Rootless Containers

Giuseppe Scrivano / Red Hat (@gscrivano)

Akihiro Suda / NTT (@_AkihiroSuda_)
Who are we?

Giuseppe Scrivano
• Software Engineer at Red Hat
• Works on Podman, Buildah, CRI-O

Akihiro Suda
• Software Engineer at NTT (the largest telco in Japan)
• Maintainer of Moby (former Docker Engine), BuildKit, containerd, and etc...
Demo: “Usernetes”

Kubernetes as a non-root user
Introduction
Rootless Containers

• “Rootless containers refers to the ability for an unprivileged user (i.e. non-root user) to create, run and otherwise manage containers.” (https://rootlesscontaine.rs/)

• Not just about running containers as an unprivileged user

• Also entails running container runtimes and orchestrators as an unprivileged user
Don’t confuse with..

- `docker run --user foo`
  - Executes the process in the container as a non-root
  - `dockerd`, `containerd`, and `runc` still running as the root

- **USER** instruction in Dockerfile
  - same as above
  - Notably you can’t `RUN dnf install ...`
Don’t confuse with..

• `usermod -aG docker foo`
  - Allows a non-root user to connect to `/var/run/docker.sock`
  - Equivalent to allow the user to gain the root! (docker run --privileged -v /:/host ...)

• `sudo docker` or `chmod +s dockerd`
  - Nope!
Don’t confuse with..

- **`dockerd --userns-remap`**
  - Execute containers as a non-root user (**dockremap**), using user namespaces
    - Inside the containers, **dockremap** can behave as if it were the root
    - Most similar to Rootless Containers, but still requires dockerd, containerd, and runc to run as the root
Motivation of Rootless Containers

• To mitigate potential vulnerability of container runtimes and orchestrator (the primary motivation)

• To allow users of shared machines (e.g. HPC) to run containers without the risk of breaking other users environments

• To isolate nested containers, e.g. “Docker-in-Docker”
Runtime vulnerabilities

• Docker “Shocker” (2014)
  – A malicious container was allowed to access the host file system, as `CAP_DAC_READ_SEARCH` was effective by default

• Docker CVE-2014-9357
  – A malicious `docker build` container could run arbitrary binary on the host as the root due to an LZMA archive issue

• containerd #2001 (2018)
  – A malicious container image could remove `/tmp` on the host when the image was pulled (not when actually launched!)
Runtime vulnerabilities

• runc #1962 (2019, found by Akihiro, analyzed and fixed by Giuseppe)
  - A malicious container could gain the write access to `/proc` and `/sys` when the host root filesystem is initrd (`DOCKER_RAMDISK`)
  - Results in arbitrary command execution as the root on the host, via `/proc/sys/kernel/core_pattern` or `/sys/kernel/uevent_helper`
  - Minikube is known to be affected (fixed in v0.33.1)

```
$ kubectl run -it --image busybox foo
# unshare -mrfp
# mount -t proc none /proc
```
Other vulnerabilities

- Kubernetes CVE-2017-1002101, CVE-2017-1002102
  - A malicious container was allowed to access the host filesystem via vulnerabilities related to volumes

- Kubernetes CVE-2018-1002105
  - A malicious API call could be used to gain `cluster-admin` (and hence the root privileges on the nodes)

- Git CVE-2018-11235 (affected Kubernetes `gitRepo` volumes)
  - A malicious repo could execute an arbitrary binary as the root when it was cloned
Play-with-Docker.com vulnerability

• Play-with-Docker.com: Online Docker playground, implemented using Docker-in-Docker with custom AppArmor profiles

• Malicious kernel module was loadable due to AppArmor misconfiguration (revealed on Jan 14, 2019)
  – Not really an issue of Docker

https://www.cyberark.com/threat-research-blog/how-i-hacked-play-with-docker-and-remotely-ran-code-on-the-host/
Caveat: Not a panacea

- Although Rootless Containers could mitigate these vulnerabilities, it is not a panacea, especially it is powerless against kernel (and hardware) vulnerabilities

- Castle approach 🏯: it should be used in conjunction with other security layers such as seccomp and SELinux
Implementation details
User Namespaces

• The key component of rootless containers.
  – Map UIDs/GIDs in the guest to different UIDs/GIDs on the host.
  – Unprivileged users can have (limited) root inside a user namespace!

