Cross-Site Request Forgery (CSRF) Attack Lab

Table of Contents:

Overview

The objective of this lab is to help students understand the Cross-Site Request Forgery (CSRF) attack. A CSRF attack involves a victim user, a trusted site, and a malicious site. The victim user holds an active session with a trusted site while visiting a malicious site. The malicious site injects an HTTP request for the trusted site into the victim user session, causing damages.

In this lab, students will be attacking a social networking web application using the CSRF attack. The open-source social networking application is called Elgg, which has already been installed in our VM. Elgg has countermeasures against CSRF, but we have turned them off for the purpose of this lab. This lab covers the following topics:

- Cross-Site Request Forgery attack
- CSRF countermeasures: Secret token and Same-site cookie
- HTTP GET and POST requests
- JavaScript and Ajax

Environment Setup

In this lab, we will use three websites. The first website is the vulnerable Elgg site accessible at www. seed-server.com. The second website is the attacker's malicious web site that is used for attacking Elgg. This web site is accessible via www.attacker32.com. The third website is used for the defense tasks, and its hostname is www.example32.com. We use containers to set up the lab environment.

Important

Before building the containers in the labsetup folder, follow the following steps -

In the normal Seed VM, run the following commands -

Command -

```
# sudo nano /etc/hosts
```

Scroll down to the CSRF Section and copy paste the following

```
10.9.0.5 www.seed-server.com
10.9.0.5 www.example32.com
10.9.0.105 www.attacker32.com
```

Note - Incase you are using the risc-gen.tech website, first switch the user to seed (su seed), then nslookup elgg, attacker and example32. Enter the respective ip address obtained in similar fashion to the VM setup as shown above.

User accounts - We have created several user accounts on the Elgg server; the username and passwords are given in the following.

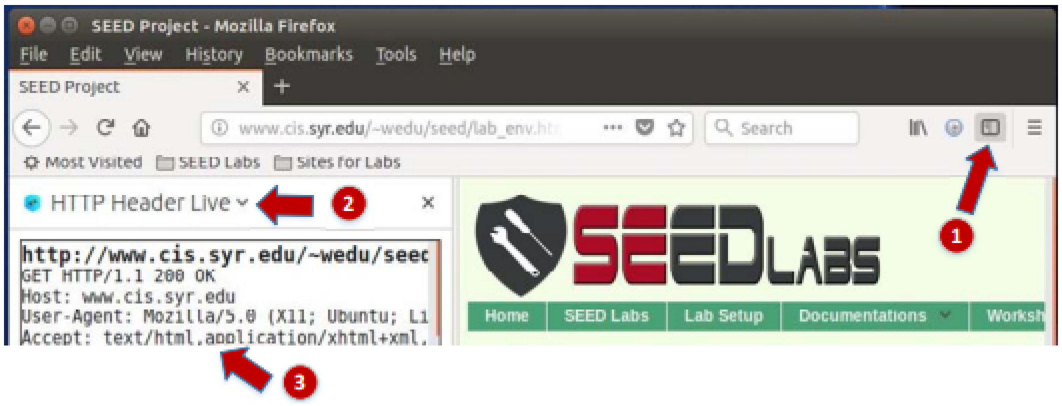
UserName	Password
admin	seedelgg
alice	seedalice
boby	seedboby
charlie	seedcharlie
samy	seedsamy

Task 1: Observing HTTP Request

In Cross-Site Request Forge attacks, we need to forge HTTP requests. Therefore, we need to know what a legitimate HTTP request looks like and what parameters it uses, etc. We can use a Firefox add-on called "HTTP Header Live" for this purpose. The goal of this task is to get familiar with this tool. Please use this tool to capture an HTTP GET request and an HTTP POST request in Elgg. In your report, please identify the parameters used in these requests, if any.

1. Open www.seed-server.com on Firefox in your VM.
2. You will be able to see the "HTTP Header Live" extension on the top right of your browser (blue coloured icon)
3. With the extension open, login to the elgg website
4. You will see the HTTP Request has been captured (Scroll to the Top)
5. In the POST request, you will be able to see the parameters i.e username and password.

