|  |
| --- |
|  |
| Capstone Experience IST 894  Carl Laneave |
| Lab 5 Report |

Table of Contents

[1.0 Introduction – Cross-Site Scripting (XSS) 3](#_Toc147687243)

[1.1 Lab Results – Cross-Site Scripting (XSS) 4](#_Toc147687244)

[2.0 Introduction – File Inclusion and Cross-Site Request Forgery (CSRF) 14](#_Toc147687245)

[2.1 Lab Results – File Inclusion and Cross-Site Request Forgery (CSRF) 15](#_Toc147687246)

[3.0 Introduction – HTML & SQL Injections 24](#_Toc147687247)

[3.1 Lab Results – HTML & SQL Injections 25](#_Toc147687248)

[5.0 References: 34](#_Toc147687249)

[6.0 Activity Log 35](#_Toc147687250)

# Introduction – Cross-Site Scripting (XSS)

During the execution of this lab, an evaluation was done to test the ability of cross site scripting. XSS is an attack using the client side by injecting malicious code. These actions occur when malicious code is pushed into web applications in which the ability to leverage data theft and user information from the site. Technologies such as cookie session manipulation as well as back door programs such as trojans are used in the overall site XSS attack.

When evaluating XSS, it’s important to understand the different types of attacks that can occur. This includes reflected, stored, and DOM-Based attacks. Each type of attack uses open principles on the site that allow the injection of code. With reflected, information can be sent directly through HTML forms that allow for manipulation of the requests going to the server from the client side of the house. With the stored attacks, attackers can submit data in a stored fashion that allows them to put malicious links and trojans directly on the front end for other clients to ingest through XSS. Lastly, the DOM method uses technology to find DOM method alerts and calls such as GET/POST that can be manipulated to take code into the calls and continuing malicious attacks from the client into the host/server.

# 1.1 Lab Results – Cross-Site Scripting (XSS)

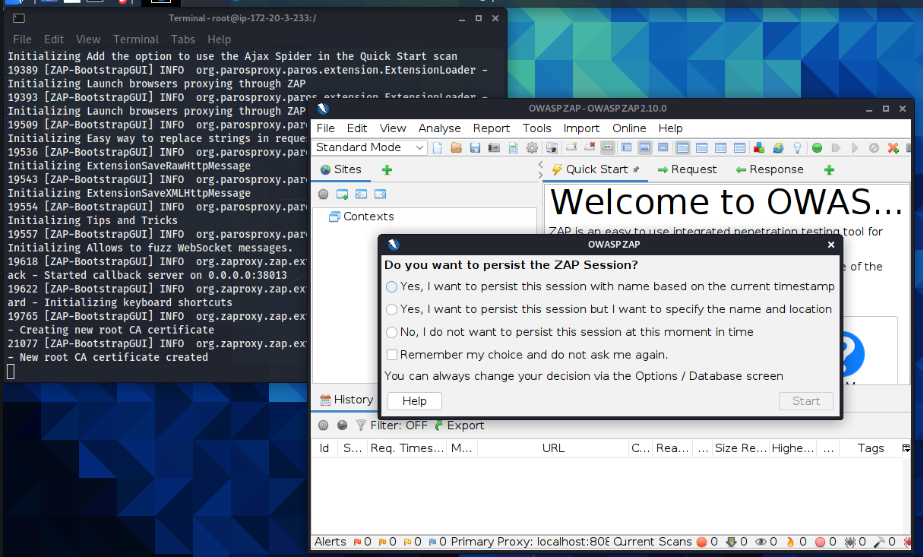


Figure 1.0 – Start OWASP ZAP application.

