
RAPPORT TP

Elèves:

Mathis FOUCADE
Jules PUGET
Axelle ROUZIER

Enseignant:

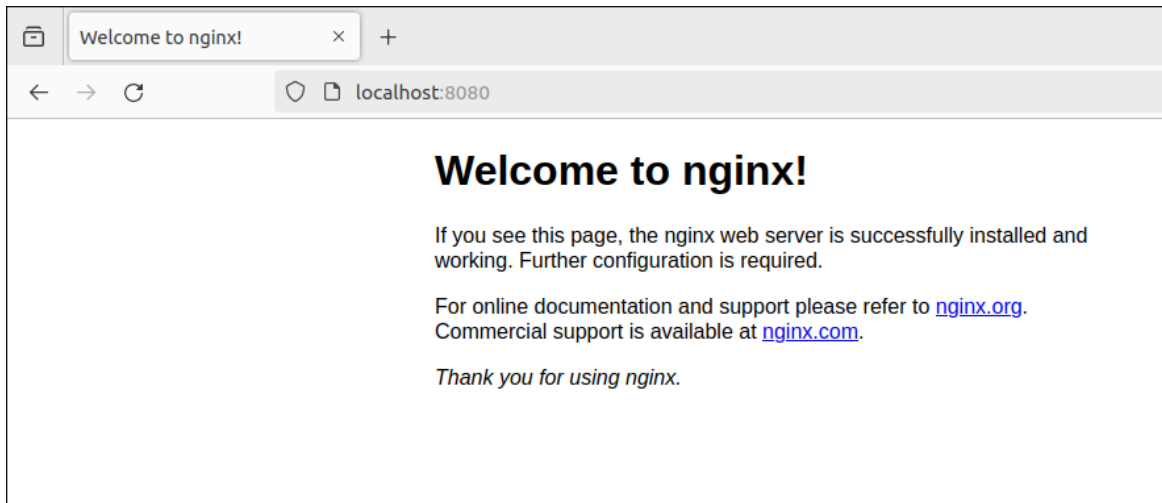
M LOUTSCH ETIENNE

Session 2: Bonnes Pratiques de Sécurité

1. Éviter l'Exposition Involontaire de Ports

```
axelle@securite-containeur:~$ docker run -d -p 8080:80 nginx  
fdc2efb3e99dfca2468a7b112dceb02fb0f9b329898e8c1c270ba30aee0ddd5e
```

```
axelle@securite-containeur:~$ ss -tuln | grep 8080  
tcp    LISTEN 0      4096      0.0.0.0:8080      0.0.0.0:*  
tcp    LISTEN 0      4096      [::]:8080      [::]:*
```



Suite à l'exécution de la commande `docker run -d -p 8080:80 nginx`, nous avons lancé un conteneur nginx qui expose son port 80 interne sur le port 8080 de notre machine hôte. La vérification avec la commande `ss -tuln | grep 8080` confirme que ce port est bien en écoute, comme le montre la sortie où nous pouvons voir deux entrées TCP en état LISTEN sur l'adresse 0.0.0.0:8080 et [::]:8080. Cela signifie que le service est accessible depuis n'importe quelle interface réseau. En accédant à l'URL localhost:8080 dans notre navigateur, nous avons pu voir la page d'accueil de nginx, confirmant que l'exposition du port fonctionne correctement. Cette exposition contrôlée est essentielle pour la sécurité, car elle permet de limiter l'accès aux services uniquement aux ports nécessaires, réduisant ainsi la surface d'attaque potentielle du système.

2. Restreindre les permissions d'accès aux fichiers sensibles

```
axelle@securite-containeur:~$ docker run -it --rm -v /etc/passwd:/mnt/passwd:ro alpine sh
/ # cat /mnt/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
```

```
/ # echo "test" > /mnt/passwd
sh: can't create /mnt/passwd: Read-only file system
```

Lors de l'exécution de la commande `docker run -it --rm -v /etc/passwd:/mnt/passwd:ro alpine sh`, nous avons monté le fichier `/etc/passwd` de l'hôte dans le conteneur Alpine en mode lecture seule (flag `:ro`). En inspectant le contenu du fichier avec `cat /mnt/passwd`, nous avons pu visualiser toutes les informations des utilisateurs du système hôte. Cependant, lorsque nous avons tenté d'écrire dans ce fichier avec `echo "test" > /mnt/passwd`, nous avons obtenu l'erreur "sh: can't create /mnt/passwd: Read-only file system", confirmant que le fichier est bien protégé contre les modifications. Cette restriction des permissions est cruciale lorsqu'on monte des fichiers sensibles dans des conteneurs, car elle empêche les processus potentiellement compromis à l'intérieur du conteneur de modifier des fichiers critiques sur le système hôte, préservant ainsi l'intégrité du système.

