Securing Container Runtimes

“How Hard Can It Be?”

```
% whoami
name:
  Aleksa Sarai
job:
  Senior Software Engineer @ SUSE
status:
  Really hoping that nobody finds
  several CVEs over the weekend
  because of this talk.
```

- Container attacks that are outright kernel bugs.
- Obvious security goofs like zero-auth remote code execution.

(Sadly that wasn’t a joke – see CVE-2014-3499

… and

CVE-2014-9357.)

HANG ON SO WHAT IS IN SCOPE?

- Everything else.
- Our focus is on bugs involving interactions between runtimes and malicious container code.

IF YOU ONLY TAKE ONE THING AWAY FROM THIS TALK

- ALWAYS.
  USE.
  USER.
  NAMESPACES.

Broken Configuration

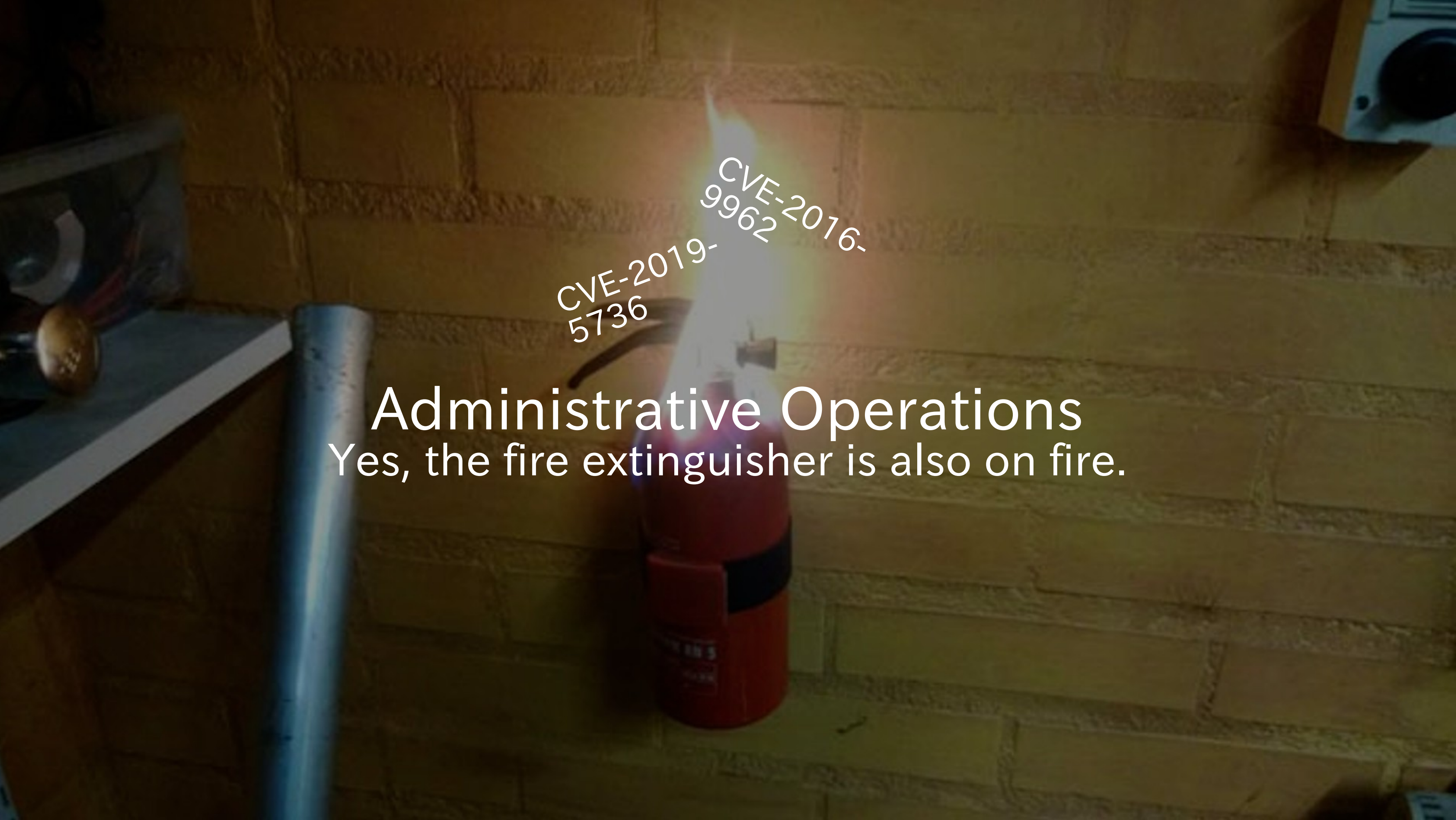
Let’s start with some kindling.

