Subject: Re: containers access control 'roadmap'
Posted by serue on Fri, 07 Sep 2007 18:18:42 GMT
View Forum Message <> Reply to Message

Quoting Serge E. Hallyn (serue@us.ibm.com):
> Quoting Herbert Poetzl (herbert@13thfloor.at):
> > > For instance CAP_IPC_LOCK doesn't really matter for
> > > CAP_HOST_ADMIN since the namespaces prevent you cross-ns
> > > access.
> >
> > hmm? maybe I am misunderstanding the entire concept
> > of CAP_HOST_ADMIN here ... maybe an example could help?
>
> I've obviously botched this so far... Let me whip up some examples of
> how it all works together and email those out tomorrow.
>
> thanks,
> -serge

Ok here is some ranting with an example:

System boots. All processes have all caps in their cap_bset.
Process 5155 does a clone(CLONE_NEWUSER|CLONE_NEWPID), returning
pid 6000, then does prctl(PR_SET_BCAP, ~CAP_HOST_ADMIN) to
take CAP_HOST_ADMIN out of it's bounding set, meaning it can
never, in any way, gain that capability.

pid 6000 is also (pidns 2, pid 1). The user owning that
process is (userns 2, uid 0).

Process 5155 does a simple clone(), returning pid 6001, and
that process does prctl(PR_SET_BCAP, ~CAP_HOST_ADMIN).

```
                PID 5155
                (pidns 1, pid 5155)
                id: (userns1,uid0)
                bcap: full

                     |
                     |
                   /   \
                 /       \
               /           \
          PID 6000       PID 6001
   (pidns 1, pid 6000)  (pidns 1, pid 6001)
    (pidns 2, pid 1)     (id: (userns 1, uid 0)
   id: (userns2, uid0)     bcap: ~CAP_HOST_ADMIN
  bcap: ~CAP_HOST_ADMIN
```

Process 6000 as root owns a file in its own chroot, let's call it
/vm1/foo. If process 5155 is still owned by root and tries
to access /vm1/foo, then since it has (CAP_HOST_ADMIN|CAP_DAC_OVERRIDE)
it will be able to access the file as root.

If process 6001 is still owned by root, it may have CAP_DAC_OVERRIDE, but
doesn' thave CAP_HOST_ADMIN, so can't cross the userns boundary into userns 2.
So it will get the 'other' perms to /vm1/foo. However CAP_DAC_OVERRIDE
will apply to let it access files in its own user namespace.

Note that if we were talking about 'host' versus 'guests', then
6001 would be a root process in the 'host'.

Note also that if pid 6000 hadn't dropped CAP_HOST_ADMIN, it would
be a 'guest' which was able to access other namespaces as though
it were the 'host' in a host-guest scheme.

When process 6000 access /vm1/foo, it is in the same userns,
and owns the file, so it can access it. If it does
setuid(1000), then it can only access /vm1/foo if it has
CAP_DAC_OVERRIDE. It doesn't need CAP_HOST_ADMIN because
it is not trying to cross a user namespace boundary.

(From here on, I'm *really* speculating, pie in the sky)

I've mentioned - and in previous patchsets started to implement -
that the inode would have a userns pointer. The filesystem
would pick who to assign an inode to - i.e. based on the superblock,
based on who mounted it, based on who created the file, whatever.
And users would get credentials for userids in other namespaces
through their keyrings.

Right now I'm thinking of taking the same idea but more
generally. Putting the userns in the inode is too restrictive.
For instance a novel filesystem might well want to ignore uids
altogether and use the keyring to determine file access.
So I'm thinking the filesystem both assigns and checks credentials
for an inode. For starters, ext2 just

1. assigns the user namespace of the process doing the
   mounting to the superblock
   OR
   if so specified at mounttime, assigns no userns so
   that all user namespaces may access the fs.
2. uses the sb->userns to enforce user namespace checks

Then as a next step it can continue to do the above, but also
allow use of credentials. Maybe the user who created a userns
with a clone(CLONE_NEWUSER) automatically gets a uid=0 credential
for the new user namespace. Or some other scheme.

Then, we can get into actually storing key hashes in an inode
xattr, and anyone with a key which hashes to the stored hash gets
access. Or some more cryptographically sound method of doing
that, please don't bother telling me all the ways that particular
example doesn't work :)

-serge