up new opportunities for automating the processes of code analysis and vulnerability detection [1]. Such methods can provide faster and more accurate threat detec-

tion. This is critically important for protecting modern information systems and infrastructures. The research is aimed at designing a system that integrates large language models to increase the efficiency and accuracy of vulnerability detection.

Scientific research in the field of vulnerability detection is gaining particular importance due to the constant complication of software. The development of innovative methods for analyzing and identifying threats is a necessary condition for ensuring the resilience of information systems to modern cyber threats. The integration of large language models into vulnerability detection processes opens up new opportunities for automating analysis, making it possible to significantly increase the efficiency and accuracy of work. Such technologies are able to process large volumes of data, identify complex vulnerabilities, and reduce the number of false positives, which makes them promising for ensuring information security.

The practical significance of vulnerability detection research relates to increasing the resilience of information systems to modern threats and reducing the risk of exploiting critical security flaws.

Therefore, research aimed at integrating machine learning technologies into vulnerability detection processes is relevant for cybersecurity specialists, developers, and business owners. It makes it possible not only to automate key analysis stages but also to ensure early detection of threats, reducing the potential impact of attacks on information systems.

2. Literature review and problem statement

In [2], the results of research on the effectiveness of machine learning algorithms in vulnerability detection are reported. It is shown that the selected methods provide high accuracy but are characterized by an increased level of false positives. The issues of their integration into penetration testing processes and adaptation to various scenarios and environments remain unresolved. A likely reason is technical difficulties and resource limitations. One of the options for overcoming these limitations is the use of large language models capable of automating routine tasks.

In [3], the results of research on the use of machine learning methods in vulnerability detection and penetration testing are reported. It is shown that algorithms such as XG Boost, Random Forest, and SVM can effectively analyze network threats, automate risk detection processes, and reduce the number of false positives. However, there are still unresolved issues related to the limitations of conventional methods in the context of flexibility, scalability, and adaptation to new scenarios. A likely reason is insufficient integration with modern tools. An option to overcome these difficulties may be to use large language models.

In [4], the results of research on the performance of language models in detecting vulnerabilities in software, in particular the DistilVulBERT model, are reported. However, there are still unresolved issues regarding the application of these models in the real penetration testing process. The solution focuses on static code analysis and detection of vulnerabilities in the code while the integration of large language models into the penetration testing process (interactive interaction with experts, context analysis, and automation of routine tasks) has not yet been studied. A likely reason is the need for significant computing resources for their deployment. An option to overcome these difficulties may be a tool using LLM.

In [5], the capabilities of LLMs in detecting vulnerabilities in software are investigated. It is shown that they can recognize complex patterns. However, they demonstrate a high proportion of false positives compared to conventional static analysis methods. It is proposed to combine LLMs with classical approaches to increase accuracy. However, challenges remain related to adaptation to real-world scenarios. A possible solution is to integrate LLMs into penetration testing processes. However, the problem of reducing false positives remains relevant. This confirms the feasibility of research aimed at designing a vulnerability detection system based on LLM to improve penetration testing efficiency.

In [6], the effectiveness of large language models in detecting vulnerabilities in code was investigated. It was shown that CodeGemma achieves the best results (F1=58 %, Recall=87 %). However, the performance of LLM depends on the specific task, and generalization of the results can be misleading. The issue of integrating LLMs into penetration testing remains open. The study considers this direction but further assessment of their effectiveness in real cyber threats is necessary. Issues related to the effectiveness of LLMs in real penetration testing conditions remain unresolved. A likely reason is limitations in adapting models to specific scenarios. An option to overcome the difficulties may be the integration of LLMs into more comprehensive approaches. This confirms the relevance of designing a vulnerability detection system that combines testing automation and support for specialists.

In [7], the results of research on the application of large language models in the penetration testing process are reported. The proposed Pentest Copilot tool uses Retrieval Augmented Generation (RAG) to improve the accuracy of the answers, as well as new file analysis methods that simplify the work with the results. However, questions related to the effectiveness of LLMs in real-world scenarios remain unresolved. A likely reason is the limitations of LLM in understanding complex attack tactics and its dependence on the quality of the input data. An option to overcome these difficulties may be the development of a tool that automates routine tasks. However, the question of its effectiveness in a real-world penetration testing environment remains open.

In [8], the authors present the CIPHER model specifically trained for penetration testing tasks. In this case, the authors emphasize that further development of the method involves scaling the models, creating more advanced benchmarks, and preserving the quality of training data in order to avoid possible false conclusions and increase the real usefulness of the model in the dynamic cybersecurity environment. However, issues related to the low performance of the model in real testing scenarios remain unresolved. An option to overcome these difficulties may be to increase the training sample and train in real scenarios.

Summarizing the foregoing studies, it can be noted that existing methods for detecting vulnerabilities in software based on conventional approaches have limitations. In particular, the main issue is the dependence on manual checks, security specifications, and a large amount of human time. Thus, we can see the need to design and implement integrated solutions that combine the capabilities of large language models with conventional static and dynamic analysis methods, which will act as an intelligent assistant for pentesting specialists. This will increase accuracy, reduce false positives, and automate the detection of complex vulnerabilities in various software environments.

3. The aim and objectives of the study

The purpose of our research is to design a vulnerability detection system based on large language models, which increases the efficiency and accuracy of penetration testing through interactive support of specialists, automation of routine tasks, and analysis of the obtained data.

