Subject: Re: [PATCH 10/10] sysfs: user namespaces: add ns to user_struct
Posted by ebiederm on Wed, 30 Apr 2008 06:47:34 GMT
View Forum Message <> Reply to Message

"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

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

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

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 10/10] sysfs: user namespaces: add ns to user_struct
Posted by ebiederm on Wed, 30 Apr 2008 22:13:53 GMT
View Forum Message <> Reply to Message

"Serge E. Hallyn" <serue@us.ibm.com> writes:

> 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)

Sounds like a reasonable set of requirements.
>
> *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.

Then lets look at this a couple of different ways.
- Filesystems have an internal user namespace that they use
  for their permissions checks, and they have some means
  of mapping that user namespace into the kernels user namespace.
  Usually this is a one-to-one mapping. But occasionally
  in cases like nfsv4 or fat there is a more interesting
  mapping scheme going on.

For filessytems on remote servers and removable media like
usb keys this concept of the filesystems user namespace is
seems relevant.

In practice this gives us two classes of filesystems we have
to work with.
- Multiple user namespace aware filesystems like nfsv4.
- Normal filesystems that do a one-to-one mapping between
kernel uids and filesystem uids.

> Do we tag each inode with a user_namespace based on some mount context?
Last time this was discussed the sane thing appeared to be tagging the
vfs_mount structure with the namespace of the mount. And having
the default permission operations reference back to the mount point.

For multiple namespace aware filesystems this seems especially useful.
As this does not pollute the pure filesystem structures like the
superblock and allows nfs do all of it's superblock merging tricks.

> Do we tag some files with a persistent 'key' which uniquely identifies a
> user in all user namespaces (and across reboots)?

I think we leave that up to the filesystem or a stackable filesystem
that rides on top of the filesystem. With a stackable filesystem
we should be able to easily implement the vserver trick of using
high bits of the uid to indicate which uid namespace we care about.

> Do we implement a
> new, mostly pass-through stackable fs which we mount on top of an
> existing fs to do uid translation?

Probably, for supporting older filesystems. Unless we choose to make
a more interesting translation layer like what nfsv4 uses and just
upgrade classic unix filesystems to use that.

> 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.
Let's make that our first target general solution. Something that handles
multiple user namespaces at the filesystem level now.

> 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.

As for the uid 0 in a container problem. I still would like to make
the default filesystem permissions checks for user equality be essentially:

vfs_mount.user_namespace == task.user_namespace
inode.uid == task.uid

That saves us from most of the trouble. And in particular almost immediately
makes most of /proc and /sysfs container safe.

> 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.

Right. We still need to find a way to describe the case of allowing the
creator of the container to have access to everything the same way root
does today.

Eric

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 10/10] sysfs: user namespaces: add ns to user_struct
Posted by serue on Fri, 02 May 2008 22:21:34 GMT
View Forum Message <> Reply to Message

Quoting Eric W. Biederman (ebiederm@xmission.com):
> "Serge E. Hallyn" <serue@us.ibm.com> writes:
>
> > 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)
>
> Sounds like a reasonable set of requirements.
> >
> > *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.
>

> Then lets look at this a couple of different ways.
> - Filesystems have an internal user namespace that they use
> for their permissions checks, and they have some means
> of mapping that user namespace into the kernels user namespace.
> Usually this is a one-to-one mapping. But occasionally
> in cases like nfsv4 or fat there is a more interesting
> mapping scheme going on.
>
> For filessytems on remote servers and removable media like
> usb keys this concept of the filesystems user namespace is
> seems relevant.
>
> In practice this gives us two classes of filesystems we have
> to work with.
> - Multiple user namespace aware filesystems like nfsv4.
> - Normal filesystems that do a one-to-one mapping between
> kernel uids and filesystem uids.
>
> > Do we tag each inode with a user_namespace based on some mount context?
> Last time this was discussed the sane thing appeared to be tagging the
> vfs_mount structure with the namespace of the mount. And having
> the default permission operations reference back to the mount point.
>
> For multiple namespace aware filesystems this seems especially useful.
> As this does not pollute the pure filesystem structures like the
> superblock and allows nfs do all of it's superblock merging tricks.
>
> > Do we tag some files with a persistent 'key' which uniquely identifies a
> > user in all user namespaces (and across reboots)?
>
> I think we leave that up to the filesystem or a stackable filesystem
> that rides on top of the filesystem. With a stackable filesystem
> we should be able to easily implement the vserver trick of using
> high bits of the uid to indicate which uid namespace we care about.
>
> > Do we implement a
> > new, mostly pass-through stackable fs which we mount on top of an
> > existing fs to do uid translation?
>
> Probably, for supporting older filesystems. Unless we choose to make
> a more interesting translation layer like what nfsv4 uses and just
> upgrade classic unix filesystems to use that.
>
> > 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.
> Let's make that our first target general solution. Something that handles
> multiple user namespaces at the filesystem level now.
>
> > 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.
>
> As for the uid 0 in a container problem. I still would like to make
> the default filesystem permissions checks for user equality be essentially:
>
> vfs_mount.user_namespace == task.user_namespace
> inode.uid == task.uid
>
> That saves us from most of the trouble. And in particular almost immediately
> makes most of /proc and /sysfs container safe.
>
> > 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.
>
> Right. We still need to find a way to describe the case of allowing the
> creator of the container to have access to everything the same way root
> does today.

>From a simple ownership point of view that wouldn't be too hard.
To do that we don't need uid-mapping in the kernel, all we need
is a concept of user_structs owning user_namespaces they created.
Then the default vfs_permission becomes:

1. if (user->uid==inode->uid && user->uidns==vfsmount->userns)
   treat as owner
2. if (user->uidns==vfsmount->userns)
   check groups
3. if (vfsmount->userns is in user->child_namespaces)
   treat as uid 0
4. treat as nobody

But as I was laying that out and trying define sane semantics,
the following obvious shortcoming sprang out. It would have
been obvious if I'd given some fs semantics requirements right
at the top:

Let's use X:Y to describe uid X in userns Y. Let's assume

the behavior described above, and that we tag vfsmounts with the user_namespace of the user_struct whose task performed the mount.

When user 500:1 creates a container with uidns2, wherein he uses uids 0:2 and 400:2, then:

1. files belonging to 500:1 should be treated no differently than files belonging to any other X:1. The container init can mount --bind it's / early on using user nobody permissions, so this is sufficient.
2. files created by 0:2 should be owned by 0:2 in the container.

BUT

3. files created by 0:2 should not be owned by uid 0 in the parent container (0:1).
4. when a task executes a file owned by 0:2 or a file owned by X:2 carrying file capabilities, the resulting task should carry privilege over objects in userns 2, not over objects in userns 1.

So simple tagging of vfsmounts can only suffice if we insist on tagging newly created files with a uid in the initial user_namespace. Then to do anything fancier - really, to do anything sufficient for system containers - we'd have to use one of the other things we described above - nfsv4, or a mostly-pass-through stackable fs, or whatever.

-serge

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers