THE LINUX FOUNDATION
OPEN SOURCE SUMMIT

# The State of Rootless Containers

Aleksa Sarai / SUSE
Akihiro Suda / NTT

@lordcyphar
@_AkihiroSuda_

THE LINUX FOUNDATION

# Who are we?

Aleksa Sarai

- Senior Software Engineer at SUSE.
- Maintainer of runc and several other Open Container Initiative projects.

Akihiro Suda

- Software engineer at NTT (the largest telco in Japan)
- Maintainer of Moby (former Docker Engine), BuildKit, containerd, and etc...

# Agenda

- What are Rootless Containers? What are they for?
  - User Namespaces
  - Network Namespaces
  - Mount Namespaces
  - cgroups
  - Current adoption status
- Demo: “Usernetes”

# Introduction to Rootless Containers

- Most container runtimes* require root privileges.
  - ... and lack sufficient protections against privilege escalation.
- What can you do if you don't have (and can't get) root privileges?
  - (Computing clusters in universities for example.)

- Rootless containers are containers that can be created and managed **without privileged codepaths** *(some caveats apply)*.
  - Requires quite a few kernel technologies, as well as some userspace tricks...

# “The Security Argument”

Another justification is to avoid privileged codepaths entirely:

- No privilege escalation if you never actually have privileges! docker:CVE-2014-9357 docker:CVE-2015-3629 docker:CVE-2015-3627
- Configuration mistakes cannot escalate privileges above the original user. docker:CVE-2016-8867
- Path traversal vulnerabilities only affect paths the user can already access. docker:CVE-2015-3630 k8s:CVE-2017-1002101 k8s:CVE-2017-1002102 docker:CVE-2018-15664

(This is not a panacea, the kernel features we use have had security flaws in the past -- especially user namespaces. But you can also restrict their usage inside rootless containers!)

# User Namespaces

- The key component of rootless containers.
  - Map UIDs/GIDs in the guest to different UIDs/GIDs on the host.
  - Unprivileged (on the host) users can have (limited) root inside!
- Root has UID 0 and full capabilities, but obvious restrictions apply.
  - Inaccessible files, inserting kernel modules, rebooting, ...
- Unprivileged users can map only their own UID/GID (to itself or root).
  - We need something better to be able to use package managers.

# User Namespaces

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

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

Provided by the admin (real root)

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

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 64K 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 (up to 4G entries) starvation, especially in LDAP environments with many users

# User Namespaces

Alternative way: Single-mapping mode + Ptrace + Xattr

- Single-mapping mode does not require `newuidmap`/`newgidmap`
- Ptrace can emulate fake sub-UIDs/sub-GIDs
  - No need to hook all syscalls (unlike gVisor)
  - Seccomp could be used as well in future
- Xattr (extended file attributes) can be used for persistent `chown(2)` emulation (see `user.rootlesscontainers`).

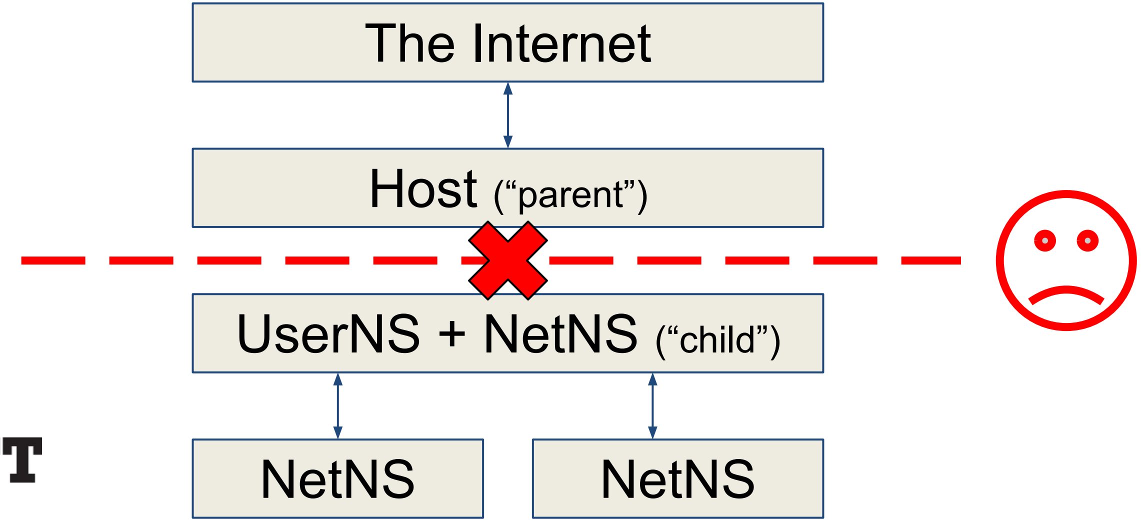
Free from potential `newuidmap`/`newgidmap` CVEs

- But slow and no real isolation across sub-UIDs/sub-GIDs
- Almost adequate for image building purpose, but not panacea

# Network Namespaces

An unprivileged user can create network namespaces by acquiring the root in a user namespace, but cannot set up the veth pair across the parent and the child (i.e. No internet connection)

- Note: isolating network namespace is not mandatory (but no iptables, bridges, no namespaced abstract UNIX sockets)



# Network Namespaces

Prior work: LXC uses SETUID binary (`lxc-user-nic`) for setting up the veth pair across the parent and the child

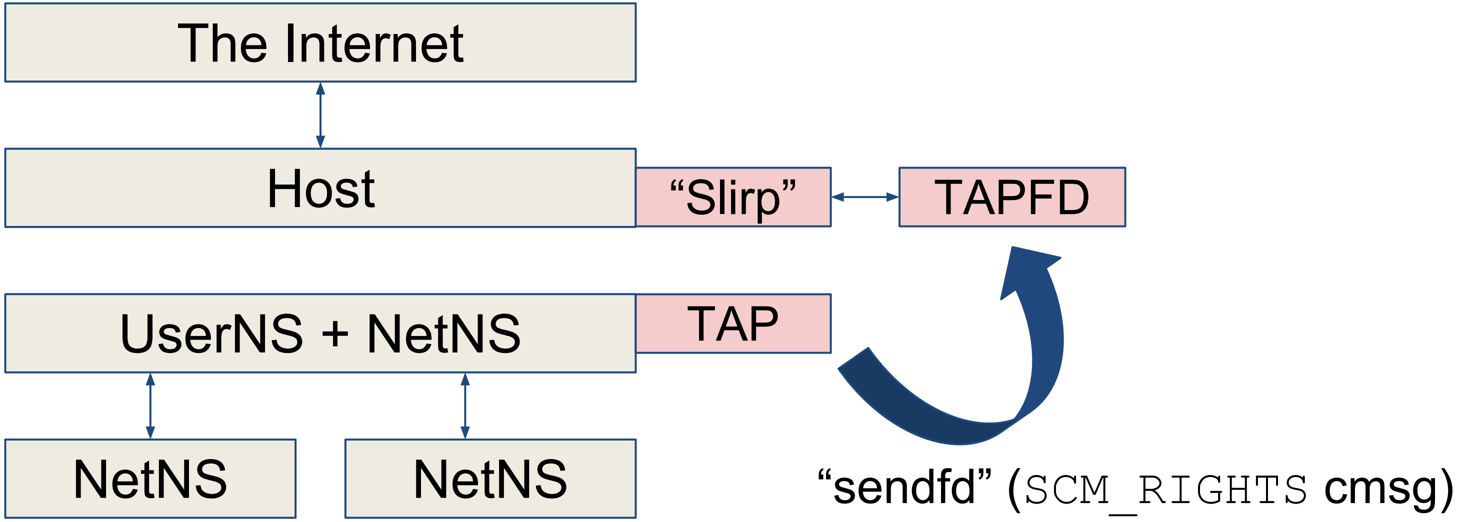
Problem: SETUID binary can be dangerous!

- CVE-2017-5985 (CVSS v3: 3.3): netns privilege escalation
- CVE-2018-6556 (NEW! disclosure: 8/10/2018): arbitrary file `open(2)`

# Network Namespaces

Our approach: use usermode network (“Slirp”) with a TAP device

- Completely unprivileged



“sendfd” (`SCM_RIGHTS` cmsg)

# Network Namespaces

Benchmark of several “Slirp” implementations:

| | MTU=1500 | MTU=4000 | MTU=16384 | MTU=65520 |
|---|---|---|---|---|
| **vde_plug** | 763 Mbps | Unsupported | Unsupported | Unsupported |
| **VPNKit** | 514 Mbps | 526 Mbps | 540 Mbps | Unsupported |
| **slirp4netns** | 1.07 Gbps | 2.78 Gbps | 4.55 Gbps | 9.21 Gbps |
| cf. rootful veth | 52.1 Gbps | 45.4 Gbps | 43.6 Gbps | 51.5 Gbps |

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

Benchmark: iperf3 (netns -> host), measured on Travis CI

See rootless-containers/rootlesskit#12

# Network Namespaces

Setting up `/etc/resolv.conf` (without chroot) is mess…

- `resolv.conf` may point to `127.0.0.X` (for systemd-resolved / dnsmasq)
- But `127.0.0.X` DNS is unaccessible from network namespaces
- We can use bind-mount for replacing `resolv.conf`, but it is often forcibly unmounted by systemd-resolved / NetworkManager

Solution: isolate `/etc`

- Mount an empty tmpfs on `/etc`
- Create the new `resolv.conf` on the new `/etc`
- Create symlinks for the real `/etc/*`, except `resolv.conf`

# Root Filesystems

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.

# Root Filesystems

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 family.)
- Unprivileged userspace overlayfs using FUSE (Linux >=4.18).

(Container images themselves have significant flaws as well.)

