New Root Emulation Mode for Charliecloud Using `seccomp`

Megan Phinney
mphinney@lanl.gov
Los Alamos National Laboratory

CANOPIE Workshop 11/13/2023

LA-UR 23-29965

Charliecloud Team (Current)

Reid Priedhorsky

Jordan Ogas

Shane Goff

Megan Phinney

Lucas Caudill

Layton McCafferty

Charliecloud

Agenda

01.
Why root emulation?

02.
seccomp

03.
pros & cons

01.

Why root emulation?

Charliecloud privilege taxonomy

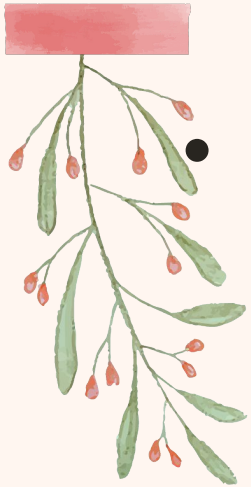
type	namespace	setup	IDs in container	examples
I	mount	privileged	shares UID and GID with host	Docker, Singularity, Podman
II	mount + privileged user	privileged	arbitrary UIDs and GIDs separate from host	Singularity, Podman (rootless)
III	mount + unprivileged user	unprivileged	only 1 UID and 1 GID in container	Charliecloud

Priedhorsky, Canon, Randles, Younge. SC21. https://dx.doi.org/10.1145/3458817.3476187

Only **Type III containers** are fully unprivileged throughout the container lifetime

Why do we need root emulation mode?

- HPC users **don't** have root access on the clusters
- We need to trick the program that we have root access even though we are **unprivileged**
- We map the `EUID` and `EGID` to zero inside the container



- Example:
  - `dnf -y install openssh` can't be installed without root emulation because:
    - it requires root access because of `chown(2)`
    - lots of other packages depend on it

02.

seccomp

What is seccomp(2)?

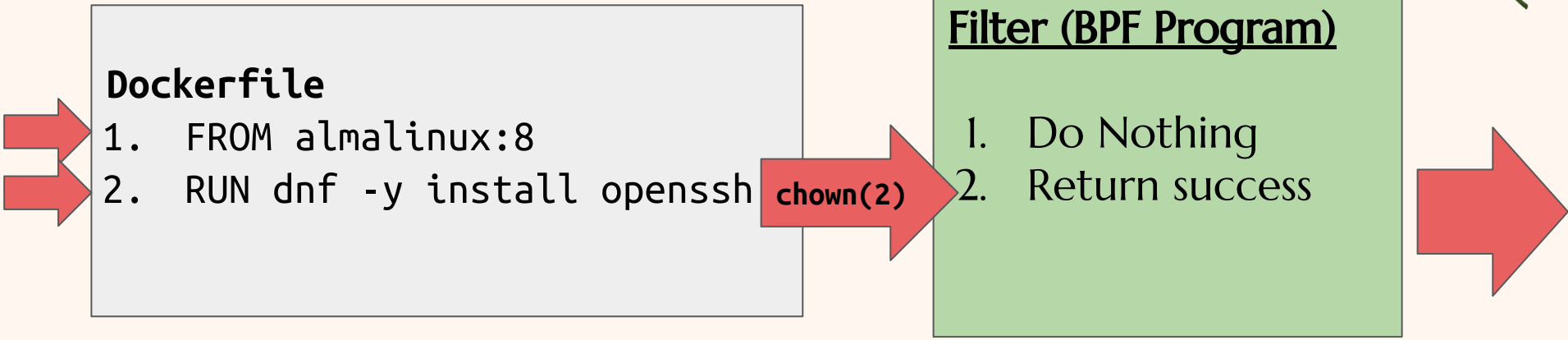
- seccomp: secure computing
  - manipulates system calls that a process may make

- High level steps:

1. Construct a fake syscall filter by writing a BPF program that does nothing and returns success
2. Install filter using `prctl()`
3. Docker `RUN` instruction go through filter

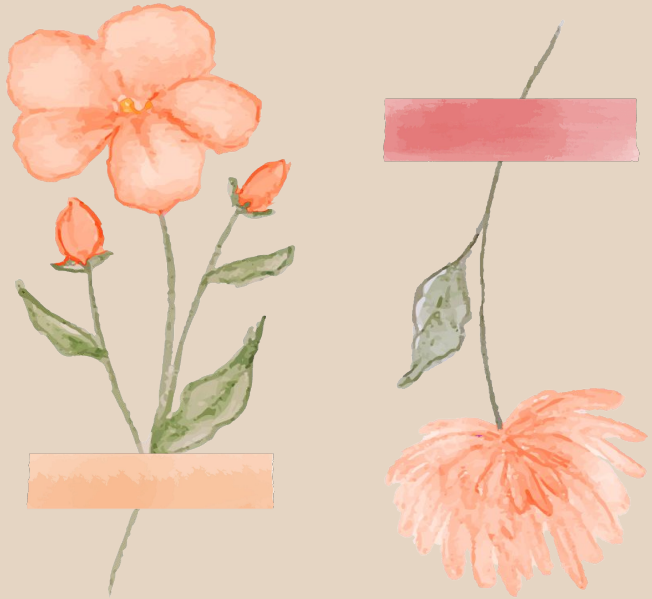
Kerrisk, Michael. "System Programming for Linux Containers." https://man7.org/training/splc/index.html. course M7D-SPLCO2. March 2023.

