

University of Southampton

Faculty of Engineering and Physical Sciences

Electronics and Computer Science

A detection method for cross-site scripting

by

Yang Ding

September 2020

Supervisor: Dr. Nawfal Fadhel

Second Examiner: Dr. Julian Rathke

A dissertation submitted in partial fulfilment of the degree
of MSC Cyber Security

University of Southampton

ABSTRACT

FACULTY OF ENGINEERING AND PHYSICAL SCIENCES
ELECTRONICS AND COMPUTER SCIENCE

Master of Science

by Yang Ding

Cross-site scripting (XSS) attack is a computer vulnerability that allow attacker to insert their own script to the web page to modify the web page in a way they want. By using XSS attack, attacker can modify the web page, stealing information from user, etc. These days, developer usually fix known XS vulnerabilities before been attack, or clean up the suspicious input using firewall before it been sent to the server. Both approaches cannot let developer react to the attack if a XSS attack really happen. To answer the question that “Through adding attack detection tools on server side, how much improvement on the efficiency can a system get when it is attacked?”, his project focuses on building a server side detection tool using pattern matching technique based on regular expression, the tool will notify the developer for possible XSS attacks and in this case make developer react earlier when XSS happened real time. We use three attack script to attack a server and use our detection tool on server side to test the effectiveness of the tool. The final result show both advantage and disadvantage of this detection technique.

Acknowledgements

I would like to thank my project supervisor for giving me lots of support on writing thesis, he also provides me much information on different part of my project when I came into problems. My second examiner provides some advice about the thesis and I really appreciate that. Also, I would like to thank my university for the online library support which helps a lot in early stage of my project. I also appreciate the help from my friend when I am having trouble extracting the input data from HTML requests.

Statement of Originality

- I have read and understood the [ECS Academic Integrity](#) information and the University's [Academic Integrity Guidance for Students](#).
- I am aware that failure to act in accordance with the [Regulations Governing Academic Integrity](#) may lead to the imposition of penalties which, for the most serious cases, may include termination of programme.
- I consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to verify whether my work contains plagiarised material, and for quality assurance purposes.

You must change the statements in the boxes if you do not agree with them.

We expect you to acknowledge all sources of information (e.g. ideas, algorithms, data) using citations. You must also put quotation marks around any sections of text that you have copied without paraphrasing. If any figures or tables have been taken or modified from another source, you must explain this in the caption and cite the original source.

I have acknowledged all sources, and identified any content taken from elsewhere.

If you have used any code (e.g. open-source code), reference designs, or similar resources that have been produced by anyone else, you must list them in the box below. In the report, you must explain what was used and how it relates to the work you have done.

I have not used any resources produced by anyone else.

You can consult with module teaching staff/demonstrators, but you should not show anyone else your work (this includes uploading your work to publicly-accessible repositories e.g. Github, unless expressly permitted by the module leader), or help them to do theirs. For individual assignments, we expect you to work on your own. For group assignments, we expect that you work only with your allocated group. You must get permission in writing from the module teaching staff before you seek outside assistance, e.g. a proofreading service, and declare it here.

I did all the work myself, or with my allocated group, and have not helped anyone else.

We expect that you have not fabricated, modified or distorted any data, evidence, references, experimental results, or other material used or presented in the report. You must clearly describe your experiments and how the results were obtained, and include all data, source code and/or designs (either in the report, or submitted as a separate file) so that your results could be reproduced.

The material in the report is genuine, and I have included all my data/code/designs.

We expect that you have not previously submitted any part of this work for another assessment. You must get permission in writing from the module teaching staff before re-using any of your previously submitted work for this assessment.

I have not submitted any part of this work for another assessment.

If your work involved research/studies (including surveys) on human participants, their cells or data, or on animals, you must have been granted ethical approval before the work was carried out, and any experiments must have followed these requirements. You must give details of this in the report, and list the ethical approval reference number(s) in the box below.

My work did not involve human participants, their cells or data, or animals.

Contents

Acknowledgements	ii
1 Introduction	1
1.1 Problem	1
1.2 Research question	2
1.3 Hypothesis	2
2 Background and Literature Review	3
2.1 XSS	3
2.1.1 Definition	3
2.1.2 How XSS happen and possible consequence	3
2.1.3 Type of XSS attack	4
2.1.3.1 Stored XSS attack	4
2.1.3.2 Reflected XSS attack	4
2.1.3.3 DOM-based XSS attack	5
2.2 Testing platform	5
2.2.1 Vagrant	5
2.3 Vulnerable web application on the target machine	6
2.3.1 About Damn Vulnerable Web Application (DVWA)	6
2.4 Tools used to attack the target	7
2.4.1 Tools been used	7
2.4.1.1 Cross Site “Scripter” (XSSer)	7
2.4.1.2 Zad Attack Proxy	8
2.4.1.3 Burp suite Community edition	8
2.4.2 Difference between these tools	9
2.5 Related works	9
3 Requirements Analysis and Project Plan	11
3.1 Functional requirements	11
3.1.1 Importance level analysis	12
3.2 Non-functional requirements	12
3.3 MoSCoW analysis	13
3.4 Risk analysis	13
3.5 Project plan	16
4 Design and Implementation	17
4.1 Program design	17
4.2 Test environment	19

4.3	Implementation	20
4.3.1	Request listening and extracting	20
4.3.1.1	Listen to HTTP request	20
4.3.1.2	Extracting the data	21
4.3.2	Checking the request data	21
4.3.2.1	Data need to check in HTTP request	21
4.3.2.2	Detection method	23
4.3.3	Extract information needed and store the suspicious request	25
5	Result and Discussion	27
5.1	Testing process	27
5.1.1	Attack using XSSer	27
5.1.2	Attack using ZAP	28
5.1.3	Attack using Burp Suite	29
5.2	Testing result	30
5.3	Discussion	31
5.3.1	Compare stored XSS and reflected XSS	31
5.3.2	Compare result from three tools	31
5.3.2.1	XSSer result	31
5.3.2.2	ZAP result	32
5.3.2.3	Burp Suite result	33
6	Conclusion and Further Work	34
6.1	Conclusion	34
6.2	Further work	35
6.3	Legal aspect	35
A	Appendix	36
A.1	Document in COMP6211 - Project Preparation	36
	Bibliography	41

List of Figures

1.1	Percentage of different vulnerabilities been found in 2019	1
2.1	DVWA application page	6
2.2	XSSer	7
2.3	ZAP user interface	8
2.4	Burp suite GUI welcome page	8
3.1	MoSCoW analysis	13
3.2	Project Plan	16
4.1	Program logic	18
5.1	XSSer attack result	28
5.2	ZAP attack steps	28
5.3	ZAP fuzzing result window	29
5.4	Burp Suite attack setting	29

List of Tables

2.1	Difference between 4 tools	9
3.1	Functional requirements	11
3.2	Importance level example	12
3.3	Importance level of functional requirements	12
3.4	Non-functional requirements	13
3.5	Risk level example	14
3.6	Importance level of functional requirements	15
4.1	Test environment	19
4.2	Tags that are frequently used for XSS attack	24
4.3	Variables that been stored into database	25
5.1	overall result	30

Chapter 1

Introduction

1.1 Problem

In summary, the problem here is that *even through XSS is one of the most vulnerable and commonly seen vulnerabilities, there isn't a good way to detect XSS attack while the server is been attack.*

According to the data from CVE security vulnerability database (2020), from 1999 to 2019, 12.5% of the recorded vulnerabilities are XSS attacks. In 2019, among all the vulnerabilities recorded in CVE, 1593 are based on XSS, make it the second common vulnerabilities of the year, Figure 1.1 is a pie chart of vulnerabilities been recorded in 2019.

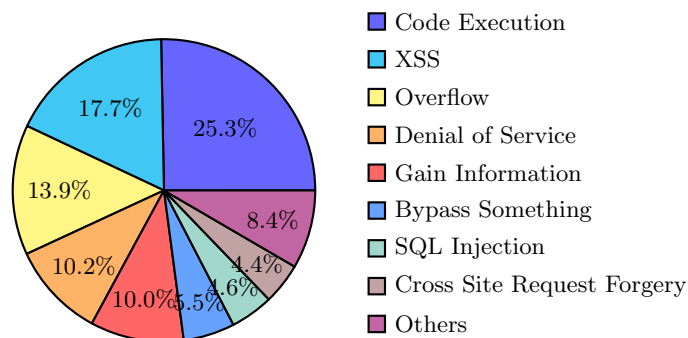


FIGURE 1.1: Percentage of different vulnerabilities been found in 2019

Usually, the common way to deal with XSS attack is to find the vulnerabilities before the hacker and fix the vulnerabilities first. Developers use different XSS vulnerability detecting/attacking scripts to find as many vulnerabilities as possible in their own system, then fix them one by one. The effectiveness of this process highly depends on how many vulnerabilities' developers can find, with another concern that the security of new code been added after the testing is still not been guaranteed.

1.2 Research question

As been mentioned in last section, common way to protect a server from XSS attack is to find the vulnerabilities before attackers make use of it. This approach will fix all the vulnerabilities been found, but it is always possible to miss some vulnerabilities and the possibility of adding new vulnerabilities after an update is also needed to be taken into consideration. Also, when the server is under XSS attack, the system has no ability to detect the attack and quickly inform the developer to allow developer take into action.

With this in mind, if a tool can be created to detect the XSS attack, the system will be able to immediately acknowledge that someone is attempting to find vulnerability of the system, then it can inform the developer to be aware of it. In this way, the developer will be able to check if the attack attempt is just aiming for already fixed vulnerabilities, or it has found a new vulnerable code that missed during self testing.

In order to prove the idea is feasible, following question will be answered:

Through adding attack detection tools on server side, how much improvement on the efficiency can a system get when it is attacked?

By answering the question, the effectiveness of the idea of building a detection tool for XSS testing can be proved. At the same time, the impact it has to the performance of the system can be measured.

1.3 Hypothesis

The hypothesis is, by using pattern matching technique on server side, it will be very effective on XSS attack detection, with a reasonable false positive rate to be considered.

Chapter 2

Background and Literature Review

2.1 XSS

XSS is a widely used attacking method in cybersecurity, it can basically be injected into anywhere that has a user input, while also have the ability to do anything.

2.1.1 Definition

Cross-site scripting (XSS) attack is a computer vulnerability that allow attacker to insert their own script to the web page to modify the web page in a way they want. Attacker inserts the script in web applications using HTML, combine with other languages like JavaScript, PHP, etc. As it basically can be inserted everywhere has a user input, with the ability to do almost anything, XSS is one of the widest spread web vulnerabilities in the world.

2.1.2 How XSS happen and possible consequence

As XSS can be inserted into different programming languages with the help of HTML tag, there are lots of ways to perform a XSS attacks. One commonly used method is to hide the code in a session during user using browser. If the browser doesn't recognize the malicious script and consider the session is from a trusted source, the script will be in the system and provide benefit for attacker to gain the access of user's system. According to OWASP, XSS attacks always occur when data enters a Web application through an untrusted source, most frequently a web request, or when data is included in dynamic content that is sent to a web user without being validated for malicious content (The Open Web Application Security Project (2020))

The result of XSS attack can be varied, as the outcome of been attack is highly depended on what the malicious code could do. So consequence of under a XSS attack has a high range of possibility.

