Vulnerability Detection System for Protected Web Applications Using Agent Technologies*

Rostyslav L. Tkachuk[1,†], Nataliya O. Maslova [1,2,†], Olena M. Liubymenko[2*,†], Andriy I. Ivanusa1,†

[1] *Lviv State University of Life Safety, 35, Kleparivska St., Lviv, 79007, Ukraine*
[2] *Donetsk National Technical University, str. Sambirska, 76, Drohobych, Lviv region, 82100, Ukraine*

Abstract
Automation of web application pentesting is a relevant direction in cybersecurity due to the increasing complexity of this process caused by modern protection mechanisms, such as WAF and IDS/IPS. This paper proposes a multi-agent system based on JADE that enables adaptive real-time testing with defense evasion. The system architecture is presented, the operation of agents is described, and experimental testing results are provided. The obtained results demonstrate the effectiveness of the multi-agent approach for automated vulnerability detection in protected web environments.

Keywords
System, Protection, Vulnerabilities, Web Applications, Agent Technologies, multi-agent systems

1. Introduction

Protection of web applications is a critically important component of information security. With the rapid development of web technologies and the increasing complexity of application logic, modern web systems face a growing range of threats, including XSS, SQL injection, CSRF, and more advanced logic-based or SSRF attacks. Despite the widespread implementation of Web Application Firewalls (WAFs), Intrusion Detection and Prevention Systems (IDS/IPS), and behavior-based filtering, many threats are still able to bypass protection using obfuscation techniques, polymorphism, or specially mutated payloads.

Penetration testing (pentesting) remains a key instrument in identifying security vulnerabilities and evaluating the robustness of web systems. However, traditional pentesting methods, especially manual or semi-automated approaches, are resource-intensive and lack the flexibility and adaptability required to handle modern, dynamic protection mechanisms. Even widely used tools such as Burp Suite, Nessus, and OpenVAS require manual configuration and cannot coordinate distributed testing or adapt in real time to WAF behavior changes.

This challenge has led to increased interest in the use of multi-agent systems (MAS) for pentesting automation. MAS offer a decentralized architecture where autonomous agents can interact in parallel, adapt their strategies dynamically, and learn from previous interactions. Prior research has explored various applications of MAS in cybersecurity, including traffic monitoring, distributed analysis, and modeling of cyber-physical system threats. However, most agent platforms either lack scalability (e.g., SPADE in heterogeneous environments) or do not support real-time adaptive payload generation.

* Corresponding author.
† These authors contributed equally.

rlvtk@ukr.net (R. Tkachuk); nataliia.maslova@donntu.edu.ua (N. Maslova); olena.liubymenko@donntu.edu.ua (O. Liubymenko); ivaaanusa@gmail.com (A. Ivanusa)
0000-0001-9137-1891 (R. Tkachuk) 0000-0002-9078-0973 (N. Maslova); 0000-0002-5935-6891 (O. Liubymenko); 0000-0001-9141-8039 (A. Ivanusa)

The scientific novelty of the research lies in the development of a full-featured MAS system based on JADE, supporting parallel agent interaction, dynamic generation of obfuscated payloads, and adaptation of attack strategies based on the results of previous attempts. The solution implements asynchronous interaction through FIPA-compliant protocols and can be deployed on physically or geographically distributed nodes.

The practical significance of the study is the possibility to apply the proposed multi-agent system for automated and adaptive pentesting of protected web applications in real time, with the capability to bypass WAFs and analyze defense system responses. Thanks to its modular architecture, support for distributed deployment, and compatibility with containerization, the system is suitable for integration into existing cybersecurity processes and can also be used as a training platform for security professionals.

2. Purpose and objectives of the study

The aim of the study is to develop and experimentally evaluate the effectiveness of a distributed multi-agent system capable of detecting web application vulnerabilities by bypassing active protection mechanisms in real time.

To achieve this goal, the following tasks were set:

- to present the architecture of the multi-agent system;
- to implement agents of various purposes based on JADE;
- to develop attack scenarios and payload mutation methods;
- to conduct experimental testing using XSS and SQLi attacks as examples.

3. Literature Review

The issue of web application security testing is actively explored in the context of both traditional penetration testing approaches and modern automated solutions. According to [1–4], pentesting remains a key tool for detecting vulnerabilities in web applications, particularly XSS, SQLi, and CSRF. Prior to the adoption of multi-agent systems, the automation of pentesting was primarily based on vulnerability scanners (Burp Suite [5], Nessus [6], OpenVAS [7]), semi-automated scripts, and frameworks (Metasploit), which had limited adaptability and scalability. These tools required significant human involvement, lacked real-time interaction, and did not learn from previous attacks. Further studies demonstrated the effectiveness of distributed computing and agent-based approaches.

Significant attention has been devoted to automating pentesting through the use of multi-agent systems (MAS). Studies [8–11] highlight the advantages of agent-oriented platforms, including flexibility and adaptability. The MAS-ML 2.0 model [12] and Java-based agent platforms [13] have proven effective under complex simulation conditions, including cyber-physical systems [14].

In the context of MAS security [15], key attack vectors and the necessity of building resilient interaction models between agents have been examined. Works [16, 17] focus on developing secure agent interaction architectures, including countermeasures against DoS attacks and ensuring consensus-based control in dynamic environments.

Regarding the use of agent systems in cybersecurity, study [18] demonstrates a wide range of MAS capabilities — from energy system monitoring to IoT threat analysis. Research [19] describes the extension of existing web pentesting frameworks, while emphasizing the challenges of integrating agent systems into scanning processes.