Take a screenshot and explain any relevant findings.



```
http://www.cis.syr.edu/~wedu/seec
GET HTTP/1.1 200 OK
Host: www.cis.syr.edu
User-Agent: Mozilla/5.0 (X11; Ubuntu; Li
Accept: text/html,application/xhtml+xml,
```

Figure 1: Enable the HTTP Header Live Add-on

Task 2: CSRF Attack using GET Request

In this task, we need two people in the Elgg social network: **Alice and Samy**. Samy wants to become a friend of Alice, but Alice refuses to add him to her Elgg friend list. Samy decides to use the CSRF attack to achieve his goal. He sends Alice a URL (via an email or a posting in Elgg); Alice, curious about it, clicks on the URL, which leads her to Samy's web site: www.attacker32.com. Pretend that you are Samy, describe how you can construct the content of the web page, so as soon as Alice visits the web page, Samy is added to the friend list of Alice (assuming Alice has an active session with Elgg).

To add a friend to the victim, we need to identify what the legitimate Add-Friend HTTP request (a GET request) looks like. We can use the "HTTP Header Live" Tool to do the investigation. In this task, you are not allowed to write JavaScript code to launch the CSRF attack. Your job is to make the attack successful as soon as Alice visits the web page, without even making any click on the page (hint: you can use the img tag, which automatically triggers an HTTP GET request).

Elgg has implemented a countermeasure to defend against CSRF attacks. In Add-Friend HTTP requests, you may notice that each request includes two weird-looking parameters, elgg ts and elgg token. These parameters are used by the countermeasure, so if they do not contain correct values, the request will not be accepted by Elgg. We have disabled the countermeasure for this lab, so there is no need to include these two parameters in the forged requests.

1. The first step now is to check how the HTTP Request looks when you add someone as your friend.

- Login as Samy, and go to the members section.

- Click on any member, for example - Charlie. Open the HTTP Header Live extension and click on Add Friend.

You should see a URL like this - http://www.seed-server.com/action/friends/add?friend=58

The number at the end is a user's **guid**. We must now create a similar request with our guid in the friend parameter, so when Alice clicks on a malicious link this request is sent and we (Samy) get added to her friend list.

- To find our guid, we can go to the member section, click on Samy and view the page source. Search for guid and you should be able to find the number.

2. Now we are ready to create our malicious request through our malicious webpage

Open the Attacker terminal, using docksh

Command:

# docksh attacker-10.9.0.105

# cd /var/www/attacker

# nano addfriend.html

In the html document, change the empty **src** field to

"http://www.seed-server.com/action/friends/add?friend=<Samy's Guid>"

Make sure to add Samy's Guid, which you have found by viewing the page source, at the end of the above URL.

3. Logout as Samy and login as Alice in the elgg website. Open www.attacker32.com in a new tab and click on the Add-Friend Attack link. Now check Alice's friend list, you should be able to find Samy added as a friend.

Please take appropriate screenshots of each step and explain your observations in detail.

Task 3: CSRF Attack using POST Request

After adding himself to Alice's friend list, Samy wants to do something more. He wants Alice to say "Samy is my Hero" in her profile, so everybody knows about that. Alice does not like Samy, let alone putting that statement in her profile. Samy plans to use a CSRF attack to achieve that goal. That is the purpose of this task.

One way to do the attack is to post a message to Alice's Elgg account, hoping that Alice will click the URL inside the message. This URL will lead Alice to your (i.e., Samy's) malicious web site www. attacker32.com, where you can launch the CSRF attack.

The objective of your attack is to modify the victim's profile. In particular, the attacker needs to forge a request to modify the profile information of the victim user of Elgg. Allowing users to modify their profiles is a feature of Elgg. If users want to modify their profiles, they go to the profile page of Elgg, fill out a form, and then submit the form—sending a POST request—to the server-side script /profile/edit.php, which processes the request and does the profile modification.

1. Our first step would be to check out the POST request when we try to update our own profile. From this we will have some idea of the fields and parameters to set in our forged request.

- Login as Samy on the elgg website. Click on your profile and click on the Edit Profile button. Open the HTTP Header Live extension
- We see many fields and boxes, where we can enter our details. For now we'll just enter "Samy is my hero" in the **"about me"** and **"brief description"** section.
- Click on save at the end of the page and in the extension we can see the POST request being captured. We can see a request is made to http://www.seed-server.com/action/profile/edit with parameters like name, *description, briefdescription, accesslevel, guid etc*. being taken.

2. Seeing the guid field is required for this request as well, we click on Alice’s profile in the members section and check the page source to find it.

