Richard Stretanski

CIS 544-001

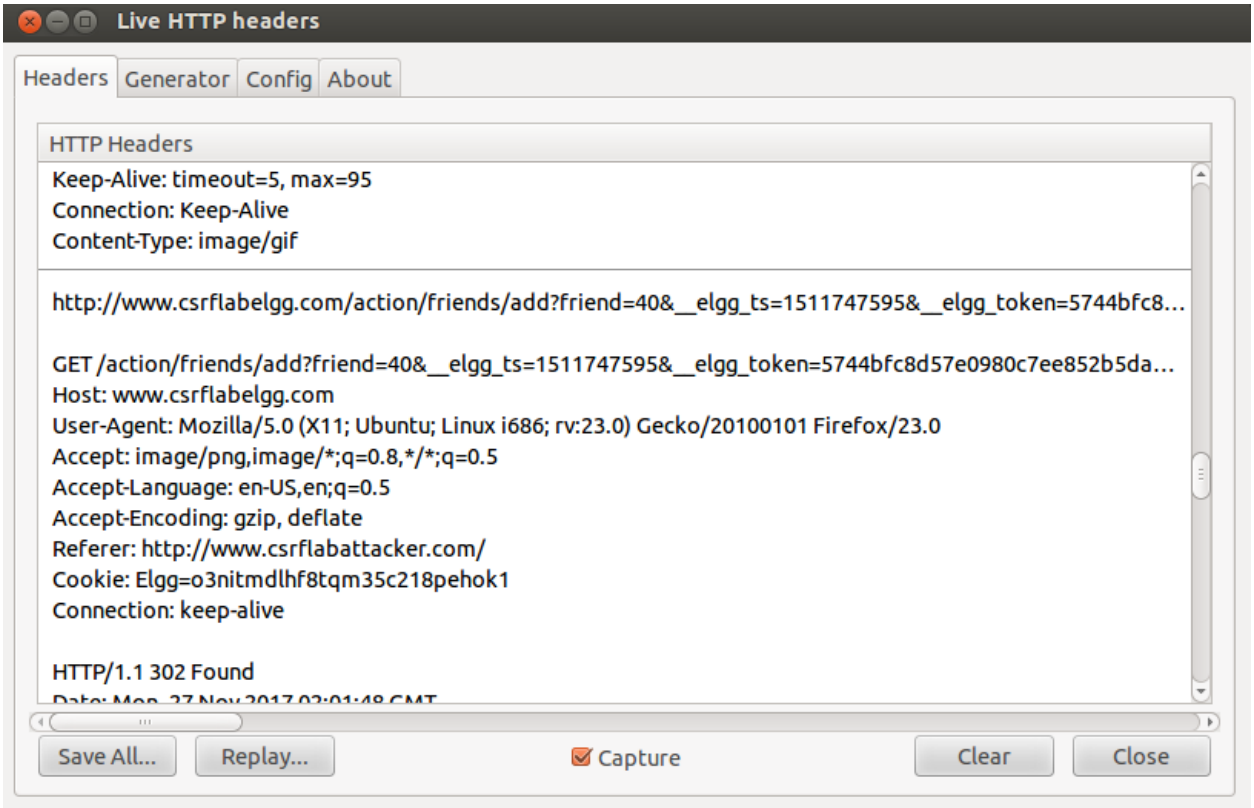
Attack Project 2

Abstract

In this paper I succeed in a CSRF attack on a user that is part of a social networking site: Elgg, which is hosted locally on a VM. The goal is to have our user (Boby) become friends with the user (Alice) when Alice clicks on a link to a malicious website. It is assumed that Alice has an active session with Elgg, and that she received the URL to the malicious website via email or a posting on Elgg.

Methodology & Results

The goal of this CSRF attack is to make the user (the victim) Alice friends with the user Boby (me). To accomplish this, I first need to obtain the GET request that is sent when a user tries to send Boby a friend request. To obtain this I will turn on the 'Live HTTP Headers' Firefox extension and then click on the button to send Boby a friend request on his page (http://www.csrflabelgg.com/profile/boby). For this I used a third user. The Live HTTP Headers extension outputted the GET request below:

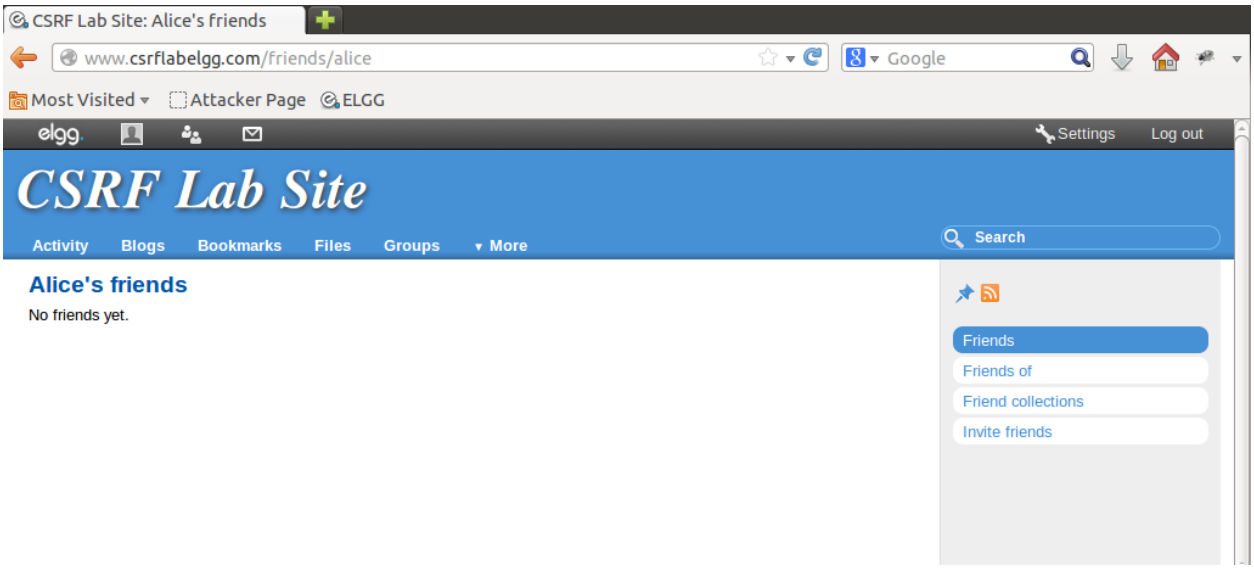


Next, I used this GET request in the malicious website that will make Alice befriend Boby. I used the website http://www.csrflabattacker.com/ which was hosted locally on the VM. I put the get request