In the domain of web application protection, multi-agent systems are applied to intrusion analysis and the enhancement of IDS/IPS and WAF functionality.

Modern protection tools such as WAFs and IDSs perform real-time traffic monitoring, attack detection, and response to emerging threats.

Their efficiency is improved through the use of machine learning methods [20–23]. For instance, models such as Naïve Bayes, k-NN, SVM, and linear regression demonstrate a detection accuracy of 92–99% for malicious HTTP requests [23]. However, modern attacks can bypass these tools using fuzzing, obfuscation techniques (Base64, URL encoding, polymorphism), and automated tools like SQLMap [24, 25].

Special attention should be paid to the comparative analysis of pentesting tools [26–28], such as Burp Suite, which, although effective, are not always accessible due to licensing restrictions, highlighting the relevance of developing free and adaptive solutions.

The conducted analysis shows that multi-agent systems hold high potential in automating web application security testing. They enable parallel task processing, real-time agent interaction, and rapid adaptation to new types of attacks — which is critical in the face of increasing threat complexity.

In the authors' previous studies [29], the SPADE platform was used to develop XSS scanners based on autonomous agents. Integrating MAS into pentesting requires addressing compatibility issues with heterogeneous environments and supporting scalability. SPADE's limitations in this context motivate the search for new platforms for building extensible agent systems. The conducted review confirms that multi-agent systems are a promising direction for automating web application security testing. Their flexibility, adaptability, and self-organization capabilities make them effective in detecting, analyzing, and preventing modern cyber threats.

4. Problem Statement

Web applications are key components of modern information systems widely used in business, government administration, finance, and other critical sectors. At the same time, the openness of their interfaces, complex user interaction logic, and rapid technological development make them vulnerable to various attacks, among which XSS, SQL injection, and CSRF remain the most common.

Traditional security testing methods, particularly manual penetration testing, are resource-intensive, laborious, and poorly scalable. Given the growing number of web applications and their dynamic changes, there is a need for automated solutions capable of quickly adapting to new types of vulnerabilities and changes in protective mechanisms, including the operation of WAFs, IDS/IPS, and others. Despite the availability of vulnerability scanning tools such as Burp Suite, their use is often limited by licensing policies, the need for manual configuration, and lack of support for full distributed request processing. This limits testing effectiveness in large-scale or distributed environments.

In this context, solutions that combine the following remain insufficiently researched and practically unimplemented:

- distributed data processing;

- use of multi-agent technologies for automating penetration testing;
- adaptation to modern protection systems in real time.

Thus, the current challenge is the development of a distributed multi-agent system capable of autonomously and coherently detecting web application vulnerabilities under the constraints imposed by modern protection mechanisms.

5. Materials and Methods

For the implementation of the experimental multi-agent system, the JADE platform (Java Agent Development Framework) was used, which complies with the FIPA specification and supports distributed interaction between agents. Within the scope of the study, the system is understood as an implemented set of agents built according to an agreed architecture that constitutes the proposed solution. The development environment included Java 11, Apache Maven for dependency management, and NetBeans IDE for designing the agent architecture. To emulate a protected web application targeted by the pentest, web applications deployed on an Apache server with the WAF module ModSecurity enabled were used, allowing the simulation of a modern protection system with signature-based analysis.

5.1. System Architecture

The system architecture involves the implementation of four agents with different purposes, which interact via ACL messages typical for JADE.

Recon-Agent performs initial analysis of the target web application, including examining HTTP headers, the presence of client-side JavaScript, and server behavior. It also detects the presence of request filtering and characterizes the WAF operation.

Mutation-Agent is responsible for generating malicious payloads using obfuscation methods. The study implements three mutation approaches: character encoding (Base64, URL encoding), syntactic transformations (altering the structure of requests), and polymorphism (dynamic XSS/SQLi payloads that evade signature detection).

Attack-Agent sends requests to the target system and records responses, including HTTP status codes (403, 406, 500) and processing times. It also detects changes in WAF behavior after each attack series.

Learning-Agent analyzes attack results, performs machine learning generalization of effective patterns, and adapts mutation and retry strategies. Based on data received from the Attack-Agent, it modifies payload mutation parameters and sends them back to the Mutation-Agent, thus forming a closed learning loop.

Each agent runs in a separate JADE container, which can be deployed locally or remotely on a separate physical or virtual node in another network. This enabled the implementation of a distributed architecture where agents coordinate yet remain autonomous, which is especially important for bypassing geolocation or IP-based filters.

Communication between agents is conducted via ACL messages, allowing asynchronous interaction without centralized control (Figure 1).

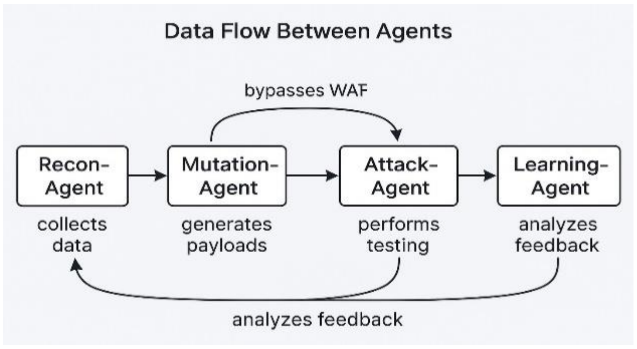


Figure 1: Data Flow Diagram between Agents of the Multi-Agent System

The diagram illustrates the sequential data exchange between agents of the multi-agent system operating in automated pentesting mode.

