Ref: Best practice proposal: avoid static reference to external resources · Issue #244 · cncf/cnf-wg (github.com)

(Wording in italics is from an existing best practice, for info, and needs removing.)

Summary

Containers consist of an image, and some configuration. This configuration might be details relating to the execution of the container, such as labels or resource requests/demands. Or it might be parameters or data to pass into the created container for the software to use.

Where a container requires external resources such as IP addresses, storage volumes, etc. these resources should be described declaratively (what outcome is desired) rather than imperatively (how that outcome is achieved). This allows for container orchestrators to dynamically manage how the desired outcome is achieved.

~~*Containers have a list of their own users independent of the host system, one of which is UID 0, the root user. Containers should run processes as a user other than root which makes it easier to run the container images securely.*~~

Motivation

This best practice benefits the CNF developer, improving the quality of the CNF and reducing the developer's likelihood of having to diagnose problems with the CNF. Validation of the best practice is the responsibility of the CNF developer.

Indirectly, improved CNF quality benefits CNF operators. The proposed tests are runnable by CNF operators as acceptance tests.

SE-Linux based environments will require dropping root privileges. Example: OpenShift

Goals

Avoiding root in containers can help to:

- Improve the security and behaviour of applications.
- Add to the defense in depth strategy against external compromises.
- Avoid compromised apps from causing more damage.

Non-Goals

This BP recommends that the application not use the UID that can override all protections. This means that file read and write protections can be established.

It does not consider what filesystem write permissions should be in order to benefit from those protections. CNF developers will additionally want to ensure filesystem permissions are tightened up appropriately so that non-root users are prevented from doing damage. This is outside the BP scope.

Proposal

When building a container, the container should be built to run its processes as a non-root user. setsid processes should not be required to do the work inside a container.

A container's root user has fewer Linux Kernel capabilities and may be distinct from the platform's root (if the container runtime enables user namespaces remap feature).

However, the container's root user does have full read/write access to the container's filesystem. It can read or modify any file. No secrets can be kept from it; it cannot be prevented from changing the content of all executable files on the system.

On a basic level, avoiding the root user means that the container filesystem permissions are enforceable against all processes running in the system. Those processes can be prevented from doing critical things like:

- viewing secrets they should not be viewing
- modifying binaries within the filesystem that will later be executed

Obviously, a well-written CNF would not be attempting to do things it should not do. But all software has bugs. Also, executing processes can be compromised by outside forces, and if this happens filesystem protection is a part of a "Defense in depth" strategy to ensure the compromise does not escalate.

User/group access enforcement will be respected. As an added advantage, fine-grained access enforcement, such as in SELinux, will also hold

Workload Context

All pod types should implement this best practice.

User Stories (Optional)

Supply chain attack user stories

[Supply chain attacks](../user-stories/supply-chain-attacks.md) are a risk at any point in the supply chain. 'Defence in depth' says that we should (a) defend against supply chain attacks but also (b) add mitigations in the case that supply chain attacks happen.

Examples include

- [A CNF downloads compromised updates](../user-stories/supply-chain-attacks.md#a-cnf-downloads-compromised-updates)
- [A CNF succumbs to code injection](../user-stories/supply-chain-attacks.md#a-cnf-succumbs-to-code-injection)
- [A CNF succumbs to malicious instructions](../user-stories/supply-chain-attacks.md#a-cnf-succumbs-to-malicious-instructions)
- [A CNF has a security-compromising bug](../user-stories/supply-chain-attacks.md#a-cnf-has-a-security-compromising-bug)

In all of these examples, the CNFs using a non-root user for their container processes, have limited the scope of damage a compromised process may cause.

See main [defense in depth for supply chain attacks](../user-stories/supply-chain-attacks.md) document for more information.

Notes/Constraints/Caveats (Optional)

Container images are frequently built from upstream image versions made from OS deployments. These will include things like setsid binaries as a part of their base configuration. If CNF developers follow this best practice they will have to audit and clean up any upstream images to respect this rule (removing files, removing packages or changing permissions as appropriate).

By default, the first process starting in a container runs as root - you have to actively take steps to shed the permissions (Dockerfile USER line, plus installing files with appropriate ownership within the Dockerfile).

We specifically want the process to run as a non-root user so that its access is limited. Of course, if access is limited then the CNF developer must ensure that access is still available to the things the CNF is going to need to access. This will involve changing permissions on data files and on working directories when the container image is constructed (they cannot be changed on startup because the application does not have the right to do that). This may also affect the use of shared volume mounts or host mounts - ownership of and rights on the root directory must permit access to the container users.

References

- [CNF WG discussion](https://github.com/cncf/cnf-wg/discussions/20)
- TAG Security: [Cloud Native Security Whitepaper - Least Privilege](https://github.com/cncf/tag-security/blob/main/security-whitepaper/v2/cloud-native-security-whitepaper.md#least-privilege)
- Sysdig: [Top 20 Dockerfile best practices](https://sysdig.com/blog/dockerfile-best-practices/) - 2021/03/09
- [Best practices for pod security in Azure Kubernetes Service (AKS)](https://docs.microsoft.com/en-us/azure/aks/developer-best-practices-pod-security) - 2020/07/28
- [RedHat OpenShift Runtime Security Best Practices](https://www.openshift.com/blog/openshift-runtime-security-best-practices)
- K8s Documentation - Securing a Cluster: [Controlling what privileges containers run with](https://kubernetes.io/docs/tasks/administer-cluster/securing-a-cluster/#controlling-what-privileges-containers-run-with)
- [Security Best Practices for Kubernetes Deployment](https://kubernetes.io/blog/2016/08/security-best-practices-kubernetes-deployment/) - 2016/08/31
- [Docker and Kubernetes — root vs. privileged](https://itnext.io/docker-and-kubernetes-root-vs-privileged-9d2a37453dec) - 2020/06/25

- Bitnami on [Use Non-Root Containers](https://docs.bitnami.com/kubernetes/faq/configuration/use-non-root/) - 2020/05/13
- [Running non-root containers on Openshift](https://engineering.bitnami.com/articles/running-non-root-containers-on-openshift.html)
- K8s 11 Ways Not to Get Hacked: [8. Run Containers as a Non-Root User](https://kubernetes.io/blog/2018/07/18/11-ways-not-to-get-hacked/#8-run-containers-as-a-non-root-user)
- Red Hat PDF [A PRACTICAL INTRODUCTION TO CONTAINER SECURITY](https://www.redhat.com/files/summit/session-assets/2017/L99901-apracticalintroductiontocontainersecurity_labguide.pdf) - 2017/05
- [CVE-2019-5736: runc container breakout](https://seclists.org/oss-sec/2019/q1/119)
- [CVE-2022-0492: Linux kernel cgroups v1 missing capabilities check when setting release_agent](https://seclists.org/oss-sec/2022/q1/123)
- [Using the rootless containers in RHEL](https://www.redhat.com/en/blog/using-rootless-containers-tech-preview-rhel-80) 2019/08
- [Understanding root inside and outside a container](https://www.redhat.com/en/blog/understanding-root-inside-and-outside-container) 2019/12

Alternatives (Optional)

These are not strictly alternatives as they can be used with non-root, but can be applied to a container running as root.

- Disable all capabilities or limit them
- Do not run the container in privileged mode

Related items include

- [Rootless](https://www.docker.com/blog/experimenting-with-rootless-docker/) containers as seen with [usernetes](https://github.com/rootless-containers/usernetes)
- Alternative runtimes like [Kata Containers](https://katacontainers.io/) for a different approach to security

Testing Objectives

An application which follows this best practice will not have any containers with processes running as root

This CBPP will be tested by the CNF Test Suite.

Static analysis

A container image can be tested for compliance:

- The container metadata should indicate that the first process started is started as a non-root user
- The container filesystem will not have setsid-root binaries.

If available, the Dockerfile can be checked to see if a non-root user is used. (Dockerfiles are not the only way to build containers, and the Dockerfile may not be a part of the CNF deliverable.)

- See USER and RUN commands, both of which allow the Dockerfile author to express which user is to be used when launching the container.

The above static analysis definitively confirms that a container cannot elevate privileges to local root as it removes all avenues for doing so.

Runtime analysis

One can check for processes launched as, or running as container root.

We offer the following applications as examples that operators might wish to evaluate for this purpose:

- [Cnitch](https://github.com/nicholasjackson/cnitch) periodically checks the list of running containers in a Docker environment to see if any are running as container root.
- [Falco](https://falco.org/) checks for processes running as container root if the non-root container policy is set.

Scanning systems that periodically check running processes or may not identify all root-owned processes, as it must conduct a scan at the moment a process is running. Similarly, process monitoring will not identify a problem if behaviour requiring a root process is not triggered. This cannot be used as a definitive guarantee of safety but is useful as a secondary check.

Scoring

This best practice results in a pass/fail on two counts, depending on role.

Static analysis (all items checked for a pass) - CNF developers (testing before delivery) or CNF operators (testing what is delivered):

- Container images indicate their processes should be started as a user other than 0
- Container images should not contain setsid-root binaries (user 0, u+s)

Runtime analysis - CNF operators:

- Operators may use runtime verification, from outside the application, to confirm that containers in processes are not owned by container root