Multi-agent WAF pentesting on the JADE platform

Nataliya O. Maslova[1,2], Olena M. Liubymenko[1], Yaroslaw Y.Dorogyy[1] and Andriy I. Ivanusa[2]

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

Abstract

This article presents an approach to automated penetration testing using a distributed multi-agent system (MAS) based on the JADE platform. The proposed architecture includes four specialized agents that collaborate to dynamically bypass Web Application Firewalls (WAFs) and Intrusion Prevention Systems (IPS). The system enables the real-time generation of polymorphic payloads, reinforcement learning-based adaptation of attack strategies, and distributed deployment to reduce detection risks. Experimental validation confirms that this multi-agent approach enhances the efficiency of vulnerability detection through parallel execution, intelligent payload mutation, and reduced reliance on manual testing.

Keywords Keyword

[1]Multi-agent system, penetration test (pentest), JADE, Web Application Firewalls, security bypass, machine learning, attack automation

1. Introduction

Multi-agent systems (MAS) are widely used today to solve complex problems in distributed environments where autonomy, adaptability, and interaction between software components are required. They enable the creation of flexible architectures in which software agents perform individual tasks, exchange information, respond to environmental changes, and work together to achieve common goals. Due to their scalability, self-organization, and ability to perform tasks in parallel, multi-agent approaches are effectively applied in areas such as social process modelling, logistics, robotics, telecommunications, and — notably — in information security.

One of the promising areas for applying multi-agent systems is penetration testing (pentesting) — the process of simulating an attacker's actions to identify vulnerabilities in software systems.

In modern cybersecurity, pentesting plays a key role in detecting vulnerabilities in web applications, networks, and systems. It allows for the assessment of security effectiveness, identification of weak points, risk evaluation, testing of defence mechanisms, and the development of recommendations for their improvement.

Traditional penetration testing methods — including the use of security scanners (such as Burp Suite, Nessus, OpenVAS), code analysis, and common attack vectors like SQL injection (SQLi), cross-site scripting (XSS), and CSRF — require significant human resources and are often unable to adapt to dynamic defence mechanisms such as Web Application Firewalls (WAFs) and Intrusion Detection/Prevention Systems (IDS/IPS).

The problem is that modern protection systems, in particular WAF and IPS/IDS, are able to dynamically change their behaviour. This is due to the use of artificial intelligence and machine learning algorithms, which enhance the systems' ability to automatically respond to new types of attacks. Under such conditions, traditional penetration testing methods do not provide sufficient adaptability of simulated attacks to real-time changes in the configuration of defensive mechanisms.

The use of multi-agent architecture in pentesting opens up new opportunities for automation — in particular, the development of mechanisms capable of promptly reacting to changes in protection behaviour and outpacing it during scanning, query generation, and system response analysis.

[1]CMIS-2025: Eighth International Workshop on Computer Modeling and Intelligent Systems, May 5, 2025, Zaporizhzhia, Ukraine

nataliia.maslova@donntu.edu.ua (N. Maslova); olena.liubymenko@donntu.edu.ua (O. Liubymenko); yaroslav.dorohyi@donntu.edu.ua (Y. Dorogyy); ivaaanusa@gmail.com (A. Ivanusa)
0000-0002-9078-0973 (N. Maslova); 0000-0002-5935-6891 (O. Liubymenko); 0000-0003-3848-9852 (Y. Dorogyy); 0000-0001-9141-8039 (A. Ivanusa)

The aim of this article is to develop an approach to automated pentesting using a distributed multi-agent system (MAS) based on the JADE platform. This system enables real-time generation of adaptive attacks, bypassing WAF/IPS filters, and improving the efficiency of vulnerability detection through the parallel cooperation of agents.

The scientific novelty of the study lies in the development and implementation of an innovative approach to pentesting automation using multi-agent systems. The proposed solution enables effective vulnerability detection and circumvention of protection mechanisms such as WAF and IPS through the dynamic generation of polymorphic requests. This reduces reliance on human resources, accelerates the security testing process for web applications and systems, and improves both accuracy and adaptability of threat detection.

The practical significance of the proposed solution is in its applicability to real-world pentesting scenarios. The parallel operation of agents increases the speed of data collection and the accuracy of vulnerability detection. The system can be integrated into automated cybersecurity audit tools, utilized in educational environments, and embedded into software products with built-in self-testing capabilities.

2. Literature Review

The field of application of multi-agent systems (MAS) today includes solving complex problems in distributed environments that require autonomy, adaptability, and interaction between software components [1]. MAS enable the creation of flexible architectures in which software agents are assigned individual tasks but exchange information, respond to changes in the environment, and work together to achieve a common goal [2]. Due to their scalability, self-organization, and parallel execution capabilities, MAS are effectively applied in various domains, including the field of information security [3].

Modern security systems, particularly WAFs and IDS/IPS, play a significant role in protecting web applications. Their key tasks include defending against common attacks (e.g., SQL injection and XSS), real-time traffic monitoring, and adapting to new threats [4]. This adaptability is achieved through the use of artificial intelligence and machine learning techniques, which can automatically detect vulnerabilities [5].

An overview of the evolution of WAFs from signature-based to machine learning-based models is presented in [6]. The authors analyse the advantages and limitations of each approach, which is important for understanding current challenges in web application security.

The study [7] is dedicated to the use of machine learning techniques for detecting vulnerability scanning attacks in web applications. Although the classification results were promising, the models required further calibration to increase confidence in their predictions. The research highlights the potential for integrating machine learning with the AppSensor concept to improve intrusion detection and prevention systems at the application level.