All agents in the figure are connected by directed arrows, reflecting the sequence and cycle of processing. The process begins with the Recon-Agent, which interacts with the target web application, collecting data on HTTP headers, client-side scripts, and server behavior:

1. Input: Collects data from target.
2. Process: Collects HTTP headers, JavaScript code and server responses.
3. Output: Passes collected data to Mutation-Agent.

The information is passed to the Mutation-Agent, which generates adaptive payloads based on it, taking into account possible protection at the WAF level:

1. Input: Receives input data from Recon-Agent.
2. Process: Generates mutated payloads; Creates obfuscated or polymorphic payloads using encoding techniques (Base64, split-XSS, etc.).
3. Output: Passes formed requests to Attack-Agent.

The simulated requests are sent to the Attack-Agent, which performs the attacks and records the target system's response, including status codes and processing time:

1. Input: Receives payloads from Mutation-Agent.
2. Process: Sends attack requests to target and logs responses; Sends requests to web application, records response codes, response time.
3. Output: Forwards collected responses to Learning-Agent.

The results are analyzed by the Learning-Agent, which identifies effective evasion strategies and adjusts the actions of the Mutation-Agent accordingly:

1. Input: Receives data on attack results from Attack-Agent
2. Process: Analyzes effectiveness and adjusts mutation strategy; Analyzes the success of attacks, adapts mutations for subsequent cycles.
3. Result: Passes recommendations back, closes the adaptation cycle.

Thus, the system forms a closed processing loop where each agent operates autonomously yet in coordination, ensuring adaptability, learning, and flexible response to defense reactions in real time.

The agents were deployed on different physical or virtual machines, simulating real distributed conditions and allowing the evaluation of WAF evasion effectiveness from various geographic zones.

Heuristic and machine learning approaches were used to generate modified requests that are not recognized by the WAF as malicious but are still capable of triggering harmful behavior in the web application. For instance, replacing SQL operators with their encoded or alternative representations allows bypassing static filtering rules.

As part of the study, three types of attack scenarios were created:

XSS with character encoding (standard injected JavaScript with modified characters);

Polymorphic SQL injection (using keyword and parameter substitution to avoid detection);

Split XSS (split payload), where the malicious script is divided into several parts that execute sequentially.

The selection of these specific attack types (XSS, split-XSS, polymorphic SQLi) is justified by their prevalence in modern web applications and their sensitivity to behavioral and signature-based WAF mechanisms.

In each test cycle, 100 requests were executed, and the results were recorded and classified based on the success of WAF evasion.

The evaluation methods included:

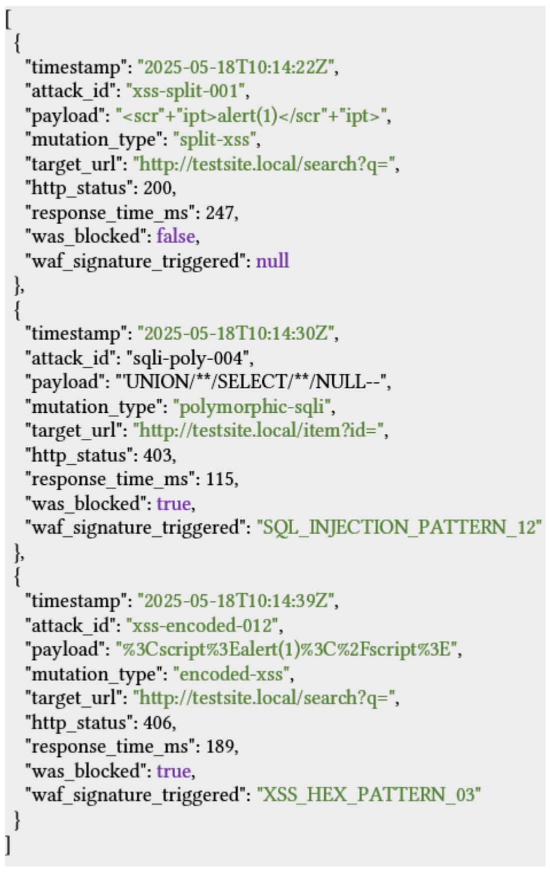
- analysis of HTTP response codes (2xx, 4xx, 5xx),
- comparison of response times for modified requests,
- statistical aggregation of successful attacks,
- visualization of payload mutation effectiveness.

The data is stored in log files, which allows the Learning-Agent to assess the effectiveness of each mutation and adjust the strategy in subsequent stages.

An example of a log file saved after a series of attacks and the use of the Learning-Agent to evaluate mutation effectiveness and adapt strategies is shown in Figure 2.

