how can we make procfs safe?

current state of the art and a few libpathrs updates

Linux Plumbers Conference | Dublin, Ireland Sept. 12-14, 2022

Aleksa Sarai (SUSE)
cyphar@cyphar.com

threat model

We are executing in an environment where a user has managed to mess with the filesystem and possibly mounts.

We want to be able to **detect** if we are being tricked into operating on a different path than the one we expected.

Main usecase is container runtimes, but basically any program operating in or on chroots would benefit.

what's special about procfs?

With openat2(RESOLVE_*), any non-procfs filesystem path can be accessed safely with various resolution restrictions.

With procfs, we require more than just the path be resolved "reasonably", we want a **specific** procfs endpoint to be reached. The core issue is that /proc/self/environ and /proc/self/sched exist.

current status
(non-magiclinks)

In Linux 5.6, openat2(2) made this safe:

- openat("/proc", O_PATH)
- Check the f_type (fstatfs) and stx_ino (statx).
- openat2(<procfsfd>, "<pid>/attr/current",
  {RESOLVE_NO_XDEV|RESOLVE_NO_SYMLINKS})

what's the issue with magiclinks?

We can't use RESOLVE_NO_XDEV because they are almost always crossing a mountpoint. **But you can also mount on top of them.**

We can't use RESOLVE_NO_SYMLINKS for obvious reasons.

current status
(magiclinks)

It turns out this is safe since Linux 5.8 (w/CAP_SYS_ADMIN):

- open_tree("/proc", OPEN_TREE_CLONE | AT_RECURSIVE)
- Check the f_type (fstatfs) and stx_ino (statx).
- openat2(<procfsfd>, "<pid>", {O_PATH,
  RESOLVE_NO_XDEV|RESOLVE_NO_SYMLINKS})
- statx(<procselffd>, "exe", AT_SYMLINK_NOFOLLOW)
  - Check whether STATX_ATTR_MOUNT_ROOT is set.
  - If not, safe to use (no races because of OPEN_TREE_CLONE).

(See this example program.)

why does this work?
(and why might it break in the future?)

The safety of this setup relies on several undocumented behaviours:

- OPEN_TREE_CLONE mounts are an O_PATH to a bind-mount and there is no way for an external process to change any overmounts.
  - Returned tree is an anonymous mount namespace.
    - Even a CAP_SYS_ADMIN user in a different mountns cannot mount into it.
  - Mount propagation to the clone is explicitly disabled.

libpathrs

- Using `openat2(RESOLVE_IN_ROOT)` correctly is non-trivial.
  - Lots of messing around with `O_PATH`.
  - No other syscalls support `RESOLVE_IN_ROOT`.
  - How do we deal with old kernels?
- **Solution:** Rust library that provides "nice" helpers that Do The Right Thing™.
  - … and it emulates `RESOLVE_IN_ROOT` on old kernels!
  - … but this requires we port programs to use it.

an update on libpathrs

Safe wrappers for procfs.

- Cache a safe open_tree'd procfs handle.
- procself_get("exe") – or something...

Sane C API (which then can be used with Go nicely).

- Go programs are used to being able to call .Close() many times.
- Should we remove the footgun-guards and just pass fds?

an update on libpathrs

How to model the (new and old) mount API?

- We want to use the new one to avoid using /proc.
- Should we abstract the whole thing or just expose the old one?

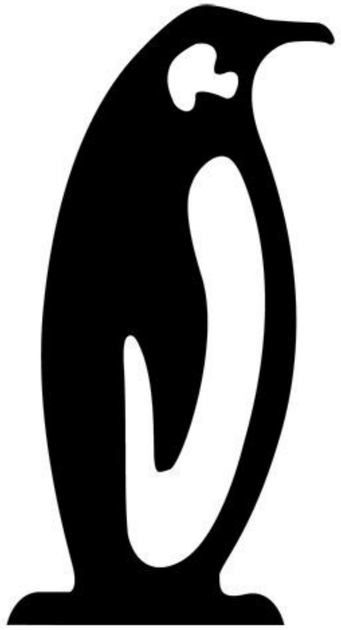
How much of the VFS API should be replicated by libpathrs?

- In theory, any operation which has AT_EMPTY_PATH is okay.
- How much should we trust library users to not footgun themselves?

an update on libpathrs

https://github.com/openSUSE/libpathrs

Still being worked on. Plan to port umoci to this first.



Linux Plumbers Conference

Dublin, Ireland September 12-14, 2022