Isolated user namespaces

Stéphane Graber
Owner, Zabbly
stgraber@stgraber.org

Aleksandr Mikhalitsyn
Kernel engineer, Canonical
aleksandr.mikhalitsyn@canonical.com

Some background

- User namespaces map a uid/gid or set of uid/gid between the namespace and its parent
- Most container runtimes using the user namespace will map through 65536 for POSIX compliance
- If you want non-overlapping maps, this limits you to around 65k containers total on the system
- A variety of software now relies on uid/gid > 65534 which causes problems and requires larger maps
- When using non-overlapping maps, mediation between various container runtimes can be a problem

- During past editions of Plumbers, we’ve been talking about a variety of ways to address this problem
- The introduction of idmapped mounts now allows for a nice separation between the concerns of process ownership and those of persistent file storage

Proposal

Let’s make uid and gid 64bit!

But hide that fact from userspace.

Extending k{u,g}id_t to be 64bit (cont.)

This extension was implemented and resulted into a small (12 commits) series of Linux kernel patches [3].

1. Extend k{u,g}id_t to contain two 32bit-width fields: one for UID/GID and another for user namespace ID (user namespace ID is the ino of the /proc/<pid>/ns/user)
2. Introduce a concept of an “isolated” k{u,g}id_t. UID/GID will be called “isolated” if it has no corresponding UID/GID in the init_user_ns.
3. For an “non-isolated” k{u,g}id_t we fill user namespace ID part of k{u,g}id_t with 0, for isolated ones we fill it with the user namespace ID.
4. Introduce /proc/<pid>/isolated_uns file that can be written from the userspace side with yes/no value (analogical to /proc/<pid>/setgroups). By default the value is “no”.

Demo

Linux kernel patches

What’s next, what needs solving

- Main current issue is that the root user needs to be mapped to a “real” uid
- There are some remaining corner cases around the integration with VFS idmap
- Upcoming changes to the VFS will make some of that work a bit harder and need some rework
- Handling nested containers
- SCM_CREDS and other cases where a uid/gid is passed across boundaries

Links

[1] Isolated dynamic user namespaces https://lpc.events/event/7/contributions/836/

[2] Simplified user namespace allocation https://lpc.events/event/11/contributions/982/

[3] Linux kernel patches: https://github.com/mihalicyn/linux/commits/isolated_userns_lpc

[4] LXC patch: https://github.com/mihalicyn/lxc/commits/isolated_userns

[5] cgroupfs and cgroup namespace:
https://github.com/torvalds/linux/blob/3ca112b71f35dd5d99fc4571a56b5fc6f0c15814/kernel/cgroup/cgroup.c#L2169

Questions ?

Stéphane Graber
Owner, Zabbly
stgraber@stgraber.org

Aleksandr Mikhalitsyn
Kernel engineer, Canonical
aleksandr.mikhalitsyn@canonical.com

A bit of context

- kuid_t/kgid_t are only for internal use and currently they always represent a UID/GID in the init_user_ns.
- uid_t/gid_t are used when we deal with a user space visible things: syscalls, taking uid/gid from disk (in filesystems), SCM_CREDENTIALS, etc
- Each user namespace has uid_map and gid_map. These structures represent mappings from user namespace local UID/GID to corresponding kuid_t/kgid_t values. Even if nested user namespaces are used these structures always represent a mapping from inside the user namespace to the host.
- Across the kernel we use helpers: make_kuid(user_namespace, uid) -> kuid_t (example: setuid)
- …and from_kuid(user_namespace, kuid) -> uid_t (example: getuid)
- In file systems we use: i_uid_read(inode) -> uid_t. This helper is used when we want to **write** (there is no mistake :-)) a uid to disk
- i_uid_write(inode, uid) -> void. This helper is used when filesystem driver **reads** uid_t from disk and fills inode->i_uid with a kuid_t value.

What’s the problem?

- Problem is in the organization of a proper management of these uid/gid mappings for user namespaces. We have /etc/subuid files, we have newuidmap utility and many other things but some software may ignore that stuff completely.
- Ideally we want to be able to have a non-intersective UIDs (on the host) for each container while having a while 32bit UID range available to use inside the container. There is a clear contradiction.