Lots of websites have been found out having XSS vulnerabilities and that include well known websites like Facebook, Twitter, YouTube and eBay. In 2010, famous open-souse foundation Apache was hacked by a hacker using XSS attack and all user password stored on that server was stolen (Threatpost (2010)). One more recent case is in January 2019, a game made by Epic games, Fortnite was found vulnerable to multiple attacks including XSS attack. By clicking the URL send by attacker, the gamer account can be taken over control and personal information are exposed to the attacker (Check Point Research (2019)).

2.1.3 Type of XSS attack

There are three types of XSS attack, stored XSS attack, reflected XSS attack, and DOM-based XSS attack.

2.1.3.1 Stored XSS attack

Persistent XSS attack (also called type-I XSS or stored XSS) is the XSS attack that aimed to store the script on vulnerable server through invalid user input. By sending unsafe parameter using forms and other kind of parameter, if the parameter been stored on the server (in database, log, etc.) successfully, when that data is been requested from other user, the scripts will be sent to the client and do harmful action on client side.

2.1.3.2 Reflected XSS attack

Also called type-II XSS or non-persistent XSS. This kind of XSS hide the script into the information in the data that can be reflected on server's respond, rather than stored XSS, this attack won't need script been stored on server side. Reflected XSS happened where user input can be shown on the web page, like a search bar, which always show your keyword in the response. Attackers to create a link that make use of the vulnerabilities, hide it in a fake website, or any other method that can trick user to click the link, when user gets the response, attacker's scripts will be run and do what attacker plans to do. This attack make victim machine believe that the script in the response are all safe, as the victim send the request in the first place.

2.1.3.3 DOM-based XSS attack

DOM-based XSS attack also known as type-0 XSS, it is related to document object model (DOM), which can making modification on the client side, without sending request to server. This is a complete different XSS attack compare to other two, because the script hacker uses not affecting HTML source code from response, so the script only render on client side and has nothing related to server.

As DOM based XSS request is fully client side attack, which means nothing has been sent to server, this XSS attack couldn't be detected on server side, so this project will focus on detecting reflected and stored XSS attack.

2.2 Testing platform

This project has some unsafe action (e.g. use tools to detect vulnerability on a web application, set up a web application on the system aim to take XSS attack) need to be done to the testing system, so virtual machines are need. This project uses Vagrant combine with VirtualBox to set up the environment.

2.2.1 Vagrant

Vagrant is a “Tool for building and managing virtual machine environments in a single workflow” (HashiCorp (2020)). By providing a config file for set up the virtual machine, vagrant can shorten the virtual machine build up time every time a new machine is set up, other useful cases is you can let other people set up a same machine with same configuration just by sharing your vagrant config file. This advantage also provide an easy way to set up the exact same machine every time the machine crash by some reason or the machine need to be in the same state every time it been tested.

In this project, vagrant is used to great all virtual machine. There are several reasons of using vagrant. First, using vagrant means the vagrant config file can be seared easily, so that the machine used for testing can be easily set up for someone that like to perform further testing about this project. Also, by using vagrant, target machine can be in exact same state at the start of the test every time. Another reason that vagrant been use is, after the XSS detection tool been installed on a target machine, an original version target machine can be build using vagrant file, and a comparison can be set up more easily.

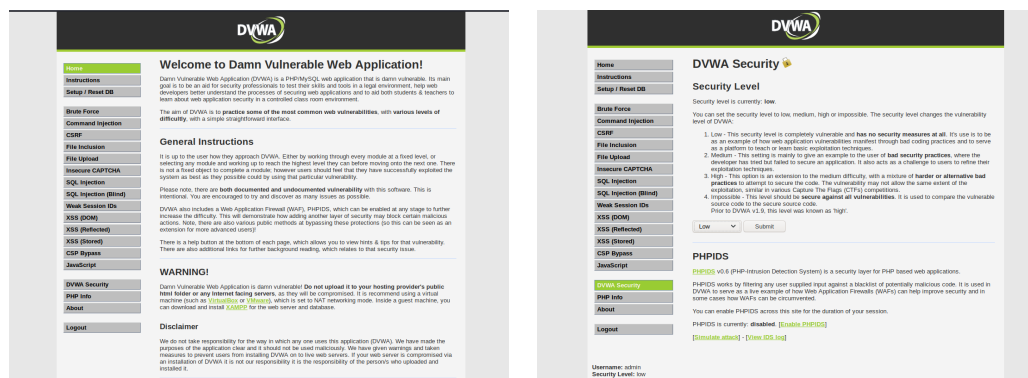
2.3 Vulnerable web application on the target machine

In this project, a target machine is need to provide a vulnerable web application. With the vulnerabilities on this machine, we can use tools to find the vulnerability, understand the attacking process, then build up a tool to detect the attack. The requirement of this web application should be able to run on local network. Also, it should be easy to modified so that we can quickly get which part of the code we need to change when building up the detection tool. There are several open source web applications can be used, in this project, we choose a project called Damn Vulnerable Web Application (DVWA).

2.3.1 About Damn Vulnerable Web Application (DVWA)

According to the GitHub page of DVWA (DVWA team (2020)), “Damn Vulnerable Web Application (DVWA) is a PHP/MySQL web application that is vulnerable. Its main goal is to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and to aid both students & teachers to learn about web application security in a controlled class room environment.”

Overall, DVWA provide a test bench, which has all kinds of vulnerabilities for user to test out their cybersecurity skills legally. It includes wide variety of vulnerabilities, including brute force, command injection, file inclusion, SQL injection, XSS and a lot more. Each vulnerability has one clearly indicated web page to test it out, with some other vulnerable points that are not indicated. Each indicated vulnerability has four security level: low, medium, high and impossible. Low is the easiest level to make use of the vulnerability, while impossible is the level that is nearly not possible.



(a) DVWA home screen

(b) DVWA security level setting

FIGURE 2.1: DVWA application page

2.4 Tools used to attack the target

To test our tools and analysis the effectiveness of it, several common used attack script-s/programs are used during the test. These tools have different cheat sheet for XSS attack, with other different included.

2.4.1 Tools been used

In total, there are 3 XSS vulnerability detection/attacking tool been used in the project. All tools are open source software, some of them are for XSS attack only and others can also detect other kinds of vulnerabilities (this project only use there XSS related functions). Following are the tools been used.

2.4.1.1 Cross Site “Scripter” (XSSer)

Cross Site “Scripter” (XSSer) (epsilon (2020)) is a python 3 based programs that can easily used to find XSS vulnerabilities on web application. Its developer describes it as “an automatic -framework- to detect, exploit and report XSS vulnerabilities in web-based applications”. It has around 1300 attack vectors preloaded in the program and support different browser. It has multiple settings to fit different testing situations, including specify different injections techniques (e.g. cookie, document object model) and options to try bypassing some common anti-XSS filter (e.g. String.fromCharCode() and Unescape() functions). It provides both command line and GUI interface to fit different users need. This project use XSSer v1.8 in the test.

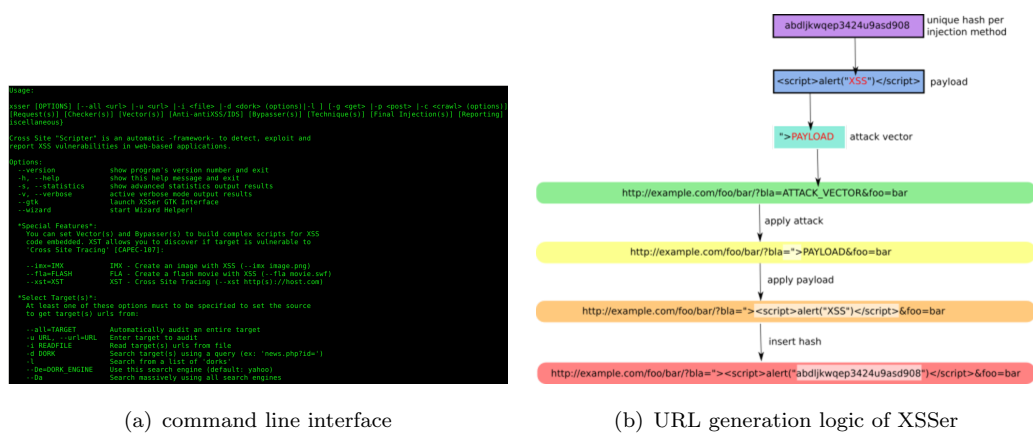


FIGURE 2.2: XSSer

2.4.1.2 Zad Attack Proxy

OWASP® Zad Attack Proxy (ZAP) is a well known web application scanner (ZAP Dev Team (2020)). It is based on Java and currently maintained by an international volunteers team. It provides scripts for multiple vulnerabilities, including XSS, cross site request forgery (CSRF) and much more. ZAP can run on both command line and GUI, and OWASP has provided a detailed documentation for users. This project use ZAP v2.9.0 during the test.

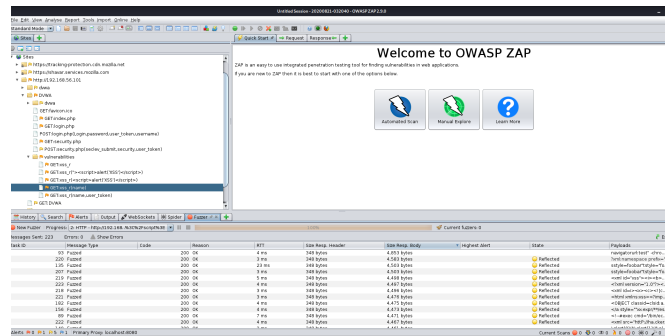


FIGURE 2.3: ZAP user interface

2.4.1.3 Burp suite Community edition

Burp suite is a Java base web application vulnerability tool. It supports GUI and has huge amount of functions including scanner, inductor, decoder, etc. The developer of burp suite aiming to “giving users a competitive advantage through superior research” (PortSwigger Ltd. (2020)). Burp suite has three editions: enterprise, professional and community. Community edition is totally free but has lots of limitations, as the community edition is enough for this project, we use community edition v2020.4 in this project.