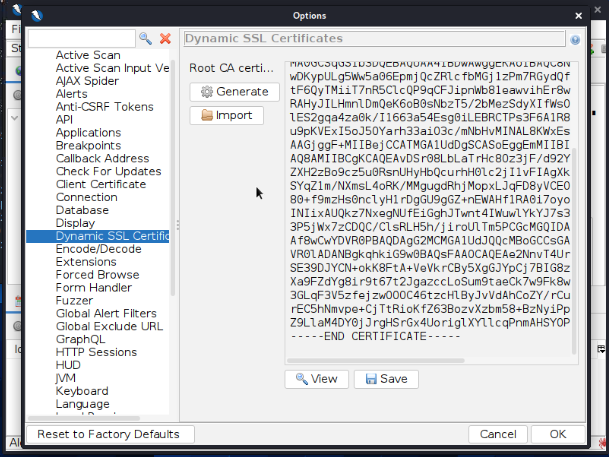


Figure 1.1 – Save the SSL certificate from ZAP for the root CA Certificate.

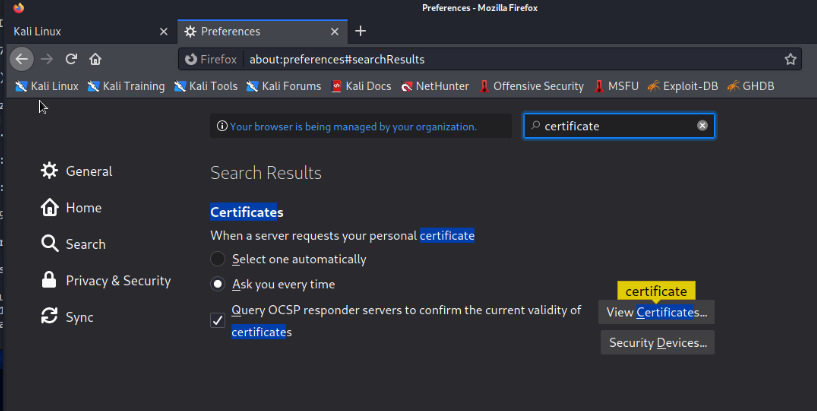


Figure 1.2 – View the certificates in the browser.

A screenshot of a computer

Description automatically generated

Figure 1.3 – Upload the downloaded certificate to the browser.

A screenshot of a computer

Description automatically generated

Figure 1.4 – Use the localhost as the proxy for both FTP and HTTPS

A screen shot of a computer

Description automatically generated

Figure 1.5 – Remove the certificate from the local machine.

A screenshot of a computer

Description automatically generated

Figure 1.6 – Open the ZAP Hud with URL bwapp.