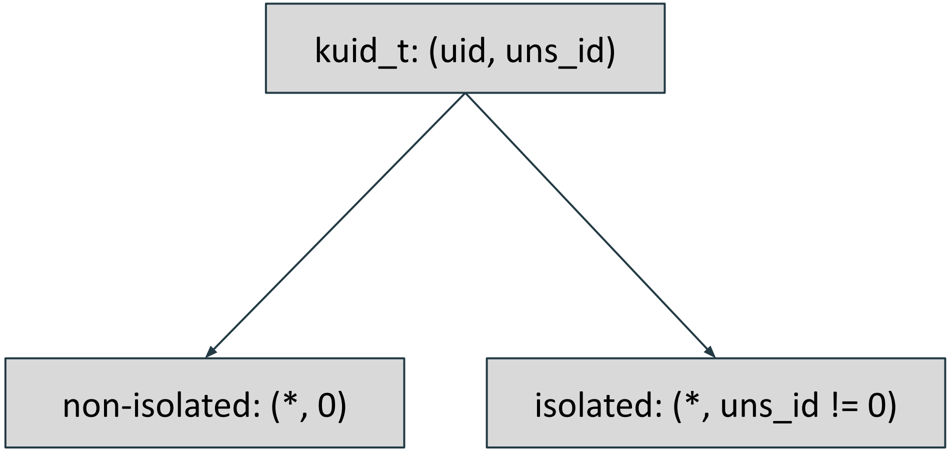
Extending k{u,g}id_t to be 64bit

Stéphane Graber and Christian Brauner gave a talks:

- LPC 2020: Isolated dynamic user namespaces [1]
- LPC 2021: Simplified user namespace allocation [2]

On of the outcome from these discussion was an idea to extend existing k{u,g}id_t types to be 64bit. One 32bit part will have the same meaning (UID or GID), while the other will be a user namespace identificator.

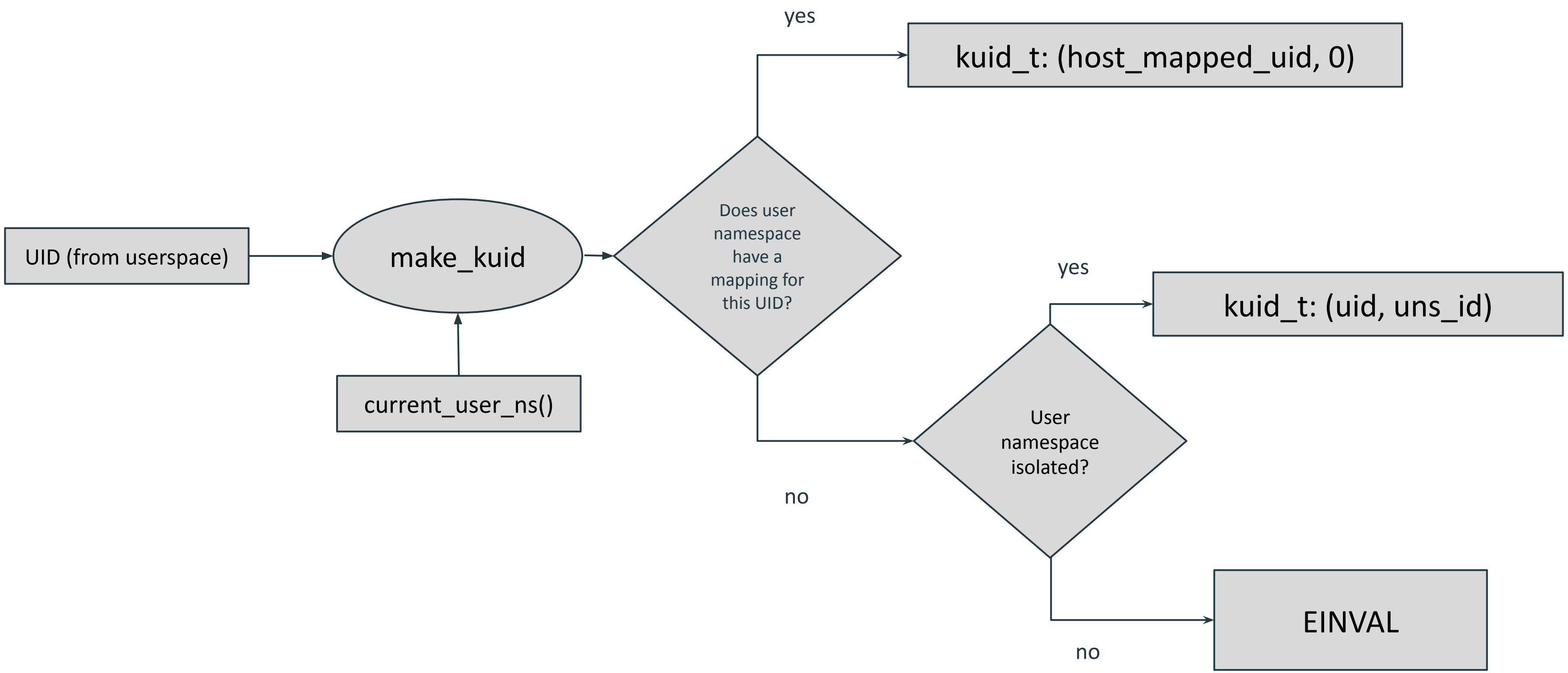
k{u,g}id_t



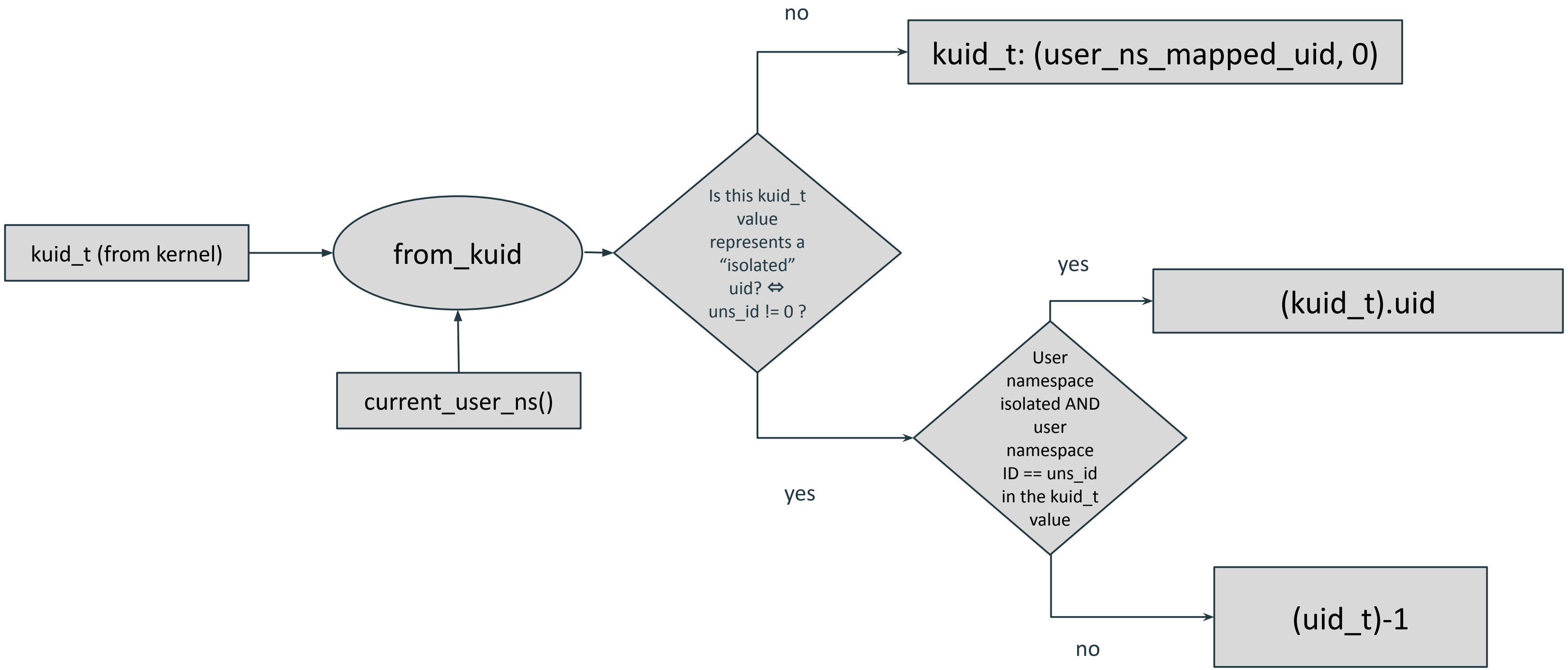
Extending k{u,g}id_t to be 64bit (cont. 2)