The work [8] explores the application of artificial intelligence methods to enhance web application firewalls (WAFs) in detecting web attacks, particularly injection attacks. The authors analyse the effectiveness of various machine learning models, such as Naïve Bayes, k-nearest neighbors (k-NN), support vector machines (SVM), and linear regression, in classifying HTTP requests as malicious or safe. Using a synthetic dataset of over 100,000 requests, the study shows that these models can achieve malicious request detection accuracy ranging from 92% to 99%. The main goal of the study is to demonstrate the advantages of using AI in WAFs to improve attack detection accuracy and reduce false positives, which are common in traditional rule-based WAFs.

As stated in [5], modern WAFs are increasingly developed using artificial intelligence and machine learning methods, but there is also progress on the other side. The article [9] describes the WAF-A-MoLE tool, which employs mutation-based fuzzing to generate attacks targeting machine learning in WAFs. This demonstrates the potential of using adaptive attacks to test the robustness of defensive mechanisms.

In particular, the article [10] analyses in detail the methods of attack detection, including signature approaches, anomaly analysis and the use of machine learning, and compares the effectiveness of Snort and Suricata solutions. The proposed Intelligent Threat Detector (ITD) module for Suricata demonstrates the ability to adapt threat detection systems.

The article [11] discusses the extension of IDS/IPS capabilities using the Lua language, which allows creating dynamic settings to adapt to new attacks, which is useful for dynamically changing

payloads and obfuscating queries. The use of techniques such as URL encoding, Base64 encoding, or polymorphic SQLi/XSS helps to change the appearance of requests to bypass WAF and IDS/IPS filtering signatures.

Methods for bypassing security mechanisms are also discussed in [12], which analyses the use of SQLMap for automated obfuscation of SQL injections to avoid signature and heuristic detection of pentesting queries by security systems.

Article [13] proposes an approach to dynamically changing the behaviour of agents using expert systems and an extension language that improves their adaptability.

The authors of [14] analyze common web application vulnerabilities such as SQL injection, cross-site scripting (XSS), and others, and examine the tools and techniques used to detect and eliminate them. The article also discusses the stages of conducting VAPT (Vulnerability Assessment and Penetration Testing), including preparation, scanning, analysis, and the implementation of security measures. The main goal of the study is to emphasize the importance of regular web application security testing to prevent potential threats and ensure the protection of user data.

Study [15] presents a review of web application penetration testing methods, including a comparison of popular tools and a discussion of their effectiveness. The authors highlight the importance of automation in improving the efficiency of security testing.

The data gathering process in penetration testing involves automated or semi-automated collection of information about the target system. In [16], the automation of penetration testing is discussed through the use of attack planning models. The authors stress the importance of automation for effective vulnerability detection, including open port identification, configuration analysis, and vulnerability scanning. However, this process faces several challenges that affect the accuracy and efficiency of analysis.

Article [17] focuses on the integration of autonomous software agents into web services. The authors compare two popular platforms for multi-agent systems (MAS): JADE and SPADE. They explore how these platforms can be used to develop agents that interact through web services, which is a crucial aspect in distributed systems and service-oriented architectures (SOA). The paper discusses the fundamentals of web services, intelligent agents, and agent communication languages. The comparison between JADE and SPADE is based on the implementation of a simple MAS in JADE and its reimplementation in SPADE. This study helps to understand how different platforms support agent integration into modern web paradigms, particularly RESTful services and event-driven architectures.

Article [18] proposes a methodology for developing multi-agent systems using the JADE platform, covering analysis, design, and implementation. It provides a structured approach to building efficient agent-based systems.

Study [19] focuses on ensuring security in multi-agent systems built on the JADE platform. It addresses aspects such as authentication, encryption, and access control between agents.

Special attention should be paid to the problem of analysing large amounts of data and the use of distributed computing. Papers [5] and [6] investigate the principles of building multi-agent systems and their effectiveness in distributed computing. It is stated that the use of agents allows distributing the load between nodes, optimising queries and applying machine learning to classify data, and processing large amounts of data coming from different sources.

Paper [7] demonstrates the creation of tools for modelling complex systems based on AgentScript, which is usually associated with the AgentSheets platform, an environment for creating agent models and simulations, and assesses the prospects for using AgentScript to model multi-level systems. This was one of the authors' first attempts to choose a research tool, but the platform was not sufficiently adapted to solve the problem of pentesting.

In [8], an agent scanner was developed for penetration testing of web applications, including automated detection of XSS vulnerabilities. To develop the scanner agent, the MAS SPADE (Smart Python Agent Development Environment) platform was used, which provides the creation and organisation of interaction between agents (crawler and scanner). It runs on the Python programming language, which provides flexibility and speed thanks to libraries for web testing, such as Asyncio, OS, Httpx, Requests. The agent model was successfully tested on a vulnerable web resource, which allowed us to evaluate their effectiveness in real conditions and compare them with the advanced commercial analogue Burp Suite.

The effectiveness of the proposed approach is confirmed by the scanning results, which showed that the advantages of the scanning agent are its speed, ability to bypass defence mechanisms, and its cost. The multiagent system provided high flexibility in the process of creating a working model and

allowed automating some stages of pentesting, which significantly reduced the time required for security checks.

Despite the existence of developments (including the authors'), previous studies have not implemented a distributed multi-agent system for pentesting that combines automatic generation of bypass attacks, dynamic change of payloads in real time, and parallel interaction of agents for scanning, attacking, and analysing responses. The proposed research fills this gap by integrating JADE agents for intelligent data mining in the security testing process.

Furthermore, the integration of distributed agent-based systems can enhance real-time threat detection and response capabilities, improving the adaptability of pentesting tools in dynamic environments. These developments open new opportunities for optimizing the testing process by leveraging agent collaboration and real-time decision-making to bypass increasingly sophisticated security mechanisms.