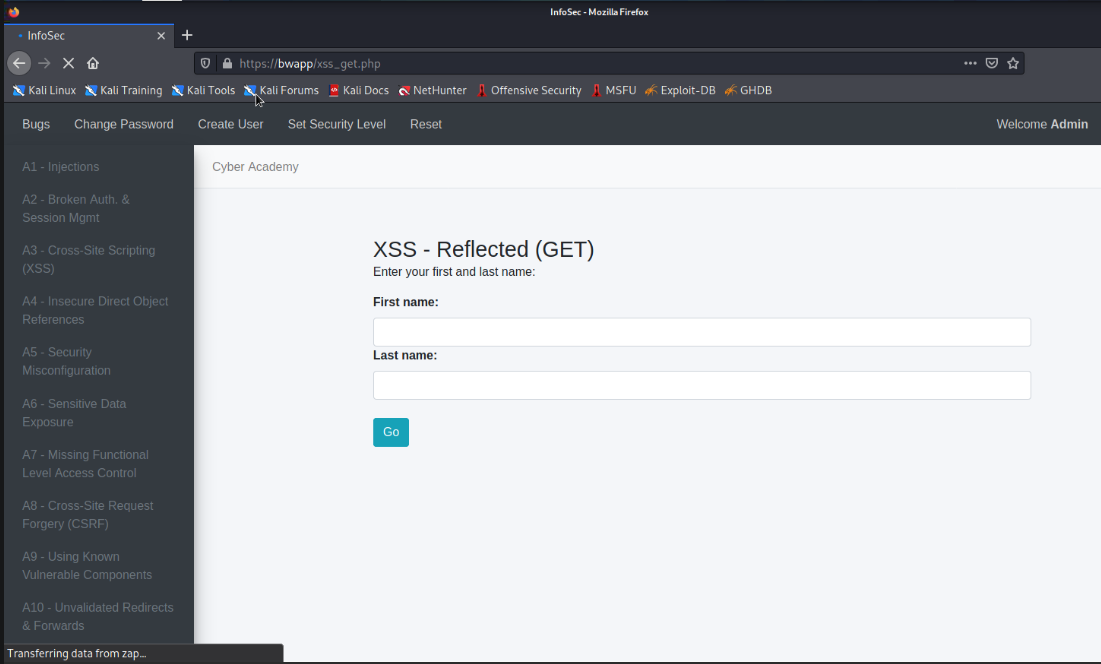


Figure 1.7 – Start the reflected attack in the bwapp.

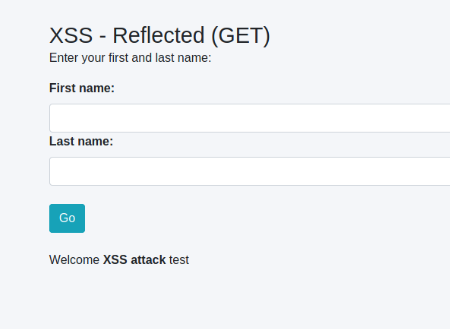


Figure 1.8 – Passing in markup language of bold reads in the welcome message.

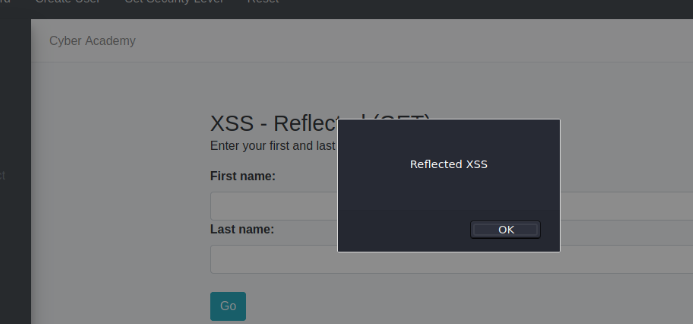


Figure 1.9 – Passing in script alert shows an alert.

A screenshot of a computer

Description automatically generated

Figure 1.10 - Create a URL link on a stored blog site using an XSS stored attack.

A screenshot of a computer

Description automatically generated

Figure 1.11 – Using a DOM XSS, we can see the key values in the URL slug.

A computer screen shot of a code

Description automatically generated

Figure 1.12 – As shown, we can see the GET request that is taking the innerHTML Function for the sink.

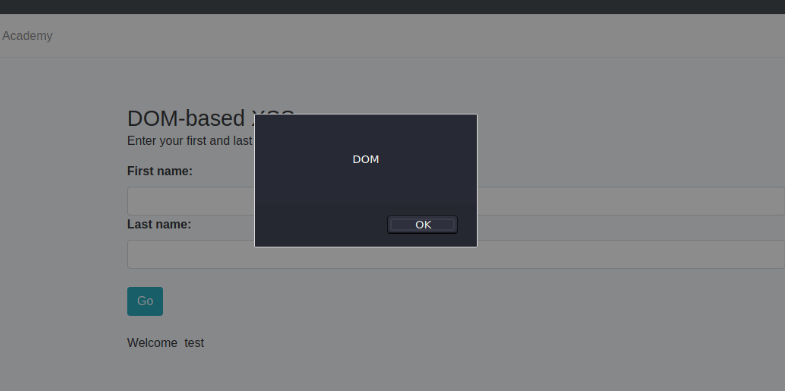


Figure 1.13 – Modify the break point to allow the GET to be modified on the DOM to pass in an alert.

# 2.0 Introduction – File Inclusion and Cross-Site Request Forgery (CSRF)

In this attack, the focus is on using cross site request forgeries to be able to upload malicious files to a website. In File inclusion and CSRF attacks, attackers can use methods on the site, such as php calls, to manipulate and upload malicious files for execution. When a site is exposed to these types of attacks, attackers can create local malicious php files and store them on their local machines. By using Python HTTP requests, they can act as their own server and upload the file to the site. Once the file is uploaded it will execute the same as the previous php file, but at this time will do so with the malicious code. This allows attackers to gain access to servers from the client as well as use it as an attack vector against other website visitors.

Through data sanitization as well as restrictions on permissions to client-side calls, attackers would be limited in the ability to upload their own files as well as see potentially vulnerable functions through call evaluations in the network.

# 2.1 Lab Results – File Inclusion and Cross-Site Request Forgery (CSRF)

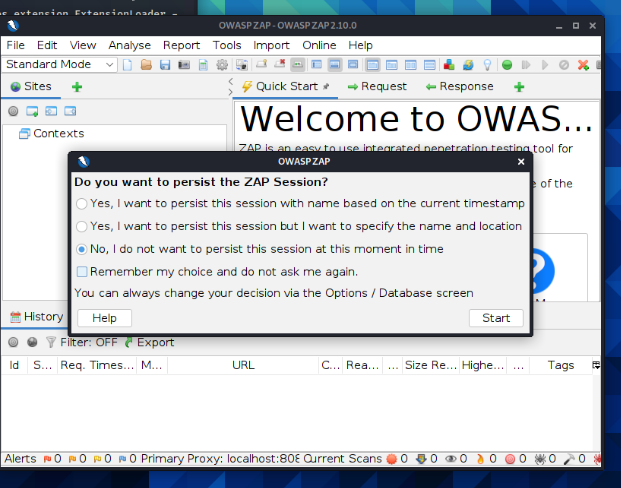


Figure 2.0 – Start OWASP Zap application.

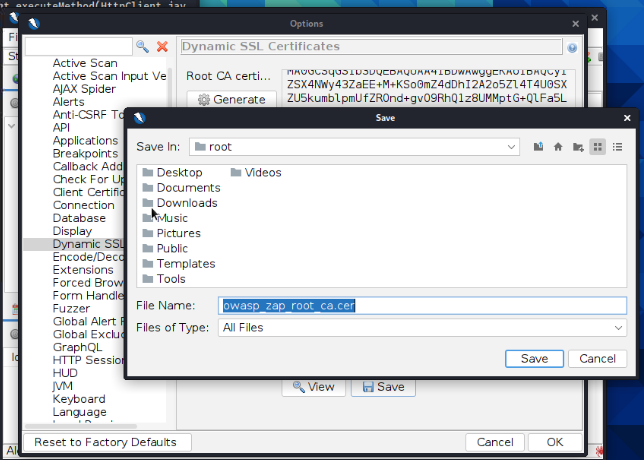


Figure 2.1 – Download the dynamic SSL Certification

A screenshot of a computer

Description automatically generated

Figure 2.2 – Important SSL certification into the browser for our session.

A screenshot of a computer

Description automatically generated

Figure 2.3 – Setup a proxy to use the localhost for resolving requests.

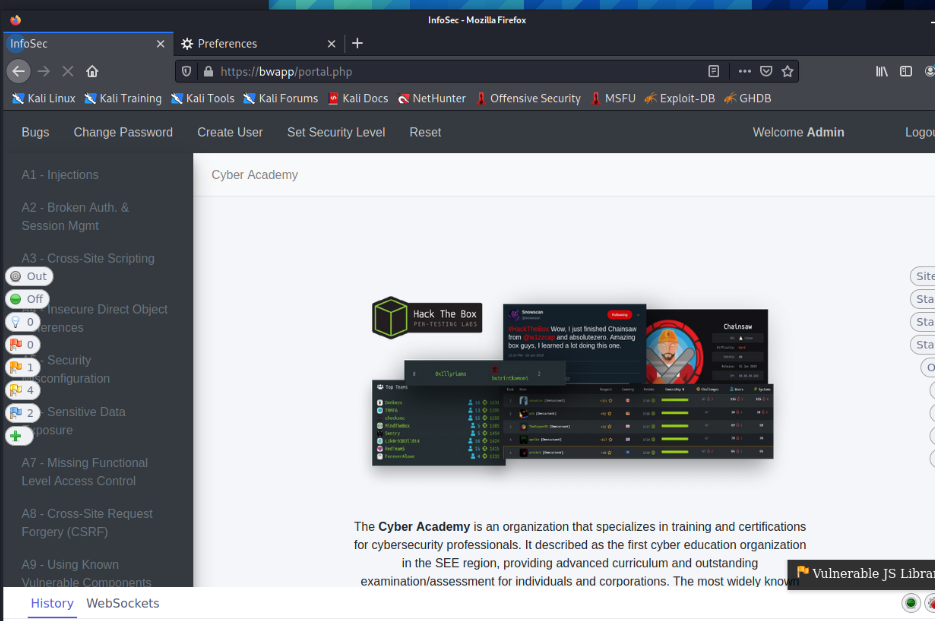


Figure 2.4 - Log into bwapp on the browser.

A screenshot of a computer

Description automatically generated

Figure 2.5 – Load the remote and local file inclusion module.

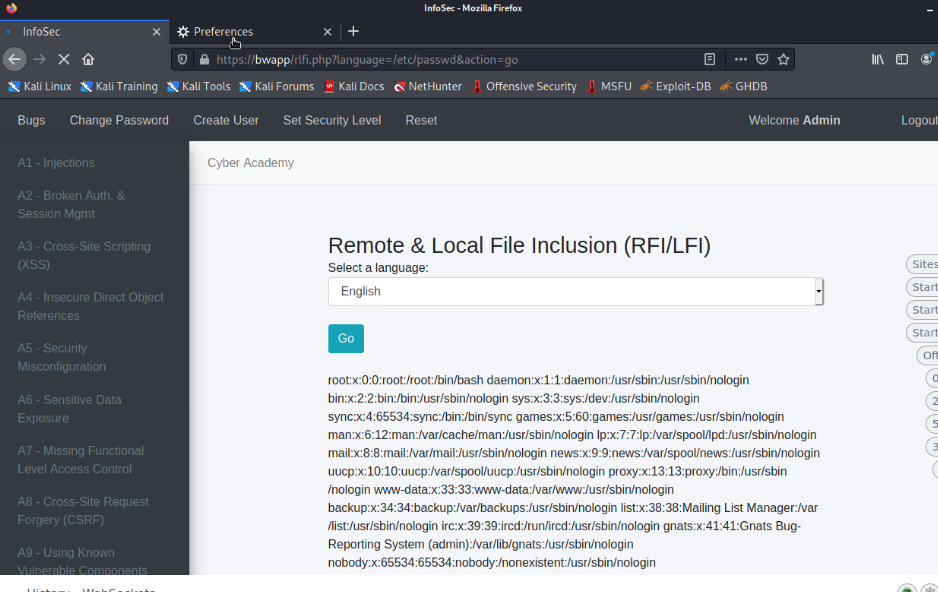


Figure 2.6 – Change the language to call /etc/passwd.

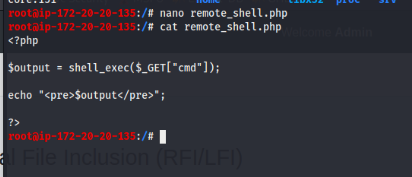


Figure 2.7 – Create a php file to execute a remote attack.

A screen shot of a computer

Description automatically generated

Figure 2.8 - Execute a local http server in python and upload your remote\_shell.php file to the site.

