Lab 3 - CSRF

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.

Lab Setup

This is the same as the previous lab. We'll be using **SEED Ubuntu-20.04 VM** for this lab. Make sure you've downloaded and installed the VM on you local machine. Follow this page for detailed instructions on how to download and install this VM. Add the following entry in `/etc/hosts` file :

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

If there's already an entry corresponding to `www.seed-server.com`, remove it and add the above. In this lab, we will use two 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`.

Container Setup

The lab uses docker containers. Two containers will be used here, one for hosting a web application and one for the database for the web application. Inside the VM, download the *Labsetup.zip* file from https://seedsecuritylabs.org/Labs_20.04/Web/Web_CSRF_Elgg/. Unzip it, enter the Labsetup folder, and use the `docker-compose.yml` file to set up the lab environment. Run `dcbuild` and `dcup` consecutively in the same folder as the `docker-compose.yml` to start the containers. Now if you hit `http://www.seed-server.com` in firefox, you'll see the vulnerable web application. For more details on working with docker, check this.. If you haven't gone through it already, its highly recommended that you get yourself familiar with the docker commands.

About the `Elgg` Web Application

We use an open-source web application called `Elgg` in this lab. `Elgg` is a web-based social-networking application. It is already set up in the provided container images; its URL is http://www.seed-server. com. We use two containers, one running the web server (10.9.0.5) , and the other running the MySQL database (10.9.0.6). The IP addresses for these two containers are hardcoded in various places in the configuration, so please do not change them from the `docker-compose.yml file`.

User Accounts

We have created several user accounts on the `Elgg` server; the user name and passwords are given in the following.

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

Lab Tasks

When you copy and paste code from this PDF file, very often, the quotation marks, especially single quote, may turn into a different symbol that looks similar. They will cause errors in the code, so keep that in mind. When that happens, delete them, and manually type those symbols.

Preparation

In this lab, we need to construct HTTP requests. To figure out what an acceptable HTTP request in `Elgg` looks like, we need to be able to capture and analyze HTTP requests. Like the previous lab, we can use a Firefox add-on called `"HTTP Header Live"` for this purpose.

Task 1

In this task, we need two people in the Elgg social network: Alice and Samy. Samy wants to become a friend to 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 an 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.

Task 2

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.

The server-side script `edit.php` accepts both GET and POST requests, so you can use the same trick as that in Task 1 to achieve the attack. However, in this task, you are required to use the POST request. Namely, attackers (you) need to forge an HTTP POST request from the victim's browser, when the victim is visiting their malicious site. Attackers need to know the structure of such a request. You can observe the structure of the request, i.e., the parameters of the request, by making some modifications to the profile and monitoring the request using the "HTTP Header Live" tool. You may see something similar to the following. Unlike HTTP `GET` requests, which append parameters to the URL strings, the parameters of HTTP `POST` requests are included in the HTTP message body (see the contents between the two ## symbols):

```
http://www.seed-server.com/action/profile/edit
POST /action/profile/edit HTTP/1.1
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:23.0) ...
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.seed-server.com/profile/elgguser1/edit
Cookie: Elgg=p0dci8baqrl4i2ipv2mio3po05
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 642
__elgg_token=fc98784a9fbd02b68682bbb0e75b428b&__elgg_ts=1403464813 ##
&name=elgguser1&description=%3Cp%3Iamelgguser1%3C%2Fp%3E
&accesslevel%5Bdescription%5D=2&briefdescription= Iamelgguser1
&accesslevel%5Bbriefdescription%5D=2&location=US
......                                                            ##
```

After understanding the structure of the request, you need to be able to generate the request from your attacking web page using JavaScript code. To help you write such a JavaScript program, we provide a sample code in the following code fence. You can use this sample code to construct your malicious web site for the CSRF attacks. This is only a sample code, and you need to modify it to make it work for your attack.

```
<html>
<body>
<h1>This page forges an HTTP POST request.</h1>

<script type="text/javascript">
function forge_post()
{
    var fields;
    // The following are form entries need to be filled out by attackers.
    // The entries are made hidden, so the victim won't be able to see them.
    fields += "<input type='hidden' name='name' value='****'>";
    fields += "<input type='hidden' name='briefdescription' value='****'>";
    fields += "<input type='hidden' name='accesslevel[briefdescription]'
    value='2'>"; ##
    fields += "<input type='hidden' name='guid' value='****'>";
    // Create a <form> element.
    var p = document.createElement("form");
    // Construct the form
    p.action = "http://www.example.com";
    p.innerHTML = fields;
    p.method = "post";
    // Append the form to the current page.
    document.body.appendChild(p);
    // Submit the form
    p.submit();
}
// Invoke forge_post() after the page is loaded.
window.onload = function() { forge_post();}
</script>
</body>
</html>
```

In Line marked ##, the value 2 sets the access level of a field to public. This is needed, otherwise, the access level will be set by default to private, so others cannot see this field. It should be noted that when copy-and-pasting the above code from a PDF file, the single quote character in the program may become something else (but still looks like a single quote). That will cause syntax errors. Replacing all the single quote symbols with the one typed from your keyboard will fix those errors.

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

- 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.
- 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? Please explain.

Task 3

Launch a CSRF attack on Alice using the techniques applied in the previous task so that it does the following things when the `Edit-Profile` page of attacker website is visited:

- set `Brief Descripttion` field to `Samy is my Hero`.
- set `Interests` to `Hacking`.
- set `Twitter username` to `Geralt`.
- Make sure these values are `public`.

Task 4

Boby loves Alice and he doesn't trust anyone other than Alice. He doesn't even open/see any message from anyone other than Alice! What a love! Samy is planning to take advantage of this insane love of Boby. He wants to send a message from Alice's account to Boby which will contain a link to Samy's website `www.attacker32.com`. Your task is to change the `Edit-Profile` page of Samy's website so that when Alice visits that page, a message would be automatically sent to Boby with the link. And poor Boby, as the message is from none other than Alice, will ignorantly click on the link and fall victim to Samy's insidious plan :)

Submission

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not result in full marks.