



Bi-Weekly Technology Watch

Technology

6 October 2026 – Cadence: Bi-weekly (Tuesday and Thursday)

Executive Summary

The SCC, CRI-O and the kernel are three successive security layers: a non-root container can be admitted, start without error, and still lose every capability.

Scenario analysed on 5 October 2026 (Red Hat Developer Blog): a non-root sidecar requests `CAP_NET_RAW`, the SCC allows it, CRI-O writes it into the Bounding/Effective/Permitted sets of the OCI spec — then the UID 0 to 1000 transition makes the kernel clear Permitted and Effective. CRI-O deliberately sets neither Inheritable nor Ambient. Mastering the SCC → CRI-O → kernel pipeline is now a first-class platform security skill.

1. Scope

Watch family: 2. Jobs, Skills & Technologies (Qualiopi indicator 24).

Look-back window: 29 September → 6 October 2026.

Execution cadence: Bi-weekly (Tuesday and Thursday).

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

2. Topic of the Day — Article Published on the Web Radar

Container Security: Why Your Non-Root Container Drops Its Capabilities (SCC, CRI-O, Kernel)

The symptom A monitoring sidecar runs as non-root and needs ICMP health checks: it requires `CAP_NET_RAW`. The SecurityContextConstraint (SCC) allows the capability, the PodSpec requests it (`capabilities.add: [NET_RAW]`), CRI-O starts the container without error — and yet ping fails with `Operation not permitted`, and `CapEff` is empty (`0000000000000000`). This is not a bug: it is the collision of three systems that do not talk to each other.

1. An SCC is an admission gate, not a runtime engine An SCC only decides whether a security configuration is *allowed into* the cluster (permitted capabilities, UID ranges, volume types, privilege escalation, SELinux context). It is evaluated by the OpenShift API server admission controller when the Pod is created. Once the Pod is admitted, the SCC's job is done: it has no runtime component, does not talk to CRI-O and configures no process capability. SCCs nevertheless remain far more granular than Pod Security Admission, which works on a coarse 3-tier model (privileged, baseline, restricted).

2. CRI-O: from SCC to OCI spec CRI-O has no concept of SCCs: it receives a `CreateContainerRequest` and translates it into a runtime spec. Simple fields (`hostNetwork`, `hostPID`, `hostIPC`) map directly to namespace modes. Capabilities go through `SpecSetPrivileges()` and diverge by mode. Privileged containers: `SetupPrivileged(true)` populates all five sets (41 known capabilities) and the PodSpec's `dropCapabilities` field is then ignored. Non-privileged containers: CRI-O merges in 9 default capabilities (`CHOWN`, `DAC_OVERRIDE`, `FSETID`, `FOWNER`, `SETGID`, `SETUID`, `SETPCAP`, `NET_BIND_SERVICE`, `KILL`) unless the PodSpec declares `drop: ALL` — hence the production best practice `drop: ALL + add: [one capability]`. CRI-O then writes the final capability into Bounding, Effective and Permitted, identically for root and non-root.



3. Kernel: five capability sets and a UID transition that wipes them Linux capabilities spread the superuser model across five sets: Bounding (absolute ceiling), Permitted (pool for effective capabilities), Effective (what the kernel actually checks), Inheritable (carried across `execve()` when the binary carries matching file capabilities) and Ambient (introduced in Linux 4.3 for non-root, repopulating Permitted and Effective after `execve()`). The fatal mechanism: the OCI runtime starts the process as root, applies the sets, then switches to the target UID (e.g. 1000) — and during that UID 0 to non-zero transition, the kernel clears Permitted and Effective. Only Bounding survives, and CRI-O never sets Inheritable or Ambient.

4. Why CRI-O skips ambient capabilities This is not an oversight but a documented security decision in the code: Kubernetes assumes that `runAsNonRoot` results in capabilities being dropped, and enabling ambient capabilities would break that implicit contract. A KEP (2763 — ambient capabilities) exists in SIG Security to make them selectively supportable: worth tracking, since it would change the game for every non-root sidecar needing fine-grained privilege.

5. The real fixes, and what they cost File capabilities (`setcap`): the most targeted approach — `setcap 'cap_net_raw=+ep' /usr/bin/ping` at build time; the kernel promotes the capability into Permitted and Effective at exec, even for non-root, as long as it remains in the Bounding set. Cost: you must modify the image; file capabilities are stored as extended attributes, so the build pipeline and storage driver must preserve them, and trust shifts to image provenance. Running as root: avoids the UID transition, but the most common production SCC (`restricted-v2`) forbids it. Privileged container: populates all five sets but disables SELinux, AppArmor and seccomp, and bypasses the entire add/drop logic — never outside debugging. The lesson: faced with a zero `CapEff` on a non-root process, you are not hunting a bug, you are re-reading the UID transition and capability semantics.

3. Impact on Aperta Scientia Curriculums

AS200 Curriculum — DevOps Platform Engineer

- **Container security & hardening (D0188 / D0430):** distinguishing admission (SCC/PSA) from execution (`CapEff`, seccomp, SELinux), reading `/proc/<pid>/status` and the SCC → CRI-O → kernel pipeline.
- **Platform & compliance (D0280 / D0380):** designing SCCs, the `drop: ALL + add` pattern, the limits of privileged mode and least-privilege trade-offs.
- **Supply chain & provenance (D0288):** file capabilities and extended attributes in images, preservation at build time, trust shifted to image provenance.

AS300 Curriculum — AI Platform Engineer

- **Inference workload security (AI267 / AI296):** non-root execution of vLLM/KServe servers and GPU sidecars, application least privilege and controlled device exposure.
- **Agents & isolation:** non-root agent containers, capability confinement and privilege-surface evaluation in production.
- **Observability & debugging:** runtime privilege diagnosis, admission/runtime correlation and platform instrumentation.

4. Sources

- **Red Hat Developer Blog — Why your non-root container dropped its capabilities (and how to fix it)** — [Link](#) (5 October 2026).



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



Contacts & Community:

- **Academic Programs:** learning@apertascientia.io
- **Disability Officer & Student Care:** care@apertascientia.io
- **X (Twitter):** [@ApertaScientia](#)
- **Discord Community:** discord.gg/eDJzR5Vk85