



Veille Technologique Bi-hebdomadaire Technique

6 octobre 2026 – Fréquence : Bi-hebdomadaire (Mardi et Jeudi)

Synthèse exécutive

La SCC, CRI-O et le noyau sont trois couches de sécurité successives : un conteneur non-root peut être admis, démarré sans erreur, et perdre tout de même ses capacités.

Cas analysé le 5 octobre 2026 (Red Hat Developer Blog) : un sidecar non-root demande `CAP_NET_RAW`, la SCC l'autorise, CRI-O l'écrit dans les jeux Bounding/Effective/Permitted de la spec OCI — puis la transition d'UID 0 vers 1000 fait effacer Permitted et Effective par le noyau. CRI-O n'active délibérément ni Inheritable ni Ambient. La maîtrise du pipeline SCC → CRI-O → kernel devient une compétence de sécurité de plateforme de premier ordre.

1. Périmètre

Famille de veille : 2. Métiers, Compétences & Technologies (Indicateur Qualiopi 24).

Fenêtre de rétrospection : 29 septembre → 6 octobre 2026.

Fréquence d'exécution : Bi-hebdomadaire (Mardi et Jeudi).

Diffusion : [Radar Web Aperta](#) — [LinkedIn Aperta Scientia](#) — [X @ApertaScientia](#).

2. Sujet du jour — Article publié sur le Radar Web

Sécurité des conteneurs : pourquoi votre conteneur non-root perd ses capacités (SCC, CRI-O, kernel)

Le symptôme Un sidecar de supervision tourne en non-root et doit exécuter des sondes ICMP : il lui faut `CAP_NET_RAW`. La SecurityContextConstraint (SCC) autorise la capability, le PodSpec la demande (`capabilities.add: [NET_RAW]`), CRI-O démarre le conteneur sans erreur — et pourtant ping échoue avec `Operation not permitted`, et `CapEff` est vide (`0000000000000000`). Il ne s'agit pas d'un bug : c'est la rencontre de trois systèmes qui ne se parlent pas.

1. La SCC est un portail d'admission, pas un moteur d'exécution Une SCC détermine uniquement si une configuration de sécurité est *autorisée à entrer* dans le cluster (capabilities autorisées, plages d'UID, types de volumes, escalade de privilèges, contexte SELinux). Elle est évaluée par le contrôleur d'admission de l'API server au moment de la création du Pod. Une fois le Pod admis, le travail de la SCC est terminé : elle n'a aucune composante runtime, ne parle pas à CRI-O et ne configure aucune capability de processus. La SCC reste toutefois plus granulaire que Pod Security Admission, qui raisonne en trois niveaux grossiers (privileged, baseline, restricted).

2. CRI-O : de la SCC au spec OCI CRI-O n'a aucune notion de SCC : il reçoit un `CreateContainerRequest` et le traduit en spec OCI. Les champs simples (`hostNetwork`, `hostPID`, `hostIPC`) se mappent directement en modes de namespaces. Les capabilities passent par `SpecSetPrivileges()` et divergent selon le mode. Conteneurs privilégiés : `SetupPrivileged(true)` remplit les cinq jeux (41 capabilities connues) et le champ `dropCapabilities` du PodSpec est alors ignoré. Conteneurs non privilégiés : CRI-O fusionne 9 capabilities par défaut (`CHOWN`, `DAC_OVERRIDE`, `FSETID`, `FOWNER`, `SETGID`, `SETUID`, `SETPCAP`, `NET_BIND_SERVICE`, `KILL`) sauf si le PodSpec déclare `drop: ALL` — d'où la bonne pratique de production `drop: ALL + add: [une capability]`. CRI-O écrit ensuite la capability



finale dans Bounding, Effective et Permitted, de façon identique en root et en non-root.

