Cross-Site Request Forgery (CSRF) Attack Lab

Austin Hansen

1001530325

3.1 Task 1: Observing HTTP Request.

![Text

Description automatically generated]()

After downloading the firefox extension “HTTP Header Live” and typing in “csrflabelgg.com”, we get this entry, this is our GET request.

![Text

Description automatically generated]()

This long statement is our POST request to elgg, to get it we sign in as alice using the username “alice” and password “aliceseed”.

![Graphical user interface, text

Description automatically generated]()

As a bonus for this task, I’d like to show what happens if we were logged in as boby and sent Alice a friend request, this will be useful in the next task, notice the second half of the request after the website, we now know that in order to add a friend, we take the path: /action/friends/add?friend = user\_id

3.2 Task 2 Task 2: CSRF Attack using GET Request.

![Graphical user interface, text, application

Description automatically generated]()

For me to launch this attack, I must first know what user id bob is, after looking around for a bit I found this when I clicked to edit boby’s profile using firefox’s inspector tool. Importantly for me, the user id is contained within the sentence “{“g”:43,” pog”:43.”c”}”, the Id is 43. Now we can make our attack.

![Text

Description automatically generated]()



Getting a bit of help from the lab document and class, here is the html code that I came up with to add boby as a friend to alice, It is disguised as an error 404 webpage, and leverages what I showed in the last image of task 1 to add boby as a friend. Also, you must restart the apache server to have this change take effect.

Graphical user interface, application, Word

Description automatically generated

Before we start let us verify that Alice has no friends.

Graphical user interface, text

Description automatically generated with medium confidence

I drafted a message as boby and sent it to alice, here is the contents of the message, Alice of course being curious clicks on the website link…

Text

Description automatically generated

Which displays this, the broken image symbol at the bottom is the code we wrote above to add alice as a friend. Were we successful?

Graphical user interface, text, application, Word

Description automatically generated

Boby was added as a friend despite us not actually adding him manually, attack successful.

Explanation of what happened: This is pretty much a textbook CSRF, we use a GET request to add boby to alice’s friend list, essentially a user on a trusted site accesses a malicious site allowing malicious code to run, and in this case forcefully add boby to the friend list.

3.3 Task 3: CSRF Attack using POST Request.



To figure out the syntax of the POST command that we are looking for we can first write a post as alice.

![Text

Description automatically generated]()

Using HTTP Header Live, we find where the first POST command, next we click on the POST command to learn more.

![Graphical user interface, text

Description automatically generated]()

Here is a closer observation of the command we saw earlier. We now know that in order to post on our profile we need to follow the path “/action/profile/edit”, in addition the content of this statement is below this output.

![Text

Description automatically generated]()

I moved content to a notepad, some parts require an access level, regardless this is useful information that we can use in our attack, indeed we can use this as a pseudo template in our new index file.

![Graphical user interface, text

Description automatically generated]()

![Graphical user interface, text, application

Description automatically generated]()

As you can see, we had to rewrite index quite a bit, I did not want to take up the entire page, so here is the top and bottom of the program, as the program itself is quite repetitive. Essentially, we use the fields given by our note pad file to have the authority to edit alice’s profile. (Note that the apostrophes when first transferring in were not really apostrophes, so I had to had edit each apostrophe to get this to work)



Just as in the first attack, I wrote a message while signed in as Boby, Alice, ever curious clicks on the link…

Graphical user interface

Description automatically generated with medium confidence

It automatically runs and changes her Brief description field to “boby is my hero”, thus our attack is successful.

**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?

![Text

Description automatically generated]()

After poking around for a bit, I found that you can inspect the members section of the website, and after abit of digging find the id of all the users, this is how boby can easily get alice’s user id for the attack (much better than the other method I used in task 2), but getting her password is a much more challenging task.

**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?

I do not need an image to answer this question, the answer is no, he cannot launch a CSRF attack without knowing the specific user id of the person that he wishes to target, this is because to edit a person’s profile not just any number will do, so this attack is limited to a targeted attack.

3.4 Task 4: Implementing a countermeasure for Elgg.

A picture containing graphical user interface

Description automatically generated

As per assignment instructions, I commented out the “return true;” from the gatekeeper function, what this does is to send two new fields, a timestamp and a unique token that we cannot see if we inspect the HTTP request.



Next, we inspect our tokens, here are our new ones.

![Text

Description automatically generated]()

![Graphical user interface, text, application

Description automatically generated]()

We attack again using the same program from the previous task, however…

![Background pattern

Description automatically generated]()

We get a series of error messages when we run the attack, clearly the tokens from the previous attacks do not work, additionally, it is not easy to get the other two fields, so conducting further attacks in this manner is impossible under these circumstances.

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

The reason that the attacker cannot send secret tokens in a CSRF attack is that they do not know the values of the timestamp and secret token that is embedded in our victims elgg page. The attacker, Boby, is hindered by the browsers access control, and by the fact that the security token itself is hashed using MD5 and is split into four pieces, the users session id, a secret value, a random string and of course our timestamp.