To achieve this goal, the following tasks were set:

– to develop a functional design of a tool that will integrate the capabilities of large language models to automate penetration testing processes;

– to evaluate the effectiveness of the proposed tool by comparing it with conventional penetration testing methods;

– to evaluate the work of the tool in scenarios of real cyber threats.

4. The study materials and methods

The object of our study is an automated vulnerability detection tool that uses the capabilities of large language models to support testing processes.

The hypothesis of the study assumes that the use of large language models to automate penetration testing processes could reduce the time and number of false positives, increasing the efficiency and accuracy of the analysis.

The research method used for this work is an experimental approach, which involves designing and testing a prototype of the tool with the subsequent comparison of its results with the results of conventional pentesting methods. Also, already known large language models that are not the subject of authorship in this study were used. The goals of the experiment were:

– to evaluate the speed of the tool compared to manual testing methods;

– to measure the accuracy and number of false positives generated by the system;

– to demonstrate the effectiveness of the tool in scenarios with real projects.

It is worth mentioning that the existing conventional approach for testing web applications and automated scanners were used. There are also other methods that were not selected for this study:

– static code analysis (SAST). Although this method is effective for finding vulnerabilities in the source code, it does not make it possible to assess real operational risks and does not take into account the context of the interaction of system components [9];

– dynamic application security analysis (DAST) is suitable for testing web applications but may be limited in detecting complex logical vulnerabilities that require deeper contextual analysis [10];

– Fuzz Testing is effective for detecting unexpected errors in software, but its use requires significant computing resources and does not always allow for a clear picture of the overall security of the system [11];

– vulnerability assessment based on behavioral analysis (UEBA, Anomaly Detection); although this method makes it possible to detect anomalous activities, it is more focused on monitoring and response rather than on active penetration testing [12].

The following technologies and tools were used to design and test the vulnerability detection system using large language models. API access to ChatGPT was used to integrate with large language models, which provides the ability to automatically generate tasks, analyze the results, and form recommendations for penetration testers. Python was chosen as the main development language, due to its numerous libraries for integration with AI, data processing and interaction with pentest tools. Python provides flexibility in developing solutions for working with data and automating testing.

To conduct a pentest, integration with the following tools was implemented:

– Nmap for network scanning [13];

– Burp Suite for checking the security of web applications [14];

– Metasploit for automating attacks [15];

– SSLyze for analyzing TLS configurations [16].

A PostgreSQL database was used to store test results, configurations, or user data.

Web content processing was performed using the Beautiful Soup [17] and Scrapy [17] libraries, which were used to parse data from web pages. The pandas and numpy libraries were used to process and analyze structured data, and tqdm provided a convenient output of task progress.

Experimental research was conducted in several stages. At the first stage, the specified tools were integrated to automate pentest tasks, such as network scanning, configuration analysis, and attack scenario generation. At the second stage, the effectiveness of the tool was tested on real cyberthreat scenarios, including processing data from web pages, pentest tool results, and text descriptions.

This approach allowed us to design an interactive system that combines the capabilities of large language models with conventional pentest methods, ensuring increased efficiency, accuracy, and convenience for testers.

5. Vulnerability detection tool results

5. 1. Development and implementation of a tool architecture based on large language models with AI assistant integration

To ensure interactivity, efficiency, and automation of the penetration testing process, a tool architecture based on the use of large language models was designed. Its architecture includes a unified terminal input handler supported by three main components, each of which has a specific functional purpose.

The test generation module is responsible for executing precise and detailed commands for performing penetration tests. Owing to integration with large language models, the module provides users with structured tasks that take into account the specificity of the target system and current testing scenarios. For example, it can automatically generate commands for port scanning using Nmap.

The reasoning module is the intelligent heart of the tool. It analyzes the current state of testing and makes recommendations for the next steps. This component allows testers to effectively manage the testing process, ensuring that they are focused on the most critical security aspects. For example, if previous testing has identified weak encryption or XSS vulnerabilities, the module can suggest additional checks to refine the results.

The parser module is responsible for analyzing the output data generated by other penetration tools, such as Nmap, Metasploit, or SSLyze. In addition, it analyzes the content of web interfaces and text data obtained during testing. For example, the module can extract key information from network scan results, such as open ports or vulnerable services, and

analyze HTTP request headers to identify security configuration issues. Owing to this component, the tool provides a convenient and informative representation of the results, which facilitates their interpretation and use.

The general high-level architecture, shown in Fig. 1, demonstrates an architectural solution for implementing the tool.

The handler is a key component of a penetration testing tool that provides an interactive user experience. The functionality of the handler includes:

1. Initializing the tool. It can automatically configure itself using pre-designed prompts that define parameters for interacting with large language models. This ensures that the testing environment is prepared with minimal user intervention.

2. Starting a new testing session. The user can initiate a new pentest session by providing basic information about the target, such as the IP address, domain name, or specific metrics to consider during testing.

3. Getting a task list. The tool can generate a structured task list, specifying the next steps to be performed. This helps penetration testers organize their workflow and focus on priority security aspects.

4. Passing information to the tool. After completing a specific task, the processor passes the test results, such as detected vulnerabilities or analysis data, to the module for further processing, interpretation, and recommendation generation.

5. Passing tool output. The tool makes it possible to pass results generated by other pentest tools for syntactic analysis and recommendation generation.

6. Passing web page content. The user can provide HTML code or content of the web page being analyzed to detect vulnerabilities related to server configuration, API integration, or web application security.

7. Passing text descriptions. To expand the context of the analysis, the penetration tester can pass a text description that contains details about the system configuration, network characteristics, or specific customer requirements.