CVE-2017-16539

CVE-2016-8867

CVE-2016-10124

CVE-2015-3630



Administrative Operations

Yes, the fire extinguisher is also on fire.

CVE-2015-1340

CVE-2015-3627

CVE-2018-15664

CVE-2015-1335

CVE-2015-1334

CVE-2015-1331

CVE-2014-6407

CVE-2019-10152

The Filesystem

"Everything is a file" – a great idea until you find out they're highly flammable.

[Honestly, most of procfs.]

The Kernel Interfaces

There are some issues.

[Your CVE Here.]

CVE-2019-10152

CVE-2016-9962

CVE-2015-1340

CVE-2018-15664

CVE-2016-10124

CVE-2015-3630

CVE-2016-8867

A Trivial Example

Creating a console for a container.

```
mfd = open(“/ctr/dev/ptmx”, O_RDWR);
num = ioctl(nfd, TIOCGPTN);
asprintf(&path, “/ctr/dev/pts/%d”, num);
sfd = open(path, O_RDWR);
/* dup2 sfd over stdio of container. */
```

```
mfd = open(“/ctr/dev/ptmx”, O_RDWR);
num = ioctl(nfd, TIOCGPTN);
asprintf(&path, “/ctr/dev/pts/%d”, num);
sfd = open(path, O_RDWR);
/* dup2 sfd over stdio of container. */
```

```
mfd = open(“/ctr/dev/ptmx”, O_RDWR);
sfd = ioctl(nfd, TIOCGPTPEER);
/* dup2 sfd over stdio of container. */
```

```
mfd = open(“/ctr/dev/ptmx”, O_RDWR);
sfd = ioctl(nfd, TIOCGPTPEER);
/* dup2 sfd over stdio of container. */
```

Solutions?

(Well, some of them.)

```c
int openat2(int dirfd,
            const char *path,
            const struct open_how *how);

// ... and fix procfs magic-links.
```

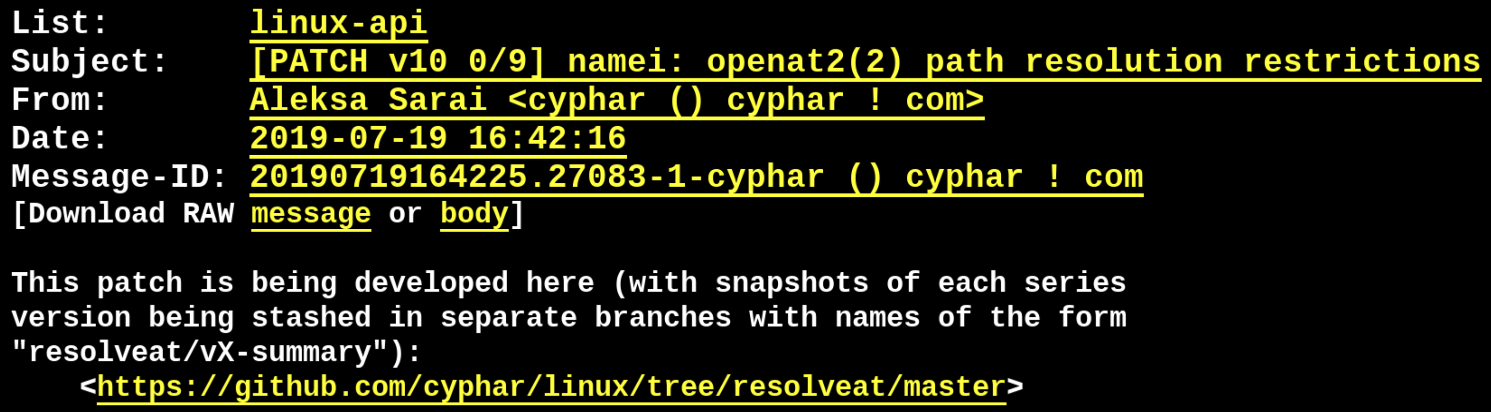
```
struct open_how {
  uint32_t flags;
  union {
    uint16_t mode;
    uint16_t upgrade_mask;
  };
  uint16_t resolve;
  uint64_t reserved[7]; /* must be zeroed */
};
```

```
struct open_how {
  uint32_t flags;
  union {
    uint16_t mode;
    uint16_t upgrade_mask;
  };
  uint16_t resolve;
  uint64_t reserved[7]; /* must be zeroed */
};
```

```c
struct open_how {
  uint32_t flags;
  union {
    uint16_t mode;
    uint16_t upgrade_mask;
  };
  uint16_t resolve;
  uint64_t reserved[7]; /* must be zeroed */
};
```

```
/* how->resolve flags for openat2(2). */
#define RESOLVE_NO_XDEV       0x01
#define RESOLVE_NO_MAGICLINKS 0x02
#define RESOLVE_NO_SYMLINKS   0x04
#define RESOLVE_BENEATH       0x08
#define RESOLVE_IN_ROOT       0x10
```

```
/* how->resolve flags for openat2(2). */
#define RESOLVE_NO_XDEV       0x01
#define RESOLVE_NO_MAGICLINKS 0x02
#define RESOLVE_NO_SYMLINKS   0x04
#define RESOLVE_BENEATH       0x08
#define RESOLVE_IN_ROOT       0x10
```



```
List:       linux-api
Subject:    [PATCH v10 0/9] namei: openat2(2) path resolution restrictions
From:       Aleksa Sarai <cyphar () cyphar ! com>
Date:       2019-07-19 16:42:16
Message-ID: 20190719164225.27083-1-cyphar () cyphar ! com
[Download RAW message or body]

This patch is being developed here (with snapshots of each series
version being stashed in separate branches with names of the form
"resolveat/vX-summary"):
    <https://github.com/cyphar/linux/tree/resolveat/master>
```

marc.info/?l=linux-api&m=156355459919115

github.com/cyphar/linux [branch: resolveat/master]

```c
int openat2(int dirfd,
            const char *path,
            const struct open_how *how);
```

Nobody is using **openat(2)** *right now!*

```
int openat2(int dirfd,
            const char *path,
            const struct open_how *how);
```

Oh, and the new interface is tricky to use correctly...

