Subject: Re: containers access control 'roadmap'
Posted by serue on Thu, 06 Sep 2007 18:26:11 GMT
View Forum Message <> Reply to Message

Quoting Herbert Poetzl (herbert@13thfloor.at):
> On Thu, Sep 06, 2007 at 11:55:34AM -0500, Serge E. Hallyn wrote:
> > Roadmap is a bit of an exaggeration, but here is a list of the next bit
> > of work i expect to do relating to containers and access control. The
> > list gets more vague toward the end, with the intent of going far enough
> > ahead to show what the final result would hopefully look like.
> >
> > Please review and tell me where I'm unclear, inconsistant, glossing over
> > important details, or completely on drugs.

Thanks for looking this over, Herbert.

> > 1. introduce CAP_HOST_ADMIN
> >
> > acts like a mask. If set, all capabilities apply across
> > namespaces.
> >
> > is that ok, or do we insist on duplicates for all caps?
> >
> > brings us into 64-bit caps, so associated patches come
> > along
> >
> > As an example, CAP_DAC_OVERRIDE by itself will mean within
> > the same user namespace, while CAP_DAC_OVERRIDE|CAP_HOST_ADMIN
> > will override userns equivalence checks.
>
> what does that mean?
> guest spaces need to be limited to a certain (mutable)
> subset of capabilities to work properly, please explain

(note that that muable subset of caps for guest spaces is what item #2,
the per-process cap_bset, implements)

> how this relates?

capabilities will give you privileged access within your own container.
Also having CAP_HOST_ADMIN will mean that the capabilities you have can
also be used against objects in other containers.

Now maybe you prefer a model where a "container" is owned by some user
in some namespaces. All capabilities apply purely within their own
namespace, and a container owner has full rights to the owned
containers. That makes container vms more like a qemu vm.

Or maybe I just punt this for now altogether, and we address
cross-namespace privileged access if/when we really need it.

> > 2. introduce per-process cap_bset
> >
> > Idea is you can start a container with cap-bset not containing
> > CAP_HOST_ADMIN, for instance.
> >
> > As namespaces are fleshed out and proper behavior for
> > cross-namespace access is figured out (see step 7) I
> > expect behavior under !CAP_HOST_ADMIN with certain
> > capabilities will change. I.e. if we get a device
> > namespace, CAP_MKNOD will be different from
> > CAP_HOST_ADMIN|CAP_MKNOD, and people will want to
> > start keeping CAP_MKNOD in their container cap_bsets.
>
> doesn't sound like a good idea to me, ignoring caps
> or disallowing them seems okay, but changing the meaning
> between caps (depending on host or guest space) seems
> just wrong ...

Ok your 'doesn't sound like a good idea' is to my blabbing though, not
the the per-process cap_bset. Right? So you're again objecting to
CAP_HOST_ADMIN, item #1?

> > 3. audit driver code etc for any and all uid==0 checks. Fix those
> > immediately to take user namespaces into account.
>
> okay, sounds good ...

Ok maybe i should make that '#1' and get going as it's the least
contraversial :)

Though I think I still prefer to start with #2.

> > 4. introduce inode->user_ns, as per my previous userns patchset from
> > April (I guess posted in June, according to:
> > https://lists.linux-foundation.org/pipermail/containers/2007-June/005342.html)
> >
> > For now, enforce roughly the following access checks when
> > inode->user_ns is set:
> >
> > if capable(CAP_HOST_ADMIN|CAP_DAC_OVERRIDE)
> > allow
> > if current->userns==inode->userns {
> > if capable(CAP_DAC_OVERRIDE)
> > allow
> > if current->uid==inode->i_uid

> > allow as owner
> > inode->i_uid is in current's keychain
> > allow as owner
> > uid==inode->i_gid in current's groups
> > allow as group
> > }
> > treat as user 'other' (i.e. usually read-only access)
>
> what about inodes belonging to several contexts?

There's no such thing in the way I was envisioning it.

An inode belongs to one context. A user can belong to several.

> (which is a major resource conserving feature of OS
> level isolation)

Sure. Let's say you want to share /usr among many servers. It exists in the host user namespace. In guest user namespaces, anyone including root will have access to them as though they were user 'other', i.e. if a directory has 751 perms, you'll get '1'.

> > 5. Then comes the piece where users can get credentials as users in
> > other namespaces to store in their keychain.
>
> does that make sense? wouldn't it be better to have
> the keychains 'per context'?

Either you misunderstood me, or I misunderstand you.

What I am saying is that there is a 'uid' keychain, which holds things like (usernamespace 3, uid 5), meaning that even though I am uid 1000 in usernamespace 1, I am allowed access to usernamespace 3 as though I were uid 5.

I expect the two common use cases of this to be:

1. uid 5 on the host system created a virtual server, and gives himself a (usernamespace 2, uid 0) key so he is root in the virtual server without having to enter it. (Meaning he can signal all processes, access all files, etc)

2. uid 3000 on the host system is given (usernamespace 2, uid 1001) in a virtual server so he can access uid 1001's files in the virtual server which has usernamespace 2.

> > 6. enforce other userns checks like signaling
> >

> > 7. investigate proper behavior for other cross-namespace capabilities.
>
> please elaborate ....

Just that we need to go through the list of capabilities and consider
what they mean with and without CAP_HOST_ADMIN. For instance
CAP_IPC_LOCK doesn't really matter for CAP_HOST_ADMIN since the
namespaces prevent you cross-ns access. Implications for CAP_NET_ADMIN
remain to be seen, when network namespaces are complete.

-serge

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