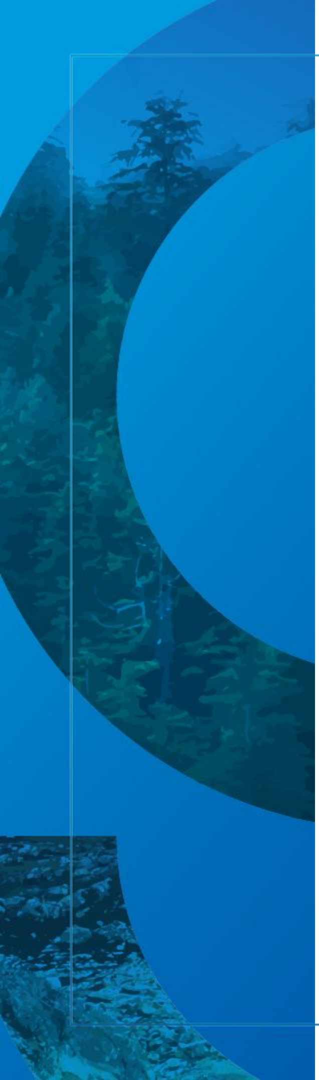
# cgroups

`/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).

There are a variety of workarounds, with various downsides:

- cgroup namespaces (with `nsdelegate`) only work in cgroupv2.
- LXC's `pam_cgfs` requires installation of a PAM module (and only works for logged-in users).



# Current adoption status

# runc

Fully supported since 1.0.0-rc4 (merged March 2017).

- Some minor features don't work because of outside restrictions.
- Originally only supported completely-unprivileged (no funny business) mode.

With 1.0.0-rc5, it supports "partially privileged" mode:

- `/sys/fs/cgroups` 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`.

`CLONE_NEWCGROUP` still not supported (but `nsdelegate` is v2-only).

# umoci and orca-build

umoci is the original generic OCI image manipulation tool.

- https://github.com/openSUSE/umoci
- Supports extraction (`unpack`) and layer generation (`repack`).
- It has supported rootless mode since the beginning.
  - Emulates `CAP_DAC_OVERRIDE` with recursive `chmod`.
  - Supports persistent xattr-based `chown(2)` emulation.

orca-build was one of the first dameon-less OCI (Dockerfile) builders.

- Built on top of umoci, skopeo, and runc.
- Supports rootless building, and is only 500 lines of Python.
- Currently have plans to merge into umoci as a `contrib/` wrapper.

# BuildKit and img

- BuildKit: next-generation backend for `docker build`
  - Integrated to Docker since v18.06, but can be also used as a standalone daemon, with support for the rootless mode
  - Uses the host network namespace at the moment
    - Not a huge problem when BuildKit itself is containerized
  - Rootless BuildKit has been used in OpenFaaS cloud

- img: rootless and daemonless image builder based on BuildKit, by Jessie Frazelle
  - Same as BuildKit but daemonless

# Kaniko

- Google's unprivileged container image builder
- Different from our approach
  - Kaniko itself needs to be executed in a container (without `--privileged`)
  - 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`

# Docker (Moby) & Podman

- Docker / Moby
  - Rootless mode is being proposed: #37375
  - Supports both slirp4netns and VPNKit for network isolation
  - Even Swarm-mode works! (except overlay NW atm)

- Podman: Red Hat’s daemonless replacement for `docker`
  - Already supports rootless mode
  - Uses slirp4netns (Thanks Giuseppe Scrivano!)

# Kubernetes & CRI runtimes

- `kubelet`, `kube-proxy`, and `dockershim` 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

- Alternative CRI runtimes:
  - CRI-O: Already supports rootless mode
  - containerd: rootless mode is on plan

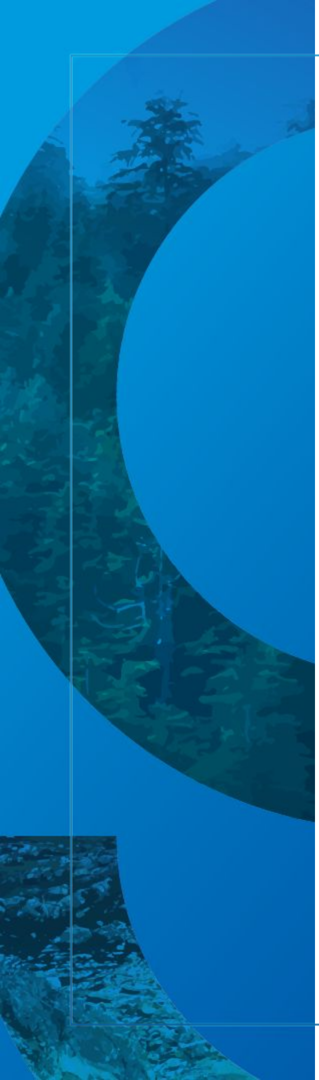
- TODO: stability improvement & multi-node network

# “Usernetes”

Experimental binary distribution of rootless Moby (Docker), CRI-O and 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..
```



# Demo: “Usernetes”

THE LINUX FOUNDATION

# OPEN SOURCE SUMMIT

THE LINUX FOUNDATION