High-Level Diagram

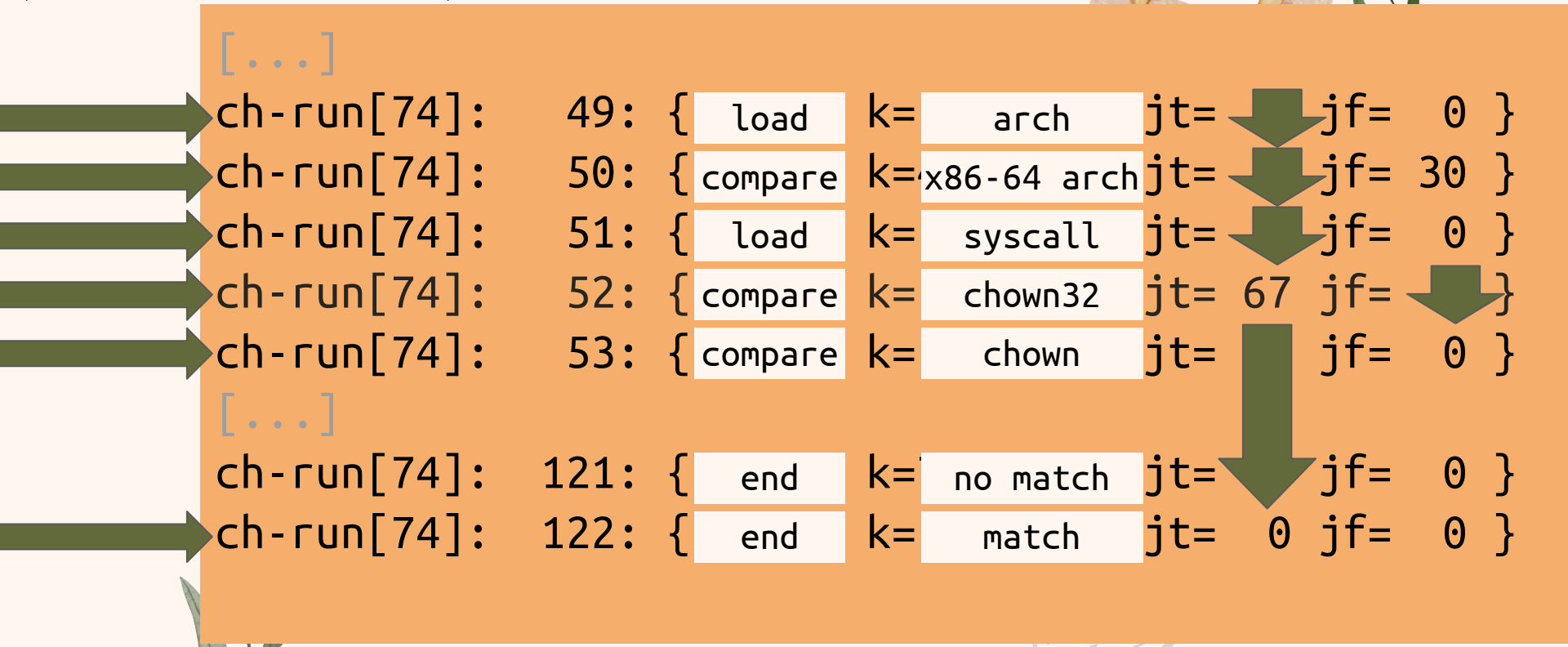


bin/ch_core.c

```
int FAKE_SYSCALL_NRS[][5] = {
   // arm64   arm32   x86     PPC64   x86-64
   // ------  ------  ------  ------  ------
   {      91,    185,    185,    184,    126 },  // capset
   {       0,    182,    182,    181,     92 },  // chown
   {       0,    212,    212,      0,      0 },  // chown32
   {      55,     95,     95,     95,     93 },  // fchown
   {       0,    207,    207,      0,      0 },  // fchown32
   {      54,    325,    298,    289,    260 },  // fchownat
   {       0,     16,     16,     16,     94 },  // lchown
   {       0,    198,    198,      0,      0 },  // lchown32
   {       0,     14,     14,     14,    133 },  // mknod
   {      33,    324,    297,    288,    259 },  // mknodat
   {     152,    139,    139,    139,    123 },  // setfsgid
   {       0,    216,    216,      0,      0 },  // setfsgid32
   {     151,    138,    138,    138,    122 },  // setfsuid
   {       0,    215,    215,      0,      0 },  // setfsuid32
   {       0,    215,    215,      0,      0 },  // setfsuid32
   {     144,     46,     46,     46,    106 },  // setgid
   {       0,    214,    214,      0,      0 },  // setgid32
   {     159,     81,     81,     81,    116 },  // setgroups
   {       0,    206,    206,      0,      0 },  // setgroups32
   {     143,     71,     71,     71,    114 },  // setregid
   {       0,    204,    204,      0,      0 },  // setregid32
   {     149,    170,    170,    169,    119 },  // setresgid
   {       0,    210,    210,      0,      0 },  // setresgid32
   {     147,    164,    164,    164,    117 },  // setresuid
   {       0,    208,    208,      0,      0 },  // setresuid32
   {     145,     70,     70,     70,    113 },  // setreuid
   {       0,    203,    203,      0,      0 },  // setreuid32
   {     146,     23,     23,     23,    105 },  // setuid
   {       0,    213,    213,      0,      0 },  // setuid32
   { -1 }, // end
};
```



Example BPF Program (ch-run -vv …)