8. Launching continuous mode. The generation module supports continuous mode, which makes it possible to automatically perform a series of operations, delving into a specific task. For example, it can be port scanning, log analysis, or TLS configuration testing.

The proposed tool is aimed at detecting vulnerabilities that are described in the OWASP Web Application Testing Guide. The main focus is on web vulnerabilities and infrastructure misconfigurations that can be used by attackers to compromise the system.

5. 2. Model performance and tool performance evaluation

The model performance was evaluated using several key metrics that allow us to assess its ability to automate penetration testing processes and provide accurate and useful recommendations to testers. Table 1 gives the metrics and their descriptions.

Table 1

Performance metrics

Metrics	Description
Vulnerability detection accuracy	The model's ability to detect vulnerabilities was assessed by comparing the results of the tool with the results of manual testing
Task completion speed	The time required to complete typical pentest tasks, such as network scanning, web application analysis, was compared
False positive and false negative rate	The model's ability to reduce the number of false positives that are not real vulnerabilities and to detect all critical vulnerabilities was determined
Integration with pentest tools	The model's ability to effectively use tools such as Nmap, Metasploit, and others was assessed

Key aspects of the evaluation included performance, accuracy, task completion time, usability, and adaptability to different scenarios.

5. 3. Evaluation of the tool's performance in real cyberthreat scenarios

A practical evaluation of the AI assistant's performance was carried out as part of comparing its results with the results obtained during manual testing. For this purpose, a test environment was used, which included a web application and network services. As well as a set of real vulnerabilities, such as SQL injections, cross-site scripting (XSS), authentication errors, and configuration vulnerabilities.

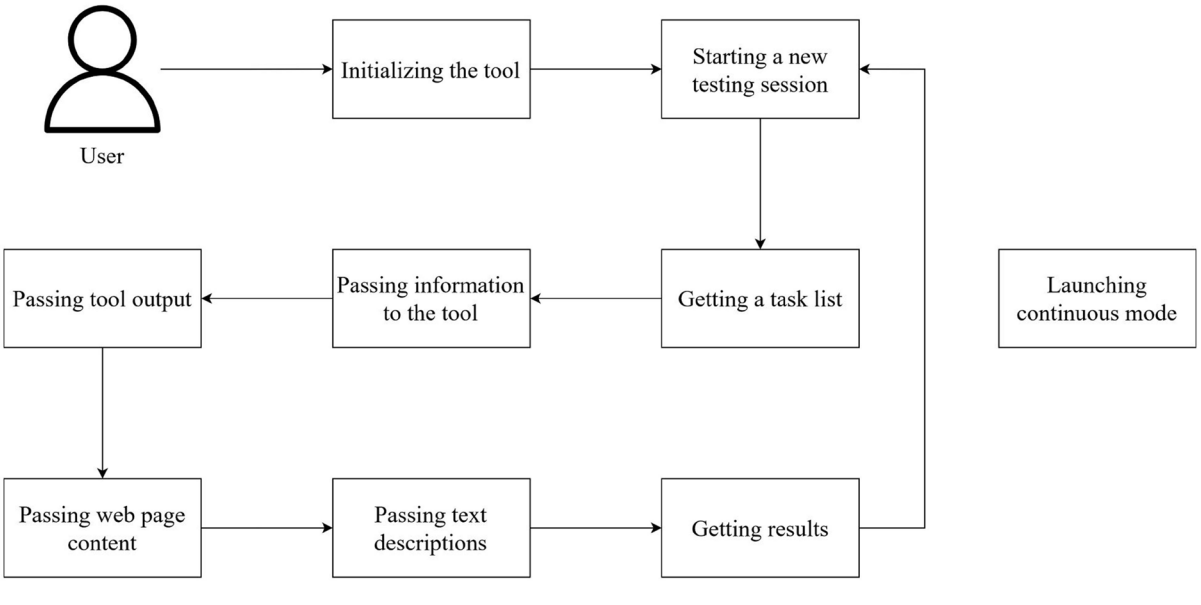

Fig. 1. Conceptual diagram of the proposed solution

Popular pentest tools were used to integrate with the AI assistant. An environment with real vulnerabilities was used for evaluation. Each scenario included tasks related to network analysis, web application testing, and TLS connection verification. The results were compared with the results obtained through manual testing. Testers analyzed the convenience of interacting with the model, including recommendation generation, input processing, and integration with tools. The following scenarios were selected for the experiment:

1. Network scenario. The AI assistant used Nmap to scan the network and generate recommendations for further steps. In the manual approach, the tester manually configured Nmap parameters and analyzed the results. Goal: to determine the efficiency of automatic command generation and the speed of transition to the next stage of testing. The workflow scheme is shown in Fig. 2.

2. Web application scenario. The web application was tested for common vulnerabilities using Burp Suite. The AI assistant automatically generated specific queries (for example, to search for XSS or SQL injections), analyzed the server responses, and suggested next steps. In manual testing, this process was performed without automation. Goal: to assess the detection accuracy and the number of false positives. The workflow is shown in Fig. 3.

3. TLS configuration scenario. The SSLyze tool was used to analyze TLS configurations. The AI assistant analyzed the results, identified weak encryption algorithms, and formulated recommendations. Goal: to determine the effectiveness of automated encryption security analysis. The workflow is shown in Fig. 4.