What the Learning-Agent analyzes:

- mutation_type — the type of mutation that was applied;
- was_blocked + http_status — the result of the request: successful or blocked;
- waf_signature_triggered — the specific signature that was triggered (if any);
- response_time_ms — an indirect indicator of additional request processing or filtering;
- payload — allows evaluation of which specific constructions are more “invisible” to the WAF.



```
[
 {
  "timestamp": "2025-05-18T10:14:22Z",
  "attack_id": "xss-split-001",
  "payload": "<scr"+"ipt>alert(1)</scr"+"ipt>",
  "mutation_type": "split-xss",
  "target_url": "http://testsite.local/search?q=",
  "http_status": 200,
  "response_time_ms": 247,
  "was_blocked": false,
  "waf_signature_triggered": null
 },
 {
  "timestamp": "2025-05-18T10:14:30Z",
  "attack_id": "sqli-poly-004",
  "payload": "'UNION/**/SELECT/**/NULL--",
  "mutation_type": "polymorphic-sqli",
  "target_url": "http://testsite.local/item?id=",
  "http_status": 403,
  "response_time_ms": 115,
  "was_blocked": true,
  "waf_signature_triggered": "SQL_INJECTION_PATTERN_12"
 },
 {
  "timestamp": "2025-05-18T10:14:39Z",
  "attack_id": "xss-encoded-012",
  "payload": "%3Cscript%3Ealert(1)%3C%2Fscript%3E",
  "mutation_type": "encoded-xss",
  "target_url": "http://testsite.local/search?q=",
  "http_status": 406,
  "response_time_ms": 189,
  "was_blocked": true,
  "waf_signature_triggered": "XSS_HEX_PATTERN_03"
 }
]
```

Figure 2: Fragment of the log file processed by the Learning-Agent

Based on these logs, the Learning-Agent compiles statistics on successful/unsuccessful mutations, excludes ineffective patterns from future attacks, adjusts the behavior of the Mutation-Agent to generate more obfuscated or altered requests, and can assign weights to each mutation type depending on its effectiveness (as part of a heuristic or reinforcement learning strategy).

The results were compared to expected blocking rates characteristic of typical signature-based WAF configurations. The research was conducted in a controlled lab environment, allowing precise tracking of the impact each mutation method had on bypassing the protection.

5.2. Implementation of distribution

The concept of distribution is implemented through the use of autonomous agents that perform their functions independently while exchanging results and coordinating actions to achieve a common goal. Agents can be deployed on different physical or virtual machines, as well as in containers, and, if possible, across diverse geographical locations to reduce the risk of blocking based on IP addresses or geographic restrictions.

For example, a Recon-Agent may operate from multiple IP addresses to avoid detection of requests as malicious originating from a single source. Similarly, Mutation-Agent and Attack-Agent can run concurrently on separate nodes, enabling parallel testing of various attack vectors. This distribution allows for multifunctional attacks from multiple sources, minimizing the likelihood of the entire system being blocked.

Several Recon-Agents can simultaneously collect information from different sections of one or multiple websites, analyzing pages, metadata, HTTP headers, request parameters, and server responses from various points. This approach provides a more comprehensive understanding of WAF configurations. The collected data is centrally stored and utilized by other agents.

A Mutation-Agent distributed across multiple nodes generates payload variants for different WAF configurations in parallel. One agent may focus on bypassing one type of protection, while another targets a different one. The generated payloads are then passed to Attack-Agents for simultaneous testing.

The distribution of Attack-Agents sending requests from different geographical locations accelerates testing and increases the likelihood of successful attacks by circumventing diverse blocking strategies. The Learning-Agent operates both centrally and in a distributed manner, gathering data on attack effectiveness, analyzing patterns, and optimizing strategies to minimize detection risk. Data from various locations are aggregated and used to enhance subsequent attacks.

The implementation of distribution in the system is based on autonomous agents that perform their functions independently but exchange results through secure communication channels using TLS protocols to ensure data integrity and confidentiality. Each agent undergoes authentication via JADE Security mechanisms, minimizing the risk of unauthorized access or message forgery.

6. Results and Discussion

In the test scenarios simulating XSS and SQLi attacks, the agents dynamically modified payloads, analyzed server responses, and learned from the results. Examples of test requests for XSS or SQLi scanning with WAF bypass are shown in Table 1.

Table 1
Examples of test requests

№	Request
1	GET/search?q=<script>alert(1)</script>
2	GET/product?id=1' OR '1'='1
3	GET/login?user=admin&pass=<img src=x onerror=alert(123)>
4	GET/comment?text=%3Csvg%20onload%3Dalert%28document.cookie%29%3E
5	POST/api/user {"username": "admin'; DROP TABLE users; --"}
6	GET/page.php?file=../../../../etc/passwd
7	GET/profile?bio=%22%3E%3Cscript%3Efetch(%27//attacker.com%2Flog%3Fcookie%3D%27%2Bdocument.cookie)%3C%2Fscript%3E

Each request had a different attack nature, objective, and The first request aimed to test for XSS vulnerability by injecting a simple script <script>alert(1)</script>. If successful, a JavaScript alert was expected to trigger. The presence of the alert indicated the absence of server-side script filtering.

The second request — an SQL injection with the payload id=1' OR '1'='1 — targeted bypassing the SQL parameter filtering logic.

The expected outcome was either incorrect data output or complete request rejection in case of blocking. If the server returned results, this indicated the presence of the injection vulnerability.

The third request used an XSS attack via an image tag: <img src=x onerror=alert(123)>.

The goal was to check whether the script executes upon handling an error in the attribute. If the browser ran the script, it meant protective mechanisms did not block such constructs.

In the fourth test, an obfuscated XSS attack was employed in the form of a URL-encoded SVG tag containing a script with alert (document. cookie). This request was intended to test the WAF's ability to recognize and decode URL-encoded inputs. Successful bypass indicated insufficient depth of the filtering check.

The fifth test checked for SQL injection through a JSON request. The "username" field contained a payload capable of initiating the deletion of the users table: "admin'; DROP TABLE users; --".

If the request executed without errors, this indicated a serious vulnerability in JSON parameter processing.

