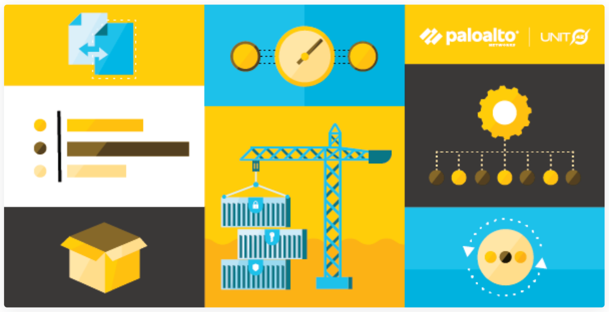
New Vulnerability Affecting Container Engines CRI-O and Podman (CVE-2021-20291)

34,903 people reacted

21 4 min. read

SHARE

By Aviv Sasson
April 14, 2021 at 6:00 AM
Category: Cloud, Unit 42
Tags: containers, CRI-O, CVE-2021-20291, Kubernetes, Podman LXC Container Security, vulnerabilities



This post is also available in: 日本語 (Japanese)

Executive Summary

As part of our initiative to improve security in the cloud-native landscape, I conducted a security audit of multiple Go libraries that Kubernetes is based on. In my research, I found CVE-2021-20291 in containers/storage that leads to a Denial of Service (DoS) of the container engines CRI-O and Podman when pulling a malicious image from a registry. Through this vulnerability, malicious actors could jeopardize any containerized infrastructure that relies on these vulnerable container engines, including Kubernetes and OpenShift.

Palo Alto Networks customers running Prisma Cloud are protected from this vulnerability through the Prisma Cloud Compute host vulnerability scanner and the Trusted Images feature.

Disclosure Process

We responsibly disclosed the vulnerability on March 10, 2021, and a fix was released on version 1.28.1. Corresponding fixes were released in CRI-O version v1.20.2 and Podman version 3.1.0.

On some platforms, depending on the user settings, the update can be downloaded automatically or it needs to be downloaded manually. We encourage the community to check their software version and update it in case it is not up to date.

We would like to thank the Red Hat security team for their prompt response to this issue and for assigning CVE-2021-20291 for the vulnerability.

How Container Images Are Pulled

To understand how this vulnerability works, we need to understand what happens when a container engine pulls an image from a registry.

The first step of pulling a container image is downloading its manifest. Each image has a manifest, a file containing instructions on how to build the image. This includes information such as the image's operation system and its CPU architecture. In this blog, I focus on the layers array, which is a list contained in the manifest that consists of layers that compose the container file system. When pulling the image, one of the operations is that the container engine reads that list and downloads, decompresses and untars each layer.

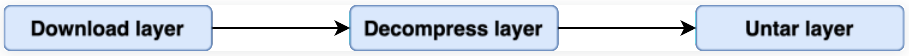


Figure 1. Retrieving layer flow.

An adversary could upload to the registry a malicious layer that aims to exploit the vulnerability and then upload an image that uses numerous layers, including the malicious layer, and by that create a malicious image. Then, when the victim pulls the image from the registry, it will download the malicious layer in that process and the vulnerability will be exploited.

The Vulnerability: CVE-2021-20291

Execution Flow

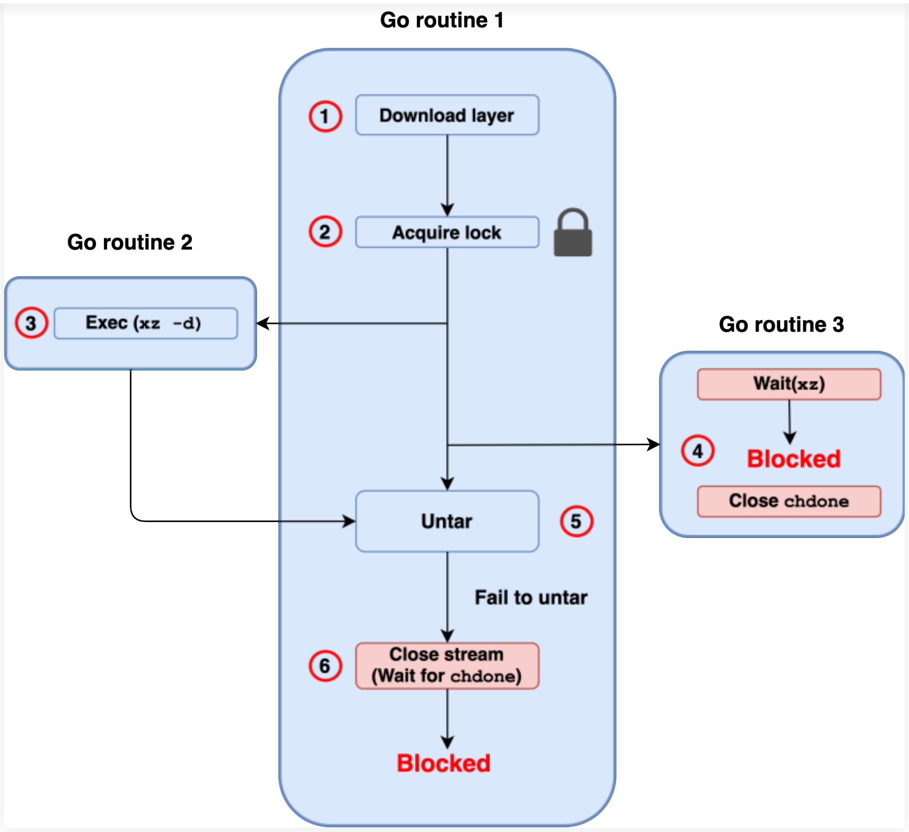


Figure 2. Vulnerability execution flow.

