A SECURITY ASSESSMENT OF EGOVERNMENT WEBSITE IN MALAYSIA

CHUR JIAN CHANG

A project report submitted in partial fulfilment of the requirements for the award of Master of Information System

Lee Kong Chian Faculty of Engineering and Science
Universiti Tunku Abdul Rahman

November 2018

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature : _____

Name : CHUR JIAN CHANG

ID No. : 17UEM00878

Date : _____

APPROVAL FOR SUBMISSION

I certify that this project report entitled **"A SECURITY ASSESSMENT OF EGOVERMENT WEBSITE IN MALAYSIA"** was prepared by **CHUR JIAN CHANG** has met the required standard for submission in partial fulfilment of the requirements for the award of Master of Information System at Universiti Tunku Abdul Rahman.

Approved by,

Signature : _____

Supervisor : DR. MADHAVAN A/L BALAN NAIR

Date : _____

Signature : _____

Co-Supervisor : _____

Date : _____

ACKNOWLEDGEMENTS

I would like to thank everyone who had contributed to the successful completion of this project. I would like to express my gratitude to my research supervisor, Dr. Madhavan a/l Balan Nair for his invaluable advice, guidance and his enormous patience throughout the development of the research.

In addition, I would also like to express my gratitude to my friend, Mr. Choong Kok Wai for his motivation and immense knowledge. He is exceptionally generous in sharing his extensive knowledge in the field of IT Security with me.

Last but not least, my sincere thanks to my family and friends who have been providing me with great support throughout the project.

ABSTRACT

Nowadays, almost every government has been using web support to improve their performance. Centralised databases have provided the websites the ease of retrieving necessary data as well as storing sensitive information such as citizen information, financial, economic statistics etc. Structured Query Language (SQL) injection and Cross Site Scripting (XSS) is perhaps one of the most common and critical attacks used by attackers to deface the website, manipulate or delete the data through injecting malicious scripts. According to the Open Web Application Security Project (OWASP), SQLi ranked highest in the vulnerability list the past few years. This study focuses on studying the vulnerability SQLi and XSS. Manual vulnerability assessment with black box testing was implemented in several Malaysia government web applications to identity their vulnerabilities, the data found was analysed to draw statistical conclusion of the present condition of government websites of Malaysia. Lastly, we also discuss the impact of both attacks and proposed possible countermeasures.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS / ABBREVIATIONS

SQL	Structured Query Language
SQLi	SQL Injection
XSS	Cross Site Scripting
HTTP	Hypertext Transfer Protocol
URL	Uniform Resource Locator

CHAPTER 1

INTRODUCTION

1.1 Introduction

Web applications have been integrated into our life recently, and they often contain sensitive information which must be protected especially government websites. While the rapidly growth in using web applications, the types of attack used to deface web application is increasing as well. To secure that sensitive information in web applications, people try to develop the system completely secure. However, it is nearly impossible as hackers are always creative enough to find out vulnerabilities to attack the system. The total number of vulnerabilities in web applications would not be a problem if they were not being exploited (Inforisk360, 2018). To mitigate the risks, people identify vulnerabilities in web applications and find mitigations.

Several technical reports and research studies have acknowledged that SQL Injection (SQLi) and Cross-Site Scripting (XSS) remain the most common and critical attacks (WhiteHat Security Threat Research Center, 2017). A non-profit organization the Open Web Application Security Project (OWASP) provides unbiased information on web applications security. In the OWASP 2017 Top 10 web application security risks, Injection remained first ranked followed by Broken Authentication, Sensitive Data Exposure, etc (Wichers and Williams, 2017).

This research will focus on the most prominent web application attacks: SQLi and XSS. Several Malaysia government web applications will be accessed to identify the existence of any vulnerabilities that might allow both mentioned attacks to be crafted, lastly mitigation frameworks will be proposed for them as well.

Motivation:

People pay no attention to the security of web application to protect sensitive information, they would realise the importance of security when they experienced in the attacks. Web applications like government websites contains much sensitive information such as identity card number, addresses, telephone number etc, hence, precautions must be taken to avoid the data being leaked.

1.2 Problem Statement

The rapid growth in relying the convenience of using web applications, most of the organisations focus on functionality of the web application and overlook the importance of security. In 2011, Sony Pictures was attacked by a group called Lulz Security with a very simple SQLi and SonyPictures.com had compromised more than a million users' confidential information including passwords, date of birth, home addresses, email addresses and other personal information (Kumar, 2011). According to OWASP (Wichers and Williams, 2017), injection is a major problem in web security which remained ranked 1 in the OWASP Top 10 web application security risks since year 2013. Injection may lead to information disclosure, data loss, authentication bypass, loss of data integrity etc. Apart from that, Acunetix vulnerability testing report in 2017 summarised that data and analysis of vulnerabilities detected by Acunetix from March 2016 to March 2017. According to the report (Ian, 2017), 50% of the sampled targets contain XSS vulnerabilities whereas SQLi vulnerabilities were found on 20% of sampled targets.

At least 33 Malaysia websites have been attacked by Distributed Denial of Services (DDoS) and defaced by an Indonesian hacker group KidsZonk who were unhappy by the flag blunder in the official souvenir booklet of the Kuala Lumpur SEA Games 2017 (Mohsen, 2017). Users have been redirected to a splash page which showing the Indonesian flag upside down with a message "Bendera Negaraku Bukanlah Mainan". Fortunately, the Indonesian hacker's intention was not about sensitive information in Malaysia websites but just a shadow of anger.

According to recent news report by The Star (Razak, 2018), the media company Media Prima Berhad which runs newspapers, advertising, TV/radio channels and digital media companies was attacked by ransomware on 8th of November 2018, reported by The Edge Financial Daily. Ransomware is a malicious software to block access to a computer system until a ransom is paid. Therefore, the company are requested a ransom of RM26.42 million to release the access to the system.

With the cases above, we should take the initiative to identify and mitigate any existence of vulnerabilities in targeted Malaysia government websites before enabling attackers to craft any attacks in the future. In the current rapid growth in Information Technology, it provides the convenience of using web applications while it also has some certain risks of using it. Therefore, it is important to identify the causes of the risks and the ways to mitigate them.

1.3 Objectives

The purpose of this research is to identify security practices of targeted Malaysia government web applications and identity the existence of SQLi and XSS vulnerabilities. Objectives have been identified as below:

1. To identify the common security practices of targeted Malaysia government web applications.
2. To identify possible vulnerabilities in targeted Malaysia government web applications.
3. To identify possible consequences caused by vulnerabilities and attacks for targeted Malaysia government web applications.
4. To propose threats mitigation frameworks for targeted Malaysia government web applications.

1.4 Research Questions

1. What is SQLi and XSS vulnerabilities?
2. What are the vulnerabilities identified in the targeted Malaysia government web applications?
3. What are the possible consequences caused by vulnerabilities and attacks for targeted Malaysia government web applications?
4. What are the mitigation frameworks to prevent both SQLi and XSS vulnerabilities?

1.5 Scope of Work

1.5.1 Identifying the Possible Vulnerabilities

Using the OWASP Top 10 web application security risks, we selected to focus on SQLi and XSS vulnerabilities. The tool Burp Spider will be adopted to gather information about targeted Malaysia government web applications and some methods to identify both vulnerabilities. Sony Pictures showed the serious impact of attack by SQLi, where hackers are able to retrieve sensitive information from the system once they have successfully injected. This is the lesson that should be learnt and take the initiative to mitigate from being injected.

1.5.2 Discussing Threats and Consequences Caused by Vulnerabilities and Attacks for Targeted Malaysia Government Websites

Once the vulnerabilities SQLi and XSS are identified, this study will also discuss about the threats will be brought according to the findings. In order showing the severity of threats to users about the importance of security, consequences will be pointed out with examples.

1.5.3 Propose Threats Mitigation Framework

From the findings of identified vulnerabilities SQLi and XSS, mitigations will be recommended to vulnerable web applications.

1.6 Contribution

From the view of Malaysia government, the result generated at the end of this research will benefit the government whether the targeted web applications are vulnerable. The result will help the government to be aware of the existence of vulnerabilities and mitigation framework will be proposed for Malaysian government to have a more systematic approach to secure their web applications. Apart from this, people who either developing or auditing web applications will have rough idea on steps to identify vulnerabilities in the system and they will be told the ways to mitigate the threats.

1.7 Novelty

There are researches have been done on identifying vulnerabilities in government web applications of South African, North Khorasan in Iran etc. However, there is no existing specific framework available for Malaysia government to follow to secure their web applications. The result will benefit to Malaysia government as they are able to be aware of if there is any existence of vulnerabilities in their websites and to propose mitigation framework for them.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

In this chapter, a series of comprehensive literature review covered on several areas for this research as the following:

- Overview of Web Application Security
- Overview of HTTP URL
- Overview of HTTP GET/POST Methods
- Overview of HTTP Requests and Responses
- Overview of HTTP Status Codes
- Introduction of SQLi
- Introduction of XSS

2.2 Web Application Security

Recently there are reports generated by WhiteHat Security Threat Research Center and Open Web Application Security Project (OWASP) describe about the TOP 10 common attacks in 2017. Among many different types of attack, they state that SQLi and XSS remain the most common and critical attacks. Other than that, a recent research done by Talebzadeh and Ghodrat in 2017 is similarly same as accessing to government or organisation websites to identify possible vulnerabilities for SQLi and XSS. In the result (Talebzadeh and Ghodrat, 2017), 3 out of 11 targeted web applications are vulnerable to XSS and 2 websites are vulnerable to SQLi. Simultaneously another research found that 309 educational web applications are found vulnerable to various types of SQLi (Delwar *et al.*, 2015).

2.3 HTTP URL

A uniform resource locator (URL) is a unique identified for a web resource (Dafydd and Marcus, 2008). The URL used to generate Hypertext Transfer Protocol (HTTP) request takes the form:

http://<host>:<port>/<path>?<searchpart>

<host> is usually the qualified domain name of network host or its IP address, *<port>* is the port number to connect to. If *:<port>* is omitted in the URL, then the port number is default to 80. The *<path>* is an HTTP selector and *<searchpart>* is a query string. The combination *<path>?<searchpart>* is optional, if it is not present then the */* may be removed. Those notations */ ; ?* are reserved within the *<path>* and *<searchpart>*.

2.4 HTTP GET/POST Methods

HTTP methods are actions performed on resources (Joel, Vincent and Caleb, 2011). In attacking web applications, GET and POST methods are most commonly used. GET is designed to retrieve information whereas POST is designed to send information. Both GET and POST can send information to the server, the important difference between them is GET leaves all the data in URL such as the *id=13* in the following URL:

http://www.sparx.com/php?id=13

but POST places the data in the body of the request which is not visible in URL. In terms of security, GET is less secure than POST as parameters are being exposed in URL.

2.5 HTTP Requests and Responses

When a client navigates to a web page, a request of the web page content will be sent to the server, the request is called HTTP request (Fielding, et al. 1999). After the server received the request, the server will interpret it and responds with an HTTP response message. Figure 2.1 is an example of request we sent to www.google.com server with GET method to retrieve content of its home page. After the request is received, the server will response to the client whether the request is being accepted by using HTTP status codes. As shown in Figure 2.2, the server accepted our request and response 200 HTTP status code with the requested content.