The sixth request involved an LFI attack, where the "file" parameter was used to display a system file (/etc/passwd).

If successful, the expected result was either the file content displayed or a specific access error. If the file was output, the system had a critical local file inclusion vulnerability.

The seventh request tested an XSS attack with data exfiltration, where JavaScript was supposed to send the user's cookies to an external server.

If the browser executed the script and triggered the external request, it meant the WAF did not block connections to third-party addresses via XSS.

Overall, these requests allowed evaluation of the agent system's bypass techniques, testing of defensive mechanisms' responses, and drawing conclusions about existing vulnerabilities in the web application.

Within the set goal, the system was tested for its ability to bypass WAF using various payload mutation techniques.

Three types of attacks were tested: character-encoded XSS, polymorphic SQLi, and split-XSS. The results are presented in the table below (Table 2).

Table 2
Overall system testing statistics

Parameter	Value
Number of tests conducted	*100*
Successful attacks	*~ 60*
Average attack success rate	*~ 60%*
The most effective method	Split XSS
Lowest success rate	Polymorphic SQLi
The type of protection most frequently bypassed	Signature-based WAF
System response to payload mutations	Reduction of blocks

Specifically, split-bypass methods for XSS demonstrated high effectiveness in bypassing behavioral filters, whereas SQL injections were often blocked by heuristic WAF algorithms, indicating the need for deeper optimization.

Analysis of agent performance showed that the Recon-Agent successfully identified the WAF type in 87% of cases, providing relevant input data for subsequent attacks.

The Mutation-Agent generated over 150 payload variants, among which split-XSS achieved the highest success rate (up to 80%).

The Attack-Agent recorded server response codes and request processing times, allowing tracking of the system's reaction to each mutation; the average response time ranged between 230 and 310 ms.

The Learning-Agent proved to be a key component of adaptation: after several attack cycles, it improved the accuracy of selecting effective payloads from 55% to 71%, demonstrating the system's ability for self-learning and strategy optimization.

The effectiveness of agents is demonstrated in the table 3.

The results presented in Table 3 confirm the feasibility of distributing functions among agents, as well as the effectiveness of their interaction in implementing automated and adaptive penetration testing of protected web applications.

Each agent plays a distinct role in the overall attack process — from information gathering to payload generation, delivery, and analysis — enabling the system to maintain a full attack cycle with feedback and learning elements.

To further evaluate the performance of the system in a dynamic and scalable environment, an additional experiment was conducted to assess how the number of agents affects the overall effectiveness of WAF bypass attempts.

The goal was to examine whether increasing the number of cooperating agents leads to a measurable improvement in penetration testing outcomes.

Table 3
Agent performance

Agent type	Main function	Key results
Recon-Agent	WAF detection, header and script collection	WAF type determined in 87% of cases; data collected for target mutation generation
Mutation-Agent	Generation of obfuscated payloads	Over 150 XSS/SQLi variants generated; split payloads yielded 80% success rate
Attack-Agent	Sending requests, capturing responses	Average system response time: 230–310 ms; most blocks in SQLi queries
Learning-Agent	Adapting attack strategies based on results	In the 3rd attack cycle, the accuracy of selecting effective mutations increased from 55% to 71%

The findings are visualized in the graph below and analyzed in detail (Figure 3).

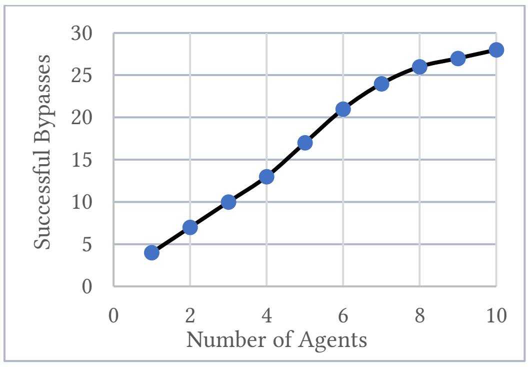


Figure 3: Relationship between the number of agents and penetration testing results

The scalability experiment of the multi-agent system demonstrated a clear positive correlation between the number of active agents and the number of successful bypasses of protection mechanisms, particularly the Web Application Firewall (WAF). The graph illustrates the results of 10 testing series, where the number of concurrently operating agents ranged from 1 to 10.

The results indicate that increasing the number of agents improves the system's capability to discover alternative attack vectors. With a single agent, the number of successful bypasses was minimal (4 cases). As new agents were added, the overall effectiveness steadily improved, reaching up to 28 successful bypasses at the 10-agent level. This can be explained by the parallel operation of agents, each performing a specialized role (e.g., reconnaissance, mutation, attack execution, and response analysis), which allowed the system to react more quickly to WAF rejections or blocks.

However, beginning at approximately 7–8 agents, a saturation effect becomes evident: the growth rate of successful attacks slows down. This suggests that the current system configuration approaches its upper efficiency limit. Such an effect may be caused by overlapping agent operations or system-level constraints (e.g., server request handling capacity or WAF throughput limitations).

Overall, the results confirm that scaling a multi-agent system increases its performance in automated penetration testing tasks, especially in the early stages of agent count expansion. This supports the feasibility of employing distributed MAS architectures in real-world cybersecurity environments.

The experimental data show that the system demonstrates stable performance in security testing tasks: the average success rate of attacks is approximately 60%. The highest effectiveness was observed for XSS attacks using split techniques, which proved successful in bypassing both signature-based and behavior-based WAF filters. This indicates the agents' ability to adapt and apply unconventional payload strategies. At the same time, polymorphic SQL injections were blocked more frequently, highlighting the need for further optimization of mutation techniques, particularly through the use of time-based or blind SQLi methods.