4. Web content scenario. The AI assistant received HTML content of web pages and analyzed it for vulnerabilities in scripts, metadata, and authentication tools. The Beautiful Soup and Scrapy libraries were used. Goal: to test the capabilities of AI to search for specific vulnerabilities in client components. The scheme of work is shown in Fig. 5.

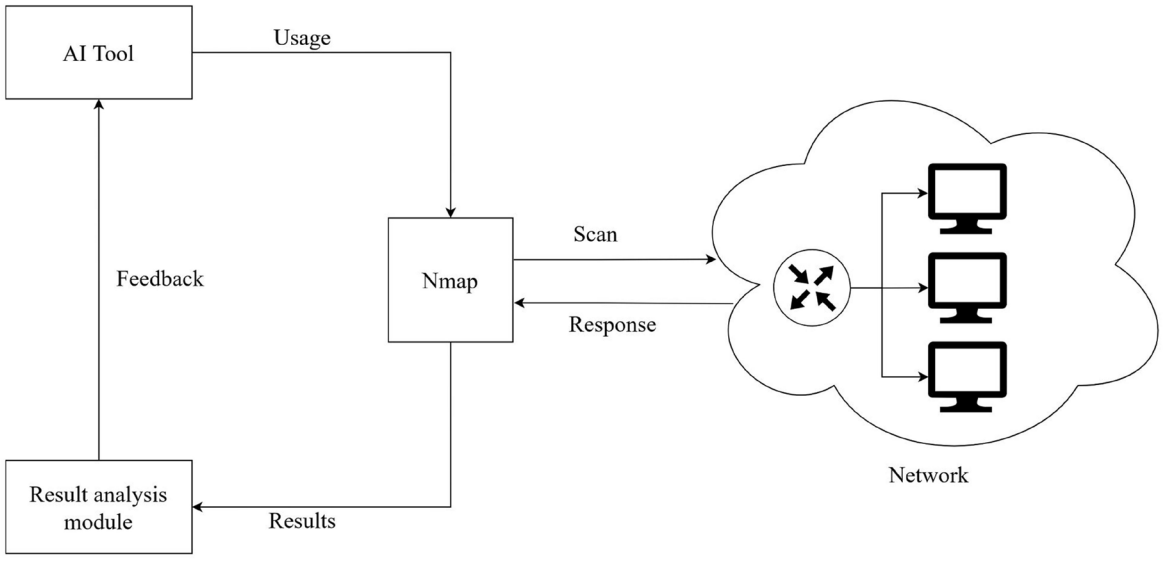

Fig. 2. Network scenario execution diagram

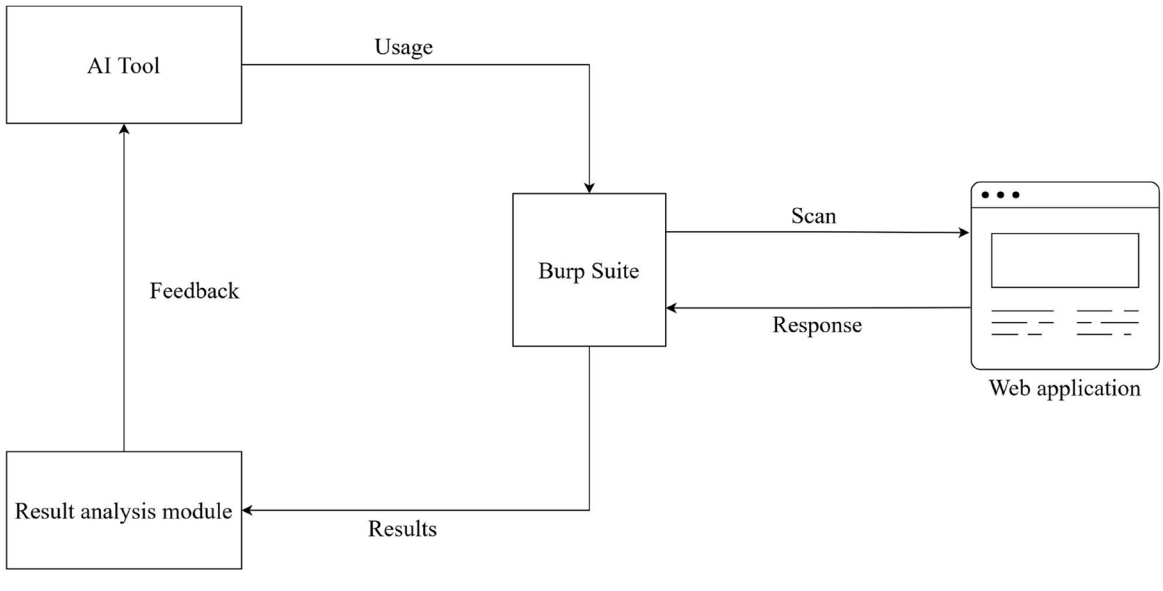

Fig. 3. Web application script execution diagram

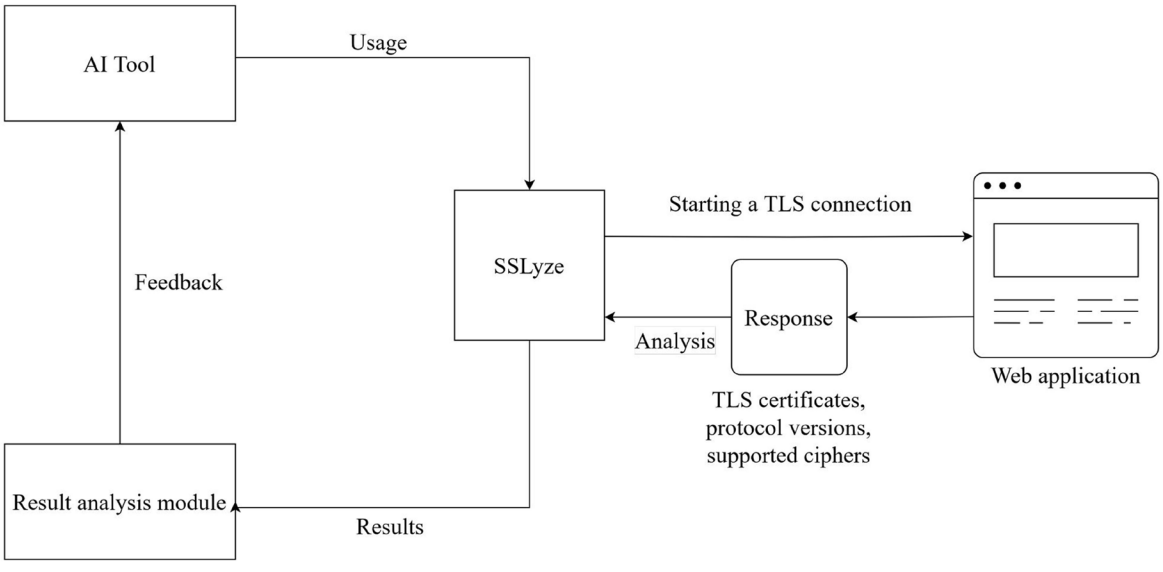

Fig. 4. TLS configuration analysis script execution flow chart

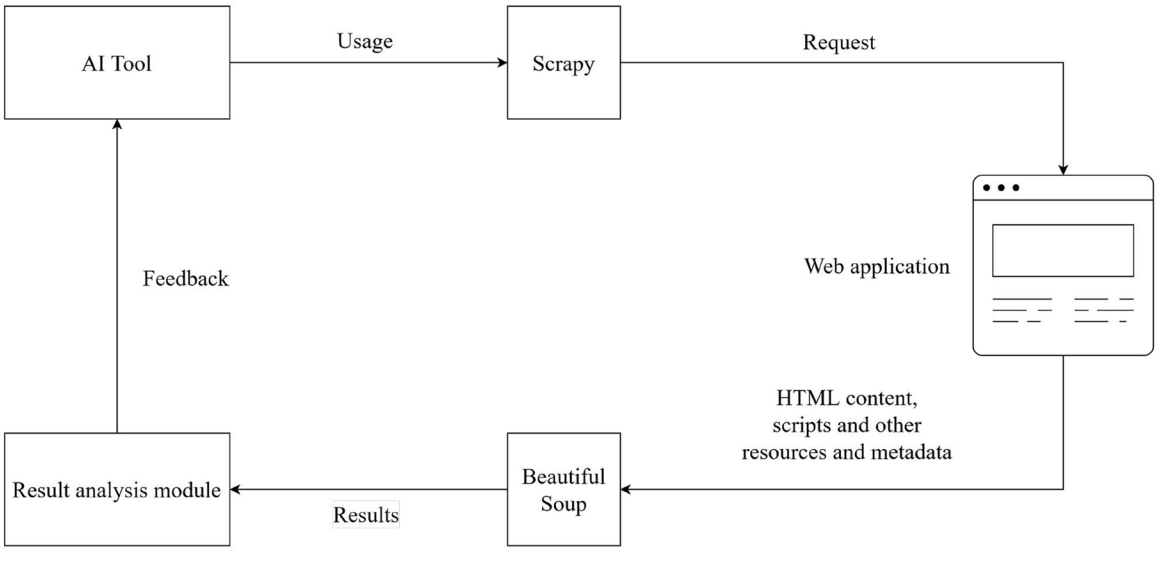

Fig. 5. Web content script execution flow chart

These scenarios were chosen to cover different levels of cybersecurity from infrastructure protection to web security. They are correlated with each other, as together they provide a comprehensive approach to penetration testing. Network analysis makes it possible to identify open services that may contain web applications, which are then analyzed in the web application testing scenario. Evaluating TLS configurations is related to web application security, as weak certificates can affect the overall security level of the system. Web content processing helps find potential problems in client components that can be entry points for attacks detected during web testing. If you use other scenarios, such as phasing or source code analysis, the tool's performance metrics will be different, as they do not evaluate the interaction of the system in a real environment, but its individual components.