3. Auditer la configuration d'un container avec Docker Bench

```
axelle@securite-containeur:~$ git clone https://github.com/docker/docker-bench-security.git
Cloning into 'docker-bench-security'...
remote: Enumerating objects: 2730, done.
remote: Counting objects: 100% (990/990), done.
remote: Compressing objects: 100% (186/186), done.
remote: Total 2730 (delta 859), reused 804 (delta 804), pack-reused 1740 (from 2)
Receiving objects: 100% (2730/2730), 4.46 MiB | 863.00 KiB/s, done.
Resolving deltas: 100% (1898/1898), done.
axelle@securite-containeur:~$ cd docker-bench-security/
axelle@securite-containeur:~/docker-bench-security$
```

Section C - Score

[INFO] Checks: 117

[INFO] Score: 6

```
axelle@securite-containeur:~/docker-bench-security$ docker run -d --name vulnerable-container vulnerables/web-dvwa
e4833a2a445c7491f21654533ff71b97d6332392803cdd7733d8ef2bf1551f9a
axelle@securite-containeur:~/docker-bench-security$ sudo sh docker-bench-security.sh
```

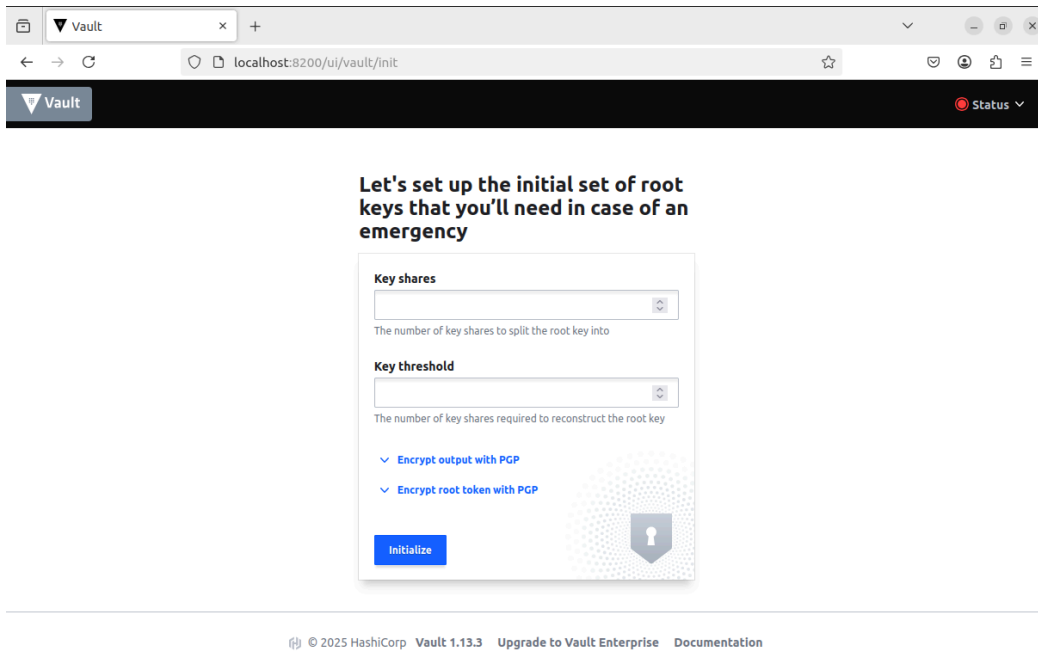
Après avoir cloné et exécuté l'outil Docker Bench for Security, nous avons obtenu un score de 6 sur une échelle non spécifiée, avec 117 vérifications effectuées. Ce score relativement bas indique que notre configuration Docker présente plusieurs vulnérabilités ou déviations par rapport aux meilleures pratiques recommandées. En comparaison, lorsque nous avons audité le conteneur *vulnerables/web-dvwa*, le score est descendu à 4 (toujours avec 117 vérifications), ce qui est logique puisque cette image est volontairement conçue avec des failles de sécurité à des fins éducatives. Les résultats montrent plusieurs problèmes potentiels comme des permissions trop larges sur les fichiers Docker, l'absence de profils AppArmor ou Seccomp, et potentiellement des conteneurs exécutés avec des privilèges élevés. Cette analyse nous permet d'identifier les domaines spécifiques nécessitant une amélioration pour renforcer la sécurité de notre environnement Docker.

