Alpaquita Linux
Keeping containers secure

Alpaquita Linux
Revision 1.0
January 2024

bellsoft

BellSoft software contains open source software. Additional information about third party code is available at https://bell-sw.com/third_party_licenses. You can also get more information on how to get a copy of source code by contacting info@bell-sw.com.

The use of any software product referenced in this document is governed by the applicable license agreement, which is not modified in any way by the terms of this notice.

Contents

1. Overview

This document provides several security recommendations and details that can help you secure containers and execution environments.

> **Note:**
>
> The examples in this document are mostly based on Podman, but you can apply all the recommendations to both Docker and Podman setup.

2. Setting up Host

The following guidelines help you set up and maintain your host securely.

- Regularly update the container host system, such as kernel and other software components. Make sure that kernel-based features related to kernel namespaces, private networking, and control groups are up-to-date with all available fixes.
- Configure the container host system to use a minimal operating system setup and apply all security best practices. Ideally such systems must be set up only to host containers and not used for anything else.

  Generally, we recommend reducing the number of services running on the same system to the required minimum. If some services are needed for the work process, consider moving all other services to run within containers controlled by Podman or transfer them to other host systems.

Set up audit to track activities

One of the most important security measures on Linux hosts is to conduct proper audit of sensitive activities on the system. The instrument that can help with tracking such activities is the audit support in Linux kernel and corresponding userspace tools/packages. The audit allows system administrators to monitor security sensitive events and report them in a log, such as `audit.log`. This log file can be created on a local or remote system for better security protection.

Due to Podman implementation to use `fork/exec` to run containers, the audit feature works more correctly and stores necessary data in the log files. In Docker, the `auditd auid` is unset in the log, which means a system administrator can see that a process associated with a container has accessed security sensitive information, but the identity is not recorded in the log. In Podman, `auid` is properly recorded in the audit log file.

For instance, a `cat` coreutils executable was used to access sensitive information and Podman provides all necessary information about an identity while in Docker `auid` is in an unset state.

```
type=SYSCALL msg=audit(02/04/23 20:31:56.759:5) : arch=x86_64 syscall=open
success=yes exit=3 a0=0x7ffe5a873ecb a1=O_RDONLY a2=0x0 a3=0x0 items=1
ppid=2531 pid=2685 auid=test uid=root gid=root euid=root suid=root fsuid=root
egid=root sgid=root fsgid=root tty=pts0 ses=unset comm=cat
exe=/usr/bin/coreutils key=(null)
```

It is recommended to use only Podman to set up container host environments and to use audit for tracking access to sensitive information.

Setting up an audit in Alpaquita requires installing the audit package and starting the audit service as follows.

```
sudo rc-service auditd start
```

Use cgroups

To properly set up resource management, the system must be configured to use Control Group v2 (**cgroup v2**). Consider performing the following actions to use cgroup v2.

Set up UIDs

Set up `/etc/subuid` and `/etc/subgid`.

Podman launches a container inside the user namespace, which is mapped to the range of UIDs defined for the user in `/etc/subuid` and `/etc/subgid`. Update those files for each user who is allowed to create containers. If you update either `/etc/subuid` or `/etc/subgid`, all running containers owned by affected users should be stopped. This can be done automatically by using the following command that stops all containers for a user and kills a pause process:

```
podman system migrate
```

Fix warnings about absent systemd

Alpaquita Linux does not use `systemd` and you may observe the following output if you run Podman on Alpaquita Linux:

```
WARN[0000] "/" is not a shared mount, this could cause issues or missing mounts
with rootless containers.
```

Run the following command to fix the setup.

```
sudo mount –make-rshared /
```

Install fuse-overlayfs

Note that `fuse-overlayfs` package is not installed by default with the Podman package. Install the

package using the `apk add` command.

```
sudo apk add fuse-overlayfs
```

Enable cgroups

Run the following commands.

```
sudo rc-update add cgroups
sudo rc-service cgroups start
```

We highly recommend using cgroups v2 that can be set by editing `/etc/rc.conf`. Assign the `unified` option to `rc_cgroup_mode`, and enable controllers. You might need to run the following command for the changes to take effect.

```
sudo rc-service cgroups restart
```

Use crun runtime

To limit resources for cgroups v2, use the corresponding OCI runtime in the setup. We recommend setting up Podman's `crun` runtime as the default runtime to use.