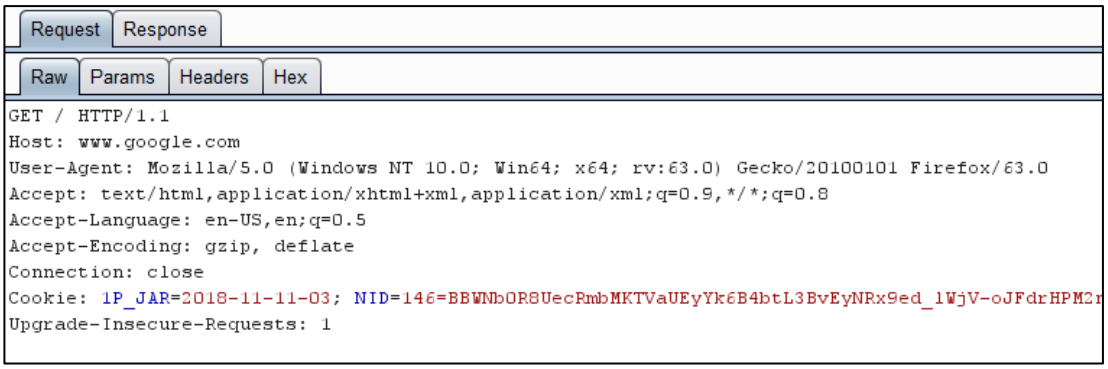

Figure 2.1: Request Sent to www.google.com

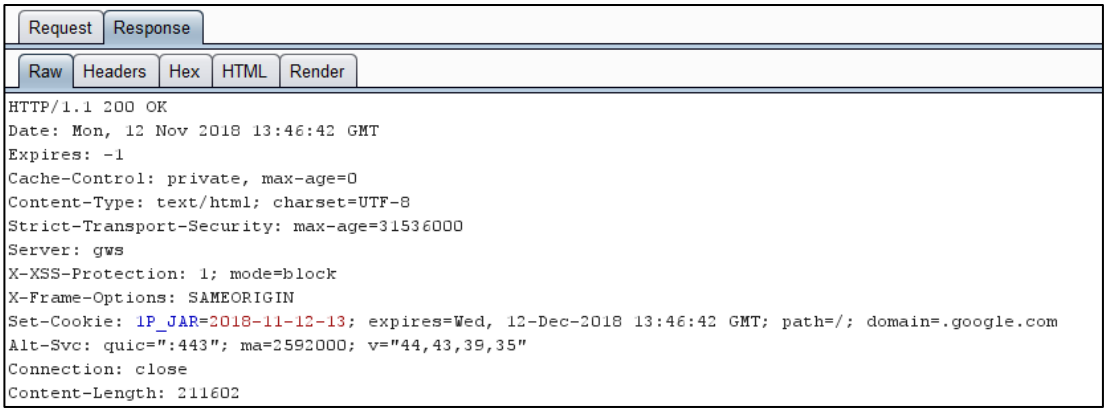

Figure 2.2: Response Get from www.google.com

2.6 HTTP Status Codes

The HTTP status code is issued by a server in response to the request made by client (Fielding, et al. 1999). Those codes are consolidated by 3-digit integer and categorised into 5 classes of response with the first digit of the status code as below:

- 1xx Informational: The request was received and continuing process.
- 2xx Success: The action was successfully received, understood and accepted.
- 3xx Redirection: Further action has to be taken to complete the request.
- 4xx Client Error: The request contains bad syntax or cannot be fulfilled.
- 5xx Server Error: The server failed to fulfil an apparently valid request.

Table 2.1, 2.2, 2.3, 2.4 and 2.5 are showing each of the status codes in every classes with corresponding meaning.

Table 2.1: 1xx Information Responses

Status Code	Meaning
100	Continue
101	Switching Protocols

Table 2.2: 2xx Success Responses

Status Code	Meaning
200	OK
201	Created
202	Accepted
203	Non-Authoritative Information
204	No Content
205	Reset Content
206	Partial Content

Table 2.3: 3xx Redirection Responses

Status Code	Meaning
300	Multiple Choices
301	Moved Permanently
302	Found
303	See Other
304	Not Modified
305	Use Proxy
307	Temporary Redirect

Table 2.4: 4xx Client Error Responses

Status Code	Meaning
400	Bad Request
401	Unauthorised
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed

406	Not Acceptable
407	Proxy Authentication Required
408	Request Timeout
409	Conflict
410	Gone
411	Length Required
412	Precondition Failed
413	Request Entity Too Large
414	Request-URI Too Large
415	Unsupported Media Type
416	Requested Range Not Satisfiable
417	Expectation Failed

Table 2.5: 5xx Server Error Responses

Status Code	Meaning
500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Timeout
505	HTTP Version Not Supported

2.7 Structured Query Language Injection (SQLi)

SQLi attacks have become the most critical web application attack. Structured Query Language (SQL) with malicious code is provided by attackers to a user input field of web interface, the SQL will be sent to the database for execution to allow attackers steal or manipulate data. Based on a survey (Diallo and Al-Sakib, 2011) and the other survey (Ossama and Mohammad, 2016), SQLi attacks are categorised into: Tautology, Illegal/Logically Incorrect Queries, Stored Procedures, Piggy-Backed Queries, Alternate Encodings and Inference.

2.7.1 Tautology

This attack helps attackers to identify injectable parameters, bypass authentication and extract data from the database. Attackers inject code in the WHERE clause of a SQL

query so that the query will always evaluates to true. The code *OR 1=1* is generally to be used for injection as the statement *1=1* will always return true, for example the query below is generated to display bank account details of owner Joel which does not exist:

*SELECT * FROM BankAccounts WHERE Owner = 'JOEL';*

After injecting *'OR 1=1--*, the query below will display all the data of BankAccounts table due to the query will be always true even if the owner Joel does not exist.

*SELECT * FROM BankAccounts WHERE Owner = 'JOEL' OR 1=1--';*

2.7.2 Illegal/Logically Incorrect Queries

This type of attack can be used as preliminary attack to gather information about the backend database, then attackers are able to craft further attacks by using the information revealed. By injecting wrong malicious queries, the database will be caused to return error messages which might include the information about the database. The following query Owner input parameter is injected by *' UNION SELECT SUM(Owner) FROM BankAccounts* which attempt to sum the NVARCHAR type Owner column in BankAccounts table:

*SELECT * FROM BankAccounts WHERE Owner = '' UNION SELECT SUM(Owner) FROM BankAccounts;*

Since there is no way to sum NVARCHAR value, this causes the database to return error messages containing information about the database and Owner column.

2.7.3 Union Query

This attack is used to bypass authentication and extract data from the database. UNION operator is injected by attackers into a vulnerable parameter that eventually will return the dataset of both the original and injected queries. For example, an additional query *' UNION SELECT * FROM AccountBalance* -- is injected into Owner column to extract data from AccountBalance table:

*SELECT * FROM BankAccounts WHERE Owner = '' UNION SELECT * FROM AccountBalance --;*

2.7.4 Stored Procedures

Most of the database provides a set of stored procedures that extend the functionality of the database. Once the attackers know which type of database is in use and the

vulnerable injectable parameter, they can craft an attack by executing stored procedures provided by the database. For example, input parameter is being injected by *'; SHUTDOWN;* -- which will run the stored procedure to cause the database to shut down:

*SELECT * FROM BankAccounts WHERE Owner = ''; SHUTDOWN; --;*

2.7.5 Piggy-Backed Queries

This attack is similar as Stored Procedures attack, in which additional queries will be added to the end of a query. Attackers can retrieve or modify data by executing additional queries, for example:

*SELECT * FROM BankAccounts WHERE Owner = ''; DROP TABLE BankAccounts --;*

If the database allows additional queries to be appended at the end of a valid SQL query, then the second query above will be executed and it will delete the BankAccounts table.

2.7.6 Alternate Encodings

This attack can be used to bypass detection methods used by defensive coding practices. Common detection methods will scan for harmful code from user input such as *SHUTDOWN*, however, attackers able to encode the input string using ASCII, hexadecimal, and so on to mask the attack. For example, *char(0x73687574646f776e)* will return the string *SHUTDOWN*.

*SELECT * FROM BankAccounts WHERE Owner = ''; char(0x73687574646f776e);*

2.7.7 Inference

This attack's intention is to identify injectable parameters, extracting data and determine database schema. Attackers modify query to be executed and return true/false from the database to derive logical conclusions. Inference SQLi can be differentiate into Blind Injection and Timing Attacks. In Blind Injection technique, true/false questions will be asked to the server to infer the information from the page. If the injected statement returns true, the site will continue function normally else the page will differ significantly although there is no error message. In Timing Attacks, attackers gain information by observing time delays in response of the site. To perform this attack, attackers normally generate the query in the form of if/then statement.

2.8 Cross Site Scripting (XSS)

XSS is the Godfather of attacks against other users (Dafydd and Marcus, 2008). XSS is another common and well-known web application attack. I. Yusof & A. Pathan have discussed about XSS (Yusof and Pathan, 2016), XSS involves injecting malicious script in HTML, Java, Flash, ActiveX, JavaScript or other browser supported technology and execute in a victim's browser. By XSS, attackers can hijack user sessions via cookie or even redirect users to malicious sites without victim's knowledge. There are few research papers discussed about different types of XSS attack, they are generally categorised into three different types: Reflected (Non-Persistent), Stored (Persistent) and DOM-based.

2.8.1 Reflected XSS (Non-Persistent)

This is a very common type of XSS also known as Type-1 XSS. It usually occurs when an application responses request on a dynamic page to display contents to users. For example, a user has logged in to a bank application as usual and is issued with a cookie containing a session token:

Set-Cookie: sessId=193a9100ed37374201a4b9672362g13459c2a652401a1

then the user requests a malicious URL crafted by attacker that embedded JavaScript such as the following:

https://bank.com/error.php?message=<script>var+i=new+Image;+i.src="http://w
ahh-attacker.com/"%2bdocument.cookie;</script>

the server responds to the user's request then attacker's script and the script will be executed in user's browser to send session token for attack to hijacks the user's session.

2.8.2 Stored XSS (Persistent)

This attack is also known as Type-2 XSS which happens when a web application accepts and stores hostile data by one user in a file, database, or other backend system then displays the unfiltered data to other users without being filtered or sanitised appropriately (Inforisk360, 2018). This is extremely dangerous for web applications that support interaction between users such as blogs, forums, or others content management systems as huge number of users see the hostile data input from hackers. The stored attack can be even worse such as a user with administrative privileges visits an infected website, attackers could potentially be stealing the user's cookie. The attackers' script can be stored in database and they can use it anytime. For example, a

forum application that allows users to post questions in a specific discussion. If a user is able to post a question that contains JavaScript without being filtered or sanitise by the application, then attackers can post a crafted question that with arbitrary scripts to execute within the browser of whoever views the question.

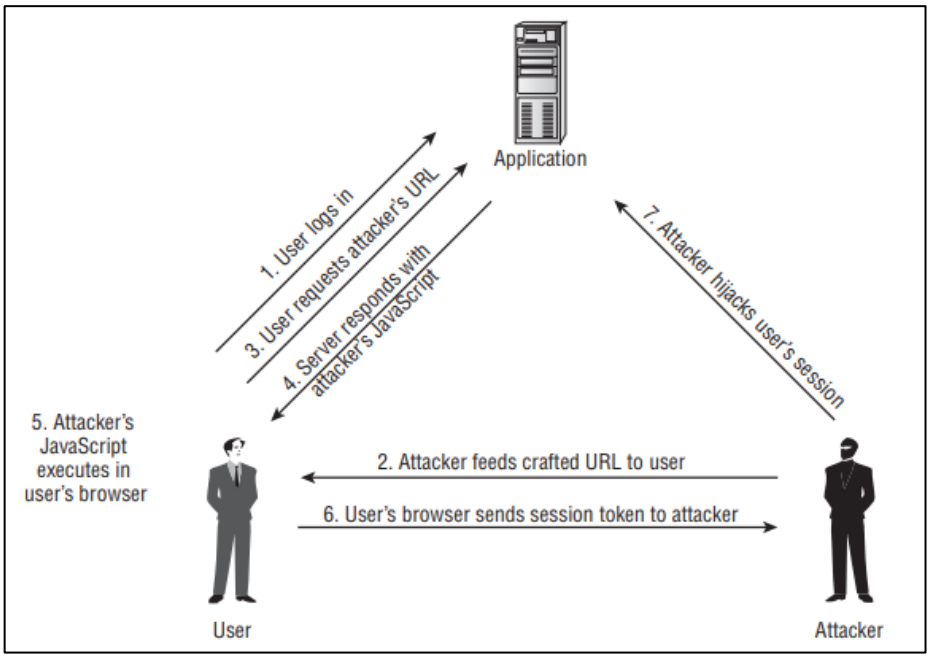

Figure 2.3: The Steps Involved in a Reflected XSS attack
Adapted from "The Web Application Hacker's Handbook Discovering and Exploiting Security Flaws" (Dafydd and Marcus, 2008)

2.8.3 DOM-Based XSS

This is known as Type-0 or Document Object Model-based XSS which is an advanced type of XSS attack. In previous examples of reflected and stored XSS, the server-side takes data from a crafted URL parameter and inserts malicious script into the page and responds to user. When the user's browser receives the response and executes the malicious script while the page loads. In DOM-based XSS, there is no malicious script inserted as part of the page but the script is executed after the page has loaded. How does it work? It works when the client-side script can access the browser's DOM, then it can determine which URL is used to load the current page and dynamically update the data from URL.