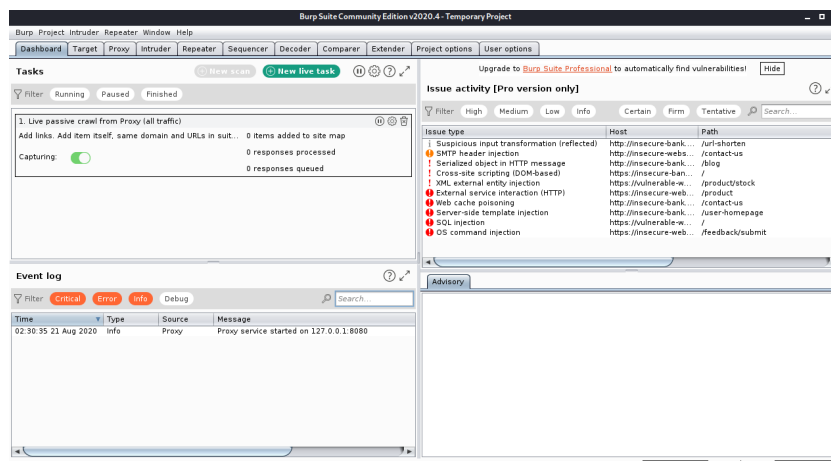


FIGURE 2.4: Burp suite GUI welcome page

2.4.2 Difference between these tools

Following is a table that shows some of the most significant difference in specs between the attack tools been used in this project (Table 2.1).

TABLE 2.1: Difference between 4 tools

Tools/specs	XSSer	ZAP	Burp suite Community
Programming language	Python 3	Java	Java
Interface	command line	command line & GUI	command line & GUI
Platform	Linux	Linux, Mac, windows	Linux, Mac, windows
vulnerabilities can find	XSS	multiple vulnerabilities	multiple vulnerabilities
Version been used in the project	1.8	2.9.0	2020.4
storage of attack scrips / cheat sheets	low	high	high
browser proxy setting	not needed	needed	needed

2.5 Related works

Currently, most of the research are aiming for finding different approaches to protect the system from XSS attack, which is relevant but not the same compare to what this project does. This project develops a tool for only detecting if there is a possible XSS attack in the system and inform the developer of the system on time, while most of the research focus on finding the vulnerabilities and fix it before attack happen.

There are also approaches that set up a firewall and use the firewall to filter the possible XSS scrips, this is a better approach compare to our tools in terms of defending XSS attack, but it usually takes more resource on the system. Shahriar and Zulkernine (2011) have provided a way to perform server side XSS protection based on the concept of “boundary injection” and “policy generation”, witch mainly focus on JSP programs.

Compare to other server side method, their method has a lower false positive rate compare to current method, although the delay on response could be a further work to be improved. This is one of the best filter approach, but it still required lots of resource.

Gupta et al. (2017) provide a system only for detection of XSS attacks just as this project do. They stated that “This work focuses on the detection of XSS attack using intrusion detection system. Here attack signature is utilized to detect XSS attack. To test the usefulness and effectiveness of proposed work a proof of concept prototype has been implemented using SNORT IDS.”

Compare to the program that aiming to fix the vulnerabilities, tool in this project focus on quickly detected the attack and inform the developers to provide a better chance for them to react to the attack on time. If use this tool before the attack protection program/scripts, it can also give developer a better idea on when there is an attacker attacking them rather than just simply block every potential attack outside the system.

Chapter 3

Requirements Analysis and Project Plan

3.1 Functional requirements

Functional requirement is a common way to list the specification or behavior of the software. Table 3.1 are the functional requirements for the tool building in this project:

TABLE 3.1: Functional requirements

Requirement	Detailed description
Command line interface	The Tool will interact with user through command line interface
Listen to port	The tool is able to listen on the port provide by user
Requests checking	The tool is able to check every request go through the port and see if it contains possible XSS attack script
Timing for request frequently	The tool is able to take time on how often one IP providing suspicious requests, once it is considered to be too frequent, the IP will be noted
Create request data	The tool will create a profile for every suspicious data
Store request data	The tool should be able to add every profile to a database
Review requests data	The user will be able to review the profile in the database
Notification	Once an IP has sent suspicious request too frequently, the tool will send user a notification

3.1.1 Importance level analysis

We set up some importance level to provide a better understanding on the complexity and time it takes to finish each functional requirement. Table 3.2 shows the example of how importance level are set up.

TABLE 3.2: Importance level example

Complexity/Time	Low	Medium	High
Short	0.0625	0.125	0.25
Medium	0.125	0.25	0.5
Long	0.25	0.5	1

For all the functional requirements, there importance level are shown in Table 3.3 below.

TABLE 3.3: Importance level of functional requirements

Requirement	Complexity	Time	Importance level
Command line interface	low	medium	0.125
Listen to port	medium	medium	0.25
Requests data checking	high	long	1
Timing for request frequently	high	long	1
Create request data	medium	medium	0.25
Store request data	high	medium	0.5
Review requests data	low	short	0.0625
Notification	low	short	0.0625

3.2 Non-functional requirements

Following are the non-functional requirements for the tool building in this project (Table 3.4), these are the requirements that is not a function or behavior, but still needed for this tool:

TABLE 3.4: Non-functional requirements

Requirement	Detailed description
Availability	The tool should run on any Linux machine
Usability	The tool will have a basic instruction of how to use it in command line to make it easy to use
Open-source	The tool will be fully open source
Modifiability	Every one download the program can modified it and make it fits their use case
Stability	The tool is stable enough to not have any bugs that make the program exit unexpectedly

3.3 MoSCoW analysis

Figure 3.1 are the scope of this project, MoSCoW method is used to separate all object to groups. It separates functional requirements in to four groups: must have, should have, could have and won't have. These four group help determine the importance of requirements and determine which are essential and which are optional.

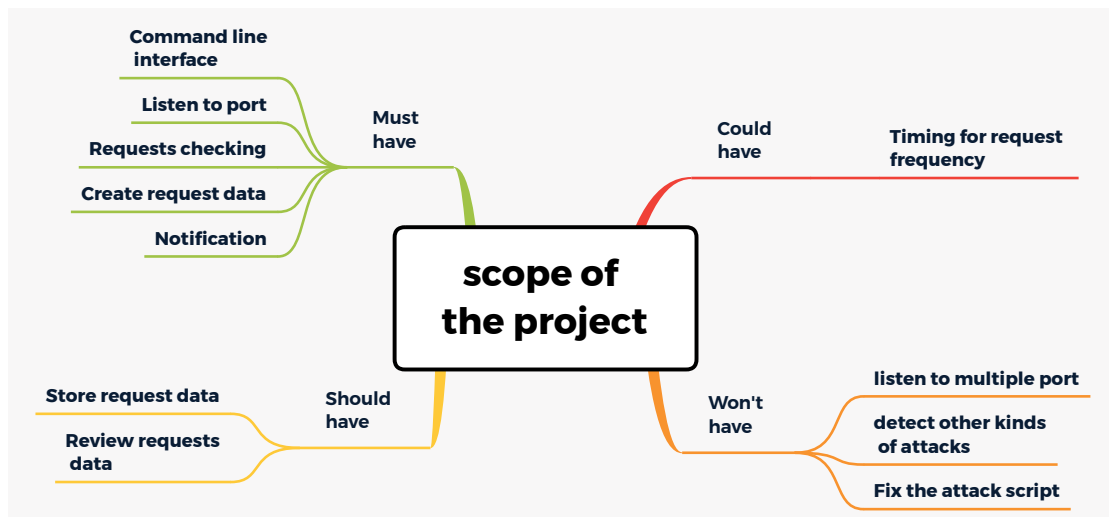


FIGURE 3.1: MoSCoW analysis

3.4 Risk analysis

A risk level similar to importance level is set up depend on the influence of consequence and possibility of every risk. Table 3.5 show the detailed information of how risk level is set up.

TABLE 3.5: Risk level example

Influence / Possibility	None	Minor	Medium	Major	Devastating
Impossible	0	0	0	0	0
Low	0	0.0625	0.125	0.1875	0.25
Medium	0	0.125	0.25	0.375	0.5
High	0	0.1875	0.375	0.5625	0.75
Certain	0	0.25	0.5	0.75	1

Table 3.6 followed show the risk level of all possible accident when using the detection tool, with the solution of every risk.

TABLE 3.6: Importance level of functional requirements

Description	Influence	Possibility	Risk level	Solution
Program crash	devastating	high	0.75	Have multiple test run before the test
Network loss	major	impossible	0	Not, possible, as the virtual machine communicate through intranet in a computer
Data package loss	major	medium	0.375	Usually cased by request coming too fast, slow done the frequency of request will work (around 5 request per second)
Data structure error	major	low	0.1875	Cause by not correctly structured the HTTP request, try as many request style as possible before the test will prevent this happened
Function error	devastating	high	0.75	Run several test run before real testing, try to find as much error as possible
Database operation error	devastating	medium	0.5	Structured the dictionary been stored into the database, limit the character and length of every parameter before send it to database
File I/O error	minor	low	0.0625	Close every file after editing before opening a new file, and check for the existence of the file before editing it

3.5 Project plan

With the functions of detection tool been decided, a grant chart is created for planing the process during preparation, Figure 3.2 is the grant chart for project planing.

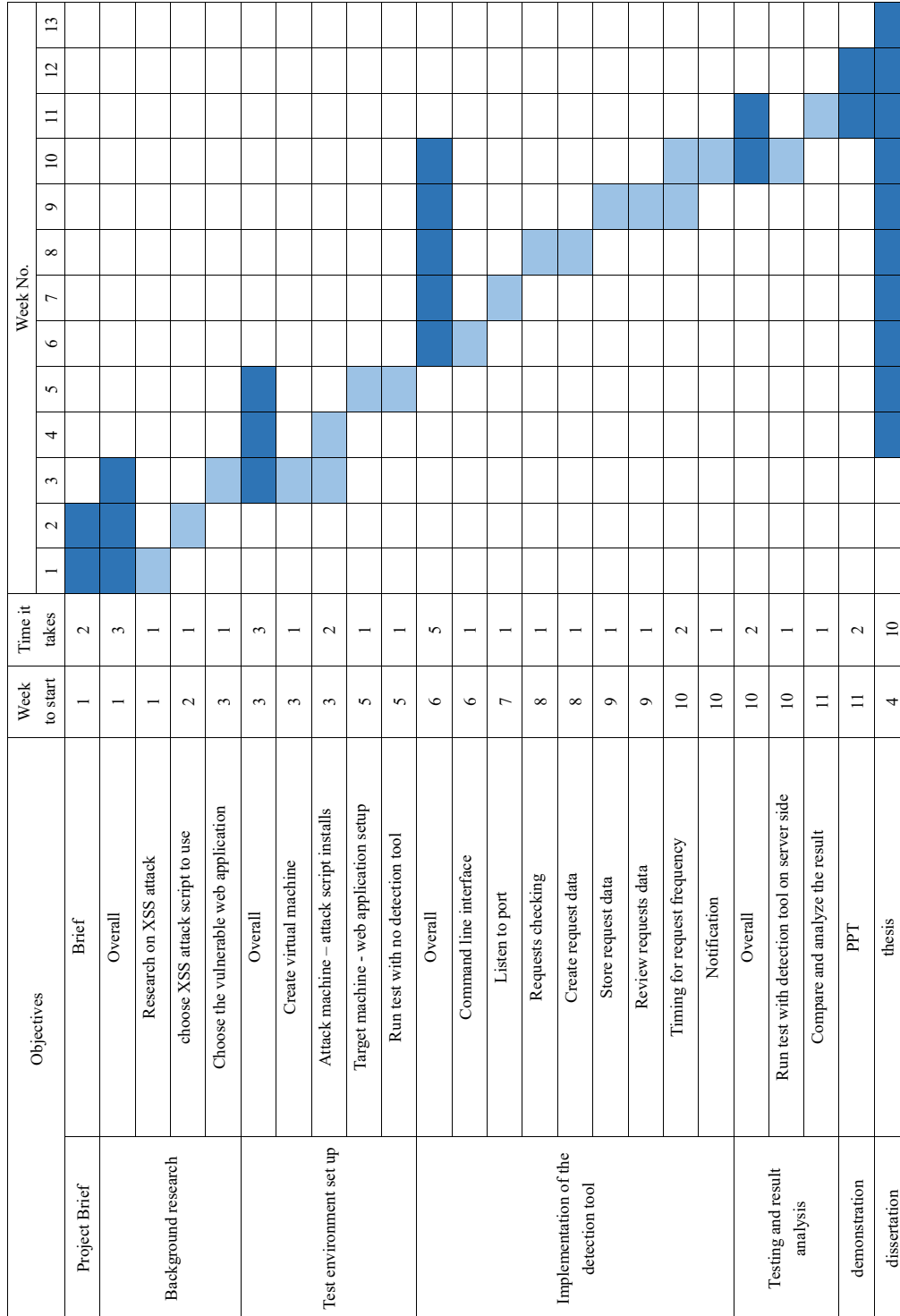


FIGURE 3.2: Project Plan

Chapter 4

Design and Implementation

4.1 Program design

The tools focus on listening to HTTP request and check if the user input in the request has a potential XSS attack script in it, if an IP has sent too many suspicious requests, the tool will inform the developer of the server.

To better explain the logic of the tools, Figure 4.1 below show the detailed operating logic of the tool and how user can access database of it.

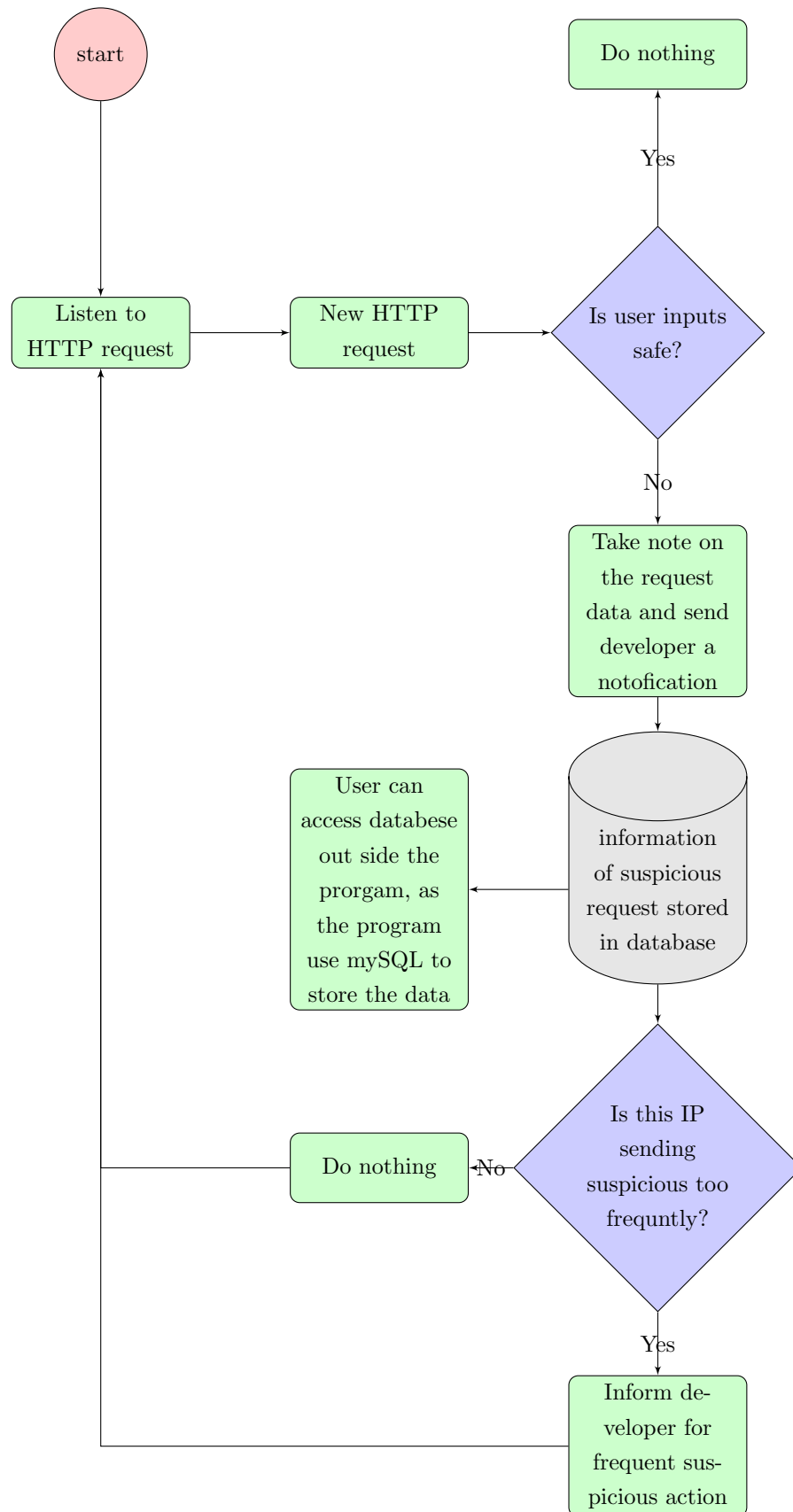


FIGURE 4.1: Program logic

4.2 Test environment

Following (Table 4.1) are the detailed information of test environment been used, All vagrant machines are running on vagrant 2.2.9 using VM VirtualBox 6.1.12 as virtual machine provider.

TABLE 4.1: Test environment

Platform	Name	Version
Windows	Vagrant	v2.2.9
Windows	VirtualBox	v6.1.12
Linux	kalilinux/rolling from vagrant cloud	v2020.2.1
Kali (attack)	Xsfer	v1.8
Kali (attack)	ZAP	v2.9.0
Kali (attack)	Burp suite community edition	v2020.4
Kali (attack)	Firefox ESR	v68.8.0esr
Kali (target & target2)	Apache	v2.4.46
Kali (target & target2)	MariaDB	v10.3.23
Kali (target & target2)	DVWA	v1.10
Kali (target2)	Python	v3.8.3rc1
Kali (target2)	pypcap	v1.2.1
Kali (target2)	dpkt	v1.9.2
Kali (target2)	mysql.connector	v8.0

Except applications/tools in Table 4.1, there are multiple Python and Linux packages been used to successfully set up the environment. As vagrant been used in this project, nearly all the setup progress has been written in to vagrant configuration file, including install necessary application, set up Apache and MySQL through MariaDB, etc. However, there are some settings need manually, including:

1. For target machine, every time it start, apache2 and MySQL need to be start manually, as vagrant configuration script only loaded when building up the machine, not every time the machine started.

2. VirtualBox additional function package is provided in the ./sync folder in Vagrant data, user can choose to install it as it is not necessary but can improve the experience when running the machine.
3. The tool developed in this project need several python package to run, but it is not installed in the machine, including pycap, dpkt and mysql.connector. So installation of them is need to run the tool.

4.3 Implementation

For building up the tool, there are several main functions need to be done, listed as follows:

1. Listen to the HTTP request and extract the data
2. Checking if the request is possibly unsafe
3. Modified the data structure to storing it

These three functions will be discussed in this section. If you'd like to check out the full coding of the tool, you can check out the Git project online (Yang Ding (2020)).

4.3.1 Request listening and extracting

To be able to check the data in the HTTP request, we first need to use tools to capture and extract it from package coming from the internet.

4.3.1.1 Listen to HTTP request

For HTTP request, to let python program be able to listen to the information, pycap is used. Pycap is a “simplified object-oriented Python wrapper for libpcap” (pynetwork (2018)), it provides the function to listen to a given port through a given network interface. As the tool we are building will only be a detection tools, which not include the function to fix/sanitize the input, the tool will only act as a sniffer.

DVWA is running on apache2 platform using HTTP, and the virtual network using in the test using Eth1, so pycap is set to Eth1 and listen to port 80.

4.3.1.2 Extracting the data

Pypcap not providing the function to unpack the network data, so we use dpkt to help extract the data we need. Dpkt is a python package that provide function for package creation and parsing to help analysis the package data. It also includes the definition for TCP/IP protocols.

We know that in TCP/IP protocol architecture model, There are several layers: application, transport, internet, data link and physical network. HTTP is application layer protocol, and for pypcap, the network package been captured has all information from application to internet layered, so what we need to do is to clean up the internet and transport layer data, approach application layer data first. By using dpkt, we can easily use data() function to get to the next layer of the package. Following code show an example of how to get HTTP request using pypcap and dpkt.

```
1 import pcap
2 import dpkt
3 #set up the network port for pypcap
4 pc = pcap.pcap('eth1', promisc=True, immediate=True)
5 pc.setfilter('tcp port 80')
6 #for every package come in
7 for ptime,pdata in pc:
8     #use dpkt to get the data
9     p=dpkt.ethernet.Ethernet(pdata)
10    #we only care about package that has IP protocol on internet layer
11    if p.data.__class__.__name__=='IP':
12        #we only care about package that has TCP protocol on transport layer
13        if p.data.data.__class__.__name__=='TCP':
14            #if the port number is 80
15            if p.data.data.dport==80:
16                #print the HTTP request data
17                print(dpkt.http.Request(p.data.data.data))
18
```

In this way, we can easily capture every HTTP request and extract the request data from the package.

4.3.2 Checking the request data

For this section, what part of the request data are needed for our tools and how our tool detect the unsafe parameter are discussed.

4.3.2.1 Data need to check in HTTP request

After getting the request package, we need to extract the information we need, then check if the user input is safe.

HTTP request data has make up by four parts:

1. Start line: This start line contain three elements: HTTP method, request target and HTTP version. HTTP method specified the method of the request, including GET, POST, DELETE, etc. Request target provide the target URL of the request. HTTP version specified the version of HTTP protocol. In our case, all the request are GET and POST request.
2. Request headers: The header of the request contain most of the information that are used in request. It has a basic structure: a case-insensitive string followed by a colon (':') and a value whose structure depends upon the header. The whole header, including the value, consist of one single line, which can be quite long. (MDN web docs (2020))
3. An empty line
4. request body: Not all request has a body, for GET and POST method, only POST has a request body, it usually includes several user input.

