

Upgrade restrictions for file descriptors

Jori Koolstra

LPC, 2026

- Context: CVE-2019-5736 in runc
- Solution: Capsicum (Drysdale, 2014)
- Solution: magic link modes (Sarai, 2019)

Context: CVE-2019-5736

- you pass procfs's magic link `/proc/self/exe` to runc
- harmless in itself, but: the container can inject code by providing alternative dynamic libraries or using `LD_PRELOAD`
- in the injected code you obtain a writable fd to the host runc
- hence `docker run <malicious image> /proc/self/exe` may overwrite the host runc¹
- initial patch (Sarai/Brauner) fixed the reopen risk with `memfd_create(2)` and sealing, then `fexecve`'ing that²

¹ <https://aws.amazon.com/blogs/compute/anatomy-of-cve-2019-5736-a-runc-container-escape/>

² <https://github.com/opencontainers/runc/commit/0a8e4117e7f715d5fbeef398405813ce8e88558b>

Is it a problem?

> - You can use `/proc/self/exe` to get a writable fd.

This is IMO the real bug.

— Andy Lutomirski

It is crazy that there are container runtimes out there that promise (...) containers without user namespaces or containers with user namespaces that allow to map the host root id to anything can be safe. They cannot (...) So if this part of the patch is just needed to handle this do we really want to do all that tricky work or is there more to gain from this that makes it worth it.

— Christian Brauner

¹ <https://lore.kernel.org/lkml/20190506165439.9155-6-cyphar@cyphar.com/>

Solution: Capsicum

- There is a FreeBSD implementation with origins at Cambridge and Google (~2010)¹
- The idea: associate rights with file descriptors, then drop ambient authority (into *capability mode*) with `cap_enter(2)`
- This blocks unrestricted reopens with `open(2)`, but you can `openat(2)` with `O_BENEATH_ONLY`
- Also blocks execution of `setuid` and `setgid` binaries, etc.
- Every right is explicitly given

¹ <https://www.cl.cam.ac.uk/research/security/capsicum/papers/2010usenix-security-capsicum-website.pdf>

Solution: Capsicum

- Drysdale (2014) only implemented `cap_rights_limit(2)`, no `cap_enter(2)`
- Invasive: every translation from an `fd` to a `struct file` needs correct annotation, and none must be missed. Example:

```
SYSCALL_DEFINE1(fchdir, unsigned int, fd)
{
- struct fd f = fdget_raw(fd);
+ struct fd f = fdgetr_raw(fd, CAP_FCHDIR);
  struct inode *inode;
  int error = -EBADF;

- error = -EBADF;
- if (!f.file)
+ if (IS_ERR(f.file)) {
+   error = PTR_ERR(f.file);
+   goto out;
+ }
```

Solution: Capsicum

- Linux has a billion different security features, it's all very random, not clean, (...)¹
— Lennart Poettering
- The LSM based implementation seems wonky
- On the flip side: Capsicum is enabled for many programs already, Linux can benefit from that
- Not too hard to convert programs
- Improvement over current capability implementation (e.g. Poettering²)

¹ <https://media.ccc.de/v/all-systems-go-2025-354-a-security-model-for-systemd#t=58>

² <https://lwn.net/Articles/1042888/>

Solution: magic link modes (Sarai, 2019)

- Idea developed by Sarai, Lutomirski, and Brauner. Added in the v8 of the `openat2(2)` patchset, but dropped in v14
- Targeted directly at the `runc` CVE
- Use the mode bits of `procfs` entries to encode `open` and `0_PATH`-upgrade restrictions
- No new field added to `struct file`, instead defines `FMODE_PATH_READ` and `FMODE_PATH_WRITE`

Solution: magic link modes (Sarai, 2019)

```
static int may_open_magiclink(fmode_t upgrade_mask,
int acc_mode)
{
    if (capable(CAP_DAC_OVERRIDE))
        return 0;

    if ((acc_mode & MAY_READ) &&
        !(upgrade_mask & (FMODE_READ |
FMODE_PATH_READ)))
        return -EACCES;
    if ((acc_mode & MAY_WRITE) &&
        !(upgrade_mask & (FMODE_WRITE |
FMODE_PATH_WRITE)))
        return -EACCES;

    return 0;
}
```

¹ <https://lore.kernel.org/lkml/20190930183316.10190-2-cyphar@cyphar.com/>

```
static int trailing_magiclink(struct nameidata *nd,
int acc_mode, fmode_t *opath_mask)
{
    fmode_t upgrade_mask = 0;
    if (!(nd->flags & LOOKUP_MAGICLINK_JUMPED))
        return 0;

    /*
     * Figure out the upgrade-mask of nd-
     >last_magiclink_mode (which
     * was stashed away during the jump with
     nd_jump_link()).
     */
    if (nd->last_magiclink_mode & S_IRUGO)
        upgrade_mask |= FMODE_PATH_READ;
    if (nd->last_magiclink_mode & S_IWUGO)
        upgrade_mask |= FMODE_PATH_WRITE;
    /* Restrict the O_PATH upgrade-mask of the
    caller. */
    if (opath_mask)
        *opath_mask &= upgrade_mask;
    return may_open_magiclink(upgrade_mask,
acc_mode);
}
```

Solution: magic link modes (Sarai, 2019)

Now that magic-link modes are obeyed for file re-opening purposes, some of the pre-existing magic-link modes need to be adjusted to be more semantically correct.

The most blatant example of this is `/proc/self/exe`, which had a mode of `a+rwX` even though tautologically the file could never be opened for writing (...)

With the new `O_PATH` restrictions, changing the default mode of these magic-links allows us to avoid delayed-access attacks such as we saw in CVE-2019-5736.

— Aleksa Sarai

Bad news...

The magic-link changes (...) have been dropped from this series (...) The main reason is that we need to restrict things other than `open(2)` (examples include `truncate(2)` as well as `mount(MS_BIND)`). This will require a fair amount of extra work, and there's no point stalling `openat2(2)` for that work to be completed.

— Aleksa Sarai

Bad news...

- truncate(2)
- open_tree(2) / mount(MS_BIND)

```
char buf[64];  
snprintf(buf, sizeof buf, "/proc/self/fd/%d", fd);  
mount(buf, "target", NULL, MS_BIND, NULL);
```

```
int mntfd = open_tree(fd, "", OPEN_TREE_CLONE | AT_EMPTY_PATH);  
move_mount(mntfd, "", AT_FDCWD, "target", MOVE_MOUNT_F_EMPTY_PATH);
```

- Worse: fsconfig(2) takes fds through FSCONFIG_SET_FD, thus per-filesystem-option handling is required

Bad news...

- More general: we need to keep track of paths that originate from procfs entries, not only in `path_openat()`, but also `truncate(2)`, `dentry_open()`, `backing_file_open()`, ...

Thanks,

Further reading:

Capsicum: practical capabilities for UNIX

Capsicum for Linux

A security model for systemd