OWASP Cross-Site Scripting Lab Exercise

The development of this document is/was funded by three grants from the DOD Grant Number H98230-19-1-0301

1 Overview

This Labtainer exercise explores Cross-site scripting (XSS). XSS is a type of vulnerability commonly found in web applications. These vulnerabilities come in two main types, reflected and stored. A reflective typically is ran once and then executed, while in contrast a stored XSS is saved and can be called and ran at a later time. XSS makes it possible for attackers to inject malicious code (e.g. JavaScript programs) into victim's web browser.

2 Lab Environment

This lab runs in the Labtainer framework, available at http://my.nps.edu/web/c3o/labtainers. That site includes links to a pre-built virtual machine that has Labtainers installed, however Labtainers can be run on any Linux host that supports Docker containers.

From your labtainer-student directory start the lab using:

```
labtainer web-xss
```

On most Linux systems, these are links that you can right click on and select "Open Link". **If you chose to edit the lab report on a different system, you are responsible for copying the completed report back to the displayed path on your Linux system before using "stoplab" to stop the lab for the last time.**

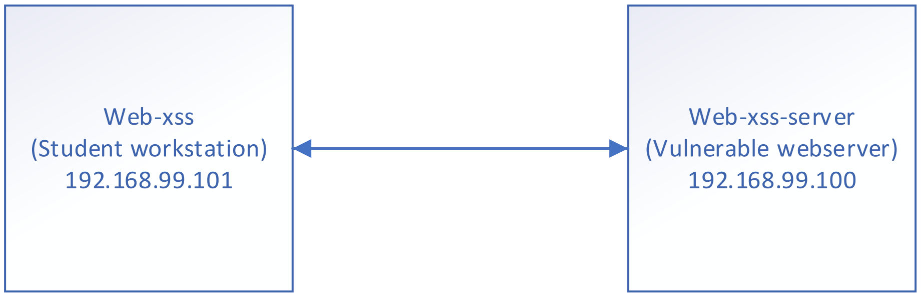
The resulting virtual terminal is connected to the student workstation; you will have OWASP ZAP and Firefox located on this workstation.

There are several accounts that are configured

Username	Password	Systems
admin@juice.org	admin123	Admin for web login
jim@juice.org	ncc-1701	web login
mc.safesearch@juice.org	Mr. N00dles	web login
Ubuntu	ubuntu	Student workstation
Ubuntu	ubuntu	vulnerable server
Ubuntu	ubuntu	Attacker workstation

3 Network Configuration

The student workstation (web-xss) is configured to have IP address `192.168.99.101` while the vulnerable webserver (web-xss-server) is `192.168.99.100;`



4 Lab Tasks

It is assumed that the student has received instruction or independent study on the basic operation of web operations

4.1 Verify connectivity between student workstation and web server

A simple ping from the student workstation system will be sufficient.

```
 ping 192.168.99.100
```

Note: to stop the ping use CTRL + C

4.2 Open Firefox and browse to the web server

At a terminal on the student workstation type:

```
firefox &
```

this will load Firefox, and type in the IP of the web server:

```
http://192.168.99.100
```

Record in Item #1 of your report why firefox might have been chosen to be the web browser used.

4.3 Open & Set up OWASP ZAP

At the terminal of the student workstation, type:

```
owasp-zap &
```

Note: if Firefox is running at the terminal and the "&" was not included then Firefox is not running in the background. Close Firefox and reopen using "Firefox &" at the terminal

OWASP ZAP Application should be open and it should be prompting the user for input.

- OWASP ZAP user input: select "yes, I want to persist this session with the name based on the current timestamp" then click start. This will open ZAP application.
- If you are prompted to "Manage Add-on" click close

4.4 Configure Firefox to use OWASP-ZAP as a Proxy

The objective of this task is to set up OWASP ZAP to be function and to allow the capture of traffic from the web-xss student workstation. Within the preference section of Firefox configure the following steps:

- In the student workstations Firefox, open "Preferences"
- In the find window type "proxy"

- In Network Proxy Setting, select “Settings”
- Select “Manual proxy configuration”
- In the HTTP Proxy section: use “127.0.0.1” and Port “8080”
- Also select “Use this proxy server for all protocols”
- Click “ok” to accept the settings

The above setting ensures that Firefox will use OWASP Zap as the proxy. Perform the following steps to ensure the Firefox is connecting and using ZAP as a proxy

- Refresh the webpage “192.168.99.100”
- A security warning stating “Your connection is not secure” will be displayed.
- This warning message must be accepted. To do that click on “Advanced”
- It will display the SSL certificate and should show a “SEC_ERROR_UNKOWN ISSUE” it is ok to use this cert, click “Add Exception”
- A confirmation window will pop up, confirm the exception by clicking the “Confirm Security Exception”

Record in Item #2 of your report why set up a proxy.

4.5 Verification and Site Navigation

The objective of this task is for the user to become familiar with the website and for the proxy tool to search out details on each visited page. According to OWASP Web Security Testing Guidelines (WSTG), one of the first steps in checking a web site or application is passive testing, also known as site exploration. During passive testing, the tester tries to understand the site or application's logic and will explore the site or application as a user. This process is used to gather information on the target site or application. A common method of passive testing is setting up a HTTP proxy that is used to observe all the HTTP requests and responses. At the end of passive testing, the tester should understand all the access points (gates) of the site or application.

Once the proxy is set up, the tester should perform some basic navigation. Make sure that in OWASP Zap, in the bottom section as the tester navigate the different pages OWASP Zap is recording the pages visited. As the tester navigates throughout the site, they need to pay attention to any data input boxes that they see.

There are two main types of data input boxes, first is a data input that allows for input and then executes without saving the input (reflected XSS). Second, is a data input filed that allows for input to be saved and executed at a later time (stored XSS).

Record in Item #3 of your report does OWASP ZAP check for vulnerabilities in each page you visit?

4.6 Site Vulnerable Check

In OWASP ZAP in the bottom panel, there are 4 tabs: History, Search, Alerts and Output. As the tester navigated to the different pages, they would be scanned by OWASP ZAP. If XSS is possible on the visited page, it would be flagged under “Alerts”

In the bottom of OWASP ZAP, navigate to

```
Alerts
```

In the alerts section, the tester should verify is OWASP ZAP has found any XSS vulnerabilities on the given page. If a vulnerability does exist, then through inspection of each of the data user inputs on the selected page will need to be verify which inputs are vulnerable to XSS.

NOTE: if the tester has not navigated to all of the pages, then pages that have not been navigated to will not show up in this section. Make sure that the tester has thoroughly explored the website before doing this step.

4.7 Verifying Data Input for Reflected XSS

The objective of this section is to test for and find reflective XSS vulnerability locations. According to OWASP WSTG, reflected XSS are the most frequent type of XSS attacks found in the wild. Reflected XSS attacks are also known as non-persistent XSS attacks and, since the attack payload is delivered and executed via a single request and response, they are also referred to as first-order or type 1 XSS. When a web application is vulnerable to this type of attack, it will pass unvalidated input sent through requests back to the client. Commonly the attacker's code is written in the JavaScript language, but other scripting languages are also used. In this section the tester will be looking for exploits using JavaScript, Malformed IMG tags, or possible non-JavaScript on error alert vulnerabilities. It's important to note that often web browsers may automatically remove the <script> ability to run, that is why the tester will be checking using two additional methods. This section will have the tester user three different strings of code to verify if a user input is vulnerable.

Testing for Reflected XSS using OWASP WSTG-INPVAL-001 guide and using Black-Box Testing method, as the tester has no previous knowledge of the system. Black-Box Testing consists of three steps. The steps are 1) detect input vectors; 2) analyze input vectors; and 3) check impact. In section 4.6 the input vectors were detected. In this section the tester will be analyzing the input vectors for reflected XSS.

