Contribution ID: **171** Type: **not specified**

Simplified user namespace allocation

Monday, 20 September 2021 07:05 (25 minutes)

The user namespace currently relies on mapping UIDs and GIDs from the initial namespace (full uint32 range) into the newly created user namespace. This is done through the use of uid_map/gid_map with the kernel allowing mapping your own uid/gid and otherwise requiring a privileged process write a more complete map.

As more and more software (not just container managers) are making use of user namespaces, a few patterns and problems have started to emerge:

- For maximum security, containers should get their own, non-overlapping uid/gid range. Non-overlapping here means no shared uid/gid with any other container on the system as well as no shared uid/gid with any user/processes running at the host level. Doing this prevents issues with configuration tied to a particular uid/gid which may cross container boundaries (user limits are/were known to do that) as well as prevent issues should the container somehow get access to another's filesystem.
- Network authentication as well as dynamic user creation is causing more "high" uid/gid being used than before. So while it was perfectly fine to give a contiguous range of 65536 uid/gid to a container before, nowadays, this often needs to be extended to 1-2M to support network authentication and may need a bunch of mapped uid/gid right at the end of the 32bit range to handle some temporary dynamic users too.
- Coordination on uid/gid range ownership in theory was done through the use of shadow's /etc/subuid, /etc/subgid and the newuidmap/newgidmap helpers. In practice those have only ever really been used when containers are created/started by non-root users and are generally ignored by any tool which operates as root. This effectively means there's no coordination going on today and its' very easy to accidentally assign a uid/gid to a user namespace which may in fact be used by the host system or by another user namespace.

The bulk of those constraints come from the fact that the user namespace has to deal with filesystem access and permissions. Having a uid in the container translated to a real uid and have that be what's used to access the VFS. If a uid cannot be translated, it's invalid and can't be used.

But now that we have a VFS layer feature to support ID shifting, we may be able to decouple the two and allow for user namespaces that have access to the full uid/gid range (uint32) yet are still technically using completely distinct kuid/kgid from everything else. This would then rely on the use of VFS based shifting for any case where data must be accessed from outside of that namespace.

This kind of approach would make it trivial to allocate new user namespaces, would drop the need for coordination and avoid conflicts with host uid/gid. Anyone could safely get a user namespace with access to all uid/gid but restricted to virtual filesystems. Then with help from a privileged helper could get specific mounts mapped into their namespace allowing for VFS operations with the outside world.

There are some interesting corner cases though, like what do we do when transferring user credentials across namespace boundaries. As in, how do we render things on the parent user namespace, whether it's the ownership of a process or the data in a ucred.

I agree to abide by the anti-harassment policy

I agree

Primary authors: GRABER, Stéphane (Canonical Ltd.); BRAUNER, Christian

Presenters: GRABER, Stéphane (Canonical Ltd.); BRAUNER, Christian

Session Classification: Containers and Checkpoint/Restore MC

Track Classification: Containers and Checkpoint/Restore MC