SQL Injection Assignment

Course: Software Security (CS 377, Drexel University 2017)

Authors: Marcello Balduccini & Ankush Israney

Date: 05/03/2017

This document outlines the steps taken to design, set-up and successfully implement a SQL injection assignment as part of course work in Software Security (CS 377) offered at Drexel University in Winter, 2017. The Assignment was set up on Drexel CCI's open stack cloud platform with assistance from Drexel's professional IT department. The students could not access the VM other than the designated apache server IP address through Drexel's VPN. The instructor(s) were allowed to access the VM through configured SSH keys. The assignment was designed as an experimentation of a timed assignment with gradual deduction of points with each passing day post the first 2 days of the start date.

Dependencies

The dependencies.txt file in the current folder outlines the steps taken to successfully install all dependencies to successfully implement this assignment. The ubuntu VM (16.0.4) with SSH capabilities, a LAMP (Apache2.0, PHP, My-SQL) Server with basic shell-scripting were required to perform this assignment.

Design & Credentials (for set-up)

The high-level Software (Assignment) design and credentials used for set-up are part of the assignmentDesign.txt File. This file contains detailed description of the expectations of this assignment from the view of both the instructor and the student.

The database connectivity configuration parameters should be changed (in retrieve.php & calculate.php) as per the designated server and My-SQL credentials which would be set up. The bind-address field in /etc/mysql/mysql.conf.d/mysql.cnf should also be changed accordingly.

Database Set-Up & Table Contents

A standard database Application exists with the following tables:

1) Users:
    a. Create string columns "Login" and "SHA1Password"
    b. "Users" table has a single user "me"

c. Set user's SHA1Password to something that is not a SHA1-produced string, e.g. "ABC".

2) RoomAssignments:
   a. Create string columns "Course" and "Room"
   b. Use Rush rooms and CS courses. (Course and Room Numbers from Winter Term Computer Science 2017 were used).

To set-up the same database for all students, we have a students.txt file which contains the official student ID's of all students registered in the course. This is done so that each student remains in his/her own assigned database (as per his/her validated Student ID) while doing the assignment.

To create a database for each student then, createdb.sh script file is used.

To run the file, use sudo bash ./createdb.sh with root privileges.

createdb.sh:

```
# Script to be used with Bash

#!/bin/bash

# If file not found
if [ ! -f 'dump.sql' ]

#Create a back-up of Application
sudo mysqldump --user=root --password=toor Application > dump.sql;
fi

#File students.txt
FILE='students.txt'

#If file not found
if [ ! -f $FILE ]
then
      #Print file does not exist
      echo "FILE $FILE NOT FOUND"
else
      #Create dummy Application'ID' db for TA
      sudo mysqladmin --user=root --password=toor create
Application14057308;

      #Import from dump.sql using < operator.
      sudo mysql --user=root --password=toor Application14057308 <
dump.sql;

      #while read p(pointer) from file
      while read p;do
```

```
#create Application ($p with pre-processing) Database
sudo mysqladmin --user=root --password=toor create
Application$"$(echo -e "$p" | tr -d '[:space:]')";

#Import from dump.sql using < operator
sudo mysql --user=root --password=toor Application$"$(echo
-e "$p" | tr -d '[:space:]')" < dump.sql;
```

#While done

```
done<$FILE fi
```

deletedb.sh:

deletedb.sh follows the same outline as created.sh, the only difference being that we use the command-line:

```
sudo mysqladmin --user=root --password=toor --force DROP Application…;
```

(including dropping the import command-lines for dump.sql.)

queryid.sh:

queryid.sh counts the number of database names in the MySQL system starting with "Application" (by including the queryid.sql script file).

```
#!/bin/bash

sudo mysql --user=root --password=toor < queryid.sql;
```

queryid.sql:

```
SELECT COUNT(*) FROM INFORMATION_SCHEMA.SCHEMATA WHERE SCHEMA_NAME
LIKE 'Application%';
```

generate600.sql:

Generate600.sql generates 600 pairs of alphanumeric (including both lower and upper case) questions and answers to these questions. The columns are output in the file numbers600.txt and are in the form:

FIB<TAB><Question><TAB>ACK<SHA1(Question)>

SHA1 is used here to avoid any plagiarism scenarios in which students can report the correct answer without doing the assignment. This generation of the answer and its relevant details are not provided to the students.

```
#/! /bin/bash
```

 # bash generate random alphanumeric strings 600 Times

```
#bash generate random 6 character alphanumeric strings
i=0;
while [[   $i    -lt   600   ]]; do
      #1 line of word_length=6, generate alphanumeric strings
      NEW_UUID=$(cat /dev/urandom | tr -dc 'a-zA-Z0-9' | fold -w 6 |
head -n 1)

      #Generate sha1 of question and cut out any trailing spaces. (The same sha1 is generated
      #For the exact same string each time)
      sha1_value=$(echo -n $NEW_UUID | sha1sum | awk '{print $1}')

      #print FIB<TAB><Question><TAB>ACK<SHA1(Question)> in > numbers600.txt
      echo "FIB"$'\t'$NEW_UUID$'\t'"ACK"$sha1_value
      let i=i+1;done >numbers600.txt
```

The file numbers600.txt is used for picking random question-answer pairs and interfacing it with Drexel's blackboard. Blackboard then scores students based on the question it supplies to each student and the corresponding answer they enter. (Those details are omitted here.)

It was required that each student had to provide the screen-shot of the answer displayed on the webpage with a valid description of his/her approach towards the problem. This policy was applied to eliminate possible plagiarism cases.

HTML Pages & PHP Code

Main.php:

The main page to begin the assignment is main.php. Therefore, students can enter <ip-address>/main.php to access this page. This file/page serves a dual purpose. The first purpose is so that students can enter their student ID information. The next is to display the course no information form (SQL injection part to modify their User's table field; password) & the login information form (link), once the students have entered their student ID information and proceed.

It is important that the students use SQL injection to reset their password to the sha1 of the password they wish to enter on the login.php page. In absence of this instruction, students will not be able to complete their assignment despite a successfully SQL injection.

```
<?php
///WELCOME STATEMENT
echo "<font size = '18' color = 'white'  face = 'Arial'>";
echo  "-----------WELCOME--------- </br></br>";
$picture ="background";
```

```
//echo ":Your Basic VM Set Up is Complete, Please Wait for Further
Instructions ";
echo "</font>";
?>

<?php
//HTML FORM TO TAKE IN COURSE NO VIA HTTP POST METHOD
?>


<html>
<head>
<link rel="stylesheet" href="hyperlink.css">
<style type="text/css">
     body{
          background-image: url('<?php echo $picture;?>');
          background-size: cover;
          background-repeat: no-repeat;
     }

</style>

<meta http:equiv="Content-Type" content="text/html; charset=utf-8"/>

<script type="text/javascript">

function show(value_id){
//The Student Id information must be inserted to proceed. This is important since students
//are linked to their own copy of the database which is named as <studentid>Application.
if (value_id==""){
alert("STUDENT ID MUST BE INSERTED TO PROGRESS!");
}
else{

//Initially form 1 is displayed and hidden after ID number is entered and submit button is
//pressed on form 1
document.getElementById("f1").style.display="none";

//Form 2 for course number is displayed once ID number is entered and submit button is
pressed on form 1
document.getElementById("f2").style.display="block";

//Form 3 for login button (link) is displayed once ID number is entered and submit button is
//pressed on form 1
document.getElementById("f3").style.display="block";

//Student ID number is passed to both form 2 and form 3
document.getElementById("student_id1").value = value_id;
document.getElementById("student_id2").value = value_id;
}
```

```
}
</script>


<title> Main Page </title>
</head>

<body>

<font  size = "12" color = "white" face = "Arial">
```

//Form 1 (STUDENT ID input)

```
<form id ="f1">
DREXEL STUDENT ID: <input type = "number" name = "student_id"
style="width: 450;"  placeholder="ENTER YOUR DREXEL UNIVERSITY STUDENT
ID HERE" required/><BR>
PRESS SUBMIT TO ENTER YOUR ASSIGNMENT PAGE: <button type="button"
onclick="show(this.form.student_id.value);"style="width:
300;">ENTER</button>
</form>
```

