LINUX
PLUMBERS
CONFERENCE
August 24-28, 2020

openat2(2)

what’s next?

Aleksa Sarai (SUSE)
cyphar@cyphar.com

LINUX
PLUMBERS
CONFERENCE
August 24-28, 2020

Current Status

openat2(2) in Linux 5.6.

- ➢ Only main missing pieces are related to magic-link hardening.
- ➢ Automount or “remote fs” restrictions might be useful.

libpathrs still under active development.

- ➢ Experimental C, Python and Go bindings.
- ➢ Still need to improve C API wrt multi-threading.
- ➢ Goal: Have first real program (umoci) ported by end-of-year.

LINUX PLUMBERS CONFERENCE
August 24-28, 2020

Remaining Issues

`procfs` is still a minefield.

- We require `/proc` but we can't trust it in containers.
- I have some proposals to work around this.
- (I still think `O_EMPTYPATH` is a good idea.)

Magic-links still allow too much reopening.

- Being able to re-open `/proc/$pid/exe` for writing is silly.
- Based on my tests, no programs break with restrictions.

(Less Important)
Remaining Issues

Can userspace safely rely on mount behaviour?

- ➢ Mainly, mounts on top of existing file descriptors and re-opening.
- ➢ Important to make sure libpathrs actually provides protection.
- ➢ Should we just add some code to VFS `selftests`?
  - ○ Probably not a bad thing to do anyway…

`readlinkat2(AT_EMPTY_PATH)`

- ➢ Given an open `O_PATH` symlink, we cannot currently `readlink` it.
- ➢ Does anyone mind if we add this?

LINUX PLUMBERS CONFERENCE
August 24-28, 2020

/proc
(Background)

We can now block most of the things we want to avoid.

- ➢ `openat2(RESOLVE_*)` is enough for most operations.
- ➢ With “safe” handles you can do most VFS operations.

However, using `/proc` safely can become complicated.

- ➢ libpathrs (currently) requires `/proc` operations in implementation.
- ➢ Container runtimes need to fiddle with `procfs` files.
- ➢ How do we make sure we are accessing the right `procfs` file?
- ➢ *Note that containers have some freedom to configure their mounts.*

LINUX PLUMBERS CONFERENCE
August 24-28, 2020

`/proc`
(The Easy-ish Stuff)

`/proc` is the root of a `procfs`.

- ➢ `fstatfs(2)` as well as `PROC_ROOT_INO` (1).
- ➢ Once we grab a handle and verify it, we’re golden.

`/proc/self/attr/exec` is the label for `$pid`.

- ➢ `openat2(RESOLVE_NO_XDEV|RESOLVE_NO_SYMLINKS)`.
- ➢ Without `openat2(2)`, not possible without races.
- ➢ (Note that `/proc/$pid/environ` and `/proc/$pid/sched` exist.)

/proc
(The Hard Stuff)

Being sure that `/proc/self/{fd/$n,exe}` is legit.

- Not currently possible, even with `openat2(2)`.
- Cannot use `RESOLVE_NO_XDEV` (blocks most magic-links).
- *Attackers can bind-mount on top of symlinks.*
- Can't do `readlink`-based lookups because we need `nd_jump_link()`.
- We need these to be safe for container runtimes and libpathrs.

`/proc`
(Proposal 1 -- “Add another hack.”)

`openat2(RESOLVE_ONLY_MAGICLINKS).`

- ➢ Only permit resolution which calls `nd_jump_link()`.
- ➢ This is sufficient to solve our `procfs` troubles.
  - ○ Lookup parent of magic-link, follow the magic-link, continue.
- ➢ But this is clearly a hack to solve *only* this one problem.
  - ○ Semantics will be strange no matter what we pick.
  - ○ Useless for “general purpose” open-this-file problems.
  - ○ Still fundamentally depends on `procfs`.

/proc
(Proposal 2 -- “Distinct Replacement APIs.”)