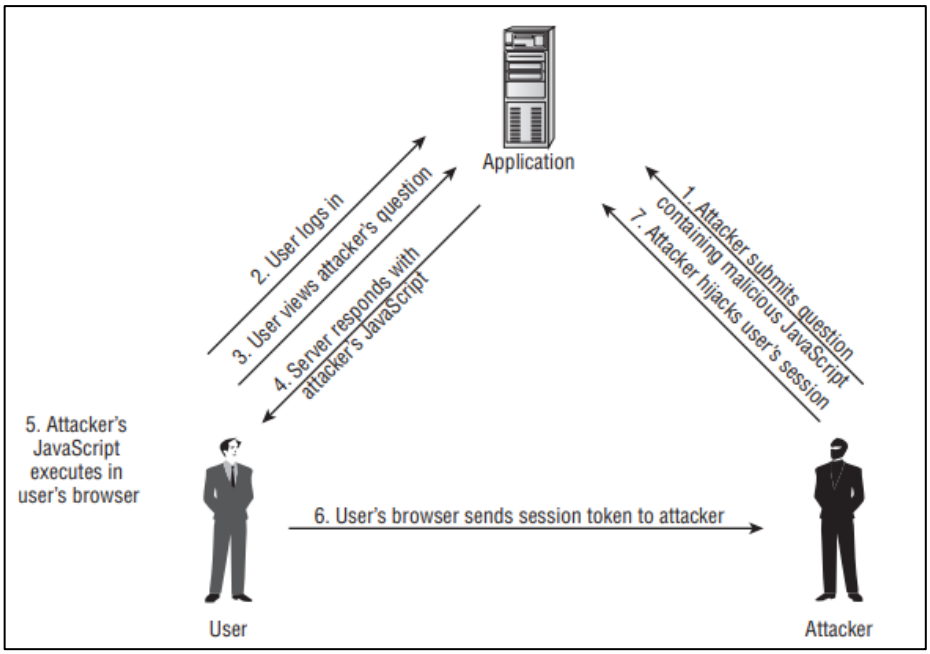

Figure 2.4: The Steps Involved in a Stored XSS attack

Adapted from "The Web Application Hacker's Handbook Discovering and Exploiting Security Flaws" (Dafydd and Marcus, 2008)

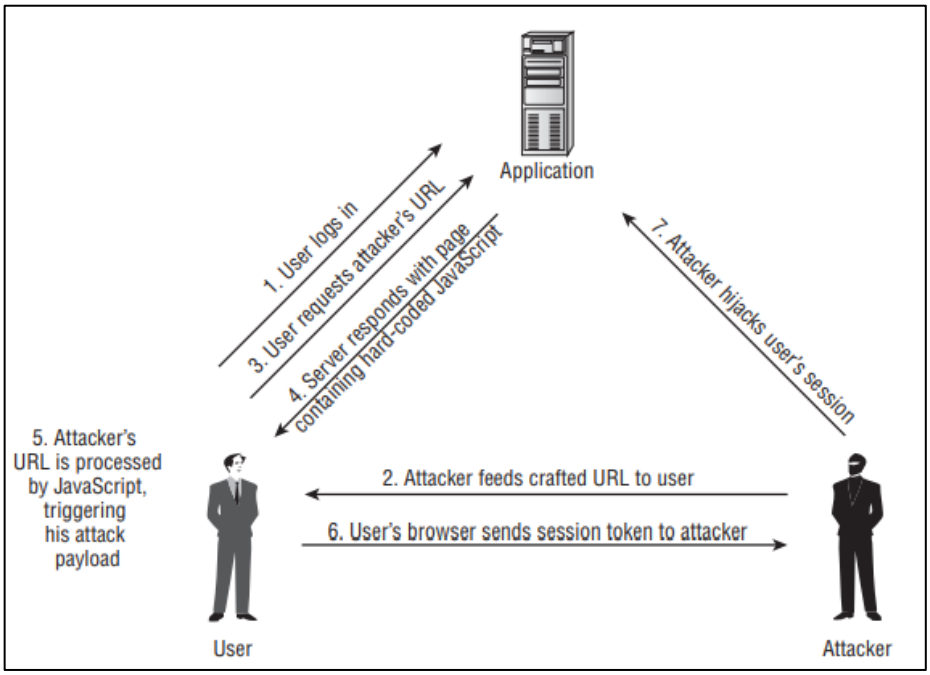

Figure 2.5: The Steps Involved in a DOM XSS attack

Adapted from "The Web Application Hacker's Handbook Discovering and Exploiting Security Flaws" (Dafydd and Marcus, 2008)

CHAPTER 3

RESEARCH METHODOLOGY

3.1 Introduction

This chapter is to demonstrate the actions taken to achieve all listed research objectives in Chapter 1. Especially the process of vulnerabilities assessment, the websites selection, the information gathering method, the discovery of vulnerabilities, and also the way to avoid being detected from web application server.

3.2 Research Approach

Under Part II of the Computer Crimes Act 1997, it is an offence if a person knowingly and intentionally accesses a computer without authorisation and causes a computer to perform any function with the intent to secure access to any program or data held in any computer (Deepak and Yong, 2019). As we must not go into the system of targeted websites, hence Black Box testing method will be used to identify the vulnerabilities. While performing Black Box testing, we might adopt some tools to identity the vulnerabilities by intercepting the traffic and observe the process.

Basically, a complete penetration testing is containing five different stages such as Reconnaissance, Profiling, Discovery, Exploitation and Reporting. Understanding the concept behind every stage which will help us to perform good assessment. Below is an overview of the five stages of penetration testing:

I. Reconnaissance: This is all about information gathering. Collecting information about the target application as much as possible for the later exploitation uses.
II. Profiling: In order to perform a good testing, understanding the target application is important as well. Usually, a dummy account will be register to observe the features, how the application works, etc.
III. Discovery: In this stage, start to discover is the application vulnerable to any attacks.
IV. Exploitation: After discovering, exploit that system.
V. Reporting: Lastly, a report will be prepared to informing the application is vulnerable to what attacks with priority to solve.

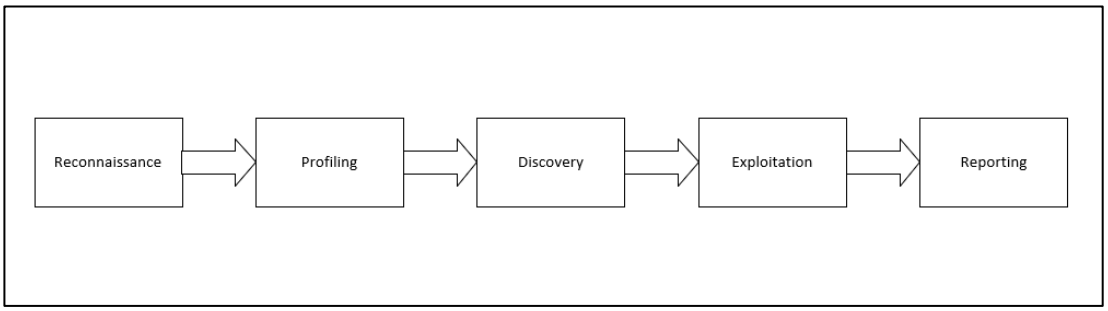

Figure 3.1: Different Stages of Penetration Testing

As this project is not authorised to perform such testing, hence only the stages Reconnaissance, Discovery and Reporting will be performed. The reasons skipping Profiling and Exploitation is not to create any data in the target web applications yet exploitation will be infringing the system operation of the web applications. This process is called as Vulnerabilities Assessment, because discovery of vulnerabilities will be done but not exploitation, once the application is found to be vulnerable then the process stops.

In Figure 3.2, shows the flow of vulnerabilities assessment for this project. Start from gather information from the application, then perform discovery. If the application is vulnerable, then stop for the application immediately. However, if the URL tested is not vulnerable, then move to next URL to continue the discovery until finish looping all the possible URL. A further discussion in each stage will be discussed later. But before that, a list of possible vulnerable websites should be ready.

3.3 Web Applications Selection

As discussed previously in Chapter 2.2, <*searchpart*> in HTTP URL scheme is a query string. If parameters are exposed in <*searchpart*> means that we are able to inject scripts into the query.

Google Search Engine is the most popular and the best in the world. According to the report (NetMarketShare, 2018), more than 76% searches were powered by Google. Its search engine algorithm provides the best results to users and the search setting is customisable for specific search. Hence, Google Search Engine is used to search for possible vulnerable Malaysia government websites by using the keywords *inurl:php?id= site.:gov.my*. This returned list of Malaysia government websites that are developed by PHP and exposing parameters in the URL, this does not mean that those web applications are vulnerable but they have the potential to be vulnerable. A

total of 21 different Malaysia government domains could be found from the list, but due to time constraint, only 10 of them are chosen for this research purpose.

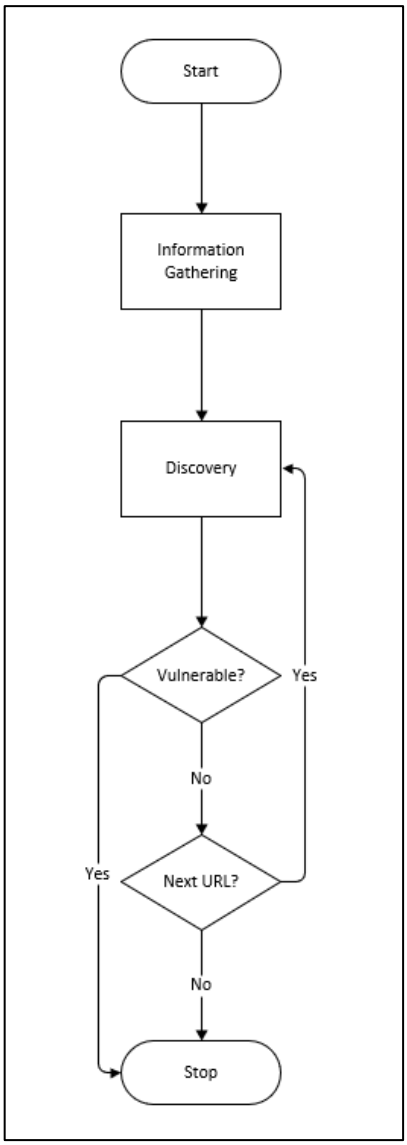

Figure 3.2: Flowchart of Vulnerabilities Assessment

3.4 Information Gathering

Information gathering is to gain accurate information about target web applications without revealing our presence or our intentions, to learn how the organisation operates, and to determine the best route of entry (Inforisk360, 2018). Information about the specific IP addresses which could be accessed over the Internet, operating system, system architecture, and the services running. Information gathering can be

categorised into Passive and Active. Passive Information Gathering, we can use tools like search engines, social networking, etc to discover information about targeted web applications without touching their systems. Active Information Gathering, we will interact directly with the systems to conduct port scans for open ports, determine what services are running, determine the details of operating system, etc.

In this project, Burp Suite Community Edition was used and the Spider function in Burp Suite to perform active information gathering. The Burp Spider is actually a scanner that perform task of scanning web sites for content, it navigates around a target web application like a normal user with a browser, by clicking links and submitting requests to the server. Eventually as in Figure 3.3, it returned list of URLs of the target web application under same domain and indicated whether there are any parameters being exposed (column Params with tick) in the URL.

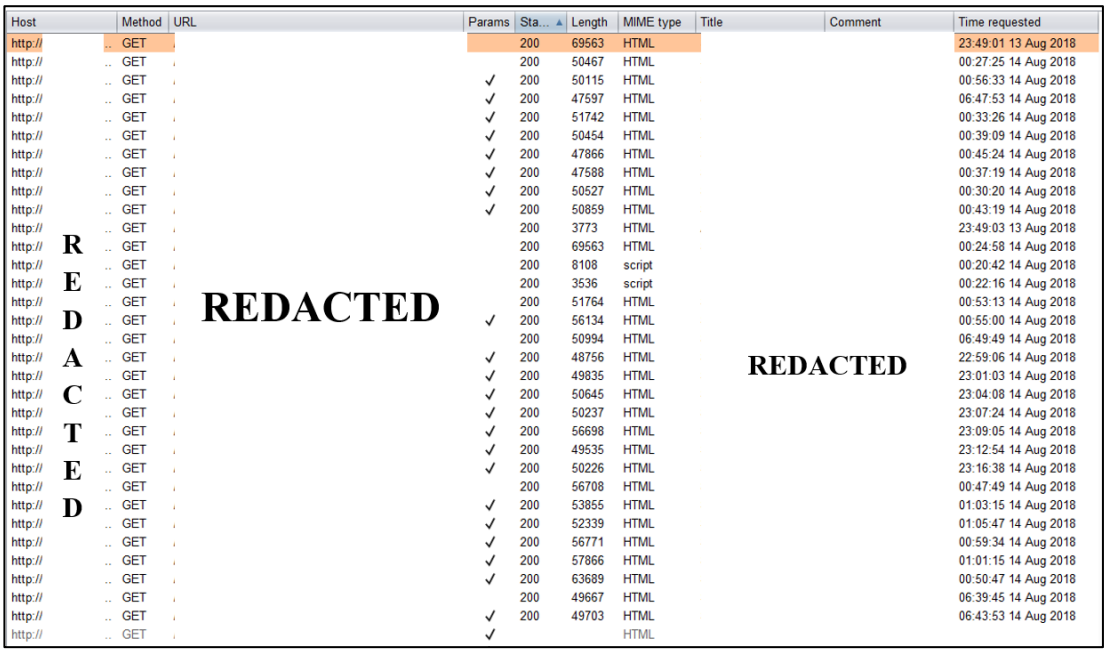

Figure 3.3: Example Results of Burp Spider

3.5 Vulnerabilities Discovery

A web application has many URLs under its domain. If we are able to show any of these URLs is vulnerable to SQLi or XSS, then we could conclude that the web application is vulnerable to the attack. An example, if an URL response with malicious SQL scripts then we are allowed to connect to the database and retrieve or manipulate

the data or do whatever we want, same goes to XSS. Hence, if an URL is vulnerable then the whole application is vulnerable.

3.5.1 Discovery of SQLi

As discussed, SQLi has several types of attack. In order not to exploit and infringe the system, we chose to perform Illegal/Logically Incorrect Queries injection by just injecting a single quote in the end of parameters exposed URL. For an example URL:

http://www.sparx.com/php?id=13

the system is expected to retrieve content *id=13* by executing query:

*SELECT * FROM dbo.PageContent WHERE Id='13'*

So, we inject a single quote at the end of URL

http://www.example.com/php?id=13'

and the system will input the value *13'* into the query like:

*SELECT * FROM dbo.PageContent WHERE Id='13''*

By executing this incorrect query, it will cause the database server to return a syntax error message. If the URL reflects with the error message, this means the application is vulnerable to SQLi then we stop the testing on this web application. Unfortunately, if the application does not response which means the application has secured for this URL. If that is the case, we stop testing on this URL and move on to the next one until we successfully show the application is vulnerable or finish testing on all URLs without responses.

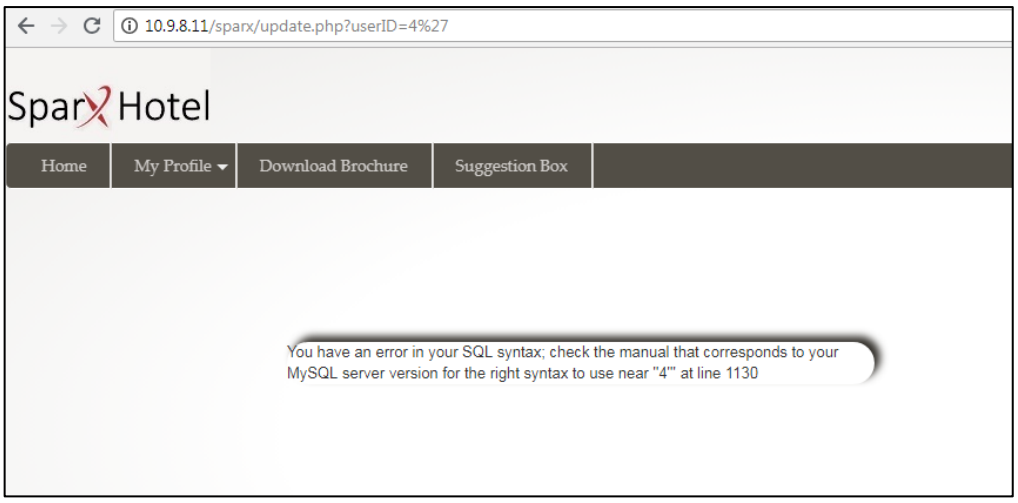

Figure 3.4: Example of SQLi with Single Quote

3.5.2 Discovery of XSS

A basic approach provided by (Dafydd and Marcus, 2008) to identify XSS vulnerabilities is to use a standard proof of concept attack string such as the following:

"><script>alert(document.cookie)</script>

This string is submitted as a value of parameter to the page of the application, and responses with the injected JavaScript that display pop-up message with browser cookie. If the request of browser cookie is responded and being shown as in Figure 3.5, then the application is very likely vulnerable to XSS.

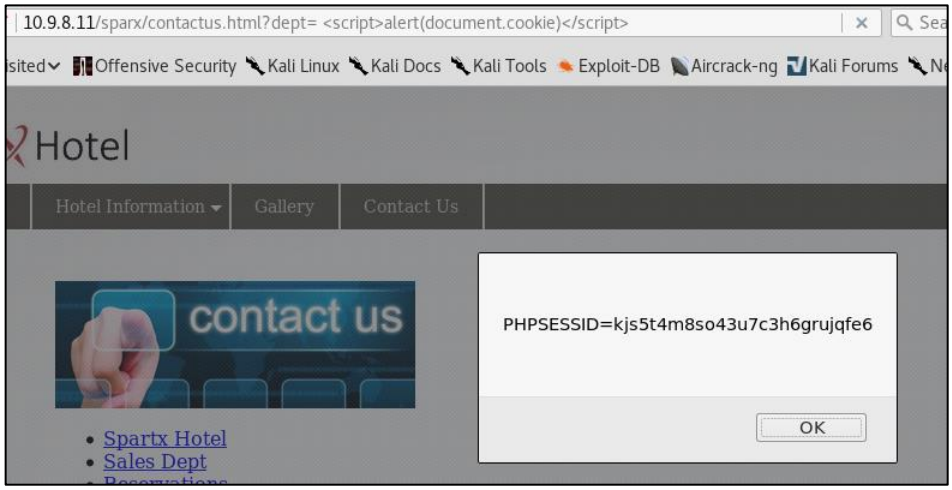

Figure 3.5: Example of XSS with Alert Box

It is possible that the application will not be identified vulnerable to XSS via the basic approach, but this does not mean that the application is not vulnerable. Most of the applications have implemented a rudimentary mitigation like blacklist-based filters to prevent XSS attacks (Dafydd and Marcus, 2008). These filters usually look for typical expression like *<script>* within the request parameters, if the expression is found then some defensive actions will be taken such as removing or encoding the expression, or even blocking the request. However, there are cases of XSS successfully exploit without using those common characters like < > / and ". Besides the basic approach, the following strings or other than them will bypass the filter and successfully result in XSS exploitation. Although these strings may be decoded, sanitized, or modified before being return in the server's response, but still able to perform XSS exploitation:

"><script >alert(document.cookie)</script >

"><ScRiPt>alert(document.cookie)</ScRiPt>

"%3e%3cscript%3ealert(document.cookie)%3c/script%3e

"><scr<script>ipt>alert(document.cookie)</scr</script>ipt>

%00"><script>alert(document.cookie)</script>

3.6 Avoiding from Detection

Nowadays, people will monitor the server to detect any uncommon or suspicious requests generated, some actions will be taken such as blocking the IP address from their server or some legal actions. In order not to be detected, time delay method is used which is in a slow pace during information gathering and find vulnerabilities.

In Burp Spider, the configuration is set to not submit any forms, as user data should not be sent or created in this project. Besides that, in order to reduce the speed of scanning which will make many HTTP request to the server, only 2 threads are set to work concurrently and 3 minutes waiting time between requests.

While identifying the SQLi vulnerabilities, only the basic approach to inject single quote *'* at the end of value of parameter will be adopted such as the following:

http://www.sparx.com/php?id=13'

If this payload does not identify the SQLi vulnerabilities, the identification will be stopped for the URL and proceed on next URL after 5 minutes.

Applying the same method on identification of XSS vulnerabilities, only the following string will be submitted as a value of parameter:

"><script>alert('Test')</script><!--

Compare this string to the basic approach as mentioned earlier, symbols *<!--* is appended to comment the remaining HTML codes. If this does not identity the XSS vulnerabilities, identification will be stopped for the URL and proceed on next URL after 5 minutes.

Theoretically, there is Cheat Sheet for SQLi or XSS that contains list of payloads to be tested one by one until the vulnerabilities is identified or end of the list. However, only one payload for one attack is used in this project. The more payloads being tried the easier to be detected.

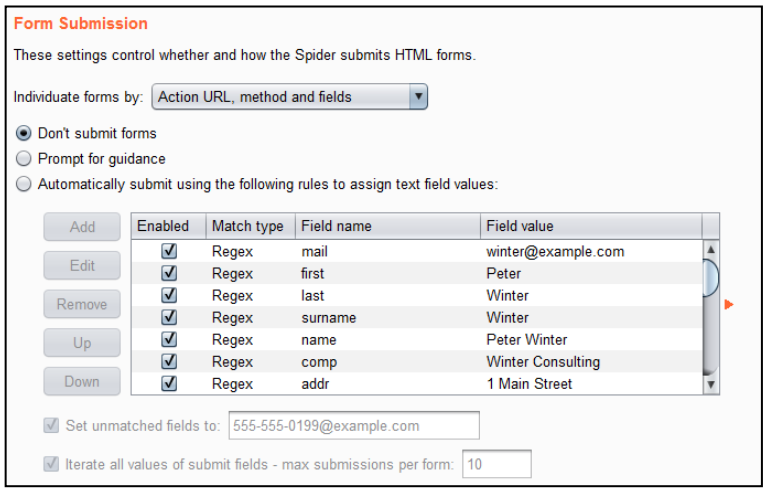

Figure 3.6: Form Submission Options in Burp Spider

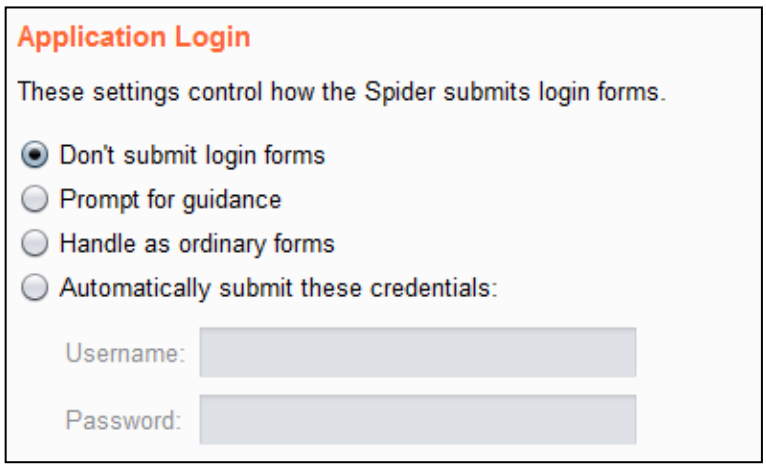

Figure 3.7: Application Login Options in Burp Spider

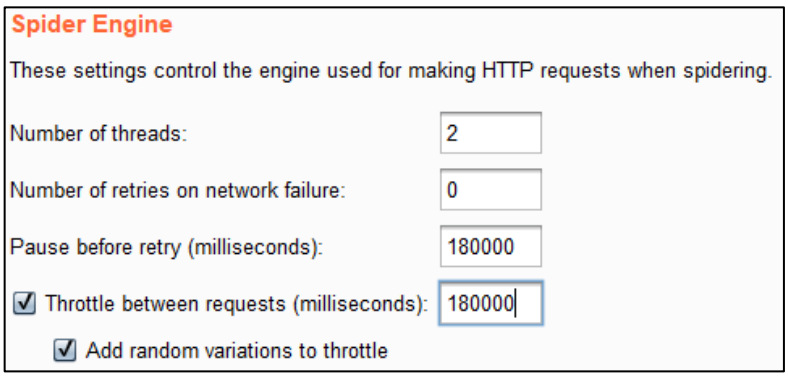

Figure 3.8: Spider Engine Options in Burp Spider

3.7 Conclusion

This chapter presented the core aspect of this study where it describes the research methods used in conducting the search. The research methodology focused on the

process of vulnerability assessment, the way of selecting possibly vulnerable Malaysia government web applications, the tools and configurations to gather information of web applications and the payloads used to identify SQLi or XSS in web applications.

Apart from that, this study applied the time delay method where all the actions were performed slowly to avoid detection from the web servers.

CHAPTER 4

RESULTS AND DISCUSSIONS

4.1 Introduction

We have completed the assessment for all the 10 government web applications with process mentioned in Figure 3.2. In this chapter, we will present the results of vulnerability assessments, perform analysis and discuss on the results. Each of the web applications is given a pseudonym to keep their information confidential, for example WebApp 1, WebApp 2, WebApp 3 and so on, and some parts of the screenshots provided will be redacted to hide their information.

4.2 Analysis of data-set based on Information Gathered

As mentioned in previous chapter, we need URLs with parameters to discover whether the web application is vulnerable to SQLi or XSS. Hence, we have adopted Spider function in Burp Suite to crawl URLs of each web application with the configurations as shown in Figure 3.6, 3.7 and 3.8 to avoid from being detected.

Table 4.1 shows the details of each web application during the information gathering stage such as number of requests have been made to the server, number of URL crawled, number of URL with parameters crawled, whether the web application is successfully completed being scanned and average of time efforts taken. We can see that WebApp3 has 8,116 the highest number of requests made, 5,820 the second highest requests made to WebApp6 and 4,341 the third highest requests made to WebApp8. The reason we found for these three web applications had the highest requests made was because they were having list of data separated into many pages for users to browse through. And hence, the average time taken for them are the longest, as the more requests were made, the longer the time taken.

Due to some unforeseen circumstances, the Burp Suite stopped scanning on WebApp 4, WebApp 5, WebApp 8 and WebApp 9. For example, power outage， short circuit, overheating etc caused the computer to automatically shut down. A special scenario occurred on WebApp 4, responses from the server were all 503 HTTP status code due to a server maintenance was performed on the web application in the middle of the scanning, and it caused us to stop the work. Therefore, the scanning was not

completed on these 4 web applications and we did not restart the scanning as we could not afford taking the risk again.

Table 4.1: Information Gathered

Target	No. of Requests Made	No. of URL Crawled	No. of URL with Parameters Crawled	Completed Scanning	Average of Time Taken (hours)
WebApp 1	1,609	1,265	170	Yes	80.45
WebApp 2	1,513	1,501	84	Yes	75.65
WebApp 3	8,116	7,984	474	Yes	405.80
WebApp 4	945	940	199	No	47.25
WebApp 5	1,063	1,011	200	No	53.15
WebApp 6	5,820	5,719	124	Yes	291.00
WebApp 7	274	274	24	Yes	13.70
WebApp 8	4,341	4,105	194	No	217.05
WebApp 9	1,147	1,123	42	No	57.35
WebApp 10	1,621	1,589	70	Yes	81.05

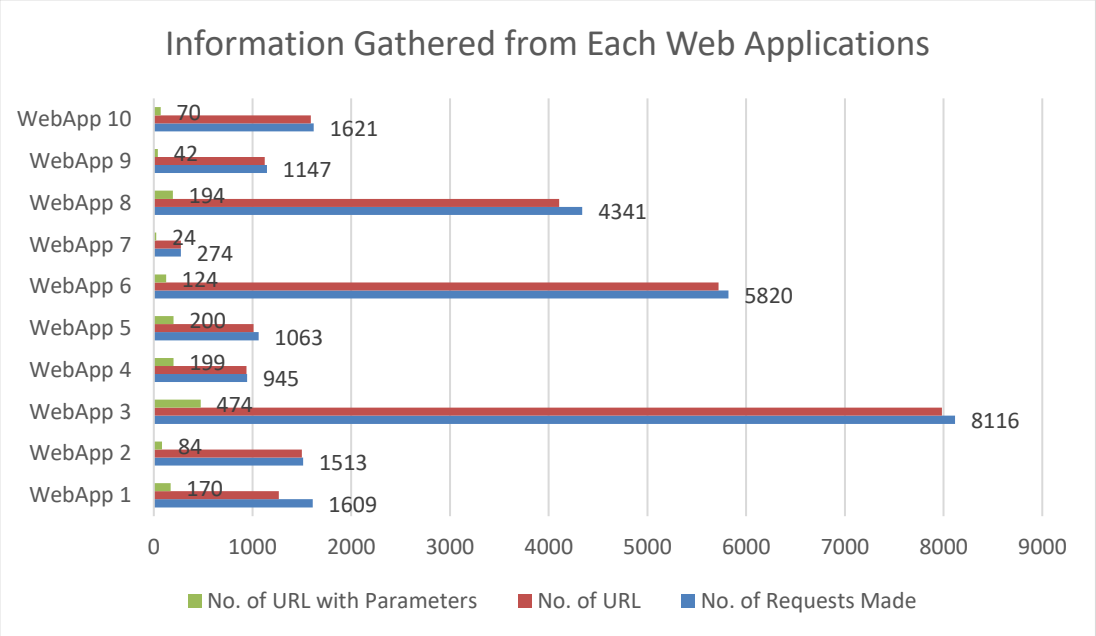

Figure 4.1: Information Gathered from Each Web Applications

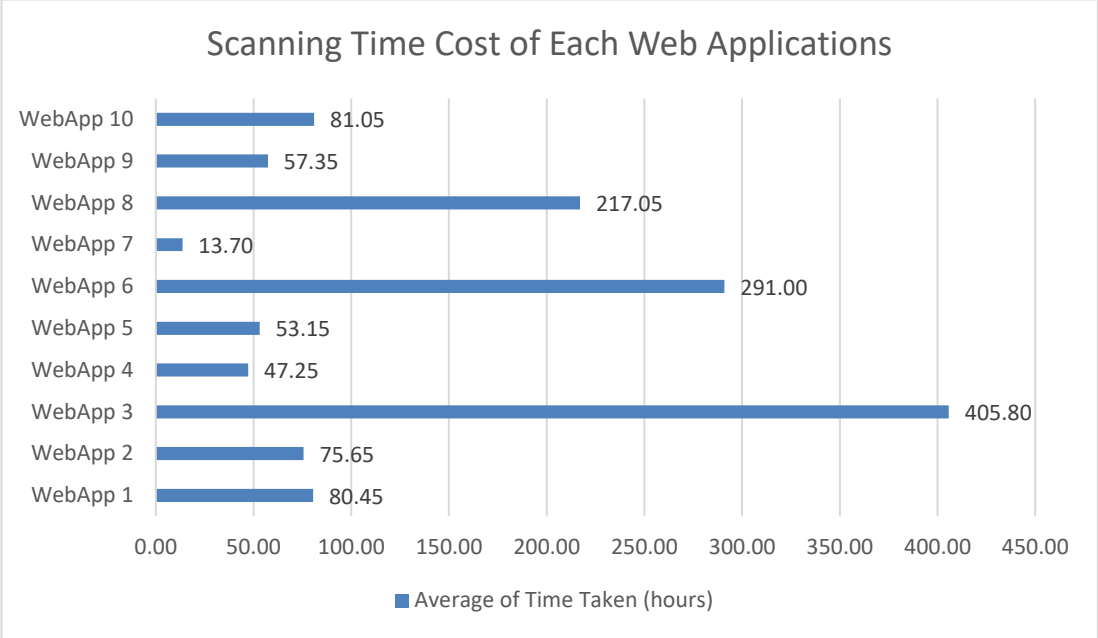

Figure 4.2: Scanning Time Cost of Each Web Applications

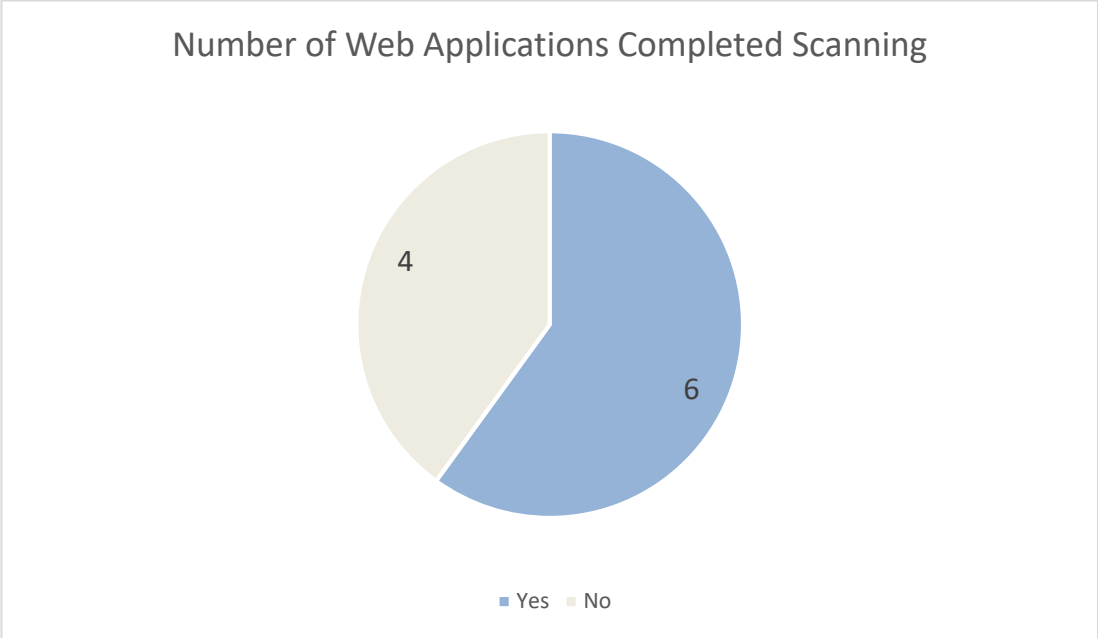

Figure 4.3: Number of Web Applications Completed Scanning

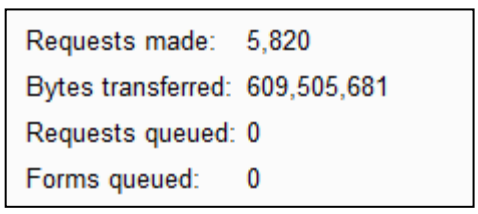

Figure 4.4: Result of WebApp 6

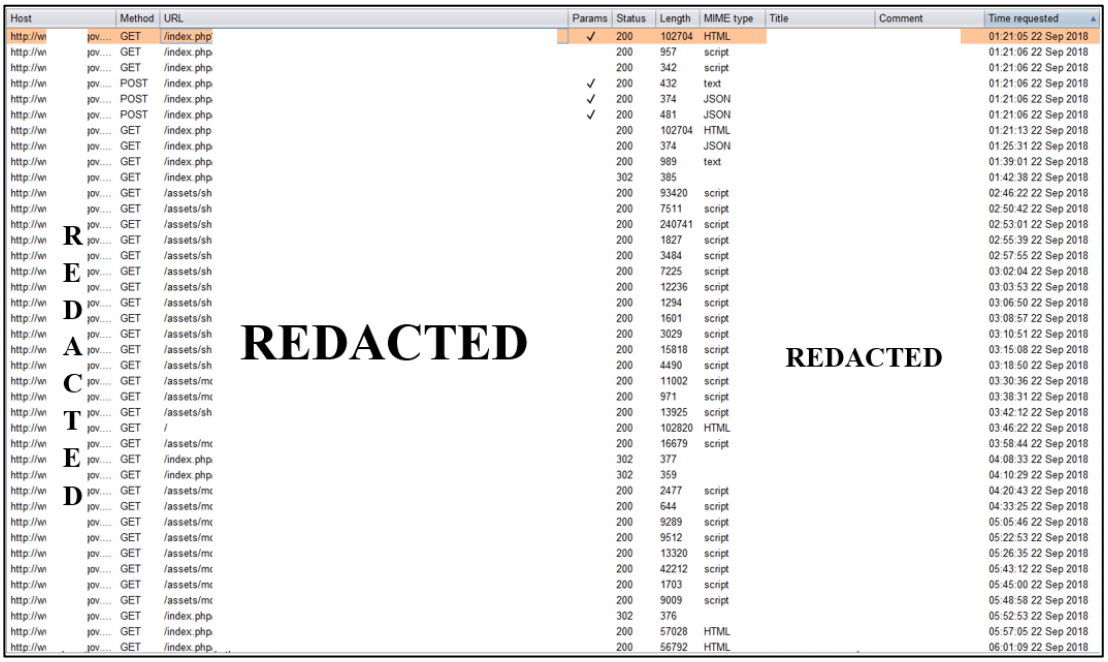

Figure 4.5: Starting of URLs Crawled from WebApp 6

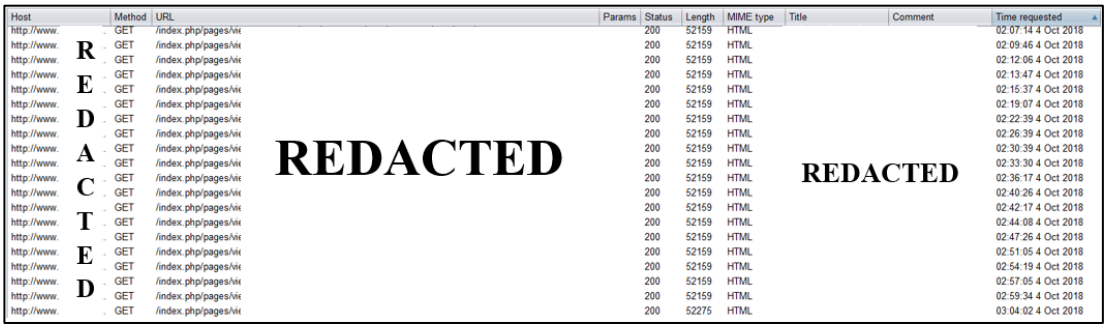

Figure 4.6: Ending of URLs Crawled from WebApp 6

4.3 Analysis of data-set based on SQLi

According to vulnerability assessment flow in Figure 3.2, we proceeded to inject single quote in the URLs with parameters one by one. Among the 10 web applications, WebApp 1 and WebApp 6 were found vulnerable to SQLi. See Figure 4.8, the error message we got from WebApp 6 server, it clearly shows they were using MySQL Server and attackers were able to inject query to retrieve and manipulate sensitive information stored in the database.

