Subject: Re: [PATCH 10/10] sysfs: user namespaces: add ns to user_struct
Posted by serue on Wed, 30 Apr 2008 21:04:15 GMT
View Forum Message <> Reply to Message

Quoting Eric W. Biederman (ebiederm@xmission.com):
> "Serge E. Hallyn" <serue@us.ibm.com> writes:
>
> >> > Index: linux-mm/include/linux/sched.h
> >> > ===
> >> > --- linux-mm.orig/include/linux/sched.h
> >> > +++ linux-mm/include/linux/sched.h
> >> > @@ -598,7 +598,7 @@ struct user_struct {
> >> >
> >> > /* Hash table maintenance information */
> >> > struct hlist_node uidhash_node;
> >> > - uid_t uid;
> >> > + struct k_uid_t uid;
> >>
> >> If we are going to go this direction my inclination
> >> is to include an array of a single element in user_struct.
> >>
> >> Maybe that makes sense. I just know we need to talk about
> >> how a user maps into different user namespaces. As that
> >
> > My thought had been that a task belongs to several user_structs, but
> > each user_struct belongs to just one user namespace. Maybe as you
> > suggest that's not the right way to go.
> >
> > But are you ok with just sticking a user_namespace * in here for now,
> > and making it clear that the user_struct-user_namespace relation is yet
> > to be defined?
> >
> > If not that's fine, we just won't be able to clone(CLONE_NEWUSER)
> > until we get the relationship straightened out.
> >
> >> is a real concept that really occurs in real filesystems
> >> like nfsv4 and p9fs, and having infrastructure that can
> >> deal with the concept (even if it doesn't support it yet) would be
> >> useful.
> >
> > I'll have to look at 9p, bc right now I don't know what you're talking
> > about. Then I'll move to the containers list to discuss what the
> > user_struct should look like.
>
> Ok. The concept present in nfsv4 and 9p is that a user is represented
> by a username string instead by a numerical id. nfsv4 when it encounters
> a username it doesn't have a cached mapping to a uid calls out to userspace to
> get that mapping. 9p does something similar although I believe less general.

>
> The key point here is that we have clear precedent of a mapping from one user
> namespace to another in real world code. In this case nfsv4 has one user
> namespace (string based) and the systems that mount it have a separate
> user namespace (uid based).
>
> Once user namespaces are fleshed out I expect that same potential to
> exist. That each user namespace can have a different uid mapping for
> the same username string on nfsv4.
>
> >From uid we current map to a user struct. At which point things get a
> little odd. I think we could swing either way. Either keeping kernel
> user namespaces completely disjoint or allowing them to be mapped to
> each other.
>
> I certainly like the classic NFS case of mapping uid 0 to user nobody
> on a nonlocal filesystem (outside of the container in our case) so the
> don't accidentally do something that root only powers would otherwise
> allow.
>
> In general I think managing mapping tables between user namespaces is
> a pain in the butt and something to be avoided if you have the option.
> I do see a small place for it though.
>
> Eric

No sense talking about how to relate uids+namespaces to user_structs to
task_structs without first laying out a few requirements. Here is the
list I would start with. I'm being optimistic here that we can one day
allow user namespaces to be unshared without privilege, and gearing the
requirements to that (in fact the requirements facilitate that):

==========
Requirement:
==========
when uid 500 creates a new userns, then:

1. uid 500 in parentns must be able to kill tasks in the container.
2. uid 500 must be able to create, chown, change user_ns, delete
   files belonging to the container.
3. tasks in a container should be able to get 'user nobody'
   access to files from outside the container (/usr ro-remount)
4. uid 400 in the container created by uid 500 must not be able
   to read files belonging to uid 400 in the parent userns
5. uid 400 in the container created by uid 500 must not be able
   to signal tasks by uid 400 in parent user_ns (*1)
6. a privileged app in the container created by uid 500 must not
   get privilege over tasks outside the container (*1)
7. a privileged app in the container created by uid 500 must not

get privilege over files outside the container (*2)

*1: this should be mostly impossible if we have CLONE_NEWUSER|CLONE_NEWPID
*2: the feasability of this depends entirely on what we do to tag fs.

Based on that I'd say that the fancier mapping of uids between containers really isn't necessary, and if needed it can always be emulated using i.e. nfsv4 to do the actual mapping of container uids to usernames known by the network fs.

But we also need to decide what we're willing to do for the regular container filesystem. That's where I keep getting stuck.

Do we tag each inode with a user_namespace based on some mount context? Do we tag some files with a persistent 'key' which uniquely identifies a user in all user namespaces (and across reboots)? Do we implement a new, mostly pass-through stackable fs which we mount on top of an existing fs to do uid translation? Do we force the use of nfsv4? Do we rely on an LSM like SELinux or smack to provide fs isolation between user namespaces? Do we use a new LSM that just adds security.userns xattrs to all files to tag the userns?

Heck, maybe nfsv4 is the way to go. Admins can either use nfsv4 for all containers, or implement isolation through SELinux/Smack, or accept that uid 0 in a container has access to uid 0-owned files in all namespaces plus capabilities in all namespaces.

Note that as soon as the fs is tagged with user namespaces, then we can simply have task->cap_effective apply only to tasks and files in its own user_ns, so CAP_DAC_OVERRIDE in a child userns doesn't grant you privilege to files owned by others in another userns. But without that, CAP_KILL can be contained to tasks within your own userns, but CAP_DAC_OVERRIDE in a child userns can't be contained.

-serge