To analyze input vectors, it is important to know that not all user inputs are vulnerable. The tester will need find which inputs are vulnerable to XSS attacks. One issue when website designer creates a website buttons, that button may perform actions that include automatic filtering. For example, some websites can sense the "<script>" and then ignore that code. Websites that follow best practices will automatic filter code such as the <script> tag and will not allow it to run. There are three test scripts listed below in varying formats that each should be tested against data user input. One or more of the scripts may run in one input while others may not, that is why it is important to use all three examples to verify if the input is exploitable. The basic example scripts used for testing are below.

It is also important to note that this lab has certain filtering outside the scope of control. Email addresses and zip codes will not be addressed in this section but will be reviewed in stored XSS section.

Example 1 - JavaScript

```
<script>alert(1);</script>
```

Example 2 – Malformed IMG tags

```
<IMG """><SCRIPT>alert("XSS")</SCRIPT>"\>
```

Example 3 – No JavaScript on error alert

```
<IMG SRC=/ onerror="alert(String.fromCharCode(88,83,83))"></img>
```

Record in Item #4 of your report why there are different types of reflected xss attacks?

Record in Item #5 of your report the locations and input areas that you found to be vulnerable.

4.8 Persisted XSS

The objective of this section is to test for and find stored also known as persisted XSS vulnerability. According to OWASP WSTG, persisted XSS occurs when a web application gathers input from a user which might be malicious, and then stores that input in a data store for later use. The input that is stored is not correctly filtered. As a consequence, the malicious data will appear to be part of the web site and run within the user's browser under the privileges of the web application. Since this vulnerability typically involves at least two requests to the application, this may also be called second-order XSS.

Testing for Stored Cross Site Scripting using OWASP WSTG-INPVAL-002, similar to testing for reflected XSS vulnerabilities using Black-Box Testing method. OWASP WSTG points out several possible locations that can be used to exploit stored XSS vulnerabilities. Those areas include user/profile pages, shopping carts, wish lists, and possibility forum or message boards. In this section the learner will go through an modify a request in transits (example 1), and through a file upload via the complaint form.

Example 1 – POST request modification
Follow the steps listed below to capture and modify a POST request when creating a new user.

- On the top left side of the screen, click on "Account" a "login" submenu pops up, click on the "login" option.
- The login screen will allow the learner to log in, if no account exists then a new account can e created. In this example, a new customer account will be created. Click "Not yet a customer?"
- This should take the learner to a "User Registration" page. Fill out the following information
  - Email: test@noemail.com
  - Password: Password#1
  - Repeat Password: Password#1
  - Security Question: Name of your favorite Pet?
  - Answer: Dogs
  - Do not hit "REGISTER"
- Navigate to OWASP Zap, and turn on the page breaks
- Navigate back to Firefox, and click "Register" while nothing should happen as OWASP Zap should have captured and paused the submission
- Navigate to OWASP Zap, it should have a "Break" tab open and in the body of the message, the learner should be able to see the information entered when creating a new user. Modify the section that has the email address. Modify the request as instructed below
  - Replace: "test@noemail.com"
  - With: "<script>alert(\"XSS\")</script>"
- Once the modification have been completed, click the "Submit and continue to the next break point" tab to allow the submission of the registration page.
- Anyone that view the administration page will run the XXS exploit. Though it may not always notify the user. Log in with the admin account and view the following page:
  - https://192.168.99.100:3000/#/administration

Example 2 – File upload
Follow the steps listed below to upload a file to check if Persisted XSS exploits are working on file uploads.

- Make sure to be logged in as a user

- On the right side click on the three lines, select "Complaint"
- In the complaint form write a message, and in the invoice section, select file upload and navigate to the "ubuntu\testfile" directory. The first complaint will be an upload of the file "testfile.pdf" do not click submit yet.
- Navigate to OWASP Zap, and turn on the page breaks
- Navigate back to Firefox, and click "Submit" while nothing should happen as OWASP Zap should have captured and paused the submission
- Navigate to OWASP Zap, it should have a "Break" tab open and in the body of the message you should see the code that it is trying to execute.
- click the "Submit and continue to the next break point" tab to allow the submission of the complaint page.
- Repeat the steps except replace the testfile from "testfile.pdf" and use the "testfile.xml"

Record in Item #6 of your report discuss why there are multiple examples and different types of stored xss scripts.