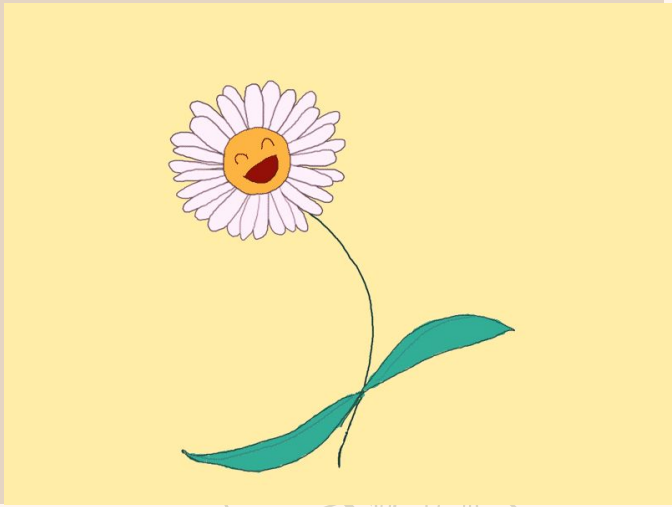


`strace` output - without seccomp filter

```
$ strace -f ch-image build .
[...]
openat(AT_FDCWD, "/usr/libexec/openssh/ssh-keysign",
O_WRONLY|O_CREAT|O_EXCL|O_TRUNC, 0666) = 49
[...]
write(49, "\177ELF\2\"..., 32768) = 32768
[...]
write(49, "B\16\..., 29104) = 29104
[...]
close(49)                           = 0
getuid()                            = 0
chown("/usr/libexec/openssh/ssh-keysign", 0, 996) = -1 EINVAL (Invalid argument)
[...]
```

`strace` output – with seccomp filter

```
$ strace -f ch-image build --force=seccomp .
[...]
openat(AT_FDCWD, "/usr/libexec/openssh/ssh-keysign",
O_WRONLY|O_CREAT|O_EXCL|O_TRUNC, 0666) = 49
[...]
write(49, "\177ELF\2\"..., 32768) = 32768
[...]
write(49, "B\16\..., 29104) = 29104
[...]
close(49)                               = 0
getuid()                                = 0
chown("/usr/libexec/openssh/ssh-keysign", 0, 996) = 0
[...]
```



03.

Pros & Cons

Pros.

- Simpler
- Faster
- Completely agnostic to libc
- Most agnostic to distribution

Cons

- Lacks consistency:

  but provides a hook to help prevent programs from asking for it

Note: *Our previous root emulation mode is called fakeroot*

Previous Root Emulation Mode: `fakeroot`

```
$ ch-image build --force=fakeroot .
1* FROM almalinux:8
2. RUN.F dnf -y install openssh
copying image from cache ...
--force=fakeroot: will use: rhel8: RHEL 8+ and derivatives
--force=fakeroot: init step 1: checking: $ command -v fakeroot > /dev/null
--force=fakeroot: init step 1: $ set -ex; [...]fi;
+ grep -Eq '\[epel\]' [... yum repo files ...]
+ dnf install -y
https://dl.fedoraproject.org/pub/epel/epel-release-latest-8.noarch.rpm
+ dnf install -y fakeroot
+ dnf remove -y epel-release
--force: RUN: new command: ['fakeroot', '/bin/sh', '-c', 'dnf -y install openssh']
--force=fakeroot: modified 1 RUN instructions
grown in 2 instructions: seccomp-test
```

New Root Emulation Mode for Charliecloud Using `seccomp`

Megan Phinney
mphinney@lanl.gov
Los Alamos National Laboratory

CANOPIE Workshop 11/13/2023

LA-UR 23-29965