//Form 2 (COURSE NO)– http post method and student_id field is hidden

```
<form id ="f2" action = "retrieve.php" method = "post" style="display:
none;">
<!--INPUT TYPE TEXT-->
<input type="hidden" name="student_id1" id="student_id1" readonly
required/><BR>
```

//*******IMPORTANT*************
//course_no. is of Type text here because we want students to exploit the SQL injection
//vulnerability over here. If we make this field numeric then students might not be able to
//exploit this flaw here. However, there are other fields in login.php where such a flaw can be
//exploited.

```
COURSE NO : CS <input type = "text" name = "course_no" style ="width:
750px" /><BR>
PRESS SUBMIT TO OBTAIN COURSE LOCATION: <input type = "submit" style
="height: 40px; width: 120px" /><BR>
</form>

<hr>
```

//Form 3 (LOGIN) – http post method and student_id field is hidden

```
<form id ="f3" action = "login.php" method = "post" style="display:
none;">
<input type="hidden" name="student_id2" id="student_id2" readonly
required/><BR>
<button type="submit" name="login" value="login" class="btn-
link">LOGIN</button>
</font>
</body>
```

```
</html>
```

Retrieve.php:

Retrieve.php is the form to retrieve location information for a particular course number. It also displays the possibility of a successful SQL injection query if more than 1 query is executed (This part can be avoided [commented out] to induce more uncertainty into the assignment).

```
<?php

//DB CONNECTION
$servername = "localhost";
$username = "client";
$password = "cs377@Client";
$dbname = "";

//CHECK IF POST METHOD HAS BEEN CALLED
if(strtoupper($_SERVER["REQUEST_METHOD"]) == "POST"){

      //CHECK IF ID IS NOT EMPTY
      if(!empty($_POST['student_id1'])){
            $dbname = "Application" . $_POST['student_id1'];
            }
      else
            echo "YOUR STUDENT ID IS EMPTY; PLEASE GO BACK AND ENTER
THE CORRECT STUDENT ID <BR>";
}
else
      echo "NO HTTP POST RECEIVED; CHECK YOUR CONNECTION";

//CREATE CONNECTION
$conn = new mysqli($servername, $username, $password, $dbname);

//CHECK CONNECTION
if ($conn->connect_error){

         die("Connection failed:" . $conn->connect_error . ":
      PLEASE ENTER A VALID STUDENT ID");
}

//CHECK IF METHOD POST IS RECEIVED
if(strtoupper($_SERVER["REQUEST_METHOD"]) === "POST"){

      //CHECK FOR COURSE NO. FIELD
      if(!empty($_POST['course_no'])){
```

//*******IMPORTANT************
//Whether course_no. is a numeric field check/validation can also be made on the server side.
//Is_numeric is a method in php to do this numeric type checking. This vulnerability has

```
//purposely been left open for students to perform the SQL injection. In an alternate version
//of this assignment students could be asked to not only perform the SQL injection but also
//fix such flaws with some guidance/hints.
            //-->if(is_numeric($_POST['course_no'])){

                  //EXECUTE QUERY
                  $course = $_POST['course_no'];
                  $query = "SELECT * FROM RoomAssignments WHERE Course =
$course";

                  //CHECK FOR ERRONEOUS RESULT
//**************IMPORTANT**************
//In this version of the assignment, mysqli_multi_query has been used to execute & check
//multiple SQL queries. If this is not used then students might not be able to very EASILY
//exploit the SQL injection due to (syntactic) restrictions posed by My-SQL.

                  if(!mysqli_multi_query($conn, $query)){

                        die('Invalid query: ' .mysql_error());
                  }


                  //PRINT OUTPUT
                  else{

                      if($result = mysqli_store_result($conn) ){

                        $row = mysqli_fetch_row($result);
                        echo "<table border = 1>";
                        echo "<tr>";

                        echo "<th>COURSE NUMBER</th>";
                        echo "<th>COURSE ROOM LOCATION</th>";
                        echo "</tr>";
                        echo "<tr>";
                        echo "<td>CS " . $row[0] . "</td>";
                        echo "<td>" . $row[1] . "</td>";
                        echo "</tr>";

                        echo "</table><br><br>";
                        if(mysqli_more_results($conn) &&
mysqli_next_result($conn))
//Display the possibility of a successful SQL injection
                              echo "THERE IS MORE THAN 1 RESULT RETURNED:
YOUR SQL UPDATE MIGHT HAVE BEEN SUCCESSFUL HERE";
                        }
                       else
                        echo "NO SUCH COURSE EXISTS IN THE DATABASE!";
                  }
            //}
```

```
            //else
                  //echo "NO SQL INJECTION SMARTY!";
      }
      else
            echo "NOTHING TO PROCESS!";
}
else
      echo "NO HTTP POST METHOD RECEIVED, CHECK YOUR CONNECTION/CODE";

mysqli_close($conn);
?>
```

Login.php:

The purpose of this page is for the students to verify their student Id information and also to enter in the corresponding database's Users table username & password. Please remember that the students had to reset the password of this table through their previous SQL injection. Although some students chose to write the SQL injection query as part of the username field of this page. **This was perfectly valid as well and students could be rewarded additional points for this insight.**

```
<?php
echo "<font size = '10' color = 'black'  face = 'times new roman'>";
echo  "LOGIN PORTAL FOR CS377:";
echo "</font>";
echo "<font size = '6' color ='red' face = 'times new roman'>";
echo "<br>*PLEASE VERFIY YOUR DREXEL UNIVERISTY ID BELOW*</br></br>";
echo "</font>";
?>

<html>
<head>
 <link rel="stylesheet" href="spacing.css">
<meta http:equiv="Content-Type" content="text/html; charset=utf-8"/>
<title>LOGIN Page </title>
</head>

<body>
```

//Calculate.php is the page to check their password and call the answers.php page.

```
<form action = "calculate.php" method = "post">
<label for="Drexel ID"><B>DREXEL ID:</label> <input name ="id" value
="<?php echo $_POST["student_id2"] ."."; ?>" readonly="readonly"/><BR>
<label for="user"><B> USERNAME:</label>     <input name = "user"
required/><BR>
<label for="password">PASSWORD:</label> <input name = "password"
required/><BR>
```

```
<label for="submit">SUBMIT:</b></label>     <input type = "submit"
style ="height: 20; width: 120px" /><BR>
</form>
</body>

</html>
```

Calculate.php:

Calculate.php checks if the username and password entered by the user is correct and if so calls the answer.php page. The name calculate.php is not entirely correct over here. The actual answer calculation is part of the answer.php page. The outline of this file is similar to the retrieve.php page with a few conceptual differences such as mysqli_multi_query.

```
<?php

//CONFIGURATION PARAMETERS
$servername = "localhost";
$username = "client";
$password = "cs377@Client";
$dbname = "";

//CHECK IF POST METHOD HAS BEEN CALLED
if(strtoupper($_SERVER["REQUEST_METHOD"]) == "POST"){

      //CHECK IF ID IS NOT EMPTY
      if(!empty($_POST['id'])){
            $dbname = "Application" . $_POST['id'];
            }
      else
            echo "YOUR STUDENT ID IS EMPTY; PLEASE GO BACK AND ENTER
THE CORRECT STUDENT ID<BR>";
}
else
      echo "NO HTTP POST RECEIVED; CHECK YOUR CONNECTION";


//CREATE CONNECTION
$conn = new mysqli($servername, $username, $password, $dbname);

//CHECK CONNECTION

if ($conn->connect_error){

         die("Connection failed:" . $conn->connect_error . ":
      PLEASE ENTER A VALID STUDENT ID");
}

//CHECK IF POST METHOD HAS BEEN CALLED
if(strtoupper($_SERVER["REQUEST_METHOD"]) === "POST"){
```

```
      //CHECK IS USER FIELD IS NOT EMPTY
        if(!empty($_POST['user'])){

                        $user = $_POST['user'];
                        $query = "SELECT * FROM Users WHERE user =
'$user'";
                        $result = mysqli_query($conn, $query);

                //QUERY IS INVALID: INTERNAL CODE ERROR
                        if(!$result){

                                die('Invalid query: ' .mysql_error());
                        }

                //QUERY EXECUTED
                        else{
                     //USERNAME IS CORRECT
                                if(mysqli_num_rows($result) > 0){

                                $row = mysqli_fetch_array($result);

                    $password = $_POST['password'];
```

