Subject: Re: [patch 1/8] add user mounts to the kernel
Posted by ebiederm on Sun, 22 Apr 2007 07:43:44 GMT
View Forum Message <> Reply to Message

Miklos Szeredi <miklos@szeredi.hu> writes:

>> > The MNT_USER flag is not copied on any kind of mount cloning:
>> > namespace creation, binding or propagation.
>>
>> I half agree, and as an initial approximation this works.
>> Ultimately we should be at the point that for mount propagation
>> that we copy the owner of the from the owner of our parent mount
>> at the propagation destination.
>
> Yes, that sounds the most sane.
>
> Ram, what do you think?
>
>> > + if (mnt->mnt_flags & MNT_USER)
>> > + seq_printf(m, ",user=%i", mnt->mnt_uid);
>> How about making the test "if (mnt->mnt_user != &root_user)"
>
> We don't want to treat root_user special. That's what capabilities
> were invented for.

For the print statement? What ever it is minor.

>> > Index: linux/include/linux/fs.h
>> > ===
>> > --- linux.orig/include/linux/fs.h 2007-04-20 11:55:02.000000000 +0200
>> > +++ linux/include/linux/fs.h 2007-04-20 11:55:05.000000000 +0200
>> > @@ -123,6 +123,7 @@ extern int dir_notify_enable;
>> > #define MS_SLAVE (1<<19) /* change to slave */
>> > #define MS_SHARED (1<<20) /* change to shared */
>> > #define MS_RELATIME (1<<21) /* Update atime relative to mtime/ctime. */
>> > +#define MS_SETUSER (1<<22) /* set mnt_uid to current user */
>>
>> If we unconditionally use the fsuid I think we can get away without
>> this flag.
>
> That coudl work if we wouldn't have to worry about breaking the user
> interface. As it is, we cannot be sure, that existing callers of
> mount(2) don't have fsuid set to some random value.

If we can get away without an extra flag it would really be
preferable.

In the container case we have an interesting and very common

scenario struct user *our_user != &root_user. our_user->uid == 0.

I.e