In the first scenario (Fig. 2) with network analysis, 10 hosts were used. Integration with the nmap tool was tested. Tasks such as scanning open ports using Nmap and using the data found to automate exploits via Metasploit were performed. The automated approach (AI module) found 20 potential vulnerabilities, of which two were false positives (false detection of outdated FTP services) and one was unnoticed (false negative). The total time for scanning and analysis was about 5 minutes. At the same time, manual testing showed 19 real vulnerabilities, with 1 false positive. The tester spent almost 12 minutes, since all Nmap parameters and subsequent interaction with Metasploit were performed manually.

In the second scenario (Fig. 3), a single web application was tested for common vulnerabilities, including XSS, SQL injection, and misconfigured security headers. Burp Suite was used as the primary analysis tool, and an AI assistant integrated with it via the Burp Extender API to automatically suggest scan settings and refine vulnerability search parameters. The tool (AI+Burp Suite) detected 10 confirmed vulnerabilities out of 14 known, but there were two false positives (false detection of XSS on pages without user input). In addition, it gave 2 false positives, marking certain pages as vulnerable to XSS, although the test did not confirm this. The average time for configuration and analysis was 25 minutes. Under manual mode, the specialist detected 13 out of

14 vulnerabilities and mistakenly marked one incorrect logic implementation as a vulnerability (1 false positive). The full testing cycle took about 60 minutes, including manual log review and additional test requests.

