Subject: Re: design of user namespaces
Posted by serue on Fri, 20 Jun 2008 20:55:08 GMT
View Forum Message <> Reply to Message

Quoting Eric W. Biederman (ebiederm@xmission.com):
> "Serge E. Hallyn" <serue@us.ibm.com> writes:
> >
> > Hi Eric,
> >
> > glad you're giving this some thought. Did you ever read over the
> > approach which I outlined in May (see
> > http://forum.openvz.org/index.php?t=msg&goto=30223&)? We agree on many
> > points. I think we basically solve the suid problem the same way.
>
> I hadn't read that post although I saw a part of it in your paper.
> Solving that problem so unprivileged users can use a user namespace
> seems to be the key to making namespaces more widely usable, and
> recursive. Plus I really like the idea of a super nobody user.
>
> > But I've moved away from a uid-to-uid mapping. Instead,
> > I expand on the relation you also describe: the user who creates a user
> > namespace owns the user namespace and introduce a persistant
> > namespace ID.
>
> I think there are management issues with a persistent namespace ID.

There are. But one key point is that the namespace ids are not
cryptographic keys. They don't need to be globally unique, or even 100%
unique on one system (though that gets too subtle).

I was wanting to keep them tiny - or at least variably sized - so they
could be stored with each inode.

> In particular: Is this user allowed to use this ID?

One way to address that is actually by having a system-wide tool with
CAP_SET_USERNS=fp which enforces a userid-to-unsid mapping. The host
sysadmin creates a table, say, 500:0 may use unsid 10-15. 500:0 (let's
call him hallyn) doesn't have CAP_SET_USERNS permissions himself, but
can run /bin/set_unsid, which runs with CAP_SET_USERNS and ensures that
hallyn uses only unsids 10-15.

It's not ideal. I'd rather have some sort of fancy collision-proof
global persistant id system, but again I think it's important that if
500:0 creates userns 1 and 400:1 creates userns 2, and 0:2 creates a
file, that the file be persistantly marked as belonging to
(500:0,400:1,0:2), distinct from another file created by
(500:0,400:1,1000:2). Which means these things have to be stored

per-inode, meaning they can't be too large.

> That is the reason I want to avoid having a generic persistent
> namespace ID. Implementing a common library and then modifying the
> filesystems to use it on a case by case basis sounds much more
> maintainable. Then picking the wrong direction does not bind us for
> all time and eternity.
>
> > I understand that you may reflexively balk at introducing a new global
> > persistant ID when we're focusing on turning everything into a
> > namespace, but I think that would be a misguided reflex (like the
> > ioctl->netlinke one of a few years ago). In particular, in your
> > approach the identifier is the combination of the uid-to-uid mapping and
> > the uids stored on the original filesystem.
>
> Not at all. I thought I had mentioned the xattr thing as one of the
> possibilities. I forgot the proper term so I probably said acls. The
> practical problem is that you then have to rev your quota support. To
> also support the xattr separation. In addition not every filesystem
> supports xattrs. Although the common ones do.

Again each fs could do whatever it wanted, and perhaps reiser would not
use xattrs but some funky reiserfs-ish thing :) I'm just talking about
xattrs bc that's what I'd use, on top of ext2/3, in any prototype.

Also, I think we're all agreed that for some filesystems, the semantics
of:

 1. filesystem is owned by current->user->user_ns who mounted it
  (complicated by mounts propagation the same way as it
  was with user mounts)
 2. access by user in another namespace == access by user nobody

make sense.

For quota support, I see where we'll have to check quota for each user
in the creator chain to which the task belongs - is that what you're
referring to?

> > I do think the particular form of the ID I suggest will be unsuitable
> > and we'll want something more flexible. Perhaps stick with the unsid
> > for the legacy filesystems with xattr-unsid support, and let advanced
> > filesystems like nfsv4, 9p, and smb use their own domain
> > identifiers.
>
> Which is why I said make it filesystem specific with support from a
> generic library. No prctl just a mount option.

Now when you say a generic library, are you referring to a userspace
daemon, queried by the kernel, which does uid translation? Do you have
a particular api in mind? What sort of commands and queries would the
kernel send to that daemon?

> > But since we seem to agree on the first part - introducing a hierarchy
> > between users and the namespaces they create - it sounds like the
> > following patch would suit both of us. (I just started implementing my
> > approach this past week in my free time). I'm not sending this as any
> > sort of request for inclusion, just bc it's sitting around...
>
> Yes. At least a loose hierarchy.
>
> It just occurred to me that with unix domain sockets, some signals, /proc
> we have user namespace crossing (child to parent) where we need to
> report the uid. In that case the simple solution appears to be to use
> the overflowuid and overflowgid that were defined as part of the 16bit
> to 32bit transition. Otherwise it would require mapping all of the
> child uids into the parent like we do with pids except using an upcall
> to userspace.

Again, in my proposal, each child user is also owned by the parent
user, so it should be possible to send the uid at the right user
namespace level the way we do with find_task_by_vpid() for pid
namespaces.

I know, there are still places in the signal code where we
fire-and-forget before we know the target, so those would be a problem.
The whole issue of putting a uid in a siginfo, when we don't know the
target and don't have a lifecycle on a siginfo, is a problem I've been
dreading for a long time now.

> Making capabilities user namespace specific if we can.
>
> Did you have any problems with adding a user_namespace argument to
> capable?

I had started that in a much earlier patchset a year or two ago. Calls
to capable(CAP_KILL) and capable(CAP_DAC_OVERRIDE) needed to be changed
to new functions accepting a target file/task. I don't recall it being
a big deal actually. Far more interesting is what to do with the
other capabilities, like CAP_NET_ADMIN.

As for bounding sets, imo we don't bother the user namespace code with
capabilities at all. We just let the admin specify bounding sets for
containers. For starters we recommend always pulling out things like
CAP_NET_ADMIN, then we can take our time discussing their safety. (i.e.
even if I don't have CAP_NET_ADMIN, if I clone(CLONE_NEWNET), is there

any reason not to give me CAP_NET_ADMIN in the result? After all I need
a capable parent in another namespace to either pass me a device or tie
any veth devices I create into a bridge before I can do any damage...)

> Just skimming through your patch I don't expect we will need the list
> of children, and not having should reduct our locking burden.

Hmm, that's true. I can't see a reason for that. Thanks!

So can you send some more info on the API you envision for the
library/daemon?

thanks,
-serge
____