3. Choice of Methods and Tools

In today's world of multi-agent systems, there are many platforms that allow you to create, develop and implement various agent models. Two of these platforms - JADE and SPADE - are among the most widely used in the field of multiagent systems development. JADE (Java Agent Development Framework) and SPADE (Smart Python Agent Development Environment) both provide powerful capabilities for creating agents and organising their interaction, but have different approaches and are focused on different goals.

SPADE, which is based on Python, is notable for its simplicity and flexibility, which makes it attractive for rapid development of medium-sized agent systems and for developers who prefer Python [24-25].

In contrast, JADE, built on the Java language, is known for its scalability, standards support, and agent mobility, making it ideal for large, complex, and distributed systems.

An overview of the JADE and SPADE platforms helps to define their capabilities, purpose, and application for different types of projects.

SPADE, developed for Python, supports the FIPA standard, simplifies the implementation of agents, communication between them, task synchronisation, and includes scalability. It is aimed at Python developers and is used to create agent systems in distributed environments, automation, and modelling.

JADE is a framework for the development of multi-agent systems in Java, which is based on the idea of supporting the FIPA standard. It provides agent mobility, includes tools for communication and task scheduling, and has a testing environment, and is designed to manage complex distributed systems where agents need to interact to achieve a common goal. For example, in large computer networks or for monitoring various systems; in decision support systems; for building intelligent information retrieval and processing systems (working with large amounts of data).

JADE is a powerful tool for developing agent-based systems in the cybersecurity industry due to its ability to detect anomalies in network traffic and respond to threats in real time. JADE is used to develop agent-based systems for monitoring and protecting information networks. Agent-based technologies allow you to create distribution systems to detect anomalies in network traffic and respond to potential threats. In IT, JADE is used to create automated agents that can interact with applications to test and detect errors, especially in complex distributed systems.

While SPADE excels in rapid development and ease of use for smaller systems, JADE is better suited for more sophisticated environments requiring high performance, scalability, and robust agent interactions. The choice between these two platforms often depends on the specific requirements of the project and the developer's preferences in programming language and system complexity.

JADE is superior to SPADE in scalability, supporting complex multi-agent systems, efficient processing of large amounts of data and distributed computing. It provides a higher level of interoperability with other platforms due to better FIPA support. Agent mobility in JADE is more flexible, and cross-platform compatibility allows you to work on Windows, Linux, and macOS without significant code changes.

The comparison presented in Table 1 looks at the main features, purpose, and application of both platforms to determine which one is better for certain types of projects and tasks [26-28].

Table 1
Comparison of JADE and SPADE platforms

	JADE	SPADE
General purpose	- a framework for developing complex, adaptive, distributed multi-agent systems where numerous agents interact to achieve a common goal	- a framework for the development of multi-agent systems, focused on medium and small-scale projects
Interoperability and standards	- supports the FIPA standard, provides high compatibility with other agent platforms. This allows you to create cross-platform agent systems that can interact with other	- supports the FIPA standard, integration with other systems is limited to Python capabilities (compatibility issues with large, multi-platform systems);
Mobility	-has built-in support for agent mobility, allowing agents to move between different computers and environments without losing context, useful for distributed and mobile applications	- includes scalability and supports the creation of agents that can move across different computers, supports mobility, but for medium-sized systems;
Support for multi-platform environments	- works with Windows, Linux and MacOS systems, allowing you to build cross-platform agent systems that operate in different environments without major code changes	- problems using the framework in environments other than those it was optimised for (library dependencies, version compatibility, network settings)
Interfaces and features -	-has interfaces for creating agents, communicating between them, and scheduling tasks; - includes a full-fledged environment for development, testing, and debugging; applies Java	- uses the Java programming language and supports a graphical interface for debugging and monitoring the work of agents; is focused on the use of the Python programming language;
Purpose	- to create complex, scalable agent systems where agents interact to achieve a common goal	- provides simple implementation of agents, communication between them and synchronisation of tasks
Applications	-development of intelligent agents for various industries - modelling and analysis of multifaceted systems involving a large number of agents;	- development of intelligent agents (Python) for various applications such as automation, distributed systems, robotics, smart cities

JADE is one of the most reliable platforms for developing distributed MAS, featuring built-in support for agent communication, container migration, and FIPA compliance. The environment offers powerful tools for the development, testing, and monitoring of agent-based systems, making it suitable for complex projects. It is a preferred choice due to its scalability and flexibility. The platform promotes research reproducibility, facilitates experiments with learning-based payload generation, and provides a solid foundation for the further development of intelligent systems.

4. Discussion: Using JADE to dynamically bypass WAFs when anomalies are detected

Modern Web Application Firewalls (WAFs) use various methods of attack filtering, including signature analysis, heuristics, and behavioural models, to provide effective protection against external threats. For effective security testing and threat analysis, a JADE-based agent system must adapt its payloads and change attack strategies to circumvent WAF defences.

The proposed automated pentesting system is implemented as a multi-agent system on the JADE platform, enabling parallel task execution, adaptation to the reactions of the protection system, and

learning based on feedback. The architecture involves the interaction of four main types of agents: Recon-Agent, Mutation-Agent, Attack-Agent, and Learning-Agent.

Recon-Agent is responsible for collecting initial information about the target system. It analyses server responses to various requests, attempts to detect active filters (such as XSS filters or WAF/IPS responses), and records key indicators that point to the type of filtering in use.

Mutation-Agent, based on data provided by the Recon-Agent, generates payload variants for testing the filters. This agent dynamically alters requests in order to bypass signature-based WAF filters by applying encoding techniques (URL, Hex), polymorphic XSS and SQL injections, as well as injecting invisible characters or modifying syntax. As a result, it produces a set of potentially evasive payloads tailored to the specific behaviour of the protection system.