• Root in a user namespace has UID 0 and full capabilities, but obvious restrictions apply.
  – Inaccessible files, inserting kernel modules, rebooting, ...
User Namespaces

• To allow multi-user mappings, shadow-utils provides `newuidmap` and `newgidmap` (packaged by most distributions).
  – SETUID binaries writing mappings configured in `/etc/sub[ug]id`

```
/proc/42/uid_map:
  0  1000  1
  1 420000 65536
```

/`etc/subuid`:
  1000:420000:65536

Provided by the admin (real root)

User can configure map UIDs after unsharing a user namespace
User Namespaces

Problems:

- **SETUID binary can be dangerous**
  - `newuidmap` & `newgidmap` had two CVEs so far:
    - CVE-2016-6252 (CVSS v3: 7.8): integer overflow issue
    - CVE-2018-7169 (CVSS v3: 5.3): supplementary GID issue
  - **Hard to maintain** `subuid` & `subgid`
    - Having 65536 sub-IDs should be ok for most cases, but to allow nesting user namespaces, an enormous number of sub-IDs would be needed
      - Potential sub-ID starvation
User Namespaces

Alternative way: Single-mapping mode

• Single-mapping mode does not require `newuidmap/newgidmap`
• There is only one UID/GID available in the container

Limit the privileges of `newuidmap/newgidmap`

• Install them using file capabilities rather than SETUID bit
  – Only CAP_SETUID and CAP_SETGID are needed
Network Namespaces

- An unprivileged user can create network namespaces along with user namespaces

- With network namespaces, the user can
  - create iptables rules
  - isolate abstract (pathless) UNIX sockets
  - set up overlay networking with VXLAN
  - run tcpdump
  - ...

redhat.
Network Namespaces

• But an unprivileged user cannot set up `veth` pairs across the host and namespaces, i.e. No internet connection
Network Namespaces

Prior work: LXC uses SETUID binary (lxc-user-nic) for setting up the veth pair across the host and containers.

Problem: SETUID binary can be dangerous!

- CVE-2017-5985 (CVSS v3: 3.3): netns privilege escalation
- CVE-2018-6556 (CVSS v3: 3.3): arbitrary file open(2)
Our approach: use completely unprivileged usermode network (“Slirp”) with a TAP device

send fd as SCM_RIGHTS cmsg via an UNIX socket
Network Namespaces

Benchmark of several “Slirp” implementations:

<table>
<thead>
<tr>
<th></th>
<th>MTU=1500</th>
<th>MTU=4000</th>
<th>MTU=16384</th>
<th>MTU=65520</th>
</tr>
</thead>
<tbody>
<tr>
<td>vde_plug</td>
<td>763 Mbps</td>
<td>Unsupported</td>
<td>Unsupported</td>
<td>Unsupported</td>
</tr>
<tr>
<td>VPNKit</td>
<td>514 Mbps</td>
<td>526 Mbps</td>
<td>540 Mbps</td>
<td>Unsupported</td>
</tr>
<tr>
<td>slirp4netns</td>
<td>1.07 Gbps</td>
<td>2.78 Gbps</td>
<td>4.55 Gbps</td>
<td>9.21 Gbps</td>
</tr>
<tr>
<td>cf. rootful veth</td>
<td>52.1 Gbps</td>
<td>45.4 Gbps</td>
<td>43.6 Gbps</td>
<td>51.5 Gbps</td>
</tr>
</tbody>
</table>

- slirp4netns (our own implementation based on QEMU Slirp) is the fastest because it avoids copying packets across the namespaces

Benchmark: iperf3 (netns -> host), measured on Travis CI. See rootless-containers/rootlesskit#12
Port forwarding