3. Kernel : cinq jeux de capacités et une transition UID qui efface tout Les capacités répartissent le modèle superuser en cinq jeux : Bounding (plafond absolu), Permitted (réservoir des capacités effectives), Effective (ce que le noyau vérifie réellement), Inheritable (transmises via `execve()` si le binaire porte les file capacités) et Ambient (introduites avec Linux 4.3 pour le non-root, elles repeuplent Permitted et Effective après `execve()`). Le mécanisme fatal : le runtime OCI démarre le processus en root, applique les jeux, puis bascule vers l'UID cible (ex. 1000) — et lors de cette transition $UID\ 0 \rightarrow \text{non-zéro}$, le noyau efface Permitted et Effective. Seul Bounding subsiste, et CRI-O ne positionne jamais Inheritable ni Ambient.

4. Pourquoi CRI-O n'active pas les capacités ambiantes Ce n'est pas un oubli mais un choix de sécurité documenté dans le code : Kubernetes suppose que `runAsNonRoot` entraîne la perte des capacités, et activer les capacités ambiantes casserait ce contrat implicite. Une KEP (2763 — `ambient capabilities`) existe côté SIG Security pour les rendre supportables de façon sélective : à suivre, car elle changerait la donne pour tous les sidecars non-root à besoin de privilège fin.

5. Les remèdes, et leur coût réel File capacités (`setcap`) : l'approche la plus ciblée — `setcap 'cap_net_raw=+ep' /usr/bin/ping` au build ; le noyau promeut la capacité dans Permitted et Effective à l'exécution, même en non-root, tant qu'elle reste dans le Bounding set. Coût : modifier l'image ; les file capacités sont stockées en attributs étendus, donc le pipeline de build et le driver de stockage doivent les préserver, et la confiance se déplace vers la provenance de l'image. Exécution en root : évite la transition UID, mais la SCC de production la plus courante (`restricted-v2`) l'interdit. Conteneur privilégié : remplit les cinq jeux mais désactive SELinux, AppArmor et seccomp, et court-circuite tout le `add/drop` — à proscrire hors debug. La leçon : devant un `CapEff` nul en non-root, on ne cherche pas un bug, on relit la transition UID et la sémantique des capacités.

3. Impact sur les Cursus Aperta Scientia

Cursus AS200 — DevOps Platform Engineer

- **Sécurité des conteneurs & durcissement (D0188 / D0430)** : distinction admission (SCC/PSA) vs exécution (`CapEff`, `seccomp`, `SELinux`), lecture de `/proc/<pid>/status` et du pipeline `SCC → CRI-O → kernel`.
- **Plateforme & conformité (D0280 / D0380)** : conception de SCC, motif `drop: ALL + add`, limites du mode privilégié et arbitrages de moindre privilège.
- **Supply chain & provenance (D0288)** : file capacités et attributs étendus dans les images, préservation au build, confiance déplacée vers la provenance de l'image.

Cursus AS300 — AI Platform Engineer

- **Sécurité des charges d'inférence (AI267 / AI296)** : exécution non-root des serveurs `vLLM/KServe` et sidecars GPU, moindre privilège applicatif et exposition contrôlée des périphériques.
- **Agents & isolation** : conteneurs d'agents non-root, confinement des capacités et évaluation de la surface de privilège en production.
- **Observabilité & débogage** : diagnostic de privilèges en runtime, corrélation admission/runtime et instrumentation de plateforme.

4. Sources

- Red Hat Developer Blog — [Why your non-root container dropped its capabilities \(and how to fix](#)



it) — [Lien](#) (5 octobre 2026).

- **Linux man-pages — capabilities(7)** — [Lien](#) (consulté le 6 octobre 2026).
- **Kubernetes Enhancement Proposal 2763 — Ambient Capabilities** — [Lien](#) (consulté le 6 octobre 2026).
- **Red Hat Blog — Managing SCCs in OpenShift** — [Lien](#) (consulté le 6 octobre 2026).



Contacts & Communauté :

- **Direction des Formations** : learning@apertascientia.io
- **Référent Handicap & Bien-être (Care)** : care@apertascientia.io
- **X (Twitter)** : [@ApertaScientia](https://twitter.com/ApertaScientia)
- **Communauté Discord** : discord.gg/eDJzR5Vk85