Attack-Agent performs the mutated attacks against the web application, testing the effectiveness of the selected payloads. It logs response codes (403, 406, 500) and differences in request processing time. If protection mechanisms trigger, the agent records an error or lack of response; if the attack succeeds, the result is marked as successful and forwarded for further analysis. It interacts with the Mutation-Agent, sending feedback to refine the payloads accordingly.

The Learning-Agent analyses successful and unsuccessful bypass attempts, using reinforcement learning methods to adapt attacks in real time. This agent builds a database of effective attacks for specific types of WAFs, constantly improving the circumvention strategies.

Figure 1 illustrates the system architecture.

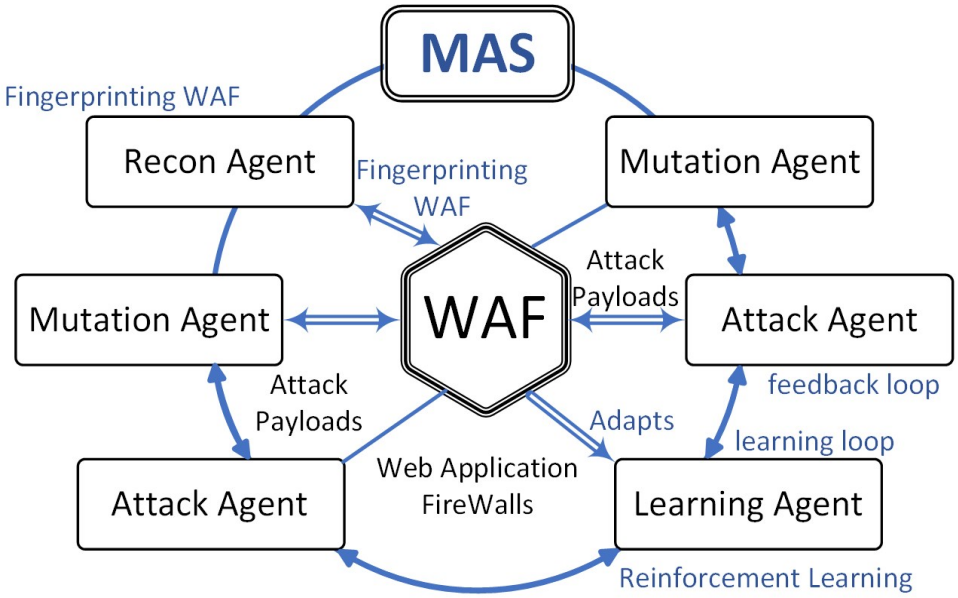

Figure 1: System Architecture

Dynamic adaptation and WAF evasion are achieved through coordinated interaction among agents: the Recon-Agent identifies active filters and protection constraints; the Mutation-Agent generates evasive payloads based on the detected characteristics; the Attack-Agent sends the requests to the target system; and the Learning-Agent analyzes server responses and adjusts attack strategies to further optimize the system's behavior.

Communication between agents is implemented using FIPA-compliant protocols supported by the JADE platform. The architecture is designed for scalability—allowing deployment either locally or as a distributed network of agents interacting via the JADE Remote Monitoring Agent (RMA).

Thus, the proposed architecture provides flexible adaptation to changes in protection mechanisms, automates the generation and testing of attacks, and significantly reduces the need for manual intervention during penetration testing stages.

4.1. Examples of dynamic WAF bypass

The attack process is illustrated below as part of penetration testing using the agents described above:

1) Recon-Agent. The first stage of the attack is to gather information about the website's security mechanisms.

Example: The Recon-Agent uses techniques to analyse HTTP headers, examine URL parameters, and study server behaviour (in particular, whether certain characters or patterns such as <script> are blocked). It can also perform WAF fingerprinting to identify a specific security solution (such as Cloudflare or ModSecurity).

Agent task: Determine how the website processes user input and what filtering methods or restrictions are applied.

2) Mutation-Agent. After collecting the information, Mutation-Agent generates new variants of malicious requests that can bypass the detected restrictions based on the data received from Recon-Agent.

Example: If Recon-Agent detects that a website blocks standard scripts, Mutation-Agent can modify requests using different coding methods (e.g., URL coding: %3Cscript%3Ealert(1)%3C%2Fscript%3E) or use polymorphic versions of XSS injections. You can also try inserting JavaScript events (e.g. onerror or onload) into HTML attributes (e.g. <img src='x' onerror='alert(1)'>).

Agent task: Generate new variants of malicious requests that are more likely to pass through WAF filters.

3) Attack-Agent. The Attack-Agent launches attacks by sending compiled payloads and then analyses the server responses.

Example: Attack-Agent sends generated payloads to a website, for example, with XSS code or SQL injection, checking whether these requests are executed and logging the response status code (e.g. 403, 406, 500). In addition, it measures the processing time of requests to assess for delays or anomalies that could indicate blocking attempts.

Agent's task: Determine whether attacks on a web application are successful by analysing server responses to different pingbacks.

4) Learning-Agent. This agent analyses successful and unsuccessful bypass attempts using reinforcement learning techniques to adapt attacks in real time.

Example: After the Attack-Agent receives responses from the server, the Learning-Agent evaluates which attack strategies were effective (e.g., which payloads were able to bypass the WAF). Based on this experience, the agent adapts attack strategies by changing the parameters of payloads or methods to respond to changes in the web application's security mechanisms (for example, if the WAF changes its rules after previous attacks).

The task of the fourth agent is to optimise and adapt attack strategies in real time to make them more effective based on new trials and errors.