Figure 2 illustrates how the vulnerability works after the container engine is requested to download a malicious image and starts downloading the malicious layer. The end result is a deadlock, which is a situation in which a lock is acquired and never gets released. This causes a DoS since other threads and processes stop their execution and wait forever for the lock to be released.

- Routine 1 - Downloads the malicious layer from a registry.
- Routine 1 - Acquires a lock.
- Routine 2 - Decompresses the downloaded layer using the `xz` binary and writes the output to stdout.
- Routine 3 - Waits for `xz` to exit and for all the data in stdout to be read. When the conditions are met, it continues and closes a channel called `chdone`.
- Routine 1 - Uses the output of `xz` as input and tries to untar the data. Since the file is not a tar archive, untar fails with "`invalid tar header`" and doesn't finish reading the rest of the data from xz's stdout. Since the data will never be read, routine 3 is now deadlocked and will never close `chdone`.
- Routine 1 - Waits for routine 3 to close `chdone` and therefore is also deadlocked.

Now that routine 1 is deadlocked, the container engine cannot execute any new requests because in order to do so, it needs to acquire the lock on step 2, which will never be freed.

Impact

The vulnerability detailed above lies in the containers/storage. This library is used by CRI-O and Podman to handle storage and download of container images. When triggering the vulnerability, their vital functionality breaks down. Some of the impacts I was able to observe after triggering this DoS vulnerability include:

CRI-O

1 - Fails to pull new images.

2 - Fails to start any new containers (even if they are already pulled).

3 - Fails to retrieve local images list.

4 - Fails to kill containers.

Podman

1 - Fails to pull new images.

2 - Fails to retrieve running pods.

3 - Fails to start new containers (even if they are already pulled).

4 - Fails to exec into containers.

Kubernetes

As of Kubernetes v1.20, Docker is deprecated and the only container engines supported are CRI-O and Containerd. This leads to a situation in which many clusters use CRI-O and are vulnerable. In an attack scenario, an adversary may pull a malicious image to multiple different nodes, crashing all of them and breaking the cluster without leaving a way to fix the issue other than restarting the nodes.

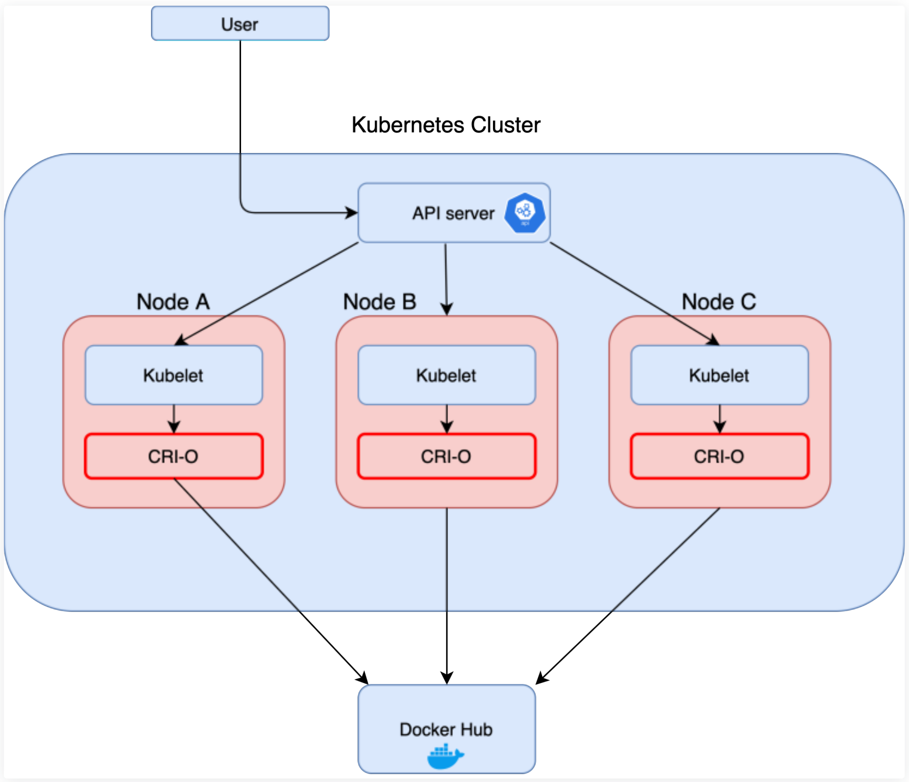
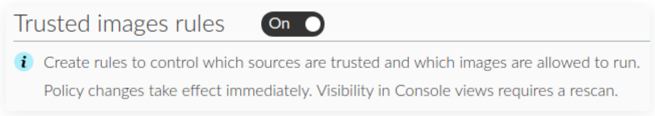


Figure 3. Kubernetes attack vector.

Conclusion

In the last several years, multiple security audits were conducted by Unit 42 and others on cloud applications and container infrastructure. Thanks to these audits, many security issues are being identified and addressed. As the transition to the cloud is rapidly increasing, adversaries will continue to shift their focus to the cloud. We are committed to supporting cloud infrastructure by conducting security audits, finding critical vulnerabilities and encouraging organizations to address them by issuing patches and applying them.



Trusted images rules — On

Create rules to control which sources are trusted and which images are allowed to run. Policy changes take effect immediately. Visibility in Console views requires a rescan.

Figure 4. Prisma Trusted Images feature.

Palo Alto Networks customers running Prisma Cloud are protected from this vulnerability through the Prisma Cloud Compute host vulnerability scanner and the Trusted Images feature.