In the third scenario (Fig. 4), the SSLyze tool was used to analyze the security of TLS configurations. During testing, the presence of weak encryption algorithms was checked (for example, support for TLS 1.0, RC4, or outdated certificates). In this experiment, 10 servers were tested. Checking the servers allowed the AI assistant (through automated SSLyze modules) to detect 16 configuration problems. In two cases, false positives were recorded for TLS 1.2. The automation runtime was about 40 minutes with full report generation. With the manual method, the tester confirmed 15 real problems, missing one non-obvious certificate configuration on a non-standard port. In total, the process took almost 90 minutes since each server was checked sequentially with manual analysis of SSLyze logs.

In the fourth scenario (Fig. 5), the tool received HTML content from web pages using the Beautiful Soup and Scrapy libraries. The task was to automate the analysis of client components, which would help identify vulnerabilities such as the absence of attributes for the user session, dangerous JavaScript functions on pages, and open configuration files. 10 web pages with different types of content (static, dynamic, pages with authorization) were tested. The tool is easily integrated into any websites independently since it is a scripting language. During the analysis, the AI assistant found 14 potential vulnerabilities, of which 2 turned out to be erroneous (for example, specific scripts for analytics), and 1 real error (incorrect inline script) remained unrecognized. Full bypass of links and generation of recommendations took about 45 minutes. The manual approach found 15 confirmed issues in 90 minutes, including complex logic flaws in components that the AI program missed. However, the tester made 1 false positive in interpreting the interaction of the iframe with the authentication script.

Fig. 6 shows a comparison between manual and automated testing.

According to the above, one can see that automated testing consumes less time. Table 2 gives data from the experiment.

Table 2

Comparison of automated testing with a tool and testing with the involvement of a specialist

Scenario	Approach	Detected	False positives	False negatives	Time, min	Precision, %	Recall, %
Scenario 1	Auto (AI)	20	2	1	5	90.00 %	94.74 %
	Manual	19	1	0	12	94.74 %	100.00 %
Scenario 2	Auto (AI)	10	2	4	25	80.00 %	66.67 %
	Manual	13	1	1	60	92.31 %	92.31 %
Scenario 3	Auto (AI)	16	2	0	40	87.50 %	100.00 %
	Manual	15	0	1	90	100.00 %	93.75 %
Scenario 4	Auto (AI)	14	2	1	45	85.71 %	92.31 %
	Manual	15	1	0	90	93.33 %	100.00 %

The evaluation results showed that the tool reduced the testing time by an average of 54.4 % compared to the manual approach. The effectiveness of the automated tool varied depending on the type of tasks: the best vulnerability detection recall rates were achieved in TLS configuration evaluation (100.00 %) and network analysis (94.7 %), while in web application testing the rate decreased to 66.7 %. Manual testing demonstrated consistently high recall rates (92–100 %) but required significantly more time. Regarding the precision of the results, the automated approach showed from 80 % to 90 %, with an average of more false positives compared to manual testing, where the precision reached 92–100 %. This is partly due to the "reinsurance" of the model, which gives a wider range of warnings. Despite this, the AI assistant provided detailed recommendations for eliminating the found threats, helped to quickly localize typical vulnerabilities and reduced the initial analysis time. The ability to quickly switch between different types of tasks reduced the cognitive load on the tester.

Automation was especially effective in network scanning, where the tool achieved the best balance between speed (5 min vs. 12 min) and quality of results (precision of about 90 %, recall of 94.7 %). The greatest time savings were observed in the analysis of TLS configurations and HTML content where automation reduced the verification time by more than half.

The evaluation confirmed that the model is an effective tool for automating penetration testing, especially in scenarios that require quick response and interactive support. However, further research remains necessary to reduce the number of false positives, increase the accuracy of detecting vulnerabilities in web applications, and improve the scalability and adaptation of the model to complex multi-vector attacks.