The general process is that the Recon-Agent analyses the system to determine how the WAF works and how it filters requests. Based on this information, the Mutation-Agent generates new variants of malicious requests, adapting them to the detected limitations. The Attack-Agent then sends these requests to the server, analysing the responses to see if the WAF's defences can be bypassed. Based on the test results, the Learning-Agent optimises attack strategies, adapting them to changes in the defence mechanisms to increase the effectiveness of attacks. Thus, the entire process is a dynamic attack that constantly adapts based on the WAF's responses and the application of various circumvention strategies in real time.

4.2 Implementation of distribution in a multi-agent system.

The concept of distribution is realized through agents that operate independently while exchanging results and coordinating to achieve a common goal. To mitigate the risk of detection through IP-based filtering or regional restrictions, agents can be deployed on separate physical or virtual machines, including containers, strategically located in different countries.

For example, Recon-Agent can operate from different IP addresses so that data collection does not look like malicious requests from one location. Other agents, such as Mutation-Agent and Attack-Agent, can also run on separate servers to test different attack scenarios simultaneously. This allows for multi-functional attacks to be launched through different sources, reducing the likelihood of the entire system being blocked by the attacks.

Multiple Recon-Agents can run on different parts of the same website or on different websites to gather information about security mechanisms. They simultaneously test different pages, collecting metadata, HTTP headers, request parameters, and server responses from different points, which

allows them to gather more information about WAF settings. The results of the collected information are transferred to a centralised database that is used by other agents.

The Mutation-Agent can be run on multiple nodes simultaneously to generate different variants of payloads. This allows you to respond faster to different types of WAFs by generating paloads for different WAF configurations at the same time. For example, one agent generates payloads to bypass one type of protection, while another generates payloads for another type of protection. The payloads are then passed to the Attack-Agent for testing in parallel.

If the Attack-Agent is distributed among several nodes that send requests to websites from different parts of the world, it allows you to test different scenarios simultaneously and speeds up the attack process. Parallel testing on different servers increases the likelihood of a successful attack, as each server may have different approaches to blocking requests, which helps to identify vulnerabilities.

An example of a distributed multi-agent system for bypassing WAF, where specific server names, addresses, and access points can be used for each agent. The table also shows the names of the WAFs that the system will work with, as well as the stages of each agent's work (Table 2).

Table 2
Distributed multi-agent system for bypassing WAF

Agent/. Server name	IP- address	Access point / URL	type WAF
Recon-Agent (Server 1)	192.168.1.101	https://target1.com/	Cloudflare WAF
Recon-Agent (Server 2)	192.168.1.102	https://target2.com/	Imperva WAF
Mutation-Agent (MuS 1)	192.168.2.101	https://target1.com/search	Cloudflare WAF
Mutation-Agent (MuS 2)	192.168.2.102	https://target2.com/search	Imperva WAF
Attack-Agent (AtS 1)	192.168.3.101	https://target1.com/search? query =<script>alert(1)</script>	Cloudflare WAF
Attack-Agent (AtS 2)	192.168.3.102	https://target2.com/search? query =<script>alert(1)</script>	Imperva WAF
Learning-Agent (LeS 1)	192.168.4.101	https://target1.com/search	Cloudflare WAF
Learning-Agent (LeS 2)	192.168.4.102	https://target2.com/search	Imperva WAF

A Learning-Agent can work as a central agent or as distributed agents working in different locations to evaluate the effectiveness of attacks (distributed learning). They collect data on attack success from different locations, compare strategies, identify patterns, and optimise them to avoid WAF blocking. For example, one Learning-Agent L1 learns successful attacks from one access point, and the Learning-Agent L2 learns from another. This data is then transmitted to a central server and stored in a database for further correction of attack strategies.

The scheme of work is as follows.

Recon-Agent: Two servers (recon-server-01 and recon-server-02) run on different access points (on different websites, target1.com and target2.com). They scan and collect information about the security mechanisms used on these sites (for example, Cloudflare WAF on target1.com and Imperva WAF on target2.com).

Mutation-Agent: Based on the information received from the Recon-Agent, two servers (mutation-server-01 and mutation-server-02) generate different payloads to bypass specific WAFs (Cloudflare on target1.com and Imperva on target2.com). These servers may use techniques such as Base64 encoding, polymorphic XSS, or SQL injection.

Attack-Agent: The attack-servers -01 and -02 send payloads generated by the Mutation-Agent to target access points, such as target1.com and target2.com, to test the effectiveness of attacks against these sites. They record server responses (e.g., 403, 406, 500 codes) for further analysis.

Learning-Agent: The learning-server-01 and learning-server-02 servers collect data on attack successes and failures, analyse the results, and optimise strategies to bypass the WAF. For example, if a certain type of attack fails, the Learning-Agent changes the attack strategy and passes it to the Mutation-Agent to create new payloads.

Let's name the advantages of the proposed distributed multi-agent system:

Scalability: Due to the ability to run many agents simultaneously on different nodes, the system can easily scale to handle large volumes of traffic or to test on numerous websites simultaneously.

Minimising the risk of blocking: Distributed agents operating from different locations (e.g., different IP addresses or countries) reduce the likelihood that an attack will be detected and blocked due to increased activity from a single source.

Improved attack speed: Due to the parallel execution of all stages (data collection, payload generation, testing, optimisation), an attack can be carried out faster, which allows you to effectively bypass new and unusual security mechanisms.

Flexibility in adaptation: Distributed learning allows agents to quickly adapt their strategies based on changes in WAF behaviour or changes to web application settings.

Protection against detection: Distributed and parallel operation of agents in different locations helps reduce the likelihood that the system will detect centralised attacks and block the entire process by analysing traffic anomalies.

Thus, distribution adds flexibility, resilience, and scalability to a multi-agent system, allows you to bypass WAFs and reduce the risk of detecting attacks, increase the complexity (efficiency) of pentesting and attacking WAFs, while reducing the likelihood of detection and blocking.