Edit `/etc/containers/containers.conf` file and change `Default OCI runtime` value to be always `crun`.

Load additional Kernel modules

Load the following modules explicitly, because they are not loaded by the default.

```
sudo modprobe tun
sudo modprobe fuse
```

AppArmor

We recommend using AppArmor in the container host setup based on Alpaquita Linux. Proper AppArmor configuration helps prevent malicious container escapes and protects host sensitive resources.

3. Execute without root privileges

A healthy security practice is to never run containers with root privileges, because the root privileges are applied to access sensitive resources on a host system itself. Since containers with applications can be downloaded from the internet or have some security flaws, it is dangerous to let them have root privileges.

In the majority of cases there is no need to run containers with root privileges. There are specific cases depending on applications where running a container as root makes sense. One of the reasons could be when containers need to access specific mounts, devices on the host, or need to listen on ports less than 1024 on the host network.

Therefore, the most typical execution mode would be rootless and while Podman supports it out of the box, it becomes the main choice for setting up container environments.

By design, rootless Podman runs as root within the container unless it is reconfigured. This policy means that all processes in the container have the default list of namespaced capabilities that allow the processes to act like root inside the user namespace, including changing their UID and changing the user and group ownership of files that are mapped into the user namespace.

4. Set up namespaces

Podman takes advantage of user namespaces, so that root within the container is mapped to a non-root UID on the host. This setup allows Podman to safely install packages and run services from within the container without impacting the host.

Administrators can use user namespace to set a user identifier (UID) and group identifier (GID) mapping for running a container. This means that a process can run as UID 0 inside the container and as UID 123456 outside the container. In other words if a container process goes outside the container, the Linux kernel will treat that process as UID 123456.

The example `--uidmap` setting instructs Podman to map a range of 5000 UIDs inside the container, starting with UID 100000 outside the container (so the range is 100000-104999) to a range starting at UID 0 inside the container (so the range is 0-4999). If a process is running as UID 1 inside the container, it is 100001 on the host.

```
sudo podman run -d bellsoft/alpaquita-linux-base sleep 1000
sudo podman top --latest user huser
USER        HUSER
root        root

sudo podman run --uidmap 0:100000:5000 -d bellsoft/alpaquita-linux-base sleep
1000
sudo podman top --latest user huser
USER        HUSER
root        100000
```

You can use the following namespace types:

- Mount (mnt): isolates mount points;
- Process ID (pid): isolates process IDs;
- Network (net): isolates network stack;
- Interprocess Communication (ipc): isolates interprocess communication resources;
- UTS: isolates hostnames and domain names;
- User ID (user): isolates user and groups IDs;
- Control groups (cgroups): isolates cgroups;
- Time: isolates time.

5. Container images and registries

It is important to verify that Podman images are received and deployed unchanged from a source registry with a trusted reputation and validated authentication. When pulling images from remote sources, ensure that the connection is protected and that HTTPS is used for the pull request. It is unsafe to use insecure image registries that are not protected by TLS. Note the following recommendations.

- Pull images by their fully-qualified names instead of a shortened name to avoid a possibility of pulling from a different registry.
- You can configure Podman to work with only trusted images from a remote registry if those images are signed and the signatures that can be validated against a local public key.
- Images can be signed similarly as packages and a Podman host can be configured to require that images from a remote registry are signed and validated before being used locally.
- Create reliably reproducible images from container files and required packages. Ensure that new images use base images and the software that you have is properly reviewed for security vulnerabilities:
    - Define a fixed version of the base image in an image Container File;
    - Define fixed versions of the package pulls in build steps of an image Container File;
    - Ensure that package pulls in the build steps use trusted and verified sources and repositories.
- Reduce to a minimum the number of packages installed on images.

We do not recommend installing unnecessary packages to new image builds. Proper review of Container Files helps remove unnecessary installation steps, and the images are used for their main purpose.

6. Other resources and security considerations

Limit CPU and memory usage

Use the `-m` option to limit memory and the `-c` option to limit CPU.