3. Now we are ready to create our malicious request through our malicious webpage

Open the Attacker terminal, using docksh

Command:

```
# docksh attacker-10.9.0.105
# cd /var/www/attacker
# nano editprofile.html
```

In the html document, we see four fields which we have to be set.

We set

The name - Alice

briefdescription - Samy is my hero

accesslevel - 2 (public access)

guid - Alice’s guid

Then to construct the form we set p.action ="http://www.seed-server.com/action/profile/edit";

4. Logout as Samy and login as Alice in the elgg website. Open www.attacker32.com in a new tab and click on the Edit-Profile Attack link. Now check Alice’s profile, you should be able to find “Samy is my hero” added in her description.

Please take appropriate screenshots of each step and explain your observations in detail.

Questions.

In addition to describing your attack in full details, you also need to answer the following questions in your report:

• 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 the different ways Boby can solve this problem.

• 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

Task 4: Enabling Elgg's Countermeasure

To defend against CSRF attacks, web applications can embed a secret token in their pages. All the requests coming from these pages must carry this token, or they will be considered as a cross-site request, and will not have the same privilege as the same-site requests.

Attackers will not be able to get this secret token, so their requests are easily identified as cross-site requests. Elgg uses this secret-token approach as its built-in countermeasures to defend against CSRF attacks. We have disabled the countermeasures to make the attack work. Elgg embeds two parameters elgg ts and elgg token in the request. The two parameters are added to the HTTP message body for the POST requests and to the URL string for the HTTP GET requests. The server will validate them before processing a request.

Secret token generation - Elgg's security token is a hash value (md5 message digest) of the site secret value (retrieved from database), timestamp, user session ID and random generated session string. The elgg web application validates the generated token and timestamp to defend against the CSRF attack. Every user action calls the validate function inside Csrf.php, and this function validates the tokens. If tokens are not present or invalid, the action will be denied and the user will be redirected. In our setup, we added a return at the beginning of this function, essentially disabling the validation.

To turn on the countermeasure, we must first get into the Elgg container, go to the /var/www/elgg/vendor/elgg/elgg/engine/classes/Elgg/Security folder, remove the return statement from Csrf.php.

Command:

```
# docksh elgg-10.9.0.5
# cd /var/www/elgg/vendor/elgg/elgg/engine/classes/Elgg/Security
# nano Csrf.php
```

Scroll down, and in the validate(Request $request) function, comment out the return statement in the first line.

Now clear out Alice's profile - remove Samy from friends and reset her descriptions. Now go to www.attacker32.com and perform the attack's again (assuming the html files have been changed according to the previous tasks), you will see that the attack fails.

Please provide screenshots of your observations with appropriate explanations.

Task 5: Experimenting with the SameSite Cookie Method

Most browsers have now implemented a mechanism called SameSite cookie, which is a property associated with cookies. When sending out requests, browsers will check this property, and decide whether to attach the cookie in a cross-site request.

A web application can set a cookie as SameSite if it does not want the cookie to be attached to cross-site requests. For example, they can mark the session ID cookie as SameSite, so no cross-site request can use the session ID, and will therefore not be able to launch CSRF attacks.

To help students get an idea on how the SameSite cookies can help defend against CSRF attacks, we have created a website called **www.example32.com** on one of the containers. Please visit the following URL (the hostname is already mapped to 10.9.0.5 in the /etc/hosts file; if you are not using the SEED VM, you should add this mapping to your machine):

Please follow the links for the two experiments. Link A points to a page on example32.com, while Link B points to a page on attacker32.com. Both pages are identical (except for the background color), and they both send three different types of requests to www.example32.com/showcookies.php, which simply displays the cookies sent by the browser. By looking at the display results, you can tell which cookies were sent by the browser.

Please go through the following links for an understanding on SameSite cookies.

https://owasp.org/www-community/SameSite
https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie/SameSite

Please do the following:

1. Open www.example32.com
2. Click on Link A, submit the GET and POST requests
3. Click on Link B, submit the GET and POST requests.

- Please describe what you see and explain why some cookies are not sent in certain scenarios. (check the **showcookies.php** for more clarity)

Questions -

1. Based on your understanding, please describe how the SameSite cookies can help a server detect whether a request is a cross-site or same-site request.

2. Please describe how you would use the SameSite cookie mechanism to help Elgg defend against CSRF attacks. You only need to describe general ideas, and there is no need to implement them

You can use the following for reference -
https://www.php.net/manual/en/function.setcookie.php

Give brief explanations for the above.

Submission

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive credits.