- Usermode port forwarder (inbound connection) can be implemented independently of Slirp (outbound connection)
  - `slirp4netns`: 7.01 Gbps (still has extra packet copy)
  - `socat+nsenter`: 9.64 Gbps
  - Our WIP implementation with `SCM_RIGHTS` optimization: 28.5 Gbps (still flaky)

https://github.com/rootless-containers/rootlesskit/pull/33#issuecomment-448992291
https://github.com/rootless-containers/slirp4netns/pull/29#issuecomment-449626896
Multi-node networking

• Flannel VXLAN is known to work
  – Encapsulates Ethernet packets in UDP packets
  – Provides L2 connectivity across rootless containers on different nodes

• Other protocols should work as well, except ones that require access to raw Ethernet
Your container root filesystem has to live somewhere. Many filesystem features used by “rootful” container runtimes aren’t available.

- Ubuntu allows overlayfs in a user namespace, but this isn't supported upstream (due to security concerns).

- Btrfs allows unprivileged subvolume management, but requires privileges to set it up beforehand.

- Devicemapper is completely locked away from us.
A “simple” work-around is to just extract images to a directory!
- It works … but people want storage deduplication.

Alternatives:
- Reflinks to a "known good" extracted image (inode exhaustion).
  - (Can use on XFS, btrfs, ... but not ext4.)
- Unprivileged userspace overlayfs using FUSE (Kernel 4.18+).
fuse-overlayfs

- Overlayfs implementation using FUSE
- Layers deduplication as for root containers
- Fast setup for a new container
- Built-in support for shifting UIDs/GIDs

- Adds complexity
- Temporary solution until unprivileged users can safely use overlay
When creating a user namespace, we must ensure proper ownership of the files in the RO layers.

The file system “lies” about the owner, so that it has the correct UID/GID in the user namespace and the same layer on disk can be used by different user namespaces.

Less expensive alternative to `cp -r` and `chown`'ing the entire image and layers.
cgroups

/sys/fs/cgroup is a roadblock to many features we want in rootless containers (accounting, pause and resume, even getting a list of PIDs!).

- By default completely owned by root (and managed by systemd).

Some workarounds:

- LXC’s pam_cgfs requires installation of a PAM module (and only works for logged-in users). It needs to be used carefully as it gives cgroupv1 write access to unprivileged users.
- cgroup namespaces (with nsdelegate) only work in cgroupv2.
Current adoption status: 
Runtimes
runc

• Supports rootless mode since 1.0.0-rc4 (merged March 2017).

• Since 1.0.0-rc5 (Feb 2018):
  – /sys/fs/cgroup can be used if they are set up to be writable.
  – Multi-user mappings are supported if they are set up with /etc/sub[ug]id.
Podman, daemonless alternative to Docker:

- Uses slirp4netns
- Uses fuse-overlayfs
- rootless storage under the user home directory
- No CLI differences between root and rootless mode
Podman containers

• When running directly a container, each container runs in its own user namespace.

• No mounts/resources leak in the host

• A container cannot join namespaces of another container as they are owned by different user namespaces
Podman pods

• Similar concept to Kubernetes pods

• A group of containers that share resources

• Deploy as a single unit

• Rootless containers in a Pod share the same user namespace
Docker

• Docker v19.03 is likely to support Rootless mode
  – PR: #38050

• Unlike Podman, fuse-overlayfs is not yet supported
LXC

• Supports unprivileged (what we call “Rootless”) containers since 2013 🇨🇳

• Unlike our work, a SETUID binary is required for setting up network namespaces

• LXD still requires the daemon to be executed as the root
Singularity

• Popular in HPC community, as it supports Rootless mode
  - Default configuration uses a SETUID helper (which we don’t call “Rootless”), but Rootless mode (\texttt{--userns}) can be enabled optionally

• Unlike our work, Rootless mode does not support creating network namespaces (with Internet connection)
CloudFoundry Garden