4. Stocker et Utiliser des Secrets

```
axelle@securite-containeur:~$ docker run --cap-add=IPC_LOCK -e 'VAULT_LOCAL_CONFIG={"storage": {"file": {"path": "/vault/file"}}, "listener": [{"tcp": {"address": "0.0.0.0:8200", "tls_disable": true}}], "default_lease_ttl": "168h", "max_lease_ttl": "720h", "ui": true}' -p 8200:8200 vault:1.13.3 server
2025-04-26T13:35:25.578Z [INFO] proxy environment: http_proxy="" https_proxy="" no_proxy=""
2025-04-26T13:35:25.581Z [WARN] no 'api_addr' value specified in config or in VAULT_API_ADDR; falling back to detection if possible, but this value should be manually set
2025-04-26T13:35:25.778Z [INFO] core: Initializing version history cache for core
==> Vault server configuration:

        Cgo: disabled
    Environment Variables: GODEBUG, HOME, HOSTNAME, PATH, PWD, SHLV, VAULT_LOCAL_CONFIG
        Go Version: go1.20.4
        Listener 1: tcp (addr: "0.0.0.0:8200", cluster address: "0.0.0.0:8201", max_request_duration: "1m30s", max_request_size: "33554432", tls: "disabled")
        Log Level:
            Mlock: supported: true, enabled: true
        Recovery Mode: false
        Storage: file
        Version: Vault v1.13.3, built 2023-06-06T18:12:37Z
        Version Sha: 3bedf816cbf851656ae9e6bd65dd4a67a9ddff5e

==> Vault server started! Log data will stream in below:
```



Vault has been initialized! Here is your key.

Please securely distribute the keys below. When the Vault is re-sealed, restarted, or stopped, you must provide at least **1** of these keys to unseal it again. Vault does not store the root key. Without at least **1** keys, your Vault will remain permanently sealed.

Initial root token



hvs.n1qaiWYXNhPPNluQVddIPEX
4

Key 1




pM+c3I6DSnhERu6YE1dodbc1/6Y
ak0eSrBZx83L4VPM=

[Continue to Unseal](#)

[Download keys](#)

Secrets Engines

Enable new engine +		
 cubbyhole/	cubbyhole_c183e881	...
per-token private secret storage		

Terminal

[cubbyhole](#) [containers](#) [mon-secret](#)

containers/mon-secret



Secret

☐ JSON

Delete

Copy

Edit secret

Key	Value	Version	created
password	 		

Create ACL policy

Name

read-secret

Policy

```
1 path "containers/mon-secret" {
2   capabilities = ["read"]
3 }
```

Enable an Authentication Method

Generic


AppRole
☐



JWT
☐


OIDC
☐


TLS
Certificates
☐


Username &
Password
☒

Cloud


AliCloud
☐



AWS
☐



Azure
☐



Google
Cloud
☐



GitHub
☐

Infra


Kubernetes
☐


LDAP
☐


Okta
☐


RADIUS
☐

[← users](#)

Create user

Username ⓘ

Password ⓘ

▼ Tokens

```
/ # curl -s -H "X-Vault-Token: hvs.n1qaiWYXNhPPNluQVddIPEX4" http://172.17.0.1:8200/v1/cubbyhole/containers/mon-secret | jq
{
  "request_id": "662d06ce-c414-686d-2963-e9616e7bd035",
  "lease_id": "",
  "renewable": false,
  "lease_duration": 0,
  "data": {
    "password": "password"
  },
  "wrap_info": null,
  "warnings": null,
  "auth": null
}
```

Après avoir lancé le conteneur Vault, nous avons accédé à l'interface utilisateur via localhost:8200. Nous avons initialisé Vault, ce qui a généré un token root (hvs.n1qaiWYXNhPPNluQVddIPEX4). Ensuite, nous avons créé une politique ACL nommée "read-secret" qui permet uniquement la lecture des secrets stockés au chemin "containers/mon-secret", en utilisant la syntaxe *path "containers/mon-secret" { capabilities = ["read"] }*. Nous avons activé la méthode d'authentification par nom d'utilisateur et mot de passe, puis créé un utilisateur "axelle" associé à cette politique. Après avoir créé le secret (un mot de passe) au chemin spécifié, nous avons vérifié sa récupération depuis un conteneur Alpine en utilisant curl. La commande curl a été exécutée avec l'en-tête X-Vault-Token contenant le token root, ce qui a permis d'accéder au secret. Cette méthode sécurisée de gestion des secrets évite l'inclusion de données sensibles directement dans les images ou les fichiers de configuration des conteneurs, réduisant ainsi considérablement les risques de fuite d'informations.

5. Trouver la clé


```

axelle@securite-containeur:~$ docker pull ety92/demo:v1
v1: Pulling from ety92/demo
f18232174bc9: Already exists
56e759f1f975: Pull complete
Digest: sha256:700771111f9e51b5aeb717a6992fe1de3890203344648bd354f151bde230e032
Status: Downloaded newer image for ety92/demo:v1
docker.io/ety92/demo:v1
axelle@securite-containeur:~$ docker history ety92/demo:v1 --no-trunc

```

IMAGE ID	CREATED BY	SIZE	COMMENT	CREATE
sha256:f8eab655bd396df5c5b96c68ae71cc6a2a8a762c36744e2a731baea711c27554	RUN /bin/sh -c apk add curl && curl -H "API-Key: U-never-will-saw-this" -L google.com # buildkit	7.42MB	buildkit.dockerfile.v0	5 weeks ago
<missing>	CMD ["/bin/sh"]	0B	buildkit.dockerfile.v0	2 months ago
<missing>	ADD alpine-minirootfs-3.21.3-x86_64.tar.gz / # buildkit	7.83MB	buildkit.dockerfile.v0	2 months ago

Après avoir téléchargé l'image *ety92/demo:v1*, nous avons examiné son historique avec la commande *docker history ety92/demo:v1 --no-trunc*. Cette analyse a révélé une clé API exposée directement dans le code source: "U-never-will-saw-this". La clé a été incluse dans une instruction RUN du Dockerfile: *RUN /bin/sh -c apk add curl && curl -H "API-Key: U-never-will-saw-this" -L google.com # buildkit*. Cette pratique est dangereuse car toute personne ayant accès à l'image peut extraire cette information sensible, même si elle n'est pas visible dans le conteneur en cours d'exécution. Pour éviter ce type de fuite, le développeur aurait dû utiliser des variables d'environnement, les secrets Docker, ou un gestionnaire de secrets comme Vault. Une solution optimale serait d'utiliser un build multi-étapes où la clé API n'est présente que dans l'étape de build et non dans l'image finale, ou d'injecter la clé au moment de l'exécution plutôt que pendant la construction de l'image.

6. Pour les plus rapide ; Rootless mode

```
axelle@securite-containeur:~$ docker run -d --name nginx -p 80:80 nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
8a628cdd7ccc: Pull complete
b0c073cda91f: Pull complete
e6557c42ebea: Pull complete
ec74683520b9: Pull complete
6c95adab80c5: Pull complete
ad8a0171f43e: Pull complete
32ef64864ec3: Pull complete
Digest: sha256:5ed8fcc66f4ed123c1b2560ed708dc148755b6e4cbd8b943fab094f2c6bfa91e
Status: Downloaded newer image for nginx:latest
f5bce4b0b03bf467b207554cdebb6808ad1d4d7406f849eab8e983c78c904629
docker: Error response from daemon: driver failed programming external connectiv
ity on endpoint nginx (bb261dcb29a59f8e4700f4427a105956912998c56ec94ece193658b31
d053810): Error starting userland proxy: error while calling PortManager.AddPort
(): cannot expose privileged port 80, you can add 'net.ipv4.ip_unprivileged_port
_start=80' to /etc/sysctl.conf (currently 1024), or set CAP_NET_BIND_SERVICE on
rootlesskit binary, or choose a larger port number (>= 1024): listen tcp4 0.0.0.
0:80: bind: permission denied.
```

```
axelle@securite-containeur:~$ docker run -d --name nginx -p 8080:80 nginx
6df62494218139dd8922feb198bd4e11792116f128f231445120795c0ed40dd8
```

Section C - Score

```
tee: log/docker-bench-security.log: Permission denied
[INFO] Checks: 117
tee: log/docker-bench-security.log: Permission denied
[INFO] Score: 4
```

Lors de l'installation de Docker en mode rootless, nous avons tenté d'exécuter un conteneur nginx avec `docker run -d --name nginx -p 80:80 nginx`. Cette commande a échoué avec l'erreur "Error starting userland proxy: error while calling PortManager.AddPort(): cannot expose privileged port 80, you can add 'net.ipv4.ip_unprivileged_port_start=80' to /etc/sysctl.conf". Cette erreur est due au fait que le mode rootless de Docker ne permet pas, par défaut, d'exposer des ports privilégiés (inférieurs à 1024) car le démon Docker s'exécute sous un utilisateur non-root. Pour remédier à ce problème, nous pouvons soit configurer le système pour permettre aux utilisateurs non privilégiés d'utiliser ces ports en modifiant la valeur de `net.ipv4.ip_unprivileged_port_start`, soit utiliser un port non privilégié (>1024), par exemple `docker run -d --name nginx -p 8080:80 nginx`. Lors de l'exécution de Docker Bench for Security en mode rootless, le score est passé de 6 à 4. Cette baisse peut sembler surprenante, mais elle s'explique par le fait que certains contrôles de sécurité ne sont pas applicables ou donnent des résultats différents en mode rootless, notamment en raison des différences d'architecture et des permissions système.