Following are two example of HTTP request, first one is a GET request and second one is POST request, these are the only two request that are related to this project.

```

1
2 -----This is a GET request,you can see userinput in fuirst line, URL part-----
3 GET /dvwa/vulnerabilities/xss_r/?name=%3Ca HTTP/1.1
4 host: 172.28.128.24
5 user-agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
6 accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
7 accept-language: en-US,en;q=0.5
8 accept-encoding: gzip, deflate
9 referer: http://172.28.128.24/dvwa/vulnerabilities/xss_r/?name=%3Cs
10 connection: keep-alive
11 cookie: security=low; PHPSESSID=01pru32dtqllrl6cfsmojpq51e
12 upgrade-insecure-requests: 1
13
14
15 -----This is a POST request,you can see userinput in HTML body part-----
16 POST /dvwa/vulnerabilities/xss_s/ HTTP/1.1
17 host: 172.28.128.24
18 user-agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
19 accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
20 accept-language: en-US,en;q=0.5
21 accept-encoding: gzip, deflate
22 referer: http://172.28.128.24/dvwa/vulnerabilities/xss_s/
23 content-type: application/x-www-form-urlencoded
24 content-length: 95
25 connection: keep-alive
26 cookie: security=low; PHPSESSID=e962h85ro7ggk5qolf6r8cbb4q
27 upgrade-insecure-requests: 1
28
29 txtName=input&mtxMessage=%3Cscript%3Ealert%28%27XSS%27%29%3C%2Fscript%3E
30
31

```

In the request, there are two parts that could contain a user input, which are URL in the start line and request body. As XSS attack happened through unsafe code injection from user input, these two parts are where XSS could happen.

With dpkt allowing us to extract the raw HTTP request, we can easily get the parameter in the request as there are only these two part that can have user input.

4.3.2.2 Detection method

As been mentioned in chapter 2, among three types of XSS, only stored and reflected XSS can be detected on server side. Both these two types of XSS attack try to inject the suspicious code to server, the difference are stored XSS try to inject the code into the server data and reflected XSS try to inject the code to the response corresponding to the request. For server, though they are aiming for different part of the server, they are the same in terms of scripts and patterns of the code. This project check the user input from HTTP request, so no matter what XSS attack it is, for the detection tools it makes no difference.

So for detection, the only task left is to find a XSS attack pattern and then compare it to every input. The way to detect a XSS attack attempt can be similar to XSS protection. Server side XSS protection has two main types, including communication between client and server, or using simple pattern matching technique to clean up possible XSS scripts. This project choose pattern matching technique similar to the technique been used in protection method to detect possible attack.

XSS attacks are attacks that focus on using code to change how web page work and then using the scripts to perform unsafe action. As our tool focus on detecting the XSS attack on web application, no matter what scripts are injected into the input, it has to be loaded by HTML code of the web page first, then by using attributes like “onerror” and “onload”, the scripts can be executed. So we can focus on detecting the HTML tag and attribute in user input and by detecting those, the tool can find possible XSS attack action.

There are over 100 tags, only some of them are valuable for XSS attack. During the research on internet and comparing the cheat sheet used in attacking tools, following 15 tags in the Table 4.2 are considered to be XSS-useable.

TABLE 4.2: Tags that are frequently used for XSS attack

a	audio	body
details	form	iframe
img	input	keygen
marquee	script	select
svg	textarea	video

You can see that there are lots of tags are word been used in normal communication, so we can not just search for these words in all input. HTML tags need a “<” to indicate itself been a tag, so a “<” can bee add in front of those words and these will significantly reduce the false positive rate during the test.

But just search for these fix strings are not enough, lots of scripts can split the tag in pieces to hide from detection. To detect HTML tags in a better way, regular expression is used. Regular expression has many advantages compare to just match a string in user input. Regular expression can express a not only a fix string, but also a pattern, and this provides much more variable when searching the tags, increase the possibility of finding the suspicious input. Also, Python provide repackage to full regular expression support, which make using regular expression easier.

By using regular expression, “(.)” are added between every character in the keyword. This make the pattern to match become not sensitive to whether there are other characters inject into the keyword. For example, for input “<scri\np\nt>alert('XSS')</script>”, by using “<(.)s(.)c(.)r(.)i(.)p(.)t>” as keyword, the program can still match the regular expression to the input. Following are some of the keywords after adding “(.)” pattern.

```

1 <(.) a
2 <(.) a(.) u(.) d(.) i(.) o
3 <(.) i(.) m(.) g
4 <(.) i(.) n(.) p(.) u(.) t
5 <(.) s(.) c(.) r(.) i(.) p(.) t
6 <(.) v(.) i(.) d(.) e(.) o
7
```

Except from these keywords, there is also possibility that attacker add a ending tag try to get out of original tag, so we can use those kayword, with a “/” at the front to find those. On the other hand, commont function can also be used for attack, so “’” and “’;” is added into the list. Finally, HTML attributes “onerror”, “onfocus”, “onload” are also common used key word. So they also been added into the list. All these keyword are modified using regular expression.

This list is provided in a .txt file in the program, so keywords can be modified to fix different need.

4.3.3 Extract information needed and store the suspicious request

Except from showing a notification on the screen, the program also store the data of malicious request into a database. For database been used, MySQL is choosen as it is one of the most used database and it support operating the database using Python by just import mysql package in the program.

for data been stored in the database, Table 4.3 shows the staucture and component of data been saved.

TABLE 4.3: Variables that been stored into database

Name	Data type	Discription
Time	varchar(25)	Time of request, in format of “yyyy-mm-dd hh-mm-ss”
IP	varchar(15)	Client IP address, only IPv4 supported
URL	varchar(2000)	URL of the request
GetParam	varchar(2000)	GET parameter of the request, in the fromat of name: script
PostParam	varchar(2000)	POST parameter of the request, in the format of txtName: input, mtXMessage: ”XSStest”
RawRequest	varchar(2000)	The raw request

In database, time, IP, URL, GET parameter and POST parameter are shown, these are some fo the most important information in a request, through database, developer can quickly find the information need.

Accourding to mysql office document from Oracle Corporation (2020), there is a length limitation for all variables stored in the table. For varchar though the theoretical maximum length of a varchar is 65,535 bytes, the effective length is depends on the maximum row size, which means all data in one row combine shoule not be larger than 65,535 bytes. As there is no length limitation on HTTP request,so although normally it should not be a problem, it is still possible for a http request to be too long to stored in mySQL database.

To fix this problem, a list of .txt files is also used to stored the data. The .txt file has a structure like below:

```
1 -----
2 Time:
3 2020-08-29 12:18:10
4 IP:
5 172.28.128.20
6 URL:
7 /dvwa/vulnerabilities/xss_r/?name=%3Ca
8 GetParam:
9 {'name': '<a'}
```

```
10 PostParam:
11 {}
12 RawRequest:
13 GET /dvwa/vulnerabilities/xss_r/?name=%3Ca HTTP/1.1
14 host: 172.28.128.24
15 user-agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
16 accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
17 accept-language: en-US,en;q=0.5
18 accept-encoding: gzip, deflate
19 referer: http://172.28.128.24/dvwa/vulnerabilities/xss_r/?name=%3Cs
20 connection: keep-alive
21 cookie: security=low; PHPSESSID=01pru32dtqllrl6cfsmojpq51e
22 upgrade-insecure-requests: 1
23
```

The .txt file has all data stored, including the variable stored in database. One .txt file will only stores suspicious requests of one IP in one minute. File name will be “yyyy-mm-dd hh-mm(%IP).txt” format. If one IP has send more than 5 malicious request in one minute, the program will send one more warning to developer as 5 malicious request in a minute shows the possible unsafe input is likely to be intended.

Chapter 5

Result and Discussion

5.1 Testing process

The test process basically include setting up the network and then try each attack tool against the target machine, each tool two times, first time without the detection tool on server side, and second time add the detection tool. In this case we can see the difference between having and not having detection tools and compare the result. In this section, how we use the attack tools will be shown. The detection tool is just an auto run program with no configuration so it does not need any guide.

To lower the stress of the detection tools, we set up a delay during every testing, if the delay is not set up, packages will came too fast and our tool won't have time to analysis every one.

5.1.1 Attack using XSSer

XSSer has one cheat sheet for both stored and reflect XSS, only different when using the post attack is adding “-p” option. Other information need to provide to XSSer is the cookie of one browser using “-cookie” option, “-u” for target, number of tries (-auto-set) and “-delay” for a fix delay between every attack.

So overall, the command been used to run XSSer are:

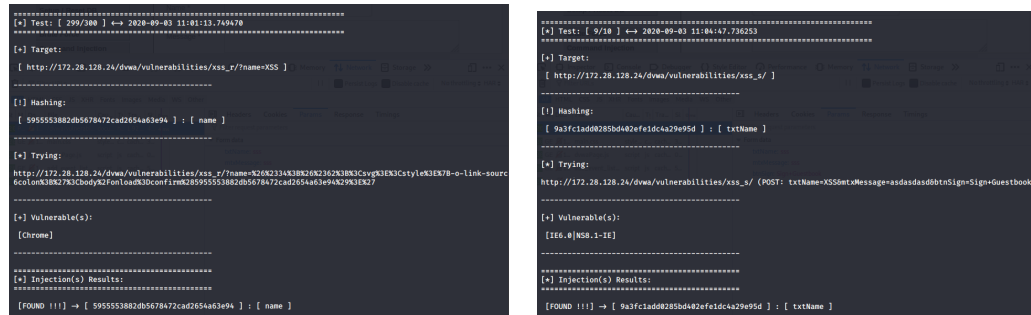
```
1 //This is for reflected XSS
2 sudo python3 xsser -u 'http://172.28.128.24/dvwa/vulnerabilities/xss_r/?name=XSS'
   --cookie='security=low;PHPSESSID=7j66eh0ami2me7sispl4un238h' --auto --
   reverse-check --delay=1 --auto-set=300
3
4 //for Stored XSS, just add -p option and add all paramater into it, with the
   parameter you want to test set to XSS, like the example below:
```

```

5 sudo python3 xsser -u 'http://172.28.128.24/dvwa/vulnerabilities/xss_s/' -p '
    txtName=XSS&mtxMessage=asdasdasd&btnSign=Sign+Guestbook' --cookie='security=
    low;PHPSESSID=7j66eh0ami2me7sispl4un238h' --auto --reverse-check --delay=1 --
6 auto-set=10

```

The result should be look like this:



(a) get parameter attack

(b) post parameter attack

FIGURE 5.1: XSSer attack result

5.1.2 Attack using ZAP

For ZAP proxy, A browser is need for testing, this test used preconfigured Firefox browser in ZAP to perform the attack, once have vulnerable web page loaded once, we can follow the step in Figure 5.2 to set up XSS payload and start fuzzing.