• Supports rootless mode using runc

• Unlike our work, SETUID binaries are required for setting up network namespaces

https://github.com/cloudfoundry/garden-runc-release/blob/master/docs/articles/rootless-containers.md
runROOTLESS and umoci

- runROOTLESS: runc-based OCI runtime (Akihiro’s work)
- umoci: OCI image manipulation tool

- No subuid/subgid configuration is needed
  - Emulates subuid/subgid with \texttt{ptrace} and \texttt{xattr}
  - Suitable for LDAP environments

- Current implementation has significant overhead due to \texttt{ptrace}
  - Future work: replace \texttt{ptrace} with Tycho Andersen’s new seccomp framework (Kernel 5.X ?)
udocker

• Docker-like CLI

• Supports both `ptrace` mode and `runc` mode
  - Unlike `runROOTLESS`, `ptrace` mode lacks support for persistent `chown` (also, `ptrace` mode isn’t used in conjunction with `runc`)

[Image]
Current adoption status:
Image builders
• Buildah: daemonless tool for building OCI images
  – User namespaces for rootless mode or root in a not privileged container
  – Uses slirp4netns
  – dnf/yum/apt/apk works
  – Can use fuse-overlayfs
  – Share configuration and storage with Podman
  – Different isolation modes
Buildah isolation modes

- Can be controlled with `--isolation=ISOLATION`
  - OCI (default): OCI compatible configuration
  - rootless: It is similar to OCI but uses a configuration that is usable for non privileged users
  - chroot: creates an environment that looks more like a chroot than a container.
BuildKit and img

• **BuildKit**: modern backend for `docker build`
  - Integrated to Docker since v18.06, but can be also used as a standalone and rootless daemon
  - Rootless BuildKit has been used in OpenFaaS cloud

• **img**: Jessie Frazelle’s image builder based on BuildKit
  - Same as BuildKit but daemonless
BuildKit and img

- Rootless BuildKit/img can be launched as an unprivileged user on the host without any extra configuration.

- However, containerized deployment of rootless BuildKit/img had required `securityContext.procMount=Unmasked`
  - Unmask `/proc/*` (e.g. `kcore`) so that build containers can mount `procfs` with dedicated PID namespaces.
  - Not real concern as long as running in rootless mode.
  - BuildKit v0.4+ no longer requires `securityContext` configuration.
    - But no PID namespace isolation across the BuildKit daemon container and build containers.
Kaniko

- Google’s unprivileged container image builder
- Different from our approach
  - Kaniko itself needs to be executed in a container
    (No securityContext configuration needed)
  - Dockerfile RUN instructions are executed without creating nested containers inside the Kaniko container
    • A RUN instruction gains the root in the Kaniko container
- Seems inappropriate for malicious Dockerfiles due to the lack of isolation
  - Potential cloud credential leakage: #106
Makisu

• Uber’s unprivileged container image builder

• Same design as Kaniko, with regard to unprivileged execution
Current adoption status: Kubernetes
Kubernetes

• kubelet and kube-proxy require a bunch of hacks for running without cgroups and sysctl
  – No hack needed for kube-apiserver and kube-scheduler
  – POC available; Planning to propose KEP to SIG-node soon
  – Future work: kubeadm integration

• CRI: Both CRI-O and containerd supports rootless mode

• CNI: Flannel VXLAN is known to work without any modification
“Usernetes”

Experimental binary distribution of rootless Kubernetes, installable under `$HOME` without mess

https://github.com/rootless-containers/usernetes

$ tar xjvf usernetes-x86_64.tbz
$ cd usernetes
$ ./run.sh

$ ./kubectl.sh run -it --image..
“Usernetes”

- `docker-compose.yml` is included for demonstrating pseudo multi-node cluster POC
  - Mix of dockershim + CRI-O + containerd
  - Flannel VXLAN is enabled by default
  - FIXME: TLS is not enabled yet 😞 (contribution wanted!)

- Usernetes-on-Kubernetes YAML is coming soon
Any questions?