SQL Injection Attack Lab – Using Collabtive

(Web Application: Collabtive)

1 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.

In this lab, we modified a web application called `Collabtive`, and disabled several countermeasures implemented by `Collabtive`. As the results, we created a version of `Collabtive` that is vulnerable to the SQL-Injection attack. Although our modifications are artificial, they capture the common mistakes made by many web developers. Students' 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.

2 Lab Environment

You need to use our provided virtual machine image for this lab. The name of the VM image that supports this lab is called `SEEDUbuntu11.04-Aug-2011`, which is built in August 2011. If you happen to have an older version of our pre-built VM image, you need to download the most recent version, as the older version does not work for this lab. Go to our SEED web page (`http://www.cis.syr.edu/~wedu/seed/`) to get the VM image.

2.1 Environment Configuration

You need to use our provided virtual machine image for this lab. The name of the VM image that supports this lab is called `SEEDUbuntu11.04-Aug-2011`, which is built in August 2011. If you happen to have an older version of our pre-built VM image, you need to download the most recent version, as the older version does not support this lab. Go to our SEED web page (`http://www.cis.syr.edu/~wedu/seed/`) to get the VM image.

2.2 Environment Configuration

In this lab, we need three things, are of which are already installed in the provided VM image: (1) the Firefox web browser, (2) the Apache web server, and (3) the `Collabtive` project management 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 need to first start the web server using the following command:

```
% sudo service apache2 start
```

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. We have also created several user accounts on the `Collabtive` server. To see all the users' account information, first log in as the admin using the following password; other users' account information can be obtained from the post on the front page.

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

Configuring DNS. We have configured the following URL needed for this lab. To access the URL , the Apache server needs to be started first:

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

The above URL is only accessible from inside of the virtual machine, because we have modified the `/etc/hosts` file to map the domain name of each URL to the virtual machine's local IP address (`127.0.0.1`). You may map any domain name to a particular IP address using `/etc/hosts`. For example you can map `http://www.example.com` to the local IP address by appending the following entry to `/etc/hosts`:

```
127.0.0.1      www.example.com
```

If your web server and browser are running on two different machines, you need to modify `/etc/hosts` on the browser's machine accordingly to map these domain names to the web server's IP address, not to `127.0.0.1`.

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/`.

2.3 Turn Off the Countermeasure

PHP provides a mechanism to automatically defend against SQL injection attacks. The method 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"`.

2.4 Note for Instructors

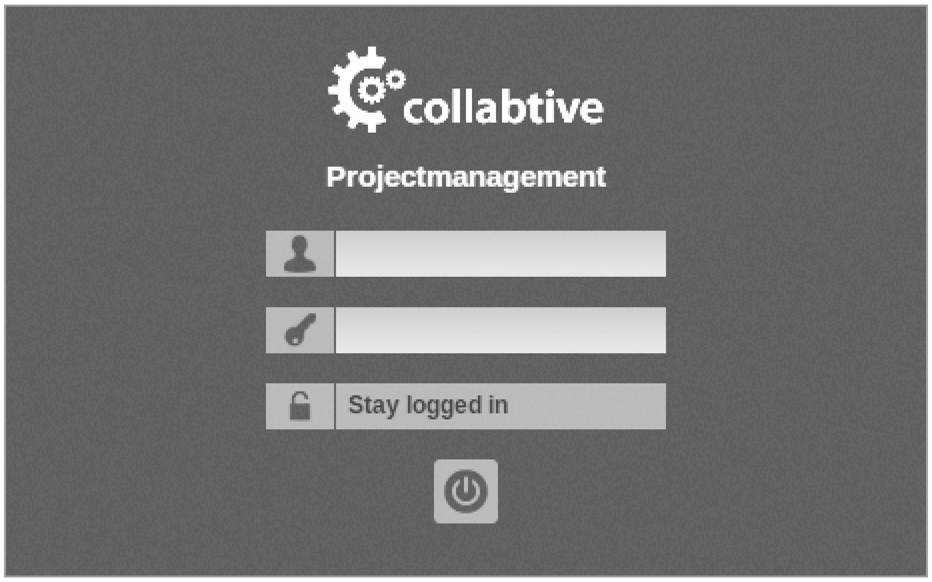
If the instructor plans to hold lab sessions for this lab, we suggest that the following background materials be covered in the lab sessions:

1. How to use the virtual machine, Firefox web browser, the `LiveHTTPHeaders` and `Tamper Data` add-ons.
2. Brief introduction to SQL: only needs to cover the basic structure of the SELECT, UPDATE, and INSERT statements. A useful online SQL tutorial can be found at `http://www.w3schools.com/sql/`.
3. How to operate the MySQL database (only the basics). The account information about the MySQL database can be found in the "*User Manual of the Pre-built Ubuntu 9 Virtual Machine*", which can be downloaded from our SEED web page.
4. Brief introduction to PHP: only needs to cover the very basics. Students who have a background in C/C++, Java, or other language should be able to pick up this script language quite quickly.

3 Lab Tasks

3.1 Task 1: 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.

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?

- **Task 1.1**: Can you log into another person's account without knowing the correct password?
- **Task 1.2**: 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.

3.2 Task 2: 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.

3.3 Task 3: 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.

- **Task 3.1: Escaping Special Characters using `magic_quotes_gpc`.** In the PHP code, if a data variable is the string type, it needs to be enclosed within a pair of single quote symbols ('). For example, in the SQL query listed above, we see `name = '$user'`. The single quote symbol surrounding `$user` basically "tries" to separate the data in the `$user` variable from the code. Unfortunately, this separation will fail if the contents of `$user` include any single quote. Therefore, we need a mechanism to tell the database that a single quote in `$user` should be treated as part of the data, not as a special character in SQL. All we need to do is to add a backslash (\) before the single quote.

PHP provides a mechanism to automatically add a backslash before single-quote ('), double quote ("), backslash (\), and NULL characters. If this option is turned on, all these characters in the inputs from the users will be automatically escaped. To turn on this option, go to `/etc/php5/apache2/php.ini`, and add `magic_quotes_gpc = On` (the option is already on in the VM provided to you). Remember, if you update `php.ini`, you need to restart the Apache server by running `"sudo service apache2 restart"`; otherwise, your change will not take effect.

Please turn on/off the magic quote mechanism, and see how it help the protection. Please be noted that starting from PHP 5.3.0 (the version in our provided VM is 5.3.5), the feature has been DEPRECATE[1], due to several reasons:

- Portability: Assuming it to be on, or off, affects portability. Most code has to use a function called `get_magic_quotes_gpc()` to check for this, and code accordingly.
- Performance and Inconvenience: not all user inputs are used for SQL queries, so mandatory escaping all data not only affects performance, but also become annoying when some data are not supposed to be escaped.

- **Task 3.2: 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, called `mysql_real_escape_string()`, 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.

- **Task 3.3: 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

[1]In the process of authoring computer software, its standards or documentation, `deprecation` is a status applied to software features to indicate that they should be avoided, typically because they have been superseded. Although deprecated features remain in the software, their use may raise warning messages recommending alternative practices, and deprecation may indicate that the feature will be removed in the future. Feature are deprecated- rather than immediately removed-in order to provide backward compatibility, and give programmers who have used the feature time to bring their code into compliance with the new standard.

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 Guidelines

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("can't 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 (SEEDUbuntu11.04-Aug-2011), Firefox is already upgraded to version 5.0, and the `"Tamper Data"` add-on is already installed.

5 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 screen shots. You also need to provide explanation to the observations that are interesting or surprising.