Subject: Re: design of user namespaces
Posted by ebiederm on Fri, 20 Jun 2008 23:00:12 GMT
View Forum Message <> Reply to Message

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

> 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.

Sure. I think they make a lot of sense. idmapd uses domain names
for this purpose. At the moment I just don't think the are necessary.
Like veth isn't necessary for network namespaces. Ultimately we
will use these identifiers all of the time but it doesn't mean
the generic code has to deal with them.

>> 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.

So my suggestion was something like this:
mount -o nativemount,uidns=1 /

Then the filesystem performs magic to ask if the owner of user namespace
is allowed to use uidns 1. That magic would consult a config file like:

[domains]
local1.mydomain 1
local2.mydomain 2
local3.mydomain 3

[users]
  bob   local1.mydomain
  bob   local3.mydomain
  nancy local2.mydomain

Or something like that. Reporting which users are allowed to use
which userid namespaces, and the mapping of those userid namespaces
to something compact for storing in the filesystem.

The magic could be an upcall to userspace.
The magic could be loading the configuration file at mount time.
The magic could be storing the config file in the filesystem
and having special commands to access it like quotas.

The very important points are that it is a remount of an existing mount
so that we don't have to worry about corrupted filesystem attacks, and
that authentication is performed at mount time.

I just think that once we get to the point of specifying the parameters at
mount time. There is no need for generic kernel configuration of a
uidns name.

>>
>> > 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.

Sounds good to me. Each fs doing whatever it wants so seems the most
sensible approach.

> 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)
Yes.
> 2. access by user in another namespace == access by user nobody
>
> make sense.

yes.

> 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?

Actually no. Although that may eventually come up. At that level user namespaces
may be completely disjoint. Mostly I was referring to the fact that quotas
as I understand them are per filesystem per uid, and don't take the xattr into
account. So you conflict with multiple uid namespaces when you reuse the uid.

>> > 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?

I was mostly thinking of something like jbd. That would allow posix
filesystems that all want to implement uid mappings the same way to
use the same code. At which point we can also reuse the same user
space support.

>> > 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.

We actually do know the target at the time the signal is queued up.
We should solve this for the pid namespace first. The patches exist
they just need to get off their sorry buts and get themselves merged.

There are a few issues that cause me to wonder if the user namespace
will not be easy to unshare for similar reasons to the pid namespace.

As for which uid to use. Duh! We use the uid of the creator of the user
namespace. How I missed that one I know. We still have the case of
completely disjoint user namespaces. In which case we do want to use
the mostly reserved overflowuid. Some non 0 uid reserved to indicate
that there is no mapping for this uid in your user namespace. Although
we have just enough mappings we may be able to get away with returning
-EIO for all the other cases.

> 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...)

Ok. This part will probably take a little more discussion.
My viewpoint is different.

I think as a fundamental fact that all capabilities and other security
identifiers (selinux lables) should be local to the user namespace.

In the case of capabilities then the result is that a capability
is either potent or impotent in the user_namespace.

So CAP_SYS_RAWIO. Would only be potent in the initial user_namespace.

CAP_NET_ADMIN would only be potent on network namespaces created in
the user namespace (and the children of the user namespace).

Likewise for the other capabilities.

Therefore the first process in a user namespace (that only unshared
the user namespace) will have all capabilities and will simply not be
able to do anything with them.

So capable becomes something like:

```
int capable(struct user_namespace *ns, int cap)
{
 return __capable(current, cap);
}

int __capable(struct task_struct *task, struct user_namespace *ns, int cap)
{
        /* The check below should check the parent user namespaces too */
        if (task->nsproxy->user_ns != ns)
         return -EPERM;
 if (!cap_raised(task->cap_effective, cap))
         return -EPERM;
        return 0;
}
```

Calls to capable initially pass in init_user_ns. And then as we fixup
things like the network namespaces we call
"capable(net->user_ns, CAP_NET_ADMIN);"

Similarly for the other paths. That should be incremental and safe.

>> 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?

Let's talk about the functionality and the API should become obvious
as a logical consequence.

In the case of read only access I don't think we need any user space
configuration at all. Just a filesystem that is ok with it.

mount -o nativemount,ro /

Eric

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