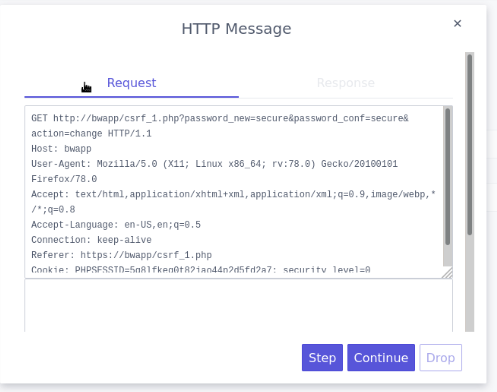


Figure 2.9 – Change your password using the break functionality to see the GET request.

A screenshot of a computer

Description automatically generated

Figure 2.10 – Input the csrf php password get request into the XSS stored blog, entry shows up.

A screenshot of a computer

Description automatically generated

Figure 2.11 – Bob’s password has been changed and he doesn’t even know it due to cookies being constant on the call.

A screenshot of a computer

Description automatically generated

Figure 2.12 – Bobs password no longer works as test and was changed to ‘changed’.

# 3.0 Introduction – HTML & SQL Injections

In this attack, a SQL Database is not cleansing or restricting the type of inputs that can go into its search. By not doing this, attackers can insert their own SQL commands to see database tables, upload their own information, or cause general disruption through deletions and manipulation of the tables. To test if a form is susceptible to potential SQL injections, attackers will input simple strings such as ‘to see the results. In the case of an exposed SQL database, errors will be returned showing that the query could not be completed as required. ‘is a way to mark the end of a string input and the start of a SQL query. Once a database is shown as exposed, attackers can pull all the data using simple commands such as 1=1 as this will always return true and all values.

SQL injections, though very uncommon now, were at one point the most common attack vector used throughout cyber-attacks.

# 3.1 Lab Results – HTML & SQL Injections

A screenshot of a computer

Description automatically generated

Figure 3.0 – Start OWASP ZAP application.

A screenshot of a computer

Description automatically generated

Figure 3.1 – Save the SSL certificate from ZAP for the root CA Certificate.

A screenshot of a computer

Description automatically generated

Figure 3.2 – View the certificates in the browser.

A screenshot of a computer

Description automatically generated

Figure 3.3 – Upload the downloaded certificate to the browser.

A screenshot of a computer

Description automatically generated

Figure 3.4 – Use the localhost as the proxy for both FTP and HTTPS.

A screen shot of a computer

Description automatically generated

Figure 3.5 – Remove the certificate from the local machine.

A screenshot of a computer

Description automatically generated

Figure 3.5 - Log into bwapp on the browser.

A screenshot of a computer

Description automatically generated

Figure 3.6 – Input html tags and hit go, the code is picked up showing it is vulnerable to reflected html injections.

A screenshot of a computer

Description automatically generated

Figure 3.7 – Inject an URL link using an attack on a stored target.

A screenshot of a computer error

Description automatically generated

Figure 3.8 – Passing in ‘ as an input shows that it allows for SQL syntax to be passed in.

A screenshot of a computer

Description automatically generated

Figure 3.9 – Passing in a query of 1=1 always returns true, showing the entire database entries.

A black background with white text and a check mark

Description automatically generated

Figure 3.10 – Lab is completed

# 5.0 References:

[1] KirstenS. (2020). Cross Site Scripting (XSS) | OWASP. Owasp.org; Owasp. https://owasp.org/www-community/attacks/xss/

[2] OWASP. (2013). SQL Injection. OWASP. <https://owasp.org/www-> community/attacks/SQL\_Injection

[3] PortSwigger. (2019). What is reflected XSS (cross-site scripting)? Tutorial & Examples. Portswigger.net. https://portswigger.net/web-security/cross-site-scripting/reflected

[4] 3 Types of Cross-Site Scripting (XSS) Attacks. (2023, May 11). Trend Micro. https://www.trendmicro.com/en\_za/devops/23/e/cross-site-scripting-xss-attacks.html

‌

‌

‌

# 6.0 Activity Log

| **Member Name** | **Task Date** | **Task Details** |
| --- | --- | --- |
| Carl Laneave | 10/08/2023 | Created Template, executed all labs, took screenshots, and completed report |
|  |  |  |
|  |  |  |