We use procfs magic-links for completely different things.
So just introduce new procfs-free APIs for each problem.

- `/proc/self/fd/$n` → `openat($n, "", O_EMPTYPATH).`
- `/proc/self/exe` → `process_get_resource(-1, PROC_EXE);`
    - Ditto for `cwd`, `root`, `ns/*`, et al.
- Lots of extra APIs and work -- is it worth it?
    - Plenty of bike-sheds to paint.
    - Might be good to cherry-pick the ones that are actually useful.
- What should we do …
    - … for `/proc/self/map_files`?
    - … if another magic-link is added?
    - … about magic-links outside of `procfs`?

/proc
(Proposal 3 -- "Process-local procfs.")

Bypass the whole "is /proc safe" question.

- ➢ API to get a fresh procfs handle that is only visible to the program.
- ➢ Unprivileged `fsopen("procfs")` with `subset=pidfs,hidepid=4`.
  - ○ … or something more fruity like `AT_FDPROCSELF` (a-la `AT_FDCWD`).
  - ○ Make sure we don't allow bypassing `mount_too_revealing()`.
- ➢ Seems like the "neatest" solution:
  - ○ Solves the whole "is /proc mounted" problem simultaneously.
  - ○ Makes lookups simpler and a program could cache this handle.
  - ○ However, doesn't help us with non-procfs magic-links.
- ➢ If we had "`subset=self`" you could pass these handles around.

LINUX PLUMBERS CONFERENCE
August 24-28, 2020

/proc
(3… 2… 1… FIGHT!)

1. "Add another hack."
   - ➢ openat2(RESOLVE_ONLY_MAGICLINKS).
2. "Distinct Replacement APIs."
   - ➢ /proc/self/fd/$n → openat2(O_EMPTYPATH).
   - ➢ /proc/self/{exe,cwd,root} → get_process_fd(pidfd).
   - ➢ Figure something out for everything else…
3. "Process-local procfs."
   - ➢ AT_PROCSELF; or
   - ➢ Unprivileged fsopen(2) for procfs with subset=pid,hidepid=4.

LINUX
PLUMBERS
CONFERENCE
August 24-28, 2020

Bonus: Magic-links

In 2019, I proposed magic-link re-opening restrictions.

- ➢ Patch on LKML (dropped from the `openat2` patchset) and LPC talk.
- ➢ *Recap:* Allow re-opening of a magic-link if the original handle has an `f_mode` which is a superset of the requested mode (`O_PATH` is special and copies magic-link modes or is `rwx` if not a magic-link).
  - ○ Add an `upgrade_mask` to `openat2(2)` for `O_PATH`.
- ➢ Is a change in behaviour, but doesn't appear to break Linux systems.

Any objections to me re-posting this patch?

LINUX
PLUMBERS
CONFERENCE
August 24-28, 2020

Bonus: Mount Behaviour?

Currently, mounts don't affect existing handles.

- ➢ (With the obvious exception of mounts to subdirectories.)
- ➢ Can userspace rely on this behaviour not changing?
  - ○ libpathrs is designed around re-opening file descriptors in a context where we assume a handle is safe after we've checked it.
  - ○ Not clear how widely-exercised this behaviour is today.
  - ○ Would breakages be noticed? Should we add more `selftests`?

LINUX
PLUMBERS
CONFERENCE
August 24-28, 2020

Bonus: `readlinkat2(...)`

We currently cannot `readlink(2)` an `O_PATH` symlink.

- ➢ `readlink("/proc/self/fd/$n")` does exactly what you expect.
- ➢ Would allow us to avoid having to do racy retry loops for `readlink`.
- ➢ Not strictly necessary for libpathrs:
  - ○ “Easy” to work around for “legacy” lookups.
  - ○ Plus we now have `openat2(2)` so it’s less critical.
- ➢ But it seems like an omission.
  - ○ We’d have to add `readlinkat2(2)` -- no flags argument.