//*******IMPORTANT*************

//As explained previously, the password entered here is checked with an sha1 of the
//password. Therefore, students should write their SQL injection queries to take this into
//account.

```
                    //PASSWORD IS CORRECT
                                if($row['user'] == $user &&
$row['password'] == sha1($password)){
                                echo "<font size = '10' color =
'black'  face = 'times new roman'>";
                                echo "WELCOME USER 'ME', YOU CAN PASTE
YOUR QUESTION IN THE BOX!<br><br>";
                                echo "</font>";
                                //SUBMIT FORM
                                echo "<html>";
                                echo "<head>";
                                echo "<link rel='stylesheet'
href='spacing.css'><meta http:equiv='Content-Type' content='text/html;
charset=utf-8'/>";

                                echo "<body>";
                                echo "<form action='answer.php'
method='post'>";
                                echo "     <label
for='question'><B>QUESTION:</label>  <input name = 'question' size =
100/><BR>";
```

```
                                    echo "  <label
for='submit'><B>ENTER:</label>   <input type = 'submit' style ='height:
20; width: 120px' /><BR>";
                                    echo "</form>";
                                    echo "</body></html>";
                              }

                        //PASSWORD IS INCORRECT
                                    else
                                          echo "PASSWORD INCORRECT
PLEASE GO BACK AND TRY AGAIN";
                              }

                        //USERNAME IS INCORRECT
                        else
                              echo "USERNAME IS INCORRECT";
                              }

           }
        else  //USERNAME IS EMPTY
                echo "YOUR USERNAME IS EMPTY; PLEASE GO BACK AND TYPE
THE CORRECT USERNAME";
}
//NO HTTP POST
else
        echo "NO HTTP POST RECEIVED : CHECK YOUR CONNECTION";

?>
```

Answer.php:

Answer.php file simply converts the typed question into ACK<sha1(question)> in the answer field of the page.

```
<?php

if(strtoupper($_SERVER["REQUEST_METHOD"]) === "POST"){

           if(!empty($_POST['question'])){
                echo "<font size = '14' color = 'red' face = 'times
new roman'>ANSWER IS: </font>";
                echo "ACK" . $_POST['question'];
           }
           else
                echo "GO BACK and TRY AGAIN";
}
else
     echo "NO HTTP POST METHOD; CHECK YOUR CONNECTION";?>
```

Possible EXTRA-CREDIT/THEORY Question-Answer:

Students can be asked if they are given root access, how could they possibly alleviate the scope of SQL injection attacks such as SQL injection UPDATE query from the Database perspective??

Answer

Remote SQL updates with the configured user credentials in this scenario were not required. Although this situation could be desirable (password reset for instance), we should have avoided remote SQL updates over here with the following administrative GRANT/REVOKE privileges.

For example,

REVOKE UPDATE ON *.* FROM 'client'@'localhost';

Where 'client' here is the username and 'localhost' is the server-name. This statement revokes all UPDATE privileges on the defined client.

#Please Note: This is just 1 such solution over here. Defining another user with different privileges (GRANT) would be an alternate solution. The purpose of this question is to make students think to fix code vulnerabilities at different layers of the client-server model.

Conclusion:

The assignment was successfully implemented & executed on Drexel's cloud with minimal difficulties (for instance, Cloud maintenance and downtime). Grading of the assignments was completely automated through Blackboard Learn. Minimal verification was required to maintain plagiarism checks & standards. Students gave positive feedback on the assignment and a significant portion of them scored full points although the assignment could easily be tweaked to be more difficult following the suggestions in this document.

Reference Contacts:

If you face any difficulties to follow this Assignment please contact:
Ankush Israney – ankushisraney@gmail.com
Dr. Marcello Balduccini – marcello.balduccini@gmail.com