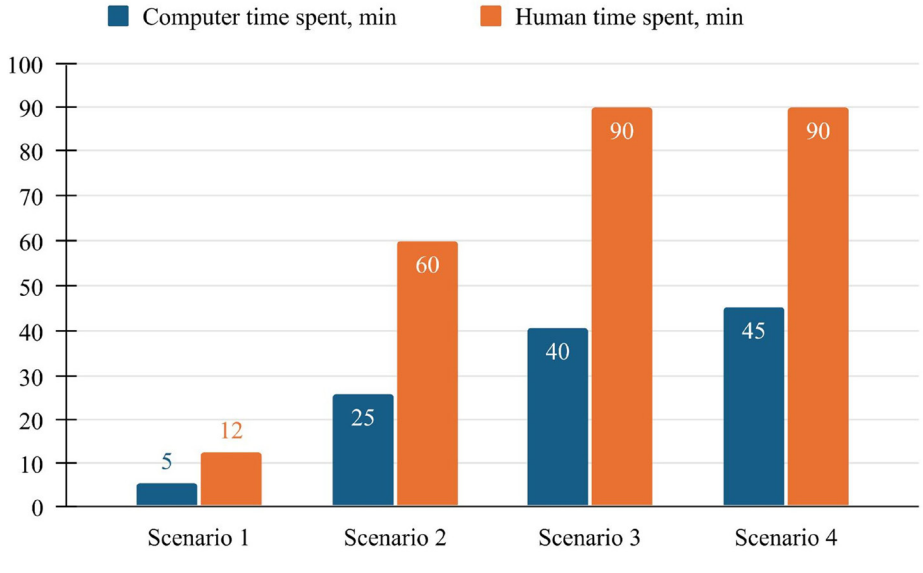

Fig. 6. Comparison of time spent between manual and automated testing

6. Discussion of results based on the research into designing an automated penetration testing tool

The modular structure of the designed tool (Fig. 1) provided a clear division of functions between the automatic generation of sequential tasks, security status analysis and detailed analysis of the output data from pentest tools. This approach makes it possible to move faster from one testing stage to another, minimize manual scanner settings, and reduce the total time for analysis, which is confirmed by the data in Fig. 6 and Table 2. In the scenarios of network analysis (Fig. 2) and TLS configuration verification (Fig. 4), the duration of work was reduced by more than half, while the precision of threat detection in a number of cases reached 90–100 %, and the recall was 94.7–100 %. Although these indicators decreased during web application testing (Fig. 3) (precision 80 %, recall 66.7 %), even in this case, time spent on routine tasks was significantly reduced. Similar trends were observed in the web content scenario (Fig. 5), where analysis of client components allowed us to detect configuration errors and malicious scripts.

These results have important practical significance. Reducing testing time by 54.4 % on average means that a cybersecurity team can conduct testing twice as often in the same period of time or significantly reduce testing costs. This allows vulnerabilities to be detected and fixed faster, thereby reducing the risk of successful cyberattacks and potential financial and reputational losses.

Unlike [2, 3], which reported high accuracy rates but had a significant percentage of false positives, the proposed system achieved a better balance between speed and accuracy by integrating large language models with proven pentest tools. This is especially important because false positives can lead to unnecessary time and resources spent analyzing non-existent threats.

Compared to [4, 5], in which the research was limited mainly to static code analysis or basic LLM applications, our work focuses on dynamic checks and full interaction with pentest tools in a real environment (scenarios 1–4, Fig. 2–5). The implemented architecture (Fig. 1) includes a specialized syntax analysis module for complex processing of results from Nmap, Metasploit, BurpSuite, and other tools. This approach makes it possible to detect not only vulnerabilities in the code but also problems related to the system configuration, network settings, and the interaction of various components.

In addition, unlike [7, 8], in which large language models are used pointwise (for example, only for generating reports or generating tips), our work implements a full cycle of interactive work – from the initialization of the test scenario to the final generation of recommendations. Finally, unlike [4–8] in general, which often focus on the analysis of individual files or code bases, the proposed tool has been successfully tested on real cyber threats (network analysis, web applications, TLS), demonstrating its practical value.

The use of large language models has eliminated the problem of excessive dependence on the manual approach and narrow integration with pentest tools since the automation of routine tasks covers various types of analysis. This significantly reduces time costs without significantly reducing the accuracy of vulnerability detection. In addition, in most cases, the integration of the tool does not require significant changes to existing processes, which simplifies its implementation in medium and small businesses.

At the same time, the proposed approach has certain limitations. First, working with large language models requires access to their API and depends on the quality of the responses. In particular, we observed that the accuracy of the responses may vary depending on the complexity of the query and the specificity of the target system. Second, in complex multi-vector attacks or with atypical configurations, the tool may require more careful manual tuning. For example, in the web application testing scenario (Scenario 2, Fig. 3), where 4 false negatives were detected, further analysis revealed that these vulnerabilities were related to complex server-side data processing logic that LLM could not fully understand without additional context. Third, dynamic web applications with complex logic may remain partially untested, which increases the risk of missing non-trivial logic vulnerabilities.

Regarding the shortcomings of the study, it is worth noting that the system requires regular updates and adaptation to new types of threats, as well as additional efforts to reduce the number of false positives in complex scenarios. In addition, the current study is limited to four testing scenarios. Although these scenarios cover a wide range of tasks, further expansion of the number of scenarios, including testing APIs, mobile applications, and cloud infrastructures, will allow for a more complete assessment of the tool's effectiveness.

Further research should focus on:

– expanding the list of supported tools and integrating with other security platforms, such as CSPM, CWPP, and SIEM systems;

– improving web vulnerability detection algorithms, in particular, by using specialized language models trained on web security data and applying machine learning methods to analyze anomalies in the behavior of web applications;

– scaling the system to work with large or distributed infrastructures, as well as designing mechanisms for automatically selecting optimal testing strategies depending on the characteristics of the target system.

Such development will improve the accuracy of detecting complex web vulnerabilities, increase the overall flexibility of the tool, and expand the scope of its practical application. In particular, automating the selection of testing strategies will allow the tool to be used not only by experienced specialists but also by less qualified users, which will expand the possibilities of using the tool in organizations with limited cybersecurity resources.

7. Conclusions

1. As a result of our research, a functional design and architecture of an automated vulnerability detection tool were developed, which integrate the capabilities of large language models into penetration testing processes. The architecture of the tool includes a test generation module, a reasoning module, and a syntactic analysis module, which provide interactive support for testers, automation of routine tasks, generation of action sequences, and analysis of the results.