Nobody is using openat2(2) *right now!*

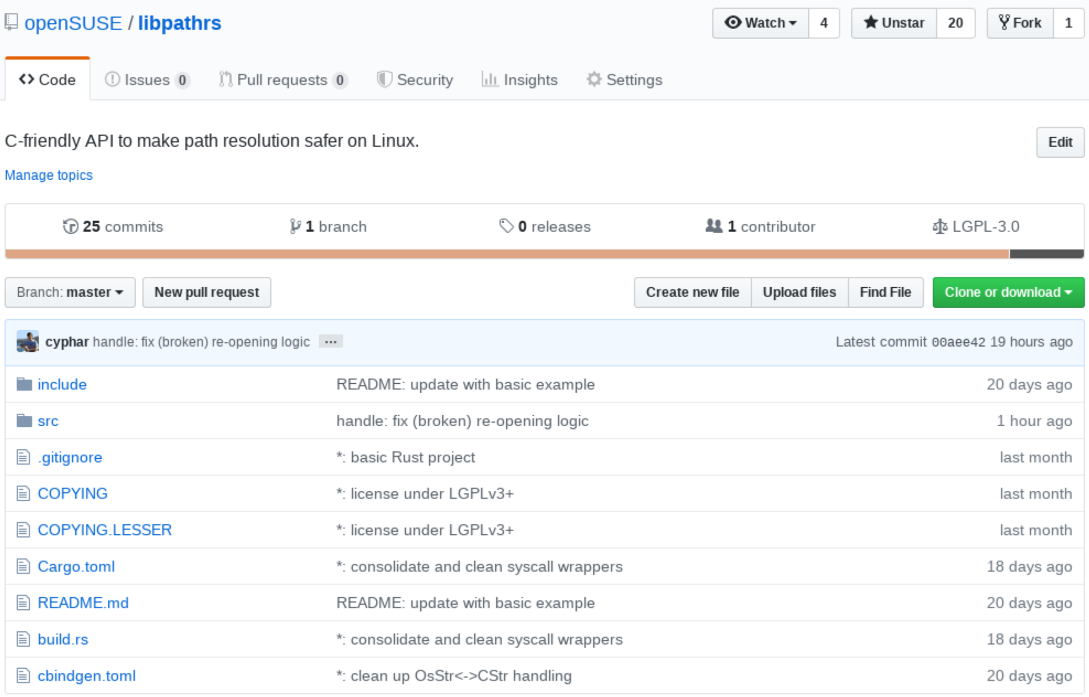
```
int openat2(int dirfd,
            const char *path,
            const struct open_how *how);
```

Also, how do we deal with old kernels?

Nobody is using openat2 right now!

The new interface is tricky to use correctly!

```
int openat2(int dirfd,
            const char *path,
            const struct open_how *how);
```



openSUSE / **libpathrs**

C-friendly API to make path resolution safer on Linux.

25 commits | **1** branch | **0** releases | **1** contributor | LGPL-3.0

cyphar handle: fix (broken) re-opening logic … Latest commit `00aee42` 19 hours ago

include	README: update with basic example	20 days ago
src	handle: fix (broken) re-opening logic	1 hour ago
.gitignore	*: basic Rust project	last month
COPYING	*: license under LGPLv3+	last month
COPYING.LESSER	*: license under LGPLv3+	last month
Cargo.toml	*: consolidate and clean syscall wrappers	18 days ago
README.md	README: update with basic example	20 days ago
build.rs	*: consolidate and clean syscall wrappers	18 days ago
cbindgen.toml	*: clean up OsStr<->CStr handling	20 days ago

```
use pathrs::*;
let root = Root::open("/path/to/root")?;
let handle = root.resolve("/etc/passwd")?;
let file = handle.reopen(libc::O_RDONLY)?;
```

```c
#include <pathrs.h>

root = pathrs_open("/path/to/root");
if (!root)
    goto err;

handle = pathrs_inroot_resolve(root, "/etc/passwd");
if (!handle)
    goto err;

fd = pathrs_reopen(handle, O_RDONLY);
if (fd < 0)
    goto err;
```

```python
import pathrs

root = pathrs.Root("/path/to/root")
with root.resolve("/etc/passwd").reopen(os.O_RDONLY) as f:
   pass # Do whatever you like with f.
```

Demo

Great! We’re all done right?

Unfortunately not.

We need (you)sers!

```
int pidfd_send_signal(int pidfd, ...);
// and CLONE_PIDFD
```

https://lwn.net/Articles/784831/

Questions?