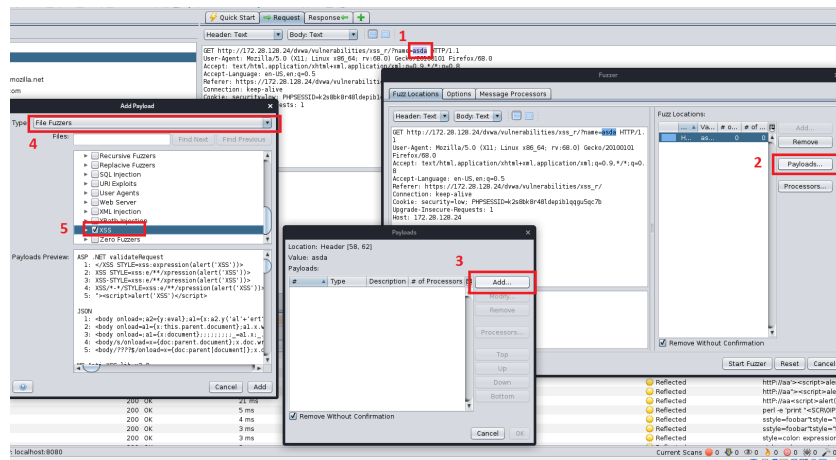


FIGURE 5.2: ZAP attack steps

This payload is also made for all kinds of XSS attack, so one payload can be used for both reflected and stored XSS attack. ZAP will provide result like the screenshot in Figure 5.3.

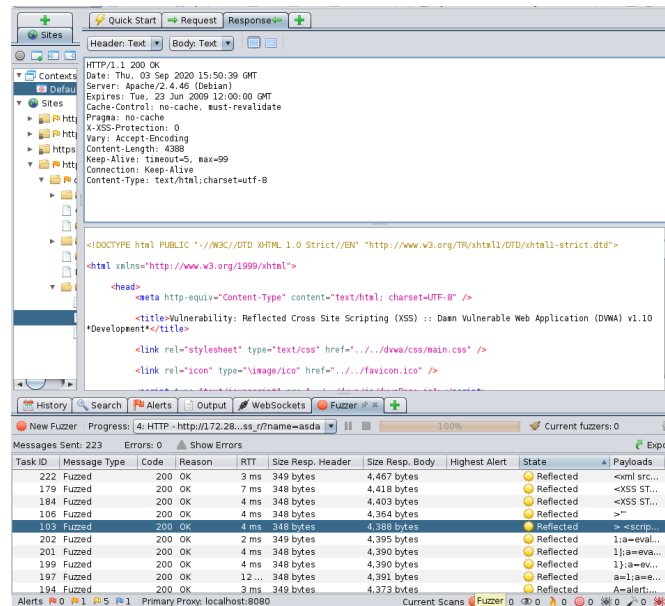


FIGURE 5.3: ZAP fuzzing result window

In the window, the section below show all the fuzzing result, and the two windows up on the right are the data of select request, including the request itself and the response data.

5.1.3 Attack using Burp Suite

For Burp Suite, the advantage of using it is we can easily load custom payload, which is exact what we done. We found a huge custom XSS payload that contains over 2000 payload and load it to Burp Suite to attack the server (macroman321 (2020)).

Firefox browser has set up the proxy to connect to Burp Suite, and we use Burp to catch the request on client side, using inductor to attack the server with attack mode set to sniper, with the payload we got. We try same payload for both reflected and stored XSS attack. Cause the payload is too large to load, we choose to randomly select 100 payloads several times when using it.

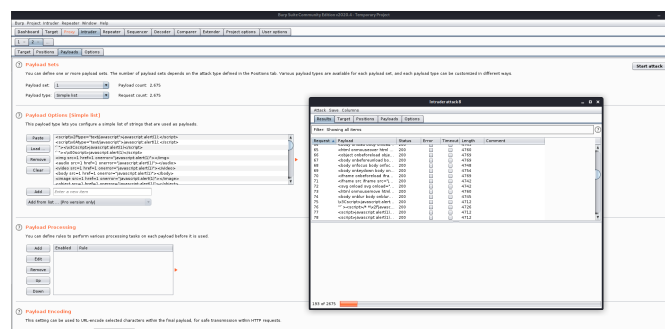


FIGURE 5.4: Burp Suite attack setting

5.2 Testing result

In conclusion, the result are different depend on the attack tool, but the difference between attacking a reflected XSS and a stored XSS is zero. Here the table shows the overall result of the testing (Burp suite community don't detect weather the attack is successful, so None). Every tool and payload been used has been tested on both reflected and stored XSS, but the result is exactly the same so it not shown in the table.

TABLE 5.1: overall result

Attack Tool	Payload Try	Payload attack tool considered to be success	Payloads detected by the tool
XSSer	1291	1287	1287
ZAP	223	112	126
Burp Suite	100 random, 3 groups	None	2 groups 100, one group 99

Here we also provide the full check list for regular expression matching below:

```

1 <(.* )a
2 <(.* )a(.* )u(.* )d(.* )i(.* )o
3 <(.* )b(.* )o(.* )d(.* )y
4 <(.* )d(.* )e(.* )t(.* )a(.* )i(.* )l(.* )s
5 <(.* )f(.* )o(.* )r(.* )m
6 <(.* )i(.* )f(.* )r(.* )a(.* )m(.* )e
7 <(.* )i(.* )m(.* )g
8 <(.* )i(.* )n(.* )p(.* )u(.* )t
9 <(.* )k(.* )e(.* )y(.* )g(.* )e(.* )n
10 <(.* )m(.* )a(.* )r(.* )q(.* )u(.* )e(.* )e
11 <(.* )s(.* )c(.* )r(.* )i(.* )p(.* )t
12 <(.* )s(.* )e(.* )l(.* )e(.* )c(.* )t
13 <(.* )s(.* )v(.* )g
14 <(.* )t(.* )e(.* )x(.* )t(.* )a(.* )r(.* )e(.* )a
15 <(.* )v(.* )i(.* )d(.* )e(.* )o
16 /(.* )a
17 /(.* )a(.* )u(.* )d(.* )i(.* )o
18 /(.* )b(.* )o(.* )d(.* )y
19 /(.* )d(.* )e(.* )t(.* )a(.* )i(.* )l(.* )s
20 /(.* )f(.* )o(.* )r(.* )m
21 /(.* )i(.* )f(.* )r(.* )a(.* )m(.* )e
22 /(.* )i(.* )m(.* )g
23 /(.* )i(.* )n(.* )p(.* )u(.* )t
24 /(.* )k(.* )e(.* )y(.* )g(.* )e(.* )n
25 /(.* )m(.* )a(.* )r(.* )q(.* )u(.* )e(.* )e
26 /(.* )s(.* )c(.* )r(.* )i(.* )p(.* )t
27 /(.* )s(.* )e(.* )l(.* )e(.* )c(.* )t
28 /(.* )s(.* )v(.* )g
29 /(.* )t(.* )e(.* )x(.* )t(.* )a(.* )r(.* )e(.* )a
30 /(.* )v(.* )i(.* )d(.* )e(.* )o

```

```

31 "(.*)>
32 "(.*)";
33 a(.*)l(.*)e(.*)r(.*)t(.*)\((
34 o(.*)n(.*)e(.*)r(.*)r(.*)o(.*)r
35 o(.*)n(.*)f(.*)o(.*)c(.*)u(.*)s
36 o(.*)n(.*)l(.*)o(.*)a(.*)d(.*)
37

```

5.3 Discussion

We will discuss the test result on two aspects: difference base by two type of attack, and difference base on using different tools.

5.3.1 Compare stored XSS and reflected XSS

As mentioned in chapter 2, the difference between these two XSS is just where there aim for. Stored XSS aiming for storing the script on the server and waits for another user load that page, while reflected XSS aiming for inject the script only to the response of current request. The script and keywords to use are always the same and for a server side detection tool that listen to the port, they all be considered as user input and what kind of XSS they are make no difference.

5.3.2 Compare result from three tools

As the pattern matching technique has been shown in implementation part, it's clear what kinds of input will be notified as a suspicious input, so we will not focus on the input been notified by the tool. Here we mainly focus on the input that not been recognized but still have XSS attack potential.

5.3.2.1 XSSer result

For XSSer, 1291 payloads has been used and 1287 is said by XSSer to be success, while detection tools detect exact 1287 vulnerable request. Notice that the 4 payloads said to be not a success attack is all different from 4 payloads considered to be safe by detection tools, So our detection tool s has 4/1287 false positive rate in this test. Following are parameters that detection tools consider to be not suspicious:

```

1 //{parameter name: value}
2 {'name': '&{4ee83fe147d0e96d666a91df83d0d5a2}';':
3 {'name': "&#34;&#62;<h1/onmouseover='%0061lert(178fb0683658f118fc61ee5fef11c443)
  '>%00"}:
4 {'name': 's%22%20%22+STYLE%3D%22background-image%3A+expression%28alert%28%27
  d434e9e018b702f0dddb3168143cdd00%3F%29%29'}:

```

```

5 {'name': '< style=x:expression\x028write(f09b133c23186f5fff023397fd642e1f)\x029
  >'}:
6
7

```

These input does not fit the regular expression at the time of performing the test. Parameter 1 use “{}” to try adding data to HTML, though its value is not working, this can be used to display variable on the web page, so it is one possible way of XSS. Second parameter is using HTML attribute not included in the list - onmouseover, this can also trigger an action like alert() or lead web page to load some script. These two input shows the limitation of using regular expression: if the attacker’s method is the way develop do not know, then the attack will be detected. On the other hand, the list of request expression can be updated, so it can be getting better and better.

5.3.2.2 ZAP result

For ZAP proxy, 223 payloads has been used and 112 is said by XSSer to be success, while detection tools detect exact 126 vulnerable request. In this result, the success attack number are not the same as the detected unsafe input. Detection tools has detected more possible attack than ZAP’s success attack number. After checking the result, we find out that all the success attack reported by ZAP has been detected, So for a detection tool, It has work as it should be detected all possible unsafe request. Other scripts that been considered to be useless on both ZAP and detection tool side are quite the same pattern as the undetected payload when using XSSer, so we won’t focus on them here.

Compare to result using XSSer, this time it has around 10% false positive rate, following are some input that detection tool considered to be unsafe but ZAP not consider it to be a success attack:

```

1 {'name': '%3Cbody%20onload=;a2=%7By:eval%7D;a1=%7Bx:a2.y('al'+ert')%7D;;;;;;_
  =a1.x;_(1);;;'};
2 {'name': '%3CDIV%20STYLE=%22background-image:%20url(&%231;javascript:alert('XSS')
  )%22%3E'};
3 {'name': '+alert(0)+'};
4 {'name': '"';//%0da=eval;b=alert;a(b(9));"}
5

```

If we try to improve it, how to improve the false positive rate will be a problem. The detection tool are not able to try render the web page to detect if the vulnerability exist, it can only make diction depend on the input structure, this is the limitation of detection tool. Another way is to build up a boundary policy in the first place to check whether by putting the input in, the boundary change, like Shahriar and Zulkernine (2011) have done, it is also a server side detection method but with lower false positive rate and a better detection method. But this concept of building up a boundary policy for every page on the website will introduce large amount of data, which required more storage

and computing power to achieve. So for now there is no good way to reduce the false positive rate without massively increase the complexity of the detection method. This false positive rate problem is one of the major disadvantage for using regular expression.

5.3.2.3 Burp Suite result

For the open-source payload list used in Burp Suite, the result is not so bad, except one script that having same structure as the one been undetected in XSSer result, all other payloads are detected by detection tool.

The reason of it is that in this payload list, nearly every payload use 'alert()' in it. Some example of the payload has been listed below:

```
1 {'name': 'xyz onerror=alert(6)'};  
2 {'name': 'style=color: expression(alert(0))', 'a': ''};  
3 {'name': '<DIV STYLE="background-image: url(javascript:alert(\'XSS\'))">'};  
4
```

As “alert()” attribute is a commonly used technique in open source XSS payload and it has been added in to the checklist, this kind of open source payload are not effective against the detection. So for this part of the test, we can see that through using regular expression has limitation on not going to detect the attack not been thought of before, and it has quite high false positive rate, but when dealing with the community based payload, which always have lots of patterns in the script, it is still very effective.

Chapter 6

Conclusion and Further Work

6.1 Conclusion

In conclusion, The detection tool has showed its ability and limitation. This server side XSS detection tool using regular expression matching technique has proved that it can perform quite well when developer has put all possible attack pattern into the regular expression keyword list. On the other hand, the need of a keyword list is also the limitation of this method, as if the attack technique is not listed in the list, the detection tool can do nothing. So in one sentence, the effectiveness of this tool is highly depend on the variety of patterns stored regular expression list.

Except from that, this tool still has advantages. As a detection tool, regular expression work pretty well as it can match the string with a pattern rather than a fix keyword. Using regular expression usually takes less computing power compare to other detection/protraction tools as they sometimes use dynamic analysis or set up complex regulation, which give more stress to the server processor.

But, with the advantage of been simple compare to other methods, there comes the draw back. Using regular expression can produce high false positive rate due to it just check the patterns in string, not consider the meaning and can not render the web page in advance to check the real effect user input has on the web page. For detection, the over 10% false positive rate is acceptable as the tool just log the data, not modifying it. But it is still a disadvantage compare to other method.

Over all, using regular expression to detect XSS attack is a very simple but efficient method, but its limitations make it hard to become better compare to much more complex attack detection system.

6.2 Further work

Through the analysis of testing result, our detection tool prov to be effective to most of the stored and reflected XSS attack, with over 95% of the attack payload been found using different tools. However, there are some further work could be done, list as follows:

1. The tool use MySQL as the database, but MySQL doesn't fit our need. MySQL has a length limitation for all variable stored in the table (Oracle Corporation (2020)), a XSS payload can get longer than that. To fix this, we also store full request in a .txt file, but we can also change the database to allow longer string input, one possible solution is MongoDB.
2. Only detection is not enough for protect the application from XSS attack, if we want to protect the server, a filter can be added to clean up the suspicious input or encode it into a safe format (like HTMLencode() function). This tool only listen to the port, it can not manipulate the request data, but if we can modify the data, then a protection method can be added.
3. As this tool use pattern machine technique, it can be used in all kinds of attack that related to user input, like SQL injection and code execution. This can be added as future function of the tool.

6.3 Legal aspect

There is one legal aspect about this project should be discussed. The main point is that our tool been build in this tool is a tool that is designed to be used for protect a server, not attacking one.

To make sure the tool won't be used for attacking a server, this tool use pycap package, which is Python package for internet package listening, it can only be used to log the package go through, not modifying it. this make sure that the tool can be used for changing the data in Package.

What's more, all the method been used in the program are not able to modify the data except the data in the file been created by the tool and the MySQL table created by this tool. So the tools won't change the data except data not related to the file.

In conclusion, the tool only has the ability to listen to the package, not modifying it, and the tool can only make changes to the data in that is related to this program, so it could not be used for attack a server as it is not designed to be able to do it.

Appendix A

Appendix

A.1 Document in COMP6211 - Project Preparation

Following are General Research Review in COMP6211 project preparation, which is related to this project:

Cross-site Scripting attacks protection methods

Yang Ding

Abstract—Cross-site scripting (XSS) attack is one of the most common used method to attack a digital system. It can hide itself in HTML, JavaScript and many other programming languages, with the ability to do anything from gaining information of the victim, to getting full control to the system been attacked. In order to deal with this problem, many developer have figure out different way to protect a system from XSS attack. This paper listed several different approach to counter XSS attack and discuss their strengths and weaknesses.

Index Terms—Cross-site scripting attacks (XSS), cyber security.

I. INTRODUCTION

Cross-site scripting (XSS) attack is a computer vulnerability that allow attacker to insert their own script to the web page to modified the web page in a way they want. Attacker could let browser sending sensitive information of the user, monitoring user action without been noticed, or even gaining access to control user's system. Usually, attacker hide its code in web applications using HTML, JavaScript, Flash and many other programming languages. According to Open Web Application Security Project (OWASP), XSS attack is at the seven place of OWASP top 10 web application security risk in 2017 [1].

II. BACKGROUND

According to the data from CVE security vulnerability database [2], from 1999 to 2019, 12.5% of the recorded vulnerabilities are XSS attacks. In 2019, 1593 vulnerabilities are based on XSS (only considering the vulnerabilities that have been identified), make it the second common vulnerabilities of the year, detailed data shown in the figure 1 below.

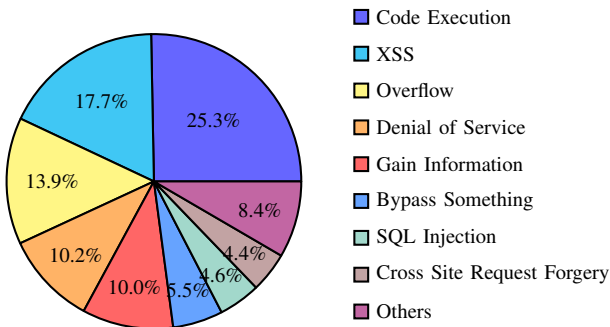


Fig. 1. percentage of different vulnerabilities been found in 2019

III. XSS ATTACK AND POSSIBLE CONSEQUENCE

As XSS can be insert into different programming languages, there are lots of ways to perform a XSS attacks. One commonly used method is to hide the code in a session during user using browser. If the browser doesn't recognized the malicious script and consider the session is from a trusted source, the script will be in the system and provide benefit for attacker to gain the access of user's system. According to OWASP, XSS attacks always occur when data enters a Web application through an untrusted source, most frequently a web request, or when data is included in dynamic content that is sent to a web user without being validated for malicious content [1].

The result of XSS attack can be vary, cause the consequence of been attack is highly depended on what the malicious code could do. For example, if the code is used for hijacking the transmission between client and server, as the result the attacker will be able to get the data been transferred and possibly even change it. If the code is for starting a connection between victim and attacker, then it could end up as attacker have full access of victim's system. In conclusion, the consequence of under a XSS attack has a high range of possibility.

Lots of websites have been found out having XSS vulnerabilities and that include well known websites like Facebook, Twitter, Youtube and ebay. In 2010, famous open-souse foundation Apache was hacked by a hacker using XSS attack and all user password stored on that server was stolen [3]. One more recent case is in January 2019, a game made by Epic games, Fortnite was found vulnerable to multiple attacks including XSS attack. By clicking the URL send by attacker, the gamer account can be took over control and personal information are exposed to the attacker [4].

IV. TYPE OF XSS ATTACK

There are three types of XSS attack, persistent XSS attack, non-persistent XSS attack, and DOM-based XSS attack.

A. persistent XSS attack

Persistent XSS attack (also called type-I XSS or Stored XSS), is the XSS attack that scripts has been inject into vulnerable server through invalid input. The script is then stored in the database, forms, user logs or other data that will be requested by client from server. When client communicate with the server, it will receive data that contain the malicious scripts and then the script can perform harmful action to the client device.

B. non-persistent XSS attack

Also called type-II XSS or reflected XSS. This kind of XSS hide the script into the information in the data that can be reflected from server's error, form or any part that can show a response that include some of information from the request. For example the search bar always show the information you are searching in the result page, this kind of operation is where non-persistent XSS could happen. When victim got a email or message from a website that ask victim to do somethings, like click a link, visit a website, or submit a form, the links/website/form can be modified by attacker and that can send the script with user request, then the vulnerable website will reflect the information been sent back to victim, as the data received from server is the responds from victim's own request, the data will be consider to be safe and the script hide in the data will be executed, which will do what attacker want to the victim.

C. DOM-based XSS attack

DOM-based XSS attack also known as type-0 XSS, it happens as a result of allowing the modification of document object model (DOM) when browser rendering the web page. Consider the hacker inject the script into the parameter in a URL, which is been used only when browser rending the page, then when victim click the URL, the request and response HTML will be no different from normal web page, but when browser rending the page based on HTML's client side script, the DOM can be modified by the parameter and the injection of hacker's script can change how the web page operate and probably run some code for hacker. This is a complete different XSS attack compare to other two, because the script hacker uses not affecting HTML source code from response, only when client rendering the web page, the script will be hid into the page and do what it wants to do, while other two attacks already got script in the code when server sending back the response.

V. DIFFERENT APPROACHES TO PROTECT THE SYSTEM FROM XSS ATTACK

Developer have different ways to protect there system from XSS attacks, none of them are perfect but all those methods have advantages and drawbacks. Five different approaches are discussed in this section.

A. Pattern filtering for user input

One simple way to protect server from XSS attack is to check every user input and us a filter to eliminate all insecure string that has similar pattern to a XSS attack code. Usually the system choose to replace/remove the malicious string. Others will use escaping method, which the string that has special meaning for computer will be modified, or just give a restriction to the input with limited amount of specified input is allowed.

This method has good result dealing with all kind of XSS attacks as the user input is always where XSS is hiding. Imran Yusof and Al-Sakib Khan Pathan build up a pattern

filtering method specifically for persistent XSS attacks [5], which has a very good result filtering the input by filtering different potential XSS attack code including event handlers, data URIs, insecure keywords, escape codes, common word in XSS payload and XSS Buddies. Some examples of the filter logics been used are: using pattern `"/on\w+=—f$command/i"` to find event handlers then replace them to `" "` (null), or remove all insecure HTML keywords in user input like `isindex`, `script`, `form` to break the logic of potential XSS code.

According to their paper [5], they used a collection of XSS cheat sheets from the Internet and successfully filtered all patterns in those cheat sheets. This show that there method is effective to known XSS patterns. But as the effectiveness of pattern filtering is highly based on the database of known vulnerabilities, the effectiveness of newly developed pattern or complex pattern is not ideal. It also has a strong demand for updating the database regularly to keep up with new attack pattern, otherwise the filter will become less and less effective.

B. client side pattern matching technique

As mentioned in second section, XSS script always hide inside the user input or data inside the response. This means the malicious code will be send to user browser, so if the browser can identified the script and protect user from been attack, the problem will be solves. Many modern browsers have a function that allow the installation of extensions, and it's possible to create an extension that check all data been send and received, then protect system from XSS attacks.

