Understanding and Undertaking Cross-site Request Forgery Attack

Aim

The aim of this lab is to understand CSRF attacks, which uses a malicious website to inject http requests to a trusted site via a victim user of said trusted site.

Introduction and Background

The focus of this lab is to launch an CSRF attack on Elgg - a popular open-source web application for social networks, modified to disable all CSRF countermeasures. The goal is to have a victim add an attacker to their friend list and then change the victim's profile to say something the attacker wants. We use three containers, one running the web server for Elgg, the second hosts the attacker's malicious website and the last is running the MySQL database. Once we destroy a container all the data inside is lost, but we do want to keep our data in the MySQL database. To achieve this we mount the data folder on the host machine. Thus, even if a container is destroyed the data folder on the host machine still remains.

Methods

First, we check we open a web browser and go to www.seed-server.com where we have hosted our modified Elgg.

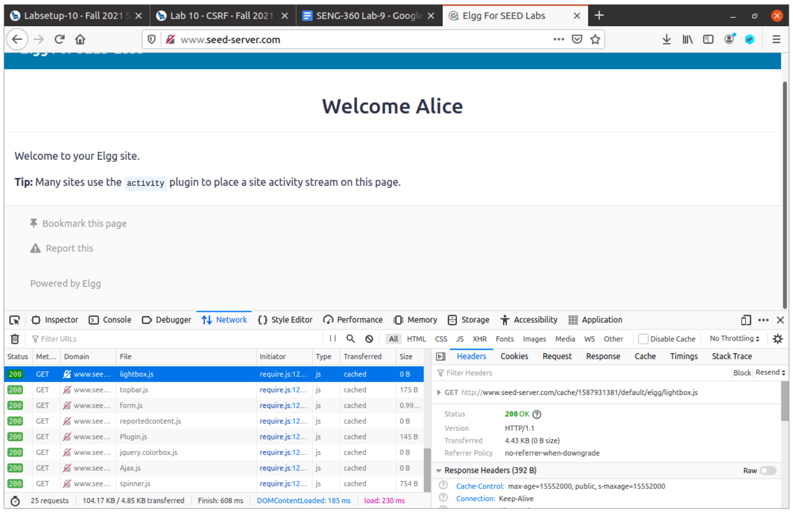
The *first task* is for us to know what a legitimate http request on Elgg looks like. To do this we use the firefox add-on “http header live” to capture an HTTP GET request and an HTTP POST request in Elgg.

The *second task* is to forge an HTTP GET request, and have Alice(victim) add Samy(attacker) to their friend list. This happens as soon as Alice visits the attacker’s malicious website hosted at www.attacker32.com. To accomplish this, we use the code our addfriend.html provided in the labsetup folder.

The *third task* is to forge an HTTP POST request. We use this to have the attacker change the victim’s profile to say what they want. We follow the same method in task2 and use the code in editprofile.html provided in labsetup folder.

Results and Discussion

For *task one* we go on to elgg and inspect the http requests.



Welcome Alice

Welcome to your Elgg site.

Tip: Many sites use the `activity` plugin to place a site activity stream on this page.

Status	Met...	Domain	File	Initiator	Type	Transferred	Size
200	GET	www.see...	lightbox.js	require.js:12...	js	cached	0 B
200	GET	www.see...	topbar.js	require.js:12...	js	cached	175 B
200	GET	www.see...	form.js	require.js:12...	js	cached	0.99...
200	GET	www.see...	reportedcontent.js	require.js:12...	js	cached	0 B
200	GET	www.see...	Plugin.js	require.js:12...	js	cached	145 B
200	GET	www.see...	jquery.colorbox.js	require.js:12...	js	cached	0 B
200	GET	www.see...	Ajax.js	require.js:12...	js	cached	0 B
200	GET	www.see...	spinner.js	require.js:12...	js	cached	754 B

25 requests | 104.17 KB / 4.85 KB transferred | Finish: 608 ms | DOMContentLoaded: 185 ms | load: 230 ms

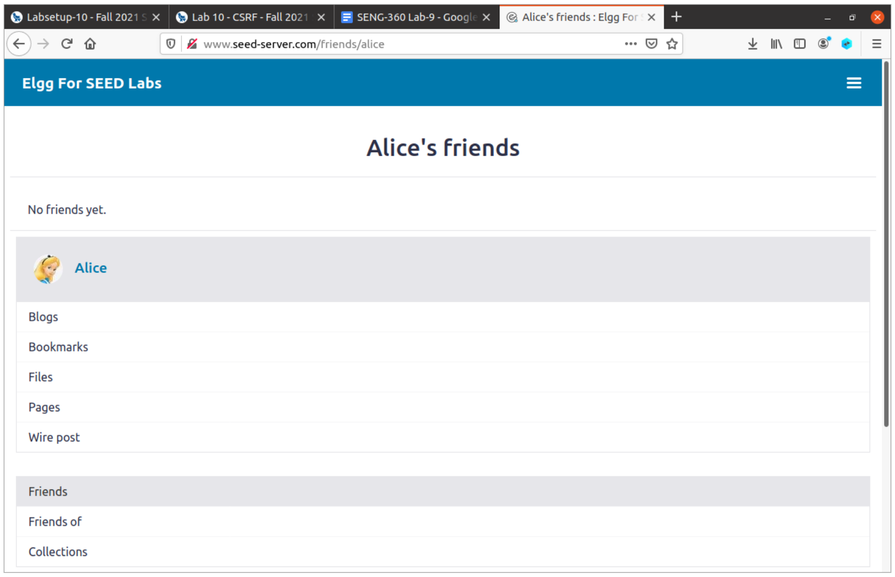
GET http://www.seed-server.com/cache/1587931381/default/elgg/lightbox.js