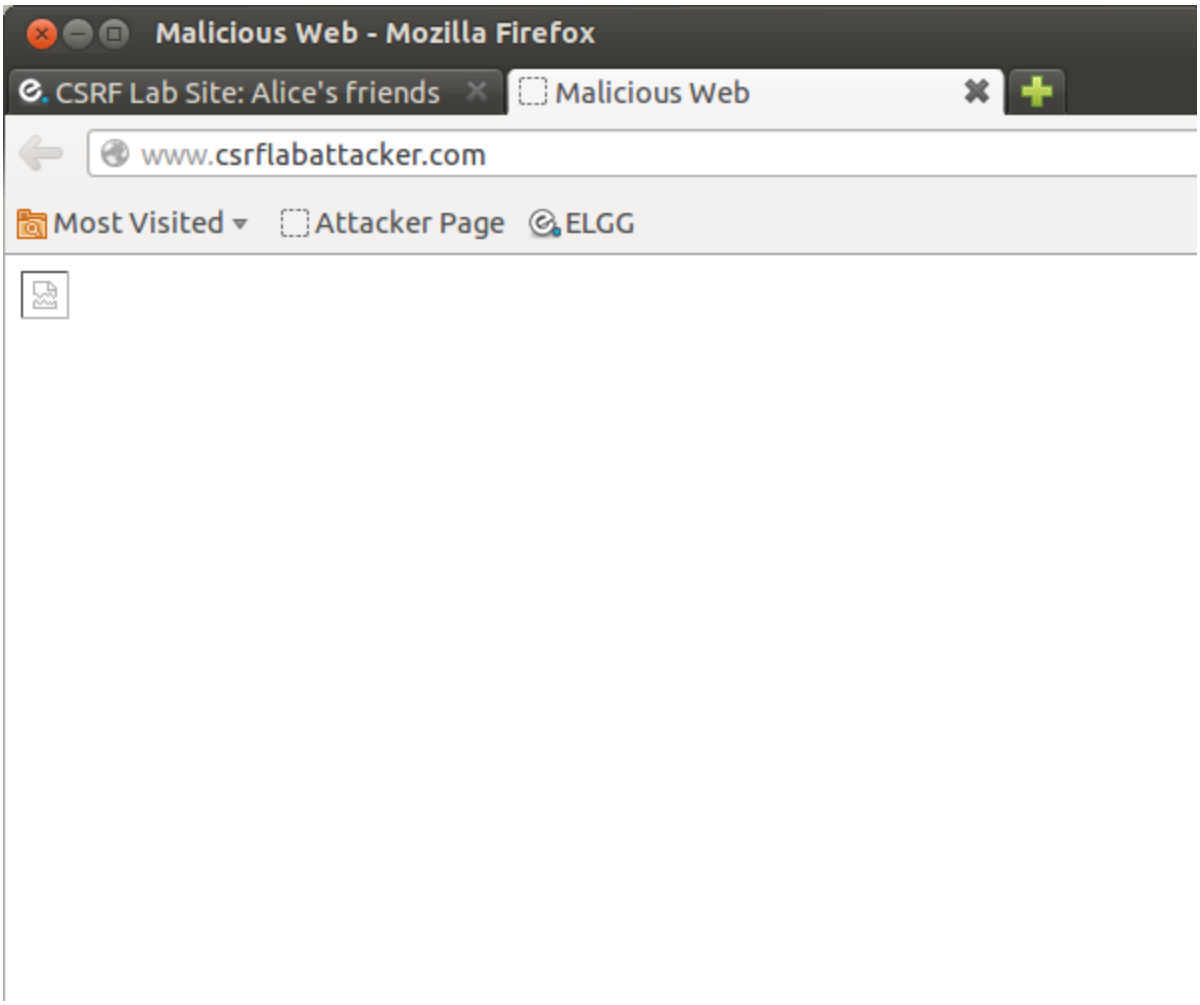
inside an HTML img tag because this tag has no restrictions on the URL that can be used as a tag attribute. Since the malicious website is hosted locally, I can edit the file “index.html” (located at var/www/CSRF/Attacker/index.html) to include the img tag and GET request in the body as shown below:

Note that this file (index.html) can only be edited with root access on the VM.