The system architecture, implemented on the JADE platform, combines automated scanning, adaptive payload generation, parallel agent interaction, and real-time learning. The four main types of agents — Recon-Agent, Mutation-Agent, Attack-Agent, and Learning-Agent — operate autonomously but remain coordinated within a shared strategy. This enables the system to adapt to changes in the defensive environment, reducing the risk of detection and providing flexibility in attack tactics.

Deploying agents across distributed nodes, including geographically separated locations, allows the system to simulate realistic attack scenarios and avoid blocking based on IP or geolocation filters. Moreover, the presence of a feedback loop between agents (especially between Attack-Agent and Learning-Agent) ensures the progressive refinement of attack strategies based on the outcomes of previous iterations.

Thus, the proposed approach demonstrates the advantages of dynamic payload mutation, distributed architecture, and agent self-learning capabilities. Future research will focus on integrating machine learning algorithms to predict WAF behavior and generate more sophisticated, context-aware attacks, thereby significantly enhancing the system's effectiveness in real-time environments.

7. Comparative analysis

To justify the feasibility of the proposed multi-agent system, a comparative analysis was conducted against existing automated pentesting tools, including Burp Suite, SQLMap, and approaches based on the SPADE platform.

Burp Suite is a widely used commercial framework for web application security analysis, enabling manual and semi-automated testing, including request interception, modification, and resubmission. However, it lacks full support for distributed execution, real-time adaptive behavior toward security systems, and requires significant user involvement. Furthermore, many of its advanced features are only available in the paid version, limiting its scalability for broad deployment.

SQLMap is a highly specialized tool designed to detect and exploit SQL injection vulnerabilities. It supports a wide range of obfuscation methods, automatic injection parameter detection, and partial WAF evasion through various techniques. Nevertheless, SQLMap is focused exclusively on a single class of attacks (SQLi) and does not provide inter-module interaction or support for multi-strategy behavior.

SPADE (Smart Python Agent Development Environment) is a platform for developing agents in Python that enables the creation of autonomous action scenarios. However, SPADE demonstrates limited flexibility when scaled or deployed in heterogeneous environments, and it lacks full agent mobility. Moreover, it does not offer comprehensive support for FIPA protocols, which complicates the development of adaptive and dynamically interacting systems.

The proposed system, based on JADE, stands out by combining several key functional advantages: support for distributed interaction between agents, the ability to operate autonomously across different nodes, adaptation to defense system responses, dynamic payload generation, and learning from attack results. This makes the system not only a scanning tool but also an intelligent mechanism for discovering evasion strategies in actively protected environments (WAF, IDS/IPS).

Table 4 presents a summarized comparison of the functionality of the mentioned approaches.

Table 4
Frequency of Special Characters

Criterion	Burd Suite	SQL Map	Spade	Jade
Automation Level	M	H(SQLi)	M	H (U)
Adaptation to WAF/IDS	L	P	L	H (D)
Distribution	N	N	P	Y
Agent Mobility	N	N	P	Y
Training/Adaptation	N	N	L	Y (LA)

Multiple Attack Types	Y	N (only SQLi)	Y	Y (XSS, other)
License Restrictions	Y	N	N	N
Openness and Modularity	P	H	H	H

Abbreviations used in the table: A – Absent, H – High, L – Limited, N – None, P – Partial, M – Medium, Y – Yes, U – Universal, D – Dynamic, LA – Learning Agent.

The comparison results indicate that the proposed JADE-based multi-agent system offers a number of advantages that ensure its effectiveness as a tool for adaptive and scalable penetration testing under modern cybersecurity challenges. Its distributed nature, learning capabilities, support for multiple attack classes, and open architecture provide a solid foundation for further integration of the system into real-world corporate environments.

The key advantages of the system were found to be:

- Parallel execution – agents operated independently, allowing simultaneous information gathering, attack generation, and learning;
- Adaptability – agents adjusted their strategies in response to WAF reactions, demonstrating features of machine learning;
- Distributed deployment – placing agents on different nodes helped reduce the risk of IP-based blocking and increased the amount of collected data;
- Coordinated interaction – the system's architecture ensured efficient data exchange between agents with minimal latency, enabled by the use of FIPA-compliant protocols provided by the JADE platform.

8. Integration aspects and system development plans

The proposed solution is based on a modular architecture that enables gradual scaling of the number of agents, with the possibility of centralized monitoring of their status. Although JADE is built in Java, the interaction interfaces between agents are implemented as REST endpoints and ACL messages, which opens the possibility for integration with Python scripts or external pentesting tools such as SQLMap, wfuzz, or Zap.

To facilitate the system's deployment in practical security environments, a simplified launch mode is provided through Docker containers with preconfigured JADE containers and configuration profiles, minimizing user workload and reducing deployment time. Future plans include the development of a web interface for managing agents and visualizing results.

Regarding attack types, the system currently focuses on XSS and SQLi, as they are common and representative vectors that are sensitive to filtering. However, the system's architecture is designed to support the extension of payload types. Future versions are planned to include support for SSRF attacks and heuristic testing for business logic flaws. For this purpose, the Mutation-Agent will be extended with templates for atypical requests, while the Learning-Agent will incorporate a response classification subsystem based not only on HTTP status codes but also on content analysis.