Status 200 OK
Version HTTP/1.1
Transferred 4.43 KB (0 B size)
Referrer Policy no-referrer-when-downgrade

Response Headers (392 B)

Cache-Control: max-age=15552000, public, s-maxage=15552000
Connection: Keep-Alive

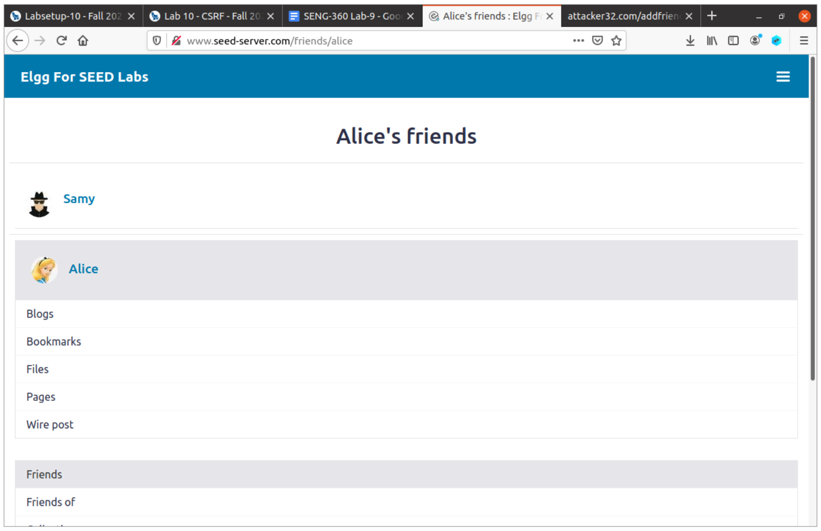
For *task two* we first check to see if Alice's friend list does not have Samy.



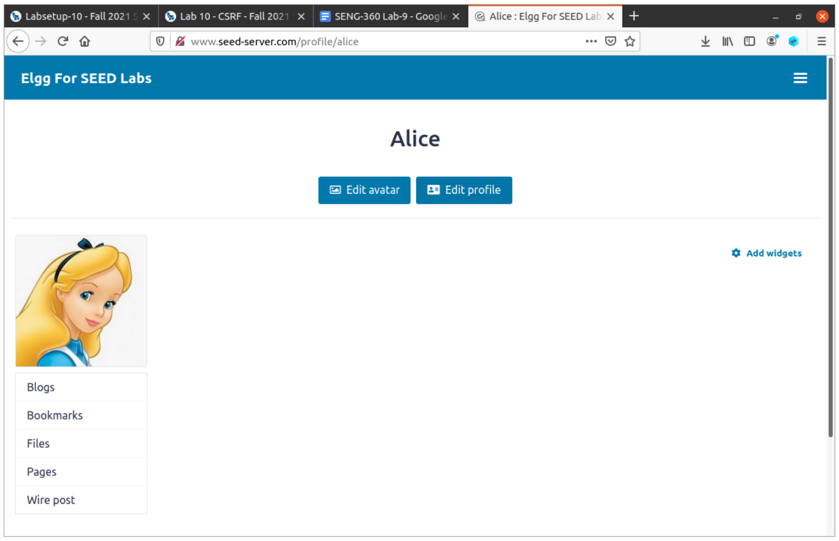
Alice's friends

No friends yet.

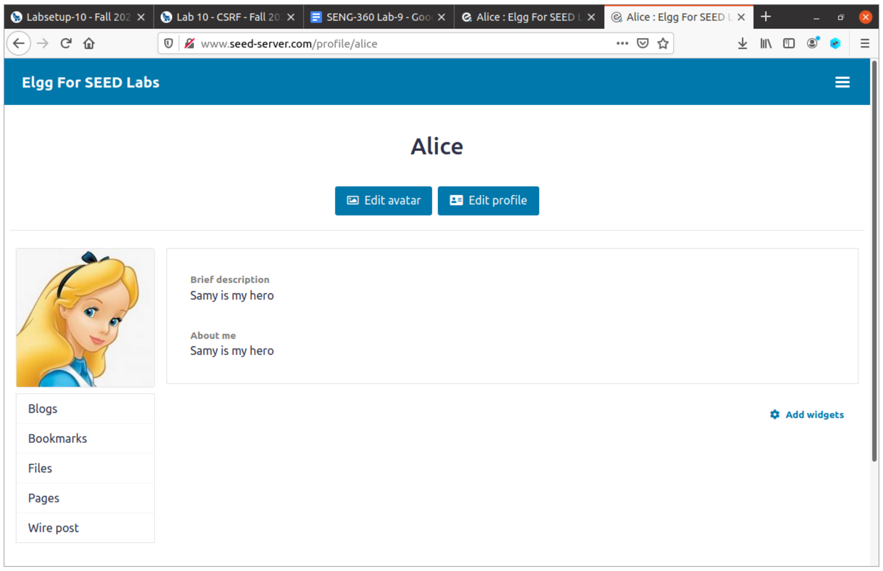
Then after editing the code in addfriend.html, we go on to the malicious website and click on the add-friend-attack button. Then we go back on to alice's friend list and see that Samy was added as a friend



For the *third task* we log on to Alice’s website and check that her status page is empty.



Next we edit the code in editprofile.html and then open our malicious website. We then click on the edit-profile-attack button. Then we go back to Alice’s status page and we see that it has changed.



Alice

Brief description
Samy is my hero

About me
Samy is my hero