The remaining eight web applications were not found vulnerable to SQLi, they might have implemented a least defence to prevent SQLi attack, but this does not mean they are not vulnerable. As we were trying to inject only one payload for the

assessment, the single quote is a common payload to be filtered in the validation and we did not try on input fields where query could be injected as well. They are affirmed not vulnerable to SQLi if and only if a complete penetration testing is performed.

Table 4.2: Web Applications Vulnerable to SQLi

Target	Vulnerable to SQLi
WebApp 1	Yes
WebApp 2	No
WebApp 3	No
WebApp 4	No
WebApp 5	No
WebApp 6	No
WebApp 7	Yes
WebApp 8	No
WebApp 9	No
WebApp 10	No

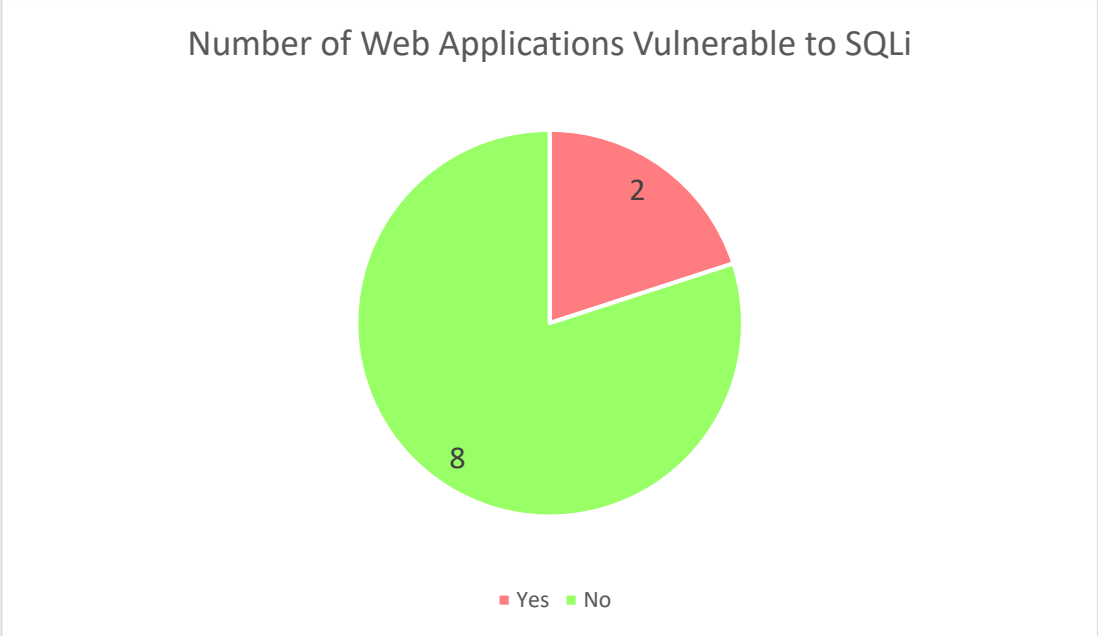

Figure 4.7: Number of Web Applications Vulnerable to SQLi

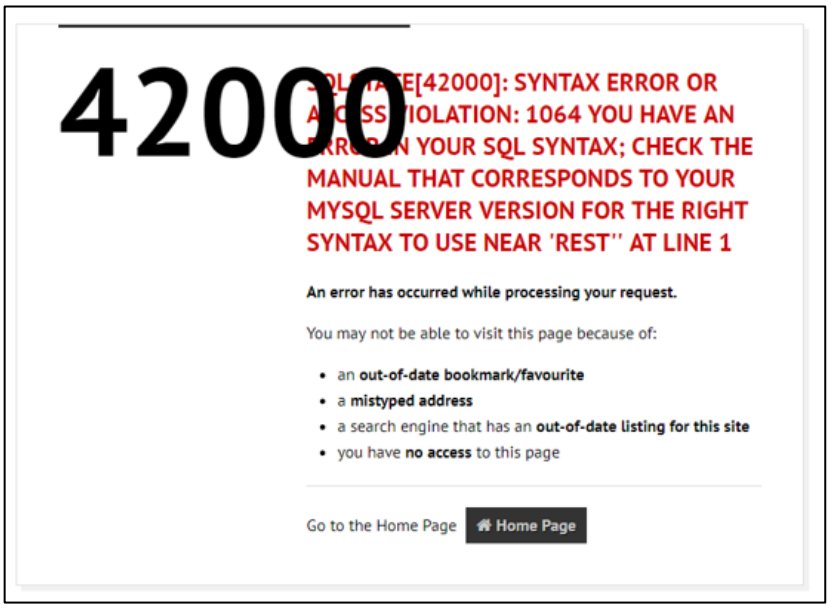

Figure 4.8: Response of WebApp 6 While Discovering SQLi

4.4 Analysis of data-set based on XSS

We did the same method of injection to identify XSS vulnerability on the 10 web applications, but with a JavaScript payload to prompt a message. The result we got was excellent that none of the 10 web applications were identified as vulnerable to XSS. Same as discussed in Chapter 4.2, these web applications were not vulnerable to XSS from our findings but this does not mean they are completely not vulnerable to XSS. With the same explanation, either they might have filtered our JavaScript payload but still accept others or the input fields might accept the payload and response to attackers.

Table 4.3: Web Applications Vulnerable to XSS

Target	Vulnerable to XSS
WebApp 1	No
WebApp 2	No
WebApp 3	No
WebApp 4	No
WebApp 5	No
WebApp 6	No
WebApp 7	No
WebApp 8	No

WebApp 9	No
WebApp 10	No

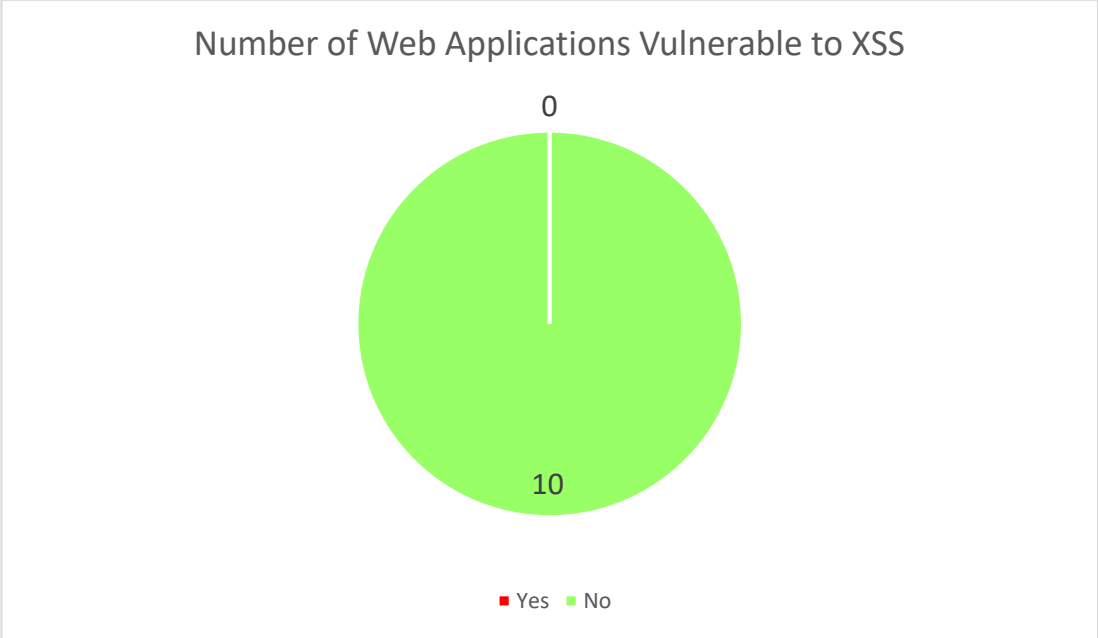

Figure 4.9: Number of Web Applications Vulnerable to XSS

4.5 Conclusion

In this chapter, we have shown and analysed the information gathered from each of the web applications. Two of the ten government web applications were found as vulnerable to SQLi but none of them were found to be vulnerable to XSS. For those not vulnerable web applications, we observed that they have some mechanism to handle suspicious requests. When a malicious URL was sent, the website will either remain and refresh the page or redirect us to the home page. Hence, we believe this is the current security practice they are using.

A total of 1322.45 hours, approximately 56 days were taken for the information gathering process. As there were four web applications that did not complete the scanning, hence we expect the exact number of days needed for the complete scanning shall be more than 56 days. There were several difficulties found during the whole vulnerability assessment process. In the information gathering process, as the time delay method was adopted, therefore the computer used to perform scanning had to be switched on for more than a week. The longer the computer was switched on, the higher possibility the computer will be shut down automatically with loss of data

because of several possible causes. In the discovery of vulnerability process, only one payload could be injected for SQLi or XSS to avoid being detected by the server, hence this would definitely reduce the accuracy of the findings.

CHAPTER 5

IMPACT AND REMEDIATION

5.1 Introduction

Previously we have shown that two web applications are vulnerable to SQLi and none of them vulnerable to XSS, but what if all of them are vulnerable to both attacks?

In this chapter, we will use the Common Vulnerability Scoring System (CVSS) v3.0 Calculator to measure the severity score and level of both attacks. It is a well-known tool used by penetration testers for generating reports to clients. Then, we will discuss about the impact of SQLi and XSS to these web applications by categorising them into the Confidentiality, Integrity and Availability (CIA triad) (Rahul and Pankaj, 2012). Lastly, the development best practices will be provided to prevent both attacks.

Table 5.1: Vulnerability Level Definition
Adapted from "Common Vulnerability Scoring System" (FIRST, 2018)

Severity	Definition
Critical	A vulnerability with high business risk and easily exploitable.
High	A vulnerability with high business risk and medium level of exploitability.
Medium	A vulnerability with medium level of business risk and difficult to exploit.
Low	A vulnerability with low business risk and no direct exploitation possible.

Rating	CVSS Score
None	0.0
Low	0.1 - 3.9
Medium	4.0 - 6.9
High	7.0 - 8.9
Critical	9.0 - 10.0

Figure 5.1: CVSS Rating Scale
Adapted from "Common Vulnerability Scoring System" (FIRST, 2018)

5.2 Web Applications Vulnerable to SQLi

Table 5.2 shows the value we select for each metric regarding to SQLi, and the score is 9.9 which is in the critical severity level.