2. The results showed that the use of large language models significantly increases the efficiency of pentesting. In particular, the tool reduced the average task execution time by 54.4 % compared to the manual approach. Recall indicators vary depending on the type of tasks: the best value (94.7 %) was achieved during network analysis, while in web application testing this indicator decreased to 66.7 %. The precision of automated tests mostly ranged from 80–90 % and was accompanied by a relatively higher percentage of false positives, while the manual approach in some scenarios reached 92–100 %. The effectiveness of the system was confirmed in real cyber threat scenarios, which indicates its reliability and practicality. These results demonstrate

the significant potential of LLMs for automating cybersecurity tasks and reducing the burden on specialists.

3. The implementation of the AI assistant significantly improved the penetration testing process, making it more structured and productive, especially in scenarios that require quick response and interactive support. The tool can be used both independently and in combination with other automation tools, making it a universal solution for organizations of various sizes. For example, small companies that do not have their dedicated cybersecurity specialists can use the tool to conduct a basic security assessment of their systems, while larger organizations can integrate it into their existing penetration testing processes to improve their efficiency.

Conflicts of interest

The authors declare that they have no conflicts of interest in relation to the current study, including financial, personal, authorship, or any other, that could affect the study, as well as the results reported in this paper.

Funding

The study was conducted without financial support.

Data availability

The data will be provided upon reasonable request.

Use of artificial intelligence

The authors used artificial intelligence technologies within acceptable limits to provide their own verified data, which is described in the research methodology section.

References

1. Tolkachova, A., Piskozub, A. (2024). Methods for testing the security of web applications. Electronic Professional Scientific Journal «Cybersecurity: Education, Science, Technique», 2 (26), 115–122. https://doi.org/10.28925/2663-4023.2024.26.668
2. Li, Z., Dutta, S., Naik, M. (2024). LLM-assisted static analysis for detecting security vulnerabilities. arXiv. https://doi.org/10.48550/arXiv.2405.17238
3. Saini, J., Bansal, A. (2024). Automated penetration testing: machine learning approach. CEUR Workshop Proceedings. Available at: https://ceur-ws.org/Vol-3682/Paper10.pdf
4. Omar, M. (2023). Detecting software vulnerabilities using language models. arXiv. https://doi.org/10.48550/arXiv.2302.11773
5. Purba, M. D., Ghosh, A., Radford, B. J., Chu, B. (2023). Software Vulnerability Detection using Large Language Models. 2023 IEEE 34th International Symposium on Software Reliability Engineering Workshops (ISSREW), 112–119. https://doi.org/10.1109/issrew60843.2023.00058
6. Sultana, S., Afreen, S., Eisty, N. U. (2024). Code vulnerability detection: A comparative analysis of emerging large language models. arXiv. https://doi.org/10.48550/arXiv.2409.10490
7. Goyal, D., Subramanian, S., Peela, A. (2024). Hacking, the lazy way: LLM augmented pentesting. arXiv. https://doi.org/10.48550/arXiv.2409.09493
8. Pratama, D., Suryanto, N., Adiputra, A. A., Le, T.-T.-H., Kadiptya, A. Y., Iqbal, M., Kim, H. (2024). CIPHER: Cybersecurity Intelligent Penetration-Testing Helper for Ethical Researcher. Sensors, 24 (21), 6878. https://doi.org/10.3390/s24216878
9. Aloraini, B., Nagappan, M., German, D. M., Hayashi, S., Higo, Y. (2019). An empirical study of security warnings from static application security testing tools. Journal of Systems and Software, 158, 110427. https://doi.org/10.1016/j.jss.2019.110427
10. Singh, R., Kumar Gupta, M., Patil, D. R., Maruti Patil, S. (2024). Analysis of Web Application Vulnerabilities using Dynamic Application Security Testing. 2024 IEEE 9th International Conference for Convergence in Technology (I2CT), 1–6. https://doi.org/10.1109/i2ct61223.2024.10543484
11. Mallissery, S., Wu, Y.-S. (2023). Demystify the Fuzzing Methods: A Comprehensive Survey. ACM Computing Surveys, 56 (3), 1–38. https://doi.org/10.1145/3623375
12. Khaliq, S., Abideen Tariq, Z. U., Masood, A. (2020). Role of User and Entity Behavior Analytics in Detecting Insider Attacks. 2020 International Conference on Cyber Warfare and Security (ICCWS), 1–6. https://doi.org/10.1109/iccws48432.2020.9292394
13. Mohammed, F., Rahman, N. A. A., Yusof, Y., Juremi, J. (2022). Automated Nmap Toolkit. 2022 International Conference on Advancements in Smart, Secure and Intelligent Computing (ASSIC), 1–7. https://doi.org/10.1109/assic55218.2022.10088375
14. Choudhary, R., Rawat, J., Singh, G. (2023). Comprehensive Exploration of Web Application Security Testing with Burp Suite Tools. International Journal For Multidisciplinary Research, 5 (6). https://doi.org/10.36948/ijfmr.2023.v05i06.11297
15. Narayana Rao, T. V., Shravan, V. (2019). Metasploit Unleashed Tool for Penetration Testing. International Journal on Recent and Innovation Trends in Computing and Communication, 7 (4), 16–20. https://doi.org/10.17762/ijritcc.v7i4.5285
16. Suga, Y. (2014). Visualization of SSL Setting Status Such as the FQDN Mismatch. 2014 Eighth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, 588–593. https://doi.org/10.1109/imis.2014.88
17. Bhoir, H., Jayamalini, K. (2021). Web Crawling on News Web Page using Different Frameworks. International Journal of Scientific Research in Science and Technology, 513–519. Internet Archive. https://doi.org/10.32628/cseit2174120