XSS Homework

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 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.

2 Lab Environment

You need to use our provided virtual machine image for this lab. The name of the VM image that supports this lab is called SEEDUbuntu12.04.zip, which is built in June 2014. If you happen to have an older version of our pre-built VM image, you need to download the most recent version, as the older version does not support this lab. Go to our SEED web page:
http://www.cis.syr.edu/~wedu/seed/lab_env.html

2.1 Environment Configuration

In this lab, we need three things, which are already installed in the provided VM image: (1) the Firefox web browser, (2) the Apache web server, and (3) the Elgg web application. For the browser, we need to use the LiveHTTPHeaders extension for Firefox to inspect the HTTP requests and responses. The pre- built Ubuntu VM image provided to you has already installed the Firefox web browser with the required extensions.

Starting the Apache Server. The Apache web server is also included in the pre-built Ubuntu image. However, the web server is not started by default. You need to first start the web server using the following command:

```
% sudo service apache2 start
```

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.

Configuring DNS. We have configured the following URL needed for this lab. To access the URL , the Apache server needs to be started first:

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 exam- ple 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.

Configuring Apache Server. In the pre-built VM image, we use 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 default in the directory "/etc/apache2/ sites-available" contains the necessary directives for the configuration:

1. The directive "NameVirtualHost ∗" instructs the web server to use all IP addresses in the ma- chine (some machines may have multiple IP addresses).

2. 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. For example, to configure a web site with URL http://www.example1.com with sources in directory /var/www/Example_1/, and to configure a web site with URL http://www.example2.com with sources in directory /var/www/Example_2/, we use the following blocks:

User	UserName	Password
Admin Alice Boby Charlie Samy	admin alice boby charlie samy	seedelgg seedalice seedboby seedcharlie seedsam
URL	Description	Directory
http://www.xsslabelgg.com	Elgg	/var/www/XSS/Elgg/

```
<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 directory /var/www/Example_1/.

Other software. Some of the lab tasks require some basic familiarity with JavaScript. Wherever neces- sary, 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. We provide a C program that can be configured to listen on a particular port and display incoming messages. The C program can be downloaded from the web site for this lab.

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:

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 available from the lab's web site.

What to submit:

1. Where you injected the javascript code and why? A screenshot that shows the alert dialog popping up.
2. A screenshot of the cookie.
3. The captured GET command from the c code that you ran.

Useful sources:

Javascript tutorial: https://www.w3schools.com/js/

Firebug & Javascript: **http://www.w3resource.com/web-development-tools/debug-JavaScript-with-Firebug.php**