Table 5.2: CVSS Metric, Values and Comments for SQLi

Metric	Values	Comments
Attack Vector	Network	Attackers are required to connect to the database over a network.
Attack Complexity	Low	A malicious SQL script is enough for an attacker craft the attacks.
Privileges Required	Low	The attacker might need an account with the authority to change user input. It varies from the design of web applications.
User Interaction	None	The attacker could exploit the database without any user interaction.
Scope	Changed	The vulnerable component is the database itself, but it might cause others linked database to be impacted.
Confidentiality	High	Once the database is exploited, the attacker could access to any sensitive data stored in the database.
Integrity	High	Once the database is exploited, the attacker could easily perform modification SQL scripts to modify the data.
Availability	High	Once the database is exploited, the attacker could execute stored procedures provided by the database to bring down the server.
Score		9.9

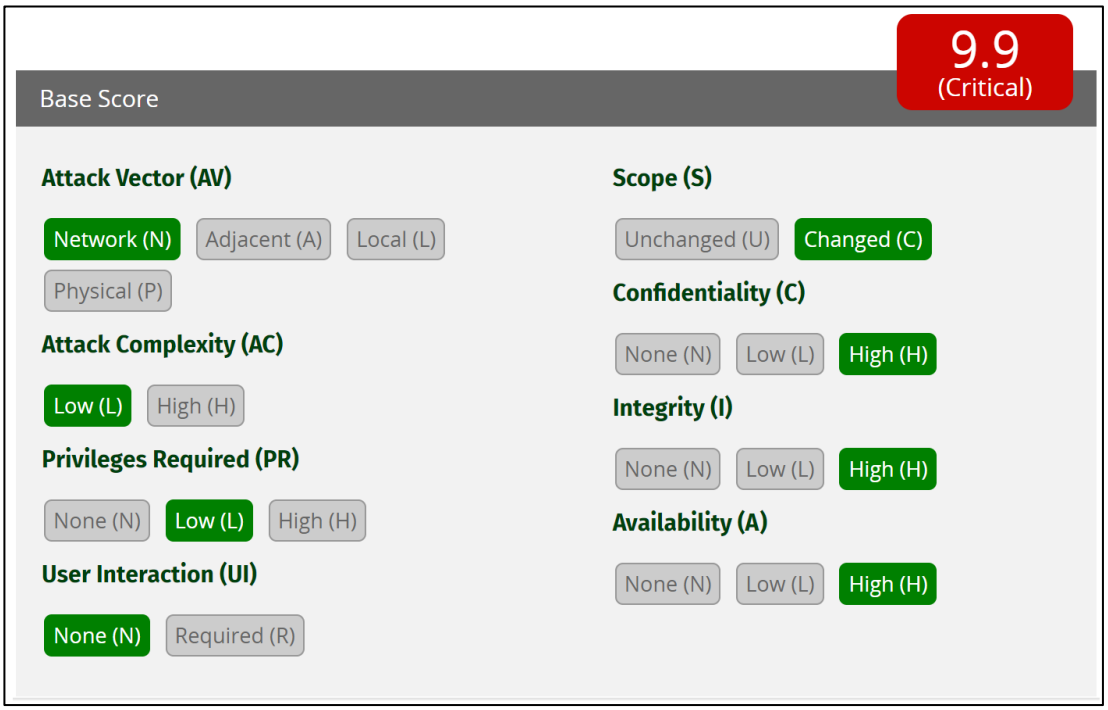

Figure 5.2: CVSS Calculation for SQLi

5.2.1 Impact of SQLi

SQL is a language to communicate with database, it can be used to retrieve, modify and delete data stored in database. In other words, if a web application is vulnerable to SQLi means attackers are able to access to the database.

The impact of SQLi categorised into confidentiality, integrity and availability triad as below:

- **Confidentiality**

  Generally, every data is stored in SQL database, including sensitive information. Among the 10 web applications, most of them do contain sensitive information like citizen name, identity number, address etc and even national secret information. Attackers are able to retrieve any data from the database, despite that the data might be encrypted but attackers still able decrypt them afterwards.

- **Integrity**

  Since data can be retrieved by SQL, then SQL allows attackers to make changes or even delete the data as well.

- **Availability**

  Some database servers allow operating system commands to be executed on the server. Attackers could craft an attack on the internal network which might bring down the service of web application.

5.2.2 Remediation of SQLi

Insufficient input validation is the root cause of SQLi attack. Hence, the following common and effective defensive coding practices for mitigating the SQLi attack.

- **Parameterised Queries**

  Parameterised query is also known as a prepared statement, it is actually a placeholder for storing a value that will be used when query runs. A parameterised query may look like this:

  *SELECT * FROM BankAccounts WHERE Owner = ?*

  where the question mark "?" is the parameter storing owner's name JOEL. If an attacker were to inject *' OR 1=1--* to bypass authentication, the parameter *?* will represent the payload. The parameterised query will not be amended instead it will look for an owner which literally matched the entire value *' OR 1=1--*.

- **Stored Procedures**

  Stored procedure is a SQL query that you could store in the database, then the query can be executed again and again. You could pass parameters to a stored procedure, so that the stored procedure can execute based on the value(s) that is passed. Stored procedure is similar to parameterised query, the main difference is that stored procedure is stored in the database but parameterised query is generated in the source code. In addition, the stored procedure is not safe to use if the SQL query is dynamically generated in the stored procedure. The stored procedure is safe as long as it does not include unsafe dynamic SQL generation.

- **White List Input Validation**

  The prepared statement is not applicable for some parts of the SQL query, for example the names of tables, columns, and sort order indicator. In some features of application, users could perform read or write from certain tables, hence the users are allowed to specify the names of tables or columns but

usually those values should be hardcoded from the source codes yet not from users' input. However, if users are allowed to make decision for the names of tables or columns, then white list input validation is the most appropriate to control the legal/expected tables or columns in the query.

```
CREATE PROCEDURE [dbo].[GetStudentName]
    -- Add the parameters for the stored procedure here
    @ID nvarchar(5)
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here
    SELECT [Name] FROM Student WHERE ID = @ID
END
GO
```

Figure 5.3: Example of Stored Procedure Creation

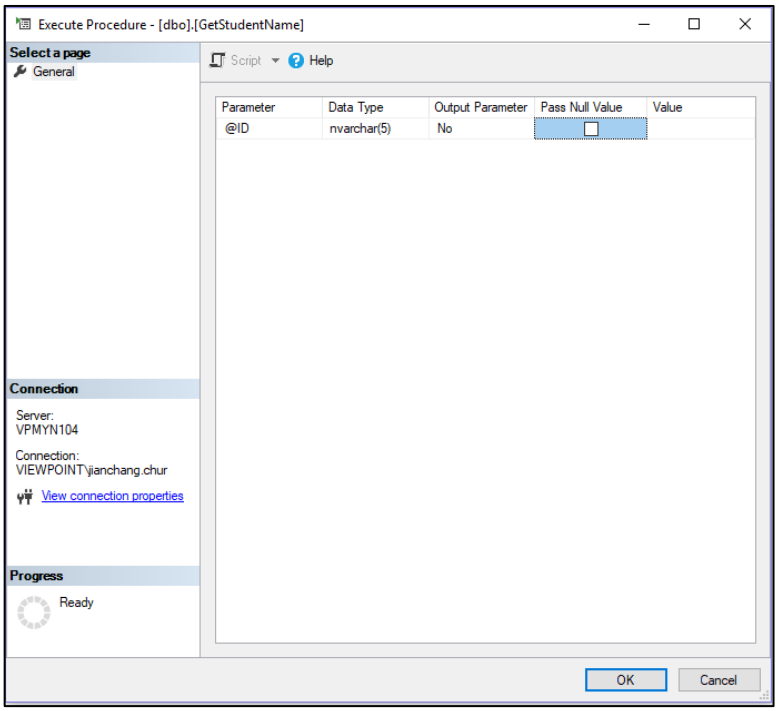

Figure 5.4: Example of Stored Procedure Execution

5.3 Web Applications Vulnerable to XSS

Table 5.3 shows the value we select for each metric regarding to XSS, and the score is 6.1 which is in the medium severity level.

Table 5.3: CVSS Metrix, Values and Comments for XSS

Metric	Values	Comments
Attack Vector	Network	The vulnerability is in the web application and reasonably requires network interaction with the server.
Attack Complexity	Low	A malicious script is enough for an attacker to obtain the valid session token.
Privileges Required	None	An attacker does not need any privileges to craft the attack.
User Interaction	Required	XSS requires the victim to visit the vulnerable component, for example: visiting a malicious URL.
Scope	Changed	Web server is the vulnerable component and victim's browser is the impacted component.
Confidentiality	Low	Information stored in the victim's browser could be read by attacker. The impact will become high only if the attacker could hijack the victim's session.
Integrity	Low	Information stored in the victim's browser could be modified.
Availability	None	The malicious script will only affect victim's browser but not the web application. Victim can easily terminate the attack by closing the browser.
Score		6.1

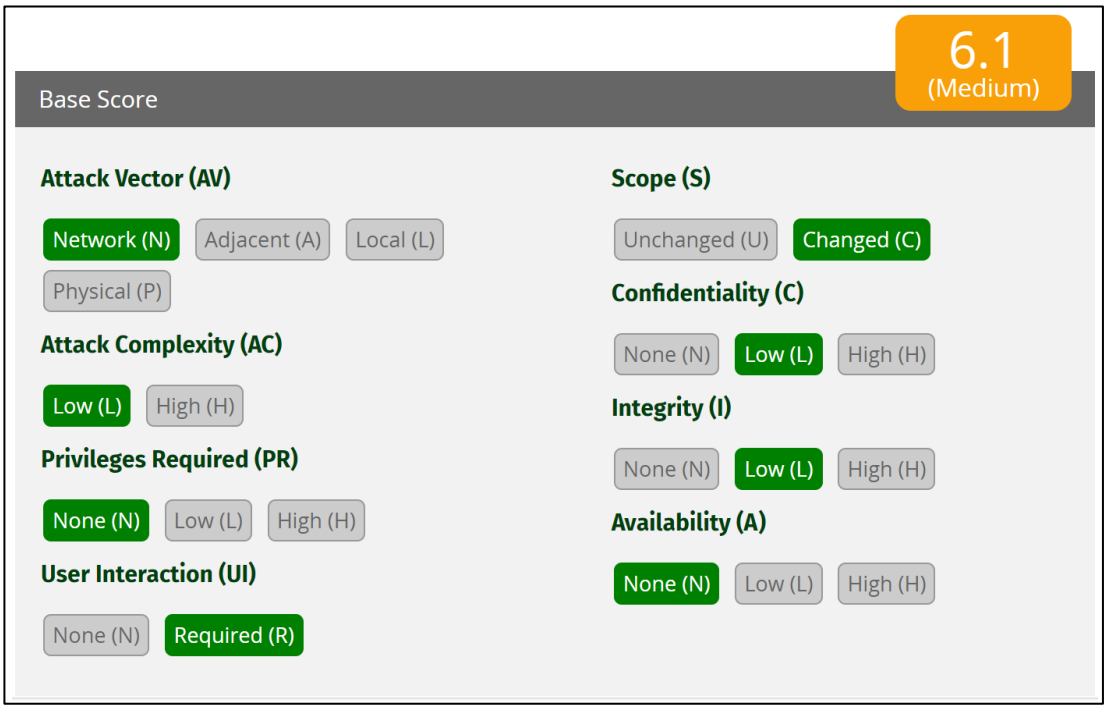

Figure 5.5: CVSS Calculation for XSS

5.3.1 Impact of XSS

The impact of XSS is the same no matter which type of XSS it is, the difference is how the payload being injected to the server. XSS could cause several types of problem for the user that range from an annoyance to user account compromise. The most serious problem is the disclosure of victim's session cookie to an attacker, which is allowing the attacker to hijack the victim's session. Other types of problem include redirecting the victim to other sites, installation of malware programs, or changing presentation of the web page content etc.

For the government web applications, the XSS vulnerability allows attackers to modify news item which might affect the country's economic or causing political issues.

5.3.2 Remediation of XSS

Content Security Policy (CSP) is a layer of security that helps to mitigate types of attack including XSS. It is a HTTP header to a web page and it grants the control what locations a client browser can load resources or what other sites are allowed to interact with the server's site. CSP is suggested to mitigate XSS without the need to modify the web application's source code.

There are two fundamentals restrictions from CSP to support XSS protection:

- **Restriction 1: Inline Scripts Will Not be Executed**

  One of the causes of XSS is the client's browser could not differentiate if the content is sent by the server or injected by an attacker. CSP enforces the code must be separated from the content and requires the code developers intend to execute have to be placed in referenced files externally.