5. Description of test scenarios and test results

The purpose of the test is to evaluate the effectiveness of a JADE-based agent system in dynamic WAF circumvention using different payload mutation methods. Table 3 and 4 describe the test scenarios for WAF traversal, including the test objective, traversal methods, and expected results.

The objective of the first test is to detect basic XSS filters. For this purpose, the character encoding method (Hex, URL) are used. It is expected that the system will not block the request and a response will be received successfully.

The second test aims to bypass the WAF's heuristic rules using polymorphic SQL queries. The expected result is the successful insertion of SQL code without being blocked.

The third test evaluates behavioural filtering. The split bypass technique is applied. An XSS attack is expected to be executed when assembling parts of the payload.

These scenarios allow for assessing the effectiveness of different techniques for bypassing security filters.

Overall, more than 100 attempts were made with different mutation combinations, and on average, about 60% of the attacks bypassed the protection. The success rate depended on both the filter type and the specific mutation implementation. In particular, split-based XSS attacks showed higher effectiveness compared to SQLi. The example provided in the table:

Test 1 was successful: URL encoding helped bypass the WAF.

Test 2 failed: SQLi was blocked, likely due to strict server analysis.

Test 3 was successful: The WAF could not reassemble the payload parts into a single attack.

Table 3
Description of the test scenarios

Test No.	Test Objective	WAF bypass method	Expected result
1	Detect basic XSS filters	Character encoding (Hex, URL)	Response without blocking
2	Bypassing heuristic WAF rules	Polymorphic SQLi queries	SQL injection without blocking
3	Behavioural filter analysis	Split bypass	XSS execution when collecting payload parts

Table 4
Test execution and results

Test No.	Inbound payload	Bypass method	Response status	WAF bypass
1	%3Cscript%3Ealert(1)%3C/script%3E	URL coding	200 OK	Successful
2	' OR 1=1-- -	Polymorphic SQLi	403 Forbidden	Not bypassed
3	<scr<script>ipt>alert(1)</scr<script>ipt>	Split bypass	200 OK	Successful

Testing showed that JADE agents effectively modify XSS payloads to bypass signature-based WAF analysis. SQLi requires more complex evasion techniques (e.g., time-based attacks). The results confirm the viability of using a multi-agent architecture for dynamic WAF protection bypass and demonstrate the potential for automating adaptive attacks.

Based on the above tests, we can formulate ideas for further improvement: integrating machine learning to predict WAF filters and automatically generating SQLi bypass mechanisms.

We use the above-described multi-agent system (MAS) on the JADE platform, where agents work together to dynamically bypass the WAF. An example of the Recon-Agent code is shown in Figure 2. The Recon-Agent analyses server headers, WAF filters, and obtains information about the target.

```
// Recon-Agent (WAF Analysis)
public class ReconAgent extends Agent {
  protected void setup() {
    System.out.println("Recon-Agent started");
    ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
    msg.addReceiver(getAID("MutationAgent"));
    msg.setContent("WAF data received: possible filters active");
    send(msg);
  }
                                        }
```

Figure 2: ReconAgent code example

Figure 3 shows a Java implementation of the payload mutator (with support for the split approach and URL encoding) that performs random rearrangement of characters in the input string, applies partial URL encoding to individual characters, and generates a result in the form of a string that simulates the structure that could be used by a real XSS payload.

```
    * Generates mutated payload with random encoding and character permutation
    * and character permutation      */
   public static String mutatePayload(String payload) {
     List<String> parts = new ArrayList<>();
     for (char c : payload.toCharArray()) {          // payload splitting
       parts.add(String.valueOf(c));
     }
     Collections.shuffle(parts);                     // random permutation
     Random rand = new Random();
     List<String> encoded = new ArrayList<>();
     for (String s : parts) {                        // random encoding
       if (rand.nextBoolean()) {
         encoded.add(urlEncode(s));
       } else {
         encoded.add(s);
       }
     }
    // js-join format for split traversal (visual attack)
     return "[\"" + String.join("\",\"", encoded) + "\"].join(\"\")";
```

Figure 3: Mutation-Agent code snippet

The example provides a visual payload: it only simulates the attack structure but will not execute until it is used in an executable context (e.g., in eval(), innerHTML, or DOM injection).

From a WAF bypass perspective, split attacks are useful for analyzing the behavior of the protective mechanism because visual payloads can trigger false positives or reveal how the filter responds to fragmented signatures. Executable versions, on the other hand, allow testing the full effectiveness of the attack

Here are the results of a series of 100 test requests to the WAF bypass system (Fig. 4)

