PID FDs: where we were, where we are and were we would like to go

Luca Boccassi, Microsoft, Linux Systems Group

Agenda

- Reminder on why we need PID FDs
- Recap of recent work to plumb PID FDs through the Linux OS low-level stack
- Current work in progress in userspace to expand the above
- What’s left to do and would be good to have as kernel interfaces

Reminder: why do we need PID FDs?

- PIDs can be recycled (CVE-2019-6133, CVE-2019-15790…)
- Cap hit at 2^22 at most, then they wrap around
  - Forking requires no privileges
- Process tracking is not only a parent/children affair, other processes need to do tracking too
  - Polkit, D-Bus, systemd, …
  - Fragile mechanism + security/authentication = sad faces all around
  - E.g.: polkit uses metadata such as starting time to try and mitigate this fragility, and was only safe to use to authenticate D-Bus peers

Some Ancient History

- Kernel feature added starting from v5.3: PID File Descriptors
- Valid only until the tracked process has been reaped, never reused
- Can be passed to other processes via SCM_RIGHTS messages (AF_UNIX)
- Not limited to parent/children
- Can be resolved to a PID at any time via /proc/self/fdinfo/<X>, -1 if process is gone

Some Recent History (1/3)

- Kernel v6.5 adds SCM_PIDFD and SO_PEERPIDFD
  - SCM_PIDFD is equivalent to SCM_CREDENTIALS but with a FD instead of a PID
  - SO_PEERPIDFD is equivalent to SO_PEERCRED but with a FD instead of a PID
- Kernel v6.9 added a new pseudo-filesystem for PID FD nodes in /proc/
  - Can now compare entries directly for equality, use statx()
- Glibc v2.39 added pidfd_spawn() and pidfd_getpid()
  - Equivalent of posix_spawn(), returning FD instead of PID, and shortcut to resolve a PID FD to a PID
- With these changes merged in and available, we can start building interesting things in userspace

Some Recent History (2/3)

- systemd v253 adds GetUnitByPIDFD() and sd_pidfd_get*() APIs
  - Query session/unit/cgroup/etc by FD instead of PID
  - PSA: if you use the *_pid_* variants, switch to the *_pidfd_* variants!
- systemd v255 spawns services using pidfd_spawn()
  - With fallback for older kernels/libc
  - Also takes advantage of CLONE_INTO_CGROUP for race-free cgroup assignments
- systemd v256 switched all internal process tracking from PIDs to PID FDs
  - Some hiccups due to kthreads also showing up, but should be all sorted
  - Exception: dealing with cgroups, which do PIDs only

Some Recent History (3/3)

- D-Bus spec, dbus-daemon v1.15.8/v1.16.0 (dev), dbus-broker v34 add ProcessFD to GetConnectionCredentials()
  - They will return it only if it has been obtained ‘safely’, i.e. via SO_PEERPIDFD
  - PSA: if you use GetConnectionCredentials or GetUnixProcessID, switch to ProcessFD instead
- Polkit v124 uses sd_pidfd* and ProcessFD internally to track processes
  - It is now safe to use to track, authenticate and authorize processes that are not communicating over D-Bus, for example: Varlink sockets
- Polkit v124 also provides a new ‘system_unit’ subject attribute
  - It will allow writing polkit rules such as ‘`if (subject.system_unit == "orchestrator.service") …`’ as an alternative to adding fixed uid/gid/user/groups to base authorization on

Work in Progress: remove setuid from Polkit Agent

- Polkit authentication agent running in the unprivileged user session -> spawns SETUID root helper binary that runs PAM session and authenticates
- SETUID binaries are considered harmful, as the environment is under the control of the unprivileged caller, so attacks are possible and have happened
- Thanks to PID FDs, we can reliably track processes outside of parent/child relationship
- Agent talks to a socket-activated service, which takes a PID FD and passes it to Polkit after authentication, so Polkit can reliably check that it is authorizing the actual process that was authenticated
- https://github.com/polkit-org/polkit/issues/169

What’s next?

- Migration to using PID FDs slowly in progress in core userspace components
- Some areas left where PID FDs cannot be used, and they have to be translated back to PIDs, or there is a lack of a programmatic API
- Would be good to provide solutions to fill these gaps
- Most are easy, one seems complex

Wishlist: resolve PID FD to PID

- Currently have to reimplement string parsing of /proc/X/fdinfo/Y
- Requires /proc being mounted, and custom string parsing is not ideal
- Glibc implemented a parser and provides a public API for it, but would be nice to remove even that
- Can we have a programmatic API instead?
  - New ioctl? E.g.:

```
#define PIDFD_GET_PID   _IOR(PIDFS_IOCTL_MAGIC, 11, int)

ioctl(pidfd, PIDFD_GET_PID, &pid)
```

Wishlist: querying creds of a PID FD

- Checking creds of a socket peer is easy via SCM_RIGHTS and friends
- But what if I don’t have a socket, I only have a PID FD?
- Have to resolve PID FD to PID, and then check /proc/PID/
  - String parsing, fine for scripts, not ideal elsewhere
  - Subject to usual races due to PID, so have to resolve PID FD, parse /proc/ manually, then again resolve PID FD to ensure nothing changed
- Can we have a programmatic API to query creds of a PID FD?
  - Avoids need for everyone to roll their own proc parser
  - Avoids need for double and triple checking that there are no races
  - New ioctl? E.g.:

```
#define PIDFD_GET_CREDS  _IOR(PIDFS_IOCTL_MAGIC, 12, struct ucred)

ioctl(pidfd, PIDFD_GET_CREDS, &creds)
```

Wishlist: <somehow> integrate PID FDs and cgroups

- Cgroups list PIDs, so we have to translate back and forth
- Main usage of PIDs left in systemd
- Can we figure out a way to somehow use PID FDs directly?
  - E.g.: I have a PID FD, what cgroup does it belong to?

```
#define PIDFD_GET_CGROUPID  _IOR(PIDFS_IOCTL_MAGIC, 13, uint64_t)

ioctl(pidfd, PIDFD_GET_CGROUPID, &cgroupid)
```

  - E.g.: I have a cgroup, can I iterate over all the processes using only FDs? Maybe new PIDFD filesystem can help, maybe something somewhat similar to /proc/N/fd/ ?

Thanks!

Questions?