Another important aspect is that real-time adaptation to WAF rule changes indeed requires computational resources. To reduce the number of false positives or negatives, a prioritization mechanism has been implemented that ranks payloads based on their historical effectiveness. Additionally, attack attempts are buffered and analyzed based on server response time, with noisy or non-informative payloads filtered out. This helps reduce the frequency of low-value requests.

As a promising direction, the use of incremental learning in the Learning-Agent is being considered to dynamically refine effective patterns without requiring full model retraining. Furthermore, integration with popular CI/CD or DevSecOps platforms will enable the automation of pentesting as part of the continuous integration pipeline.

9. Conclusions

As a result of the conducted research, a distributed multi-agent system for automated web application security testing was implemented. The proposed solution is based on the JADE platform and integrates the functionality of several types of agents (Recon, Mutation, Attack, Learning), which interact autonomously in real time. A closed learning loop between the agents was established, where the effectiveness of attacks is used to dynamically update the payload mutation strategy, enabling the system to adapt to the behavior of WAF/IDS in real time. This approach provides parallel request processing, adaptation to protective mechanisms, and learning capability based on obtained results.

The system demonstrated stable performance in penetration testing tasks: the average attack success rate was approximately 60%, with the best results achieved for split-XSS. Polymorphic SQL injections proved less effective due to countermeasures from heuristic WAF filters. The distributed operation of agents reduced the risk of IP blocking and increased the system's overall resilience to detection.

Comparative analysis confirmed the advantages of the proposed system over traditional tools (Burp Suite, SQLMap) and agent-based platforms built on SPADE. Key differentiators include dynamic adaptation to defenses, distributed architecture, openness, and support for multiple attack classes.

Unlike previously described platforms, the proposed system demonstrates the ability to scale and process requests in a distributed environment with support for various attack types, including XSS (notably split-XSS) and polymorphic SQLi, with prospects for extension to SSRF and logic vulnerabilities.

Heuristic analysis and request obfuscation tools integrated into the system proved effective in realistic tests—the average success rate of attacks reached 60%, and up to 80% for split-XSS.

At the same time, results indicate the need for further improvement in supporting time-based SQLi, blind-XSS, and optimizing performance when processing large volumes of requests. The proposed system features a modular architecture and open interfaces, ensuring efficient integration with diverse external tools and platforms. Use of REST endpoints and Docker containerization support simplify deployment and scaling in practical environments.

Future work envisions integrating machine learning algorithms to predict WAF responses and generate more complex payloads. The developed system can be utilized both for research purposes and as a foundation for practical cybersecurity solutions.

Acknowledgements

We express our gratitude to all members of the group who diligently and persistently conducted this research. Special thanks are extended to Professor Y.Dorogyy and PhD candidate V. Kravchuk for their significant contributions to prior studies. We also convey our appreciation to the teams of the Departments of Information Security Management and Applied Mathematics and Computer Science, as well as to the postgraduate and master's students whose ideas greatly contributed to the modernization of the research and its integration into the educational process, and we hope the research will be further developed in their future scientific work.

Declaration on Generative AI

During the preparation of this work, the authors used generative AI tools by using the activity taxonomy in ceur-ws.org/genai-tax.html to improve grammar, spelling, and enhance the visual quality of Figure 1. After using these tools and services, the authors thoroughly reviewed and edited all AI-assisted content, ensured its accuracy, and presented it in their own words. The authors are fully responsible for the final content of the publication."

References

[1] Valentina, A., Vishwashri, R., and S. Rajadurai, "Finding Vulnerability in Web Application by using Pentesting," Int. J. Multidiscip. Res., 2024. doi: 10.36948/ijfmr.2024. v06i04.24517.

[2] M. Olivares-Naya, J. C. de Gracia, and A. S'anchez-Maci'an, "Adding web pentesting functionality to PTHelper," ArXiv, vol. abs/2410.12422, 2024. URL:https://api.semanticscholar.org/CorpusID:273375081

[3] L. De Lima, M. Horstmann, D. Neto, A. Grégio, F. Silva, and L. Peres, "On the Challenges of Automated Testing of Web Vulnerabilities," in 2020 IEEE 29th Int. Conf. Enabling Technol. Infrastruct. Collaborative Enterprises (WETICE), 2020, pp. 203–206. doi: 10.1109/WETICE49692.2020.00047.

[4] Y. Wijaya, "Web-Based Dashboard for Monitoring Penetration Testing Activities Based on OWASP Standards," J. Ilm. Tek. Elektro Komput. Inform., 2020. doi: 10.26555/jiteki.v16i1.17019.

[5] "Burp Suite," PortSwigger, URL: https://surl.gd/kteluz.

[6] B, S., NRK, S., J, T., & S, S. (2024). A Comparative Analysis of Vulnerability Management Tools: Evaluating Nessus, Acunetix, and Nikto for Risk Based Security Solutions. *arXiv*. URL: https://doi.org/10.48550/arXiv.2411.19123

[7] K. Vimala and S. Fugkeaw, "VAPE-BRIDGE: Bridging OpenVAS Results for Automating Metasploit Framework," in 2022 14th Int. Conf. Knowl. Smart Technol. (KST), 2022, pp. 69–74. doi: 10.1109/KST53302.2022.9729085.

[8] K. Kravari and N. Bassiliades, "A survey of agent platforms," J. Artif. Soc. Soc. Simul., vol. 18, 2015. doi: 10.18564/jasss.2661

[9] N. Gilbert and S. Bankes, "Platforms and methods for agentbased modeling," Proc. Natl. Acad. Sci. U. S. A., vol. 99, pp. 7197–7198, May 2002. doi: 10.1073/PNAS.072079499.