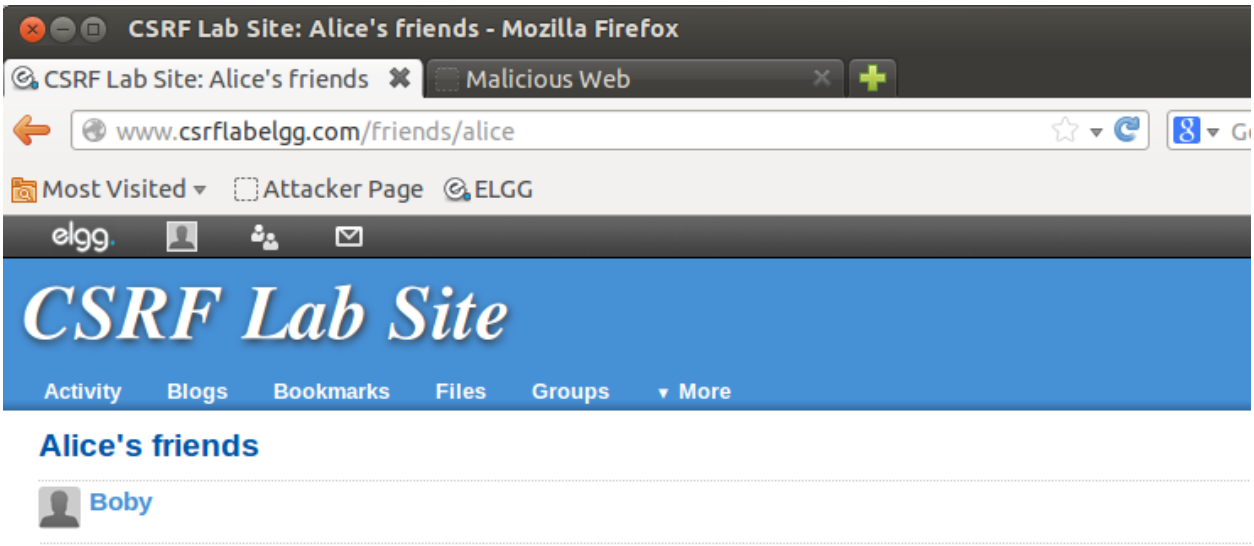
At this point in time we are assuming that Alice has already logged in to Elgg and has yet to visit the malicious website. A session identifier is stored in a cookie in Alice’s browser (Firefox). Here is a screenshot of Alice’s friends page before she receives the malicious link:



I then assume that Alice receives the link to the site http://www.csrflabattacker.com/ via email or blog post and clicks on it. It takes her to a page that looks like the screenshot below:



The malicious site contains an image that sends the GET request to the trusted site from Alice's browser, which attaches the session cookie to it. The social networking site http://www.csrflabelgg.com believes the GET request is legitimate and processes the request. As a result, Alice unknowingly is now friends with Boby as shown in the screenshot below:



Analysis

It is my understanding that my CSRF attack on Alice succeeded because of two key vulnerabilities. First, I was easily able to obtain the GET request URL by using an extension that displayed HTTP headers and second, the Elgg site accepted the illegitimate cross-site request.

The GET request was obtained by signing in to a third user account on Elgg. I turned on the HTTP headers extension that records the live headers, and clicked on the friend request button while signed in to the third account. From there I just had to copy the GET request link that appeared in the header and paste it into an image tag on the malicious webpage. The first vulnerability can be solved by using custom headers. If the website receives a request with a header that doesn't match its custom header, it will not validate the request.

After Alice visited the page, the browser did all the work. The browser automatically attached a cookie to the request and the illegitimate cross-site request was verified. The second vulnerability can be solved in this system by forcing the user to perform a verification after a cross-site request is received. The user verification could be one of many things including requiring the user to re-login, or requiring them to fill out a captcha.

Conclusion

I was successfully able to perform a CSRF attack which allowed my user account to befriend the victim's user account on a social networking site. While this attack seems mostly harmless, a similar attack on a different but also vulnerable system could result in a loss of account ownership, the purchase of goods at the victim's expense, or a transfer of funds. All of this could be done without the knowledge of the victim, at least without knowledge until after the damage is done.

To me, something especially interesting about this specific attack is that, given the right set of circumstances (vulnerabilities, and Alice logging in before clicking the malicious link) this was extremely low effort and low complexity in comparison to a buffer overflow attack. I think this attack project was an excellent introduction to CSRF attacks and I feel like I have a better understanding of CRSF attacks than the previous buffer overflow project as a result. In a future attack project, it would be interesting to utilize JavaScript to exploit POST requests, however I think if you included it with this project, it would likely be too much and I realize there is a tradeoff between the variety and depth of attacks.

References

[1] Elgg documentation: http://docs.elgg.org/wiki/Main_Page.

[2] JavaScript String Operations.
http://www.hunlock.com/blogs/The_Complete_Javascript_Strings_Reference

[3] Session Security Elgg. http://docs.elgg.org/wiki/Session_security.

[4] Forms + Actions Elgg http://learn.elgg.org/en/latest/guides/actions.html

[5] PHP:Session id-Manual: http://www.php.net//manual/en/function.session-id.php

[6] Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet: https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet#CSRF_Specific_Defense