There is a research on these kind of extension in 2018 [6]. In the paper they developed a Chrome extension called CounterXSS and try to identify three common XSS attack pasterns based on HTML5. The extension can identified direct attack patterns, which is script like `onscroll = "script"` been hide directly into a HTML element, a multiple format attack patterns that use attributes like `src` to directly put URL with modified parameter into the web page, or DOM-base pattern that use attribute like `onerror`, `onload` to modified HTML code using JavaScript when browser rendering the page. The extension can run in the background and will send a alert to user when a pattern has been found. Not much detailed on how the extension performed under multiple testing, but this approach show a new way to prevent XSS.

This research didn't provide a powerful method like the first one provide, but the idea is what can be borrowed. According to NetMarketShare [7], from May 2019 to April 2018, over 68% Internet users is using Google Chrome, taking other Chromium based browsers like new Microsoft Edge and Opera into consideration, the number will be easily above 70%. With all this high amount of user using browser that support Chromium extension, using a extension to provide XSS protect service will be a very effective way to improve the security of whole Internet. But there are also limitations. Just like the extension been developed in the research, extensions are usually small piece of code that couldn't acquire too much processing power, and it can only protect one browser, not the whole system. Client side approach also is said to be not effective toward web content manipulation [9].

C. server side approach

Beside from preventing XSS attack from client side, there are also server side approaches to mitigate the possibility of XSS attack. Usually server side approach need to cooperate with client side browser, server will generate the content that should be generated by browser, check it for any XSS possible content, then send the already generate page to browser. This will increase the work done by server and can also to be found not so effective [8]. So Hossain Shahriar and Mohammad Zulkernine create a new way to perform server side XSS protection based on the concept of “boundary injection” and “policy generation”, witch mainly focus on JSP programs [8].

In their approach, the server look for the dynamic content on every pages, then add a “boundary” with a specified token to it, for every content, it also generate a expected content features (attributes, JavaScript method name and other information of the content) as the “policy” and stored the policy on the server. When a client request for a web page, the server generate the response and then compare it to the boundary and policy of original page, if the feature in the boundary is different from what was expected in policy, the XSS attack is consider to be found and will deal with it using an attack handler program. If no policy deviation, server then check for boundary informations based on tokens, if there are new boundaries found or boundaries with same token, the server will remove all those boundaries. Finally the server sent the response to the server.

In there paper, the false positive rate of there approach vary between 0% - 5.2%, with a 2% - 6% delay compare to normal response due to the processing on server side, they also point out that their method has detect some advanced XSS attack other server side method failed to detect [8]. Compare to other server side method, their method is no doubt a improvement to current method, although the delay on response could be a further work to be improve.

D. static analysis method

Different from other method that focus on improving the system to be more resistant to XSS attacks, static analysis focus on identifying the vulnerabilities in server code. Static analysis usually tracked the input that are not trusty, see how the information flows and see if the data has reach a part, which it can be considered as a part of statement such as HTML or JavaScript statement.

Traditional static analysis can quickly detect XSS vulnerabilities, but it also suffers from problems like always giving wrong positive result [9]. To improve this method, Gary Wassermann and Zhendong Su bring string analysis to the traditional method [10]. They translate the testing code to a static single assignment(SSA) to encode data dependencies, they also use other techniques like context free grammar(CFG) to help improve the checking of blacklisted string values [9].

According to Gary Wassermann and Zhendong Su, there technique has improved the accuracy of static analysis, but couldn't analysis arbitrarily complex code, and the black-list policy rather than white-list policy also cause some problem. What's more, the new method still couldn't fix the weakness

that static analysis won't detect any DOM-based XSS attack [10].

E. static analysis combined with dynamic analysis

With the goal of improving static analysis, developers start to combine dynamic analysis with static analysis, in oder to solve the problems of producing too many false positive result. Based on a open source static analysis project [11], Davide Balzarotti and his colleagues build up a method that use both static and dynamic analysis to give a more accurate vulnerability analysis result [12].

For static analysis, they added a over-approximation for each testing string at every point of the program, so that they can use the value to check whether the string really poses a security risk when reaching that sink of the code. The overall performance is nearly the same. For dynamic analysis, they take the suspicious program path for every string, and try to identify if it can really harm the system. By simulating the program operating the string and check the result, dynamic analysis tried to identified false positive automatically for developer [12].

The method has proved to be more effective than static analysis, but it still has several problems. It contain some programing errors when dealing with regular expressions, the dynamic analysis process can still be insufficient [12]. As it is still a server based analysis, DOM-based XSS attack is still not been tested just like static analysis.

VI. CONCLUSION

Overall, 5 different approaches have been introduce, every-one of them has their own advantage and draw back. Pattern filtering is one of the most easiest way to counter XSS attack and it's very effective to known vulnerabilities. But the needed of a database with a regular update make it hard to enforce after deploy.

Client side pattern matching is deployed on client side, which make it a personal protection method for XSS attack, the pay back is that it won't be able to have complex protection like a normal server does, as client has much less computing power compare to server machine. it also won't detect the attack if web content manipulation is involved [9].

Opposite to client side approach, server side approach can be much more powerful than client side giving more processing power, with more method could be used to detect a XSS attack. Some server side approach is too complex that even need to corporate with client side browser, but there are new approach that are more effective and won't need the help of client browser [8].

Static analysis approach focus on if the vulnerability exist rather than defending an attack. This method need to track down the input string so it's not effective when input string is too complex, it also has issue about high false positive rate, with no ability to find vulnerability related to DOM-based attack [9].

To improve static analysis, developers combine it with dynamic analysis to get better performance. New method have lower false positive rate, but the method couldn't handle

regular expression effectively, and it still not going to test DOM-based XSS attack.

Considering all methods are far from perfect, and XSS attack is still one of the most popular vulnerabilities in the world, which means attacker is only going to find more ways to use XSS in attacking, the importance to develop more effective approach to prevent from XSS attack will keep increasing.

REFERENCES

- [1] "OWASP Top Ten" [Online]. Available: <https://owasp.org/www-project-top-ten/>
- [2] "Vulnerability distribution of cve vulnerability by types" [Online]. Available: <https://www.cvedetails.com/vulnerabilities-by-types.php>
- [3] "Apache Foundation Hit by Targeted XSS Attack — Threatpost" [Online]. Available: <https://threatpost.com/apache-foundation-hit-targeted-xss-attack-041310/73815/>
- [4] "Hacking Fortnite Accounts - Check Point Research" [Online]. Available: <https://research.checkpoint.com/2019/hacking-fortnite/>
- [5] I. Yusof and A. K. Pathan, "Preventing persistent Cross-Site Scripting (XSS) attack by applying pattern filtering approach," *The 5th International Conference on Information and Communication Technology for The Muslim World (ICT4M)*, Kuching, 2014, pp. 1-6.
- [6] A. P. Sivanesan, A. Mathur and A. Y. Javaid, "A Google Chromium Browser Extension for Detecting XSS Attack in HTML5 Based Websites," *2018 IEEE International Conference on Electro/Information Technology (EIT)*, Rochester, MI, 2018, pp. 0302-0304.
- [7] "Market share for mobile, browsers, operating systems and search engines — NetMarketShare" [Online]. Available: <https://netmarketshare.com/>
- [8] H. Shahriar and M. Zulkernine, "S2XS2: A Server Side Approach to Automatically Detect XSS Attacks," *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing, Sydney, NSW*, 2011, pp. 7-14.
- [9] L. K. Shar and H. B. K. Tan, "Defending against Cross-Site Scripting Attacks," in *Computer*, vol. 45, no. 3, pp. 55-62, March 2012.
- [10] G. Wassermann and Z. Su, "Static detection of cross-site scripting vulnerabilities," *2008 ACM/IEEE 30th International Conference on Software Engineering*, Leipzig, 2008, pp. 171-180.
- [11] N. Jovanovic, C. Kruegel and E. Kirda, "Pixy: a static analysis tool for detecting Web application vulnerabilities," *2006 IEEE Symposium on Security and Privacy (S&P'06)*, Berkeley/Oakland, CA, 2006, pp. 6 pp.-263.
- [12] D. Balzarotti et al., "Saner: Composing Static and Dynamic Analysis to Validate Sanitization in Web Applications," *2008 IEEE Symposium on Security and Privacy (sp 2008)*, Oakland, CA, 2008, pp. 387-401.

Bibliography

- Check Point Research. Hacking fortnite accounts. [online], 2019. Aviliable:
<https://research.checkpoint.com/2019/hacking-fortnite/>, accessed on June 16 2020.
- CVE security vulnerability database. Vulnerability distribution of cve vulnerability by types. [online], 2020. Aviliable:
<https://www.cvedetails.com/vulnerabilities-by-types.php>, accessed on June 10, 2020.
- DVWA team. Dvwa: Damn vulnerable web application- github. [online], 2020. Aviliable:
<https://github.com/digininja/DVWA>, accesed on July 27, 2020.
- epsylon. Cross site "scripter" - github. [online], 2020. Aviliable:
<https://github.com/epsylon/xsser>, accessed on Jult 23, 2020.
- K. Gupta, R. Ranjan Singh, and M. Dixit. Cross site scripting (xss) attack detection using intrusion detection system. In *2017 International Conference on Intelligent Computing and Control Systems (ICICCS)*, pages 199–203, 2017.
- HashiCorp. Vagrant by hashicorp. [online], 2020. Aviliable:
<https://www.vagrantup.com/intro>, accessed on July 20, 2020.
- macroman321. xss-payload-list. [online], 2020. Aviliable:
<https://github.com/payloadbox/xss-payload-list>, accessed on August 15, 2020.
- MDN web docs. Http messages - http. [online], 2020. Aviliable:
<https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages>, accessed on August 7, 2020.
- Oracle Corporation. Mysql 8.0 reference manual. [online], 2020. Aviliable:
<https://dev.mysql.com/doc/refman/8.0/en/char.html>, accessed on September 12, 2020.
- PortSwigger Ltd. Burp suite comminity edition. [online], 2020. Aviliable:
<https://portswigger.net/burp>, accessed on July 19, 2020.

- pynetwork. pypcap - python libpcap module. [online], 2018. Aviliable:
<https://github.com/pynetwork/pypcap>, accessed on August 5, 2020.
- H. Shahriar and M. Zulkernine. S2xs2: A server side approach to automatically detect xss attacks. In *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*, pages 7–14, 2011.
- The Open Web Application Security Project. Owasp top ten. [online], 2020. Aviliable:
<https://owasp.org/www-project-top-ten/>, accessed on June 15, 2020.
- Threatpost. Apache foundation hit by targeted xss attack. [online], 2010. Aviliable:
<https://threatpost.com/apache-foundation-hit-targeted-xss-attack-041310/73815/>, accessed on June 15, 2020.
- Yang Ding. A detection method for cross-site scripting. [online], 2020. Aviliable:
https://git.soton.ac.uk/yd4u19/msc-project_yd, accessed on September 15, 2020.
- ZAP Dev Team. Owasp zap. [online], 2020. Aviliable:
<https://www.zaproxy.org/>, accessed on July 25, 2020.