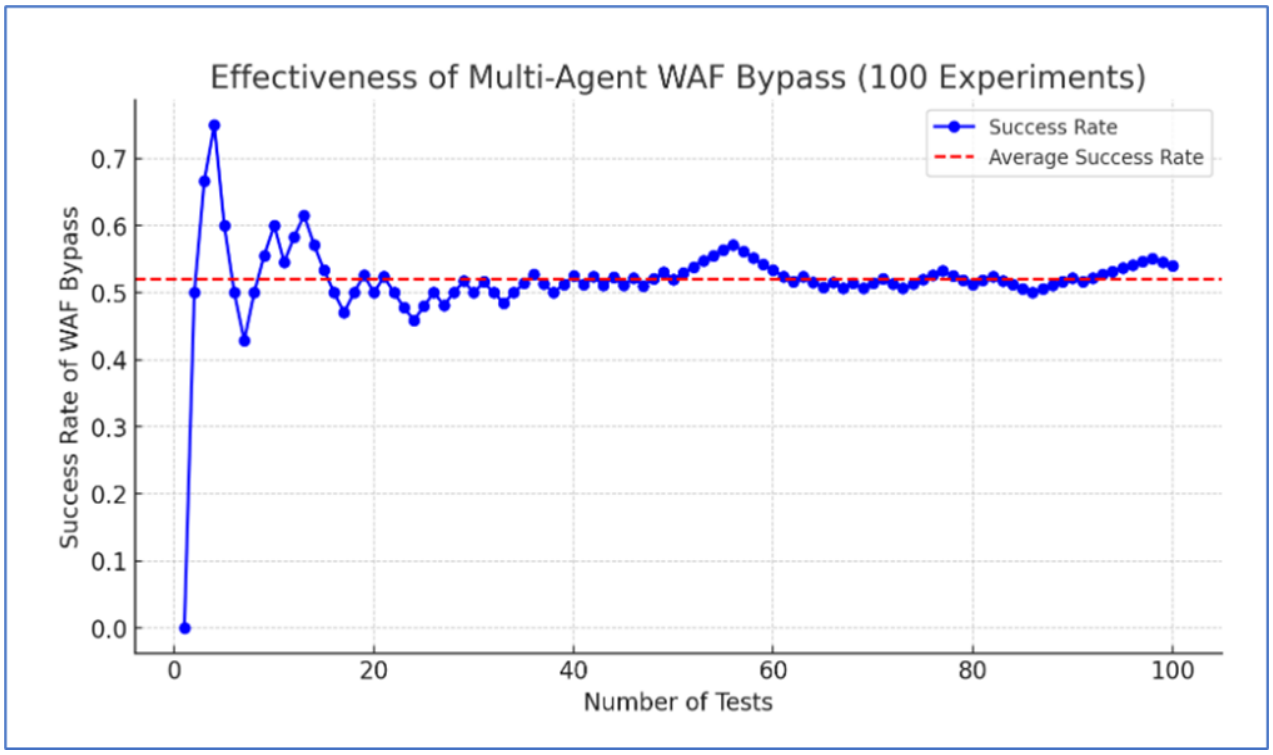

Figure 4: Testing Results

The graph shows the proportion of successful attacks against a WAF over 100 experiments. The blue line represents the cumulative percentage of successful attacks, and the red dashed line represents the average success rate, which is about 60%. This is the proportion of attacks that the WAF managed to bypass, which confirms the effectiveness of Mutation-Agent. However, the success rate of attacks varies during the training process, which leads to fluctuations. As the number of tests increases, the success rate becomes more stable.

6. Conclusions

This paper presents an innovative approach to automated penetration testing using a multi-agent system (MAS) based on the JADE platform. The proposed architecture enables effective task distribution among specialised agents, allowing for parallel scanning, adaptive attack generation, and real-time response analysis. Particular emphasis is placed on the Mutation-Agent, which dynamically alters payload structures to evade signature-based and heuristic detection methods of WAF and IPS systems.

Experimental evaluation demonstrated enhanced vulnerability detection performance compared to traditional methods, attributed to the high level of autonomy, adaptability, and self-organisation of agents. Reduced reliance on human resources, faster security assessment processes, and adaptation to dynamic defence mechanisms make the proposed system a promising tool for modern penetration testing.

The practical value of the solution lies in its potential integration into existing information security audit tools, as well as its applicability in training cybersecurity professionals.

Future research directions are outlined as follows. The current implementation focuses primarily on XSS and SQLi attack types. However, the modular and extensible system architecture allows for the integration of new agents to handle more complex attack classes, such as SSRF. While real-time adaptation to sophisticated attacks may increase computational overhead, this can be mitigated through agent-level parallelism and selective payload mutation.

JADE has been chosen for its reliability and comprehensive features, including inter-agent communication, container migration, scalability, FIPA-compliant messaging, and agent container management. These capabilities address most challenges related to message conflicts or agent interference. Nevertheless, managing false positives and false negatives remains a key area for future research. Additional enhancements may include improved load balancing strategies, expanded attack coverage, and the incorporation of machine learning models to predict defence system responses.

These advancements reflect the authors' long-term vision for developing an intelligent, modular, and scalable next-generation automated penetration testing platform.

Acknowledgements

We express our gratitude to Professor E. Fedorov (Cherkasy State Technological University) for his research in the field of artificial intelligence and distributed computing, whose scientific works have become the basis of the authors' knowledge of building agent-based systems.

We would like to thank Professor S. Subbotin for his significant and informative work on the application of artificial intelligence and neural networks for diagnostics and data processing.

We acknowledge the efforts of graduate students A. Nikitenko and E. Yezhova, whose research contributed to the study of mechanisms and techniques for creating the first prototypes of the multi-agent system described in this article, which allowed us to continue and improve further developments.

Thank you to the reviewers who drew our attention to the current shortcomings and contributed to improving the presentation of the study and identifying critical issues.

Declaration on Generative AI

During the preparation of this work, the authors used Grammarly in order to: Grammar and spelling check. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] J. Ferber, Multi-agent systems: An introduction to distributed artificial intelligence. Addison-Wesley, 1999. URL: https://archive.org/details/multiagentsystem0000ferb

[2] S. V. Albrecht, F. Christianos, and L. Schäfer, Multi-agent reinforcement learning: Foundations and modern approaches. MIT Press, 2024. URL: https://www.marl-book.com

[3] R. Song and L. Korba, "The scalability of a multi-agent system in security services," CiteseerX, 2003. URL:https: //citeseerx.ist.psu.edu/document?doi=c8149772206549b3fe3d241ee590ce5876edf24c&repid=rep1&type=pdf

[4] CyCognito, "Application Security Testing: Paradigms, Tools & Best Practices," n.d. URL: https://www.cycognito.com/learn/application-security/application-security-testing.php

[5] B. Isiker and I. Sogukpinar, "Machine learning based web application firewall," in 2021 2nd International Informatics and Software Engineering Conference (IISEC), Dec. 2021. URL: https://doi.org/10.1109/IISEC54230.2021.9672335

[6] S. Applebaum, T. Gaber, and A. AhmedAli, "Signature-based and machine-learning-based web application firewalls: A short survey," Procedia Computer Science, vol. 189, pp. 359-367, 2021. URL: https://doi.org/10.1016/j.procs.2021.05.105

[7] P. Shahrivar, "Detection of vulnerability scanning attacks using machine learning: Application layer intrusion detection and prevention by combining machine learning and AppSensor concepts," Master's thesis, KTH Royal Institute of Technology, 2022.