5. If a user namepace is isolated then it is allowed to create and use isolated UID/GIDs. (Currently we don’t apply any limits on that and in the isolated user namespace the whole 32bit UID/GID range is available to use by default.)
6. When user space calls setuid syscall kernel decides if this UID is “isolated” or not dynamically. For example, if the user namespace has uid_map that maps 10 (inside) to 1000 (in init_user_ns) and user calls setns(10) then non isolated kuid_t will be created with the UID value 1000. (Same behavior as we have now.) If, for example, UID 100000 is not mapped to the host and user calls setuid(100000) then, without isolated user namespace we will get EINVAL. But if the user namespace is isolated then isolated kuid_t will be created with UID = 100 000 and user namespace ID equal to the current user namespace ID.

Mapping procedure (setuid syscall)



Inverse mapping procedure (getuid syscall)


no
kuid_t: (user_ns_mapped_uid, 0)
kuid_t (from kernel)
from_kuid
Is this kuid_t value represents a "isolated" uid? ⇔ uns_id != 0 ?
yes
(kuid_t).uid
current_user_ns()
User namespace isolated AND user namespace ID == uns_id in the kuid_t value
yes
(uid_t)-1
no

Corollaries

1. We can represent any UID as kuid_t value in a isolated user ns
2. If a user ns has mapping for UID then it will be represented as a “normal” (non-isolated) kuid_t
3. If we have two different user namespaces (and both isolated) and we have UID 1 that can’t be mapped to the host kuid_t in a classical way then it will be represented as kuid_t like (1, first_uns_id) and (1, second_uns_id). It means that we can’t map (1, first_uns_id) kuid_t value back from the host to the user namespace 2, because first_uns_id != second_uns_id.

VFS idmap integration

To be able to create an inode on the filesystem from inside of isolated user namespace we require:

- UID/GID have to be mapped (non-isolated case)
- UID/GID have can be non-mapped, but:
  - Filesystem superblock user namespace (sb->s_user_ns) is the same as writer's one
  - Filesystem superblock user namespace is init_user_ns AND idmapped mount is used with idmap user namespace same as writer's one

From the first glance this looks a little bit weird, but if you think about how idmapped mount works when filesystem is mounted in the initial user namespace and idmapping is attached to the writer's user namespace you will see that this is very-very intuitive thing. (In this case local UID inside the user namespace will be written on disk as it is without any transformation!)

Problems to solve

- We expect that isolated user namespace will have some non isolated UID/GIDs, at least UID zero mapped to some UID on the host.
- While we have support for VFS integration when filesystem superblock attached to the container user namespace (sb->s_user_ns = container_user_ns) and when filesystem superblock attached to the host AND we use idmapped mount for this filesystem with the idmap that holds the same user namespace as container has. There is a problem with cgroupfs.
- We have not played with nesting here and it’s still a good question
- Integration with unix sockets (SCM_CREDENTIALS) between processes from the host and container and between two different (isolated) containers?

What’s the problem with cgroupfs

- Cgroupfs superblock is shared between all containers on the machine
- When we enable cgroup namespace that enables cgroupfs containerization it just shifts the root dentry of cgroupfs mount when cgroupfs is mounted by a process inside the cgroup namespace [5].
- Why it’s a problem?
- Because we have inode->i_{u,g}id fields for each inode which holds a k{u,g}id_t values. For procfs and sysfs, when they are mounted inside the container these values will be isolated ones! Another example is ext4 mounted on the host and idmapped bindmount of it inside the container, in this case k{u,g}id_t values will be transformed by the VFS idmap machinery to fit with the isolated user namespace context.
- Procfs, sysfs are allowed to be mounted inside the user namespace and superblock is unique. (To be precise, for sysfs it’s unique in different network namespaces and for procfs it will always has a unique superblock for each mount (of course, if it is not a bind mount).)