- **Restriction 2: Strings May Not Become Code**

  (Sid, Brandon and Gervase, 2010) *eval()* is a dangerous function which will execute a string of characters as code. In normal circumstances, attackers need *<script> </script>* tags to bypass whatever encoding or filters to execute codes injected. If *eval()* is able to operate on user input without the need of script tags. Therefore, the JavaScript function *eval()* and related functions which generating code from strings are blocked by CSP.

Besides the restrictions from CSP, we could specify the policy by using *Content-Security-Policy* HTTP header like *Content-Security-Policy: <policy>*, the *<policy>* is a string describing the policy using a series of policy directives. Most of the directives control where a resource may be loaded from, for example *default-src*, *img-src*, *media-src*, *script-src* and etc.

The *default-src* is used to whitelisting where the sources may be loaded from, for example:

Content-Security-Policy: default-src 'self'

which the developers allow client browsers to load resources from the web application's origin only, and this exclude subdomains; and

*Content-Security-Policy: default-src 'self' *.example.com*

which resources are loaded from the web application's origin and the *example* domain only; and

*Content-Security-Policy: default-src 'self'; media-src example1.com example2.com; img-src *; script-src examplescript.com*

which by default the resource is permitted from the web application's origin only, but with exceptions below:

1. Media is loaded from example1.com and example2.com only.
2. Image may be loaded from anywhere.
3. Executable script is loaded from examplescript.com only.

The CSP is not shown with the HTTP header, it should be hidden from the header. It is a risk if the CSP of the web application is shown, because the attacker will have the configuration of CSP and craft further attack.

5.4 Principle of Least Privilege

The principle of giving least privilege (PoLP) is one of the most important security policies in IT security, this principle is limiting users' authority to the minimum that they need to perform their work. This is a principle to improve the data protection as well as the application functionality from malicious behaviour. By applying PoLP could help to restrict attacker's access to the web application. For example, the victim has the access to what they need only and so the compromised victim's account to attacker will have access to limited resources. Therefore, applying this principle will reduce the consequences.

5.5 Conclusion

This chapter presented the severity and consequences of SQLi and XSS with the CVSS, and provided comments on every metrics. The SQLi has the critical severity level which score 9.9 out of 10, this has shown why SQLi has been being the Top 1 vulnerability reported by OWASP, because the impact brought to the web application could be ranged from losing confidentiality of data to availability of the application. Besides that, XSS scores 6.1 a medium severity level. The consequences of both attacks might affect the country's economic or even causing political issues. Hence, several remediations are proposed for preventing the attacks and mitigating the risks. There are several research objectives and research questions were developed in this research as a guideline in performing the vulnerability assessment. Therefore, this study has fulfilled all the research objectives and research questions. Below is the summary of the research fulfilment on the research objectives and research questions:

Research Objective 1:

To identity the common security practices of targeted Malaysia government web applications.

Research Question 1:

What is SQLi and XSS vulnerabilities?

Refer to Chapter 2 (2.7, 2.8), a comprehensive review on the SQLi and XSS. There are seven different types of SQLi which is Tautology, Illegal/Logically Incorrect Queries, Union Query, Stored Procedures, Piggy-Backed Queries, Alternate Encodings and Inference. And there are there types of XSS which is Reflected, Stored and DOM-Based. We discussed about the current security practices the web applications are using in Chapter 4.5, they are refreshing the page or redirecting us to the home page when suspicious request is detected.

Research Objective 2:

To identify possible vulnerabilities in targeted Malaysia government web applications.

Research Question 2:

What are the vulnerabilities identified in the targeted Malaysia government web applications?

Refer to Chapter 3, it shows the assessment flow clearly from information gathering to identify SQLi and XSS, the time delay method was adopted to avoid from being detected by the server. Refer to Chapter 4 (4.3 and 4.4), the analysis of data sets has shown what are the vulnerabilities found in the web applications.

Research Objective 3:

To identity possible consequences caused by vulnerabilities and attacks for targeted Malaysia government web applications.

Research Question 3:

What are the possible consequences caused by vulnerabilities and attacks for targeted Malaysia government web applications?

We have discussed about those impacts of SQLi in the categories of confidentiality, integrity and availability. Data stored in SQLi vulnerable web applications is unsafe, as the SQL allows attackers to retrieve, modify and delete the data, more serious is the attackers are possible to bring down the service of web applications. Next, the impacts of XSS was discussed shown to cause several types of problem for the user that range from an annoyance to user account compromise.

Research Objective 4:

To propose threats mitigation frameworks for targeted Malaysia government web applications.

Research Question 4:

What are the mitigation frameworks to prevent both SQLi and XSS vulnerabilities?

The remediations have been proposed in this chapter. Three specific development practices were proposed to prevent SQLi which is parameterised queries, stored procedures and white list input validation. Then the CSP is proposed to prevent XSS, which is a setting in HTTP header to control what locations a client browser to load resources can what other sites are allowed to interact with the server's site. Lastly, the principle of giving least privilege is the general practice to improve the data protection as well as the application functionality by restricting attacker's access to the web application.

REFERENCES

Abdulrahman, A. et al., 2017. Web Application Security Tools Analysis. *2017 IEEE 3rd International Conference on Big Data Security on Cloud*.

Abirami, J., Devakunchari, R. & Valliyammai, C., 2015. A Top Web Security Vulnerability SQL Injection attack - Survey. *2015 Seventh International Conference on Advanced Computing (ICoAC)*.

Acunetix, *Types of XSS: Stored XSS, Reflected XSS and DOM-based XSS*. Available at: https://www.acunetix.com/websitesecurity/xss/.

Ahmad, G., 2017. A Hybrid Method for Detection and Prevention of SQL Injection Attacks. *Computing Conference 2017*.

Ali, S. & Yashar, H., e-Government Services Vulnerability.

Ankit, S., Santosh, C. & Ashish, K., 2016. XSS Vulnerability Assessment and Prevention in Web Application. *2016 2nd International Conference on Next Generation Computing Technologies*.

Arianit, M. et al., 2017. Testing Techniques and Analysis of SQL Injection Attacks. *2017 2nd International Conference on Knowledge Engineering and Applications*.

Aureliano, C. & Diego, T., 2008. alert('A JavaScript agent').

Berners-Lee, T., Masinter, L. & McCahill, M., 1994. Uniform Resource Locators (URL). *The Internet Society*.

Chou, T.-S., 2013. Security Threats on Cloud Computing Vulnerabilities. *International Journal of Computer Science & Information Technology (IJCSIT)*, 5(3).

Dafydd, S. & Marcus, P., 2008. *The Web Application Hackers's Handbook*, Wiley Publishing, Inc

Dave, W. et al., 2018. *SQL Injection Prevention Cheat Sheet*. Available at: https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet.

Dave, W., 2017. *Input Validation Cheat Sheet*. Available at: https://www.owasp.org/index.php/Input_Validation_Cheat_Sheet.

Dave, W. & Jeff, W., 2017. OWASP Top 10 - 2017. *The OWASP Foundation*.

Deepak, P. & Yong, S.H., 2018. *Cybersecurity 2019 Malaysia*. Available at: https://iclg.com/practice-areas/cybersecurity-laws-and-regulations/malaysia.

Delwar, A. et al., 2015. SQLi Vulnerability in Education Sector Websites of Bangladesh. *International Conference on Information Security and Cyber Forensics*.

Diallo, K.A. & Al-Sakib, P.K., 2011. A Survey on Sql Injection: Vulnerabilities, Attacks, and Prevention Techniques. *2011 IEEE 15th International Symposium on Consumer Electronics*.

Fielding, R. et al., 1999. Hypertext Transfer Protocol -- HTTP/1.1. *The Internet Society*.

FIRST, 2018. *Common Vulnerability Scoring System v3.0: Specification Document.* Available at: https://www.first.org/cvss/specification-document.

Ian, M., 2017. *Acunetix Vulnerability Testing Report 2017*. Available at: https://www.acunetix.com/blog/articles/acunetix-vulnerability-testing-report-2017/.

Imran, Y. & Al-Sakib, P.K., 2016. Mitigating Crss-Site Scripting Attacks with a Content Security Policy. *The IEEE Computer Society*.

Inforisk360, 2018. *Advanced Web Hacking & Defense*, Inforisk360.

Inforisk360, 2018. *Hacking and Security Vulnerability Management*, Inforisk360.

Jeff, W., Jim, M. & Neil, M., 2018. *XSS (Cross Site Scripting) Prevention Cheat Sheet*. Available at: https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet.

Jerome, S.H. & Michael, S.D., The Protection of Information in Computer Systems. *IEEE*.

Joel, S., Vincent, L. & Caleb, S., 2011. *Hacking Exposed Web Applications* 3rd ed., The McGraw-Hill Companies.

Joel, W. et al., 2011. A Systematic Analysis of XSS Sanitization in Web Application Frameworks. *Springer-Verlag Berlin Heidelberg*.

Kumar, M., 2011. *Sony Pictures hacked and Database Leaked by LulzSec*. Available at: https://thehackernews.com/2011/06/sony-pictures-hacked-and-database.html.

Lwin, K.S. & Hee Beng, K.T., 2012. Predicting Common Web Application Vulnerabilities from Input Validation and Sanitization Code Patterns. *Association for Computing Machinery (ACM).*

Mingyi, Z., Jens, G. & Peng, L., 2015. An Empirical Study of Web Vulnerability Discovery Ecosystems. *Association for Computing Machinery (ACM)*.

Mohsen, A.S., 2017. Indonesian hackers retaliate flag blunder by defacing M'sian sites. *Indonesian hackers retaliate flag blunder by defacing M'sian sites*. Available at: http://www.thesundaily.my/news/2017/08/22/indonesian-hackers-retaliate-flag-blunder-defacing-msian-sites.

Nuno, A. & Marco, V., 2012. Defending Against Web Application Vulnerabilities. *IEEE Computer Society*.

NetMarketShare, 2018. *Search Engine Market Share*. Available at: https://netmarketshare.com/search-engine-market-share.aspx.

Ossama, B.A. & Mohammad, A.A., 2015. Survey of Web Application Vulnerability Attacks. *2015 4th International Conference on Advanced Computer Science Applications and Technologies*.

Pratik, S., Douglas, K. & Salim, H., 2016. Anomaly Behavior Analysis of Website Vulnerability and Security.

Rahul, J. & Pankaj, S., 2012. A Survey on Web Application Vulnerabilities (SQLIA, XSS) Exploitation and Security Engine for SQL Injection. *2012 International Conference on Communication Systems and Network Technologies*.

Razak, A., 2018. Media Prima hit by ransomware, hackers demand RM26mil in bitcoins, says report. *Media Prima hit by ransomware, hackers demand RM26mil in bitcoins, says report*. Available at: https://www.thestar.com.my/news/nation/2018/11/13/media-prima-hit-by-ransomware-hackers-demand-rm26mil-in-bitcoins-says-report/.

Schneider, F.B., 2003. Least Privilege and More. *The IEEE Computer Society*.

Shashank, G. & Lalitsen, S., 2012. Exploitation of Cross-Site Scripting (XSS) Vulnerability on Real World Web Applications and its Defense. *International Journal of Computer Applications*, 60(14).

Sid, S., Brandon, S. & Gervase, M., 2010. Reining in the Web with Content Security Policy. *International World Wide Web Conference Committee*.

Talebzadeh, P.F. & Ghodrat, S., 2017. Assessments Sqli and Xss vulnerability in Several Organizational Websites of North khorasan in Iran and Offer Solutions to Fix these Vulnerabilities. *2017 3th International Conference on Web Research (ICWR)*, pp.44–47.

The Commissioner of Law Revision Malaysia, 2006. Computer Crimes Act 1997. *Percetakan Nasional Malaysia Berhad*.

The OWASP Foundation, 2018. *Cross-site Scripting (XSS)*. Available at: https://www.owasp.org/index.php/Cross-site_Scripting_(XSS).

The OWASP Foundation, 2016. *SQL Injection*. Available at: https://www.owasp.org/index.php/SQL_Injection.

WhiteHat Security Threat Research Center, 2017. 2017 WhiteHat Security Application Security Statistics Report.

William, H.G.J., Jeremy, V. & Alessandro, O., 2006. A Classification of SQL Injection Attacks and Countermeasures. *IEEE*.

Yuma, M. & Vitaly, K., 2015. Evaluation of Web Vulnerability Scanners. *The 8th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications*.