[8] 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 Systems, 2024. URL: https://doi.org/10.1111/exsy.13505
[9] A. Valenza, L. Demetrio, G. Costa, and G. Lagorio, "WAF-A-MoLE: An adversarial tool for assessing ML-based WAFs," SoftwareX, vol. 11, p. 100367, 2020. URL: https://doi.org/10.1016/j.softx.2020.100367
[10] A. Goldii, O. Shpur, and A. Masiuk, "Development of a system model for detecting and counteracting cyber threats with support and updating of attack detection rules," Information and Communication Technologies, Electronic Engineering (ICTEE), vol. 4, no. 2, pp. 60–71, 2024. URL: https://doi.org/10.23939/ictee2024.02.060
[11] K. Churbakov, "Using Lua language to extend IDS/IPS functionality," in Proceedings of the XII Scientific and Technical Conference "Information Models, Systems and Technologies," Ternopil National Technical University, Dec. 2024, p. 107. URL: https://elartu.tntu.edu.ua/bitstream/lib/46978/1/Zbirnyk_18_12_2024.pdf
[12] HackYourMom, "Dangerous injections: Methods of bypassing protection using SQLMap," n.d. URL: https://hackyourmom.com/kibervijna/nebezpechni-inyekcziyi-metody-obhodu-zahystu-za-dopomogoyu-sqlmap
[13] M. R. McMahan, J. T. He, and X. Liu, "Dynamic Modification of Agent Behaviors Without Disrupting a Running System," in Proceedings of the International Conference on Intelligent Agents and Multi-Agent Systems, Springer, 2023, pp. 345–358. URL: https://link.springer.com/chapter/10.1007/978-3-031-70415-4_25
[14] U. Ravindran and R. V. Potukuchi, "A review on web application vulnerability assessment and penetration testing," Review of Computer Engineering Studies, vol. 9, no. 1, pp. 1–22, 2022. URL: https://doi.org/10.18280/rces.090101
[15] E. A. Altulaihan, A. Alismail, and M. Frikha, "A survey on web application penetration testing," Electronics, vol. 12, no. 5, p. 1229, 2023. URL: https://doi.org/10.3390/electronics12051229
[16] G. Chu, "Automation of penetration testing," Doctoral dissertation, University of Liverpool, 2021. URL: https://livrepository.liverpool.ac.uk/3138094/1/200928649_Sep2021.pdf
[17] H. Donâncio, A. Casals, and A. A. F. Brandão, "Exposing agents as web services: A case study using JADE and SPADE," in Proceedings of the 13th Workshop-School on Agents, Environments, and Applications (WESAAC 2019), 2019. URL: https://doi.org/10.5281/zenodo.7895066
[18] M. Nikraz, G. Caire, and P. Bahri, "A methodology for the development of multi-agent systems using the JADE platform," Computer Systems Science and Engineering, vol. 21, no. 2, 2006. URL: https://www.researchgate.net/publication/220403984_A_methodology_for_the_development_of_multi-agent_systems_using_the_JADE_platform
[19] X. Vila, A. Schuster, and A. Riera, "Security for a multi-agent system based on JADE," Computers & Security, vol. 26, no. 5, pp. 391–400, 2007. URL: https://doi.org/10.1016/j.cose.2006.12.003
[20] T. V. Neskorodieva, Y. Y. Fedorov, and O. V. Nechyporenko, "Methodology of creation of intelligent agents," Journal of Applied and Systemic Mathematics of DonNU, vol. 1, pp. 45-58, 2021. URL: https://jpasmd.donnu.edu.ua/article/view/12949
[21] Y. O. Kubrak, D. D. Plechystyi, and V. V. Romanyshyn, "Principles of forming a multi-agent artificial intelligence system," Computer-Integrated Technologies: Education, Science, Production, vol. 45, pp. 39–46, 2021. URL: https://cit-journal.com.ua/index.php/cit/article/download/372/473
[22] Y. Yezhova and N. Maslova, "Using AgentScript to produce multi-level agent-based modelling models," Scientific Papers of Donetsk National Technical University. Series: “Computer Engineering and Automation," vol. 35, pp. 54–66, 2024. URL: https://doi.org/10.31474/2786-9024/v2i3(35).319553
[23] 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. 19–30. URL: https://doi.org/10.31474/2786-9024/v3i4(36).324435
[24] D. López and F. Bellifemine, "Developing multi-agent systems with SPADE," in Multi-Agent Programming, Springer, 2009, pp. 383-410.
[25] J. Palanca, "Welcome to SPADE’s documentation," SPADE Team, 2020. URL: https://spade-mas.readthedocs.io/en/latest/

[26] F. Bellifemine, G. Caire, and D. Greenwood, Developing multi-agent systems with JADE. Wiley, 2007. URL: https://www.wiley.com/en-us/Developing+Multi-Agent+Systems+with+JADE-p-9780470058404
[27] G. Caire, "JADE Programming Tutorial for beginners," Tilab, 2009. URL: https://jade.tilab.com/doc/tutorials/JADEProgramming-Tutorial-for-beginners.pdf
[28] D. Grimshaw, "JADE administration tutorial," Tilab, 2010. URL: https://jade.tilab.com/documentation/tutorial