Managing serialization versions for LUO objects

Pratyush Yadav <pratyush@kernel.org>

Agenda

LUO object versions

Phase 1: list versions in vmlinux

Phase 2: support multiple versions

Agenda

- LUO object versions
- Phase 1: list versions in vmlinux
- Phase 2: support multiple versions

LUO object versions

- As LUO and subsystems evolve, the serialization formats will change.
- To fix bugs, improve performance, support new features, etc.
- If the next kernel doesn't understand the versions, live update will fail.

Agenda

LUO object versions

Phase 1: list versions in vmlinux

Phase 2: support multiple versions

List versions in vmlinux

- Put all the LUO version strings in a vmlinux section (say `.liveupdate_versions`).
- Userspace tooling (like luod) can read the list and figure out if there is incompatibility.
- Don't reboot if a participating FD has incompatible version.

Data format

```
struct liveupdate_ver_hdr {
        u32 magic;
        u32 version;
};

struct liveupdate_ver_table {
        struct liveupdate_ver_hdr hdr;
        char versions[][LIVEUPDATE_HNDL_COMPAT_LENGTH];
};

LIVEUPDATE_FILE_HANDLER(memfd_luo_handler, MEMFD_LUO_FH_COMP
                                &memfd_luo_file_ops);
```

Question:

What ABI guarantees do we make for this?

Current status

- RFC of this posted this Thursday: link.

Agenda

LUO object versions

Phase 1: list versions in vmlinux

Phase 2: support multiple versions

Support multiple versions

- Actually lets you roll out a new serialization version in your fleet.
- More flexible rolling forward and back.
- Upstream gets core LUO infra, and upstream can decide how many versions it supports.
- CSP can have additional versions downstream.

Listing all versions in vmlinux

- Add the concept of a LUO “type”.
- Each LUO type is one file handler or FLB (example: “memfd”).
- Each type can have multiple versions.

```
struct liveupdate_ver {
        char type[LIVEUPDATE_HNDL_COMPAT_LENGTH];
        char version[LIVEUPDATE_HNDL_COMPAT_LENGTH];
};

struct liveupdate_ver_table {
        struct liveupdate_ver_hdr hdr;
        struct liveupdate_ver versions[];
};
```

LUO file handler set

- Set of file handlers for the same type.
- LUO chooses the right handler based on what userspace configured.

```
struct liveupdate_fh_set {
        char type[LIVEUPDATE_HNDL_COMPAT_LENGTH];
        struct list_head handlers;
        struct liveupdate_file_handler *current;
        // ...
};
```

Choosing the version

- Based on version info of both kernels, userspace (luod) can "negotiate" the versions LUO should use.
- New ioctl: `LIVEUPDATE_CMD_SET_VERSION`.

```
struct liveupdate_ioctl_set_version {
        __u32 size;
        __u8 type[LIVEUPDATE_HNDL_COMPAT_LENGTH];
        __u8 version[LIVEUPDATE_HNDL_COMPAT_LENGTH];
};
```

Alternative: serialize all versions

- Suggested by Jason Gunthorpe earlier.
- Always create serialized state of all the versions LUO knows.
- No need for UAPI exposure.
- More expensive for performance.
- Can be inflexible: two different versions might need the file to be in different states.

Thank you for attending the talk!