Record in Item #7 of your report the locations and input areas that you found to be vulnerable.

4.9 Persistent XSS without Front-End Access

The objective of this section is to perform a persistent XSS exploit without accessing the front end of the web server using curl. Follow the proceeding steps to identify product and try to exploit the server using a PUT request. The goal of this task is to see if the learner can modify or updating products.

- Search for a product, the user should see that the URL doesn't change between the different product. The user can view all product detail by navigating to the following URL:

  `https://192.168.99.100:3000/api/Products/`

- This gives the learner a list of product IDs and will be used in modifying data in the proceeding steps. Take note of "Orange Juice" which is product ID2 and pay attention to the description.
- From the student workstation terminal, the learner will use curl to see if the options for a specific product. Type the following command at the terminal:

  ```
  curl -X OPTIONS -D –
  'http://192.168.99.100:3000/api/Products/2'
  ```

- Pay attention to the Access-Control-Methods, if the learner is unsure of the different types then a review of HTTP requests will need to be reviewed. The fact that it accepts PUT, should mean that we can push updates via the terminal.
- From the student workstation terminal, type the following command

  ```
  curl -X PUT "http://192.168.99.100:3000/api/Products/2" -
  H "Content-Type: application/json" --data-binary
  '{"description":"TEST"}'
  ```

- check the Products page to see if the description was updated. The product URL is:

https://192.168.99.100:3000/api/Products/

- Modify the curl PUT command to check the description to include an XXS error

4.10 Persistent XSS without Front-End Access using Curl

In the previous step the learner using Curl was able to adjust and create an XXS exploit in the description. In this section the learner will take what they have learned from the previous steps. From the terminal of the student workstation, write a curl statement that will change the price of product ID #6. Pick one additional product IDs and modify it so that one of them will display an alert (basic java exploit as shown in step 4.7) when viewing the price.

Record in Item #8 of your report if the curl statement for modifying the price of product #6 was successful. Also, record the curl statement used.

Record in Item #9 of your report if the curl statement for modifying the price of an additional product was able to accept the commands for an XXS exploit.

4.11 Posting a Malicious Message to Display Cookies

The objective of this task is to be able to view a user's cookie. Log in as Jim and under the "track orders" section, type in the following code to see the session cookie. The session cookies may not always be available or even viewable; if the cookie doesn't show up that is ok. The code looks like the following example:

```
<script>alert(document.cookie);</script>
```

Take not of the response, does it say cookie/session, or does it say token? Review the guide on "Session vs Token" [1]

Record in Item #10 of your report discuss the main difference between session and token, and which one is more secure; explain your response.

4.12 Generate Report

In OWASP ZAP once the tester has found the different XSS vulnerabilities, in the ZAP application under Report on the top and save a HTML report. Save the report to the home directory of the web-xss workstation, call the file "`report_zap`"

Record in Item #14 of your report any interesting finding of the report. Find three areas of interest and explain why they were important and how they may have an impact on the security of this website.

4.13 Review Journal output

This task allows the user to see output that the server would be generating when certain attacks or exploits have run against them. The learner will review journal output on the server.

- On the shell of the web server type the following command:
  - Sudo journalctl -u juice-shop
- Review the output and space bar until the end, record anything of note and if there are any possible exploits that had ran. Support your claim with output from the journal. Record this at the end of your report under question 15.

5 Stop the Labtainer

When the lab is completed, or you'd like to stop working for a while, run

```
stoplab web-xss
```

from the host Labtainer working directory. You can always restart the Labtainer to continue your work. When the Labtainer is stopped, a zip file is created and copied to a location displayed by the stoplab command. When the lab is completed, send that zip file to the instructor.

References

1. Session VS. Tokens Overview. Available at the following URL:

   https://www.linkedin.com/pulse/cookie-vs-token-authentication-rajveer-gangwar/