[10] S. Railsback, S. Lytinen, and S. Jackson, "Agent-based simulation platforms: Review and development recommendations," Simulation, vol. 82, pp. 609–623, Sep. 2006. doi: 10.1177/0037549706073695.

[11] C.-V. Pal, F. Leon, M. Paprzycki, and M. Ganzha, "A review of platforms for the development of agent systems," Inf., vol. 14, p. 348, Jul. 2020. DOI: 10.3390/info14060348

[12] E. Gonçalves, M. Cortés, G. Campos, Y. S. Lopes, E. Freire, V. Silva, K. Oliveira, and M. A. De Oliveira, "MAS-ML 2.0: Supporting the modelling of multi-agent systems with different agent architectures," Journal of Systems and Software, vol. 108, pp. 77–109, Oct. 2015. doi: 10.1016/j.jss.2015.06.008.

[13] P. Vrba, "JAVA-based agent platform evaluation," Sep. 2003, pp. 47–58. doi: 10.1007/978-3-540-45185-3_5.

[14] M. Ahmed, O. Kazar, and S. Harous, "Cyber-physical system model based on multi-agent system," IET Cyber-Physical Systems: Theory & Applications, Jun. 2024. doi: 10.1049/cps2.12096.

[15] R. Cavalcante, I. Bittencourt, A. Silva, M. Silva, E. Costa, and R. Santos, "A survey of security in multi-agent systems," Expert Systems with Applications, vol. 39, p

[16] K. Pan, Y. Lyu, and Q. Pan, "Adaptive formation for multiagent systems subject to denial-of-service attacks," IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 69, pp. 3391–3401, Aug. 2022. doi: 10.1109/TCSI.2022. 3168163.

[17] X. Yang, L. Yang, L. Dong, W.-H. Jin, M. Zhang, F. Yang, and Y. Lin, "Consensus tracking control for uncertain non-strict feedback multi-agent system under cyber-attack via resilient neuroadaptive approach," International Journal of Robust and Nonlinear Control, vol. 32, pp. 4251–4280, Feb. 2022. doi: 10.1002/rnc.6035.

[18] A. Dorri, S. Kanhere, and R. Jurdak, "Multi-agent systems: A survey," IEEE Access, vol. 6, pp. 28573–28593, Apr. 2018. doi: 10.1109/ACCESS.2018.2831228

[19] L. De Lima, M. Horstmann, D. Neto, A. Grégio, F. Silva, and L. Peres, "On the Challenges of Automated Testing of Web Vulnerabilities," in 2020 IEEE 29th Int. Conf. Enabling Technol. Infrastruct. Collaborative Enterprises (WETICE),

[20] B. Isiker and I. Sogukpinar, "Machine learning based web application firewall," in *Proc. 2nd Int. Informatics and Software Eng. Conf. (IISEC)*, 2021. doi: 10.1109/IISEC54230.2021.9672335

[21] S. Applebaum, T. Gaber, and A. AhmedAli, "Signature-based and machine-learning-based web application firewalls: A short survey," *Procedia Comput. Sci.*, vol. 189, pp. 359–367, 2021. doi: 10.1016/j.procs.2021.05.105

[22] P. Shahrivar, "Detection of vulnerability scanning attacks using machine learning," Master's thesis, KTH Royal Inst. of Technology, 2022.

[23] J.-Á. Román-Gallego, M.-L. Pérez-Delgado, M. Luengo Viñuela, and M.-C. Vega-Hernández, "Artificial Intelligence Web Application Firewall for advanced detection of web injection attacks," *Expert Syst.*, e13505, 2024. doi: 10.1111/exsy.13505

[24] A. Valenza, L. Demetrio, G. Costa, and G. Lagorio, "WAF-A-MoLE: An adversarial tool for assessing ML-based WAFs," *SoftwareX*, vol. 11, 2020. doi: 10.1016/j.softx.2020.100367

[25] HackYourMom, "Dangerous injections: Methods of bypassing protection using SQLMap," URL:https://hackyourmom.com/kibervijna/nebezpechni-inyekcziyi-metody-obhodu-zahystu-za-dopomogoyu-sqlmap/

[26] A. Doupé, M. Cova, and G. Vigna, "Why Johnny Can't Pentest: An Analysis of Black-Box Web Vulnerability Scanners," 2010, pp. 111–131. doi: 10.1007/978-3-642-14215- 4_7.

[27] K. Vimala and S. Fugkeaw, "VAPE-BRIDGE: Bridging OpenVAS Results for Automating Metasploit Framework," in 2022 14th Int. Conf. Knowl. Smart Technol. (KST), 2022, pp. 69–74. doi: 10.1109/KST53302.2022.9729085.

[28] M. Albahar, D. Alansari, and A. Jurcut, “An Empirical Comparison of Pen-Testing Tools for Detecting Web App Vulnerabilities,” Electronics, 2022. doi: 10.3390/ electronics11192991.

[29] V. Kravchuk, N. Maslova, and I. Dorohyi, "Automated xss vulnerability detection in web applications based on a multi-agent approach," Scientific Papers of Donetsk National Technical University. Series: “Computer Engineering and Automation," vol. 3, no. 4(36), pp., 2025. - 19–30. https://doi.org/10.31474/2786-9024/v3i4(36).324435 .