Limit container file access

When creating and launching containers, limit file access using the `-v <host dir>:<container dir>:ro` option or `--readonly` flag. It is safer to explicitly create volumes for applications running inside a container. Monitoring file changes in these volumes can help prevent security breaches. Volumes that are dedicated for container write access must be cleaned up regularly. Here is an example of how to use the `:ro` option and mount a host directory in a way that the host directory/file is read only for a container:

```
podman run -v /host_directory:/container_directory:ro bellsoft/alpaquita-linux-base
```

`host_directory` is the host directory/files to be mounted as a volume and become available in `container_directory` in the container. We strongly discourage you from mounting the following sensitive host system directories at container runtime. `/`, `/boot`, `/dev`, `/etc`, `/lib`, `/usr`, `/sys`, `/proc`, and so on.

Limit container restarts

Malicious or accidental denial-of-service might happen to a container that produces many errors, so limiting container restarts is a good practice and can be done using the `--restart=on-failure:N` option when creating or launching a container.

Limit networking access from containers

It is an excellent practice to limit network access from a container unless it is needed for running certain applications. When publishing ports to the host, specify the IP address of the interface that a port will be bound to so that the attack surface is reduced to the network interface where the container should be listening to. Podman publishes to all interfaces (0.0.0.0) by default if an IP address is not specified when using the `--publish` option.

Review the following recommendations:

- Do not run SSH inside containers;
- Do not map privileged ports (< 1024) inside containers;
- Do not use the `--net=hostname` mode option when starting a container. This option gives the container full access to local system services and is insecure.

Review Kernel capabilities in containers

The following kernel capabilities are usually granted to a container by default. Review the capabilities and disable unnecessary ones.

```
- CHOWN
- DAC_OVERRIDE
- FSETID
- FOWNER
- NET_RAW
- SETGID
- SETUID
- SETFCAP
- SETPCAP
- NET_BIND_SERVICE
- SYS_CHROOT
- KILL
```

The `--privileged` option disables the security measures used for isolating a container from the host, so it should not be used unless absolutely required. This option removes barriers from isolated capabilities, limited devices, read-only mount points and volumes, and so forth.

Monitor any malicious resource usage in container

Podman includes the necessary features to monitor container resource usage, such as memory consumption, CPU time, I/O, and network usage. Monitoring container resource usage for performance, error detection, and abnormal behavior like suspicious traffic or unexpected user activity, helps to detect security flaws as well.

Clean a container host system regularly

Images and containers that are not needed should be removed from the host system to avoid image and container garbage and to protect from the accidental execution of an old, unused image, or container that might have possible security flaws. Podman has the `auto-update` option to automate the process of updating for new image/container versions according to their auto-update policy.

Limit system calls in containers

By default, Podman containers limit the system calls available to containers based on the calls defined in the `/usr/share/containers/seccomp.json` file. This list is valid for general purpose containers and is compatible with most containers, so there is no need to add more system calls.

However, you can narrow down system calls needed for a particular container to run. Create the seccomp generated profile using the following command:

```
sudo podman run --annotation io.containers.trace-syscall=”of:<file_path>”
<other_options> <container_name>
```

You can reuse it later with the `--security-opt seccomp=<file_path>` option.

Inspect 'runlabel' before execution

Podman has a useful feature to create shortcuts for `podman run` with all necessary metadata, so the actual container execution can be done as follows:

```
podman container runlabel <my_label> <image_name>
```

Before actual execution, we recommend checking such shortcuts by executing the `runlabel --display` option to avoid any malicious `podman run` command. The option does not execute the

label command, but only displays what will be executed.

Secure remote API usage

Podman has a remote API implementation called `varlink` that works over a socket. The socket configuration is similar to a Unix socket, restricting it to local use only. Security consideration is to run this socket as a non-root user especially if a container has no requirement to run as a privileged user. A socket can be created by running the following Podman command:

```
podman varlink --timeout=0 unix:/run/user/$(id -u)/podman/io.podman
```

By creating a socket with Podman command, you ensure that proper permissions are set on the socket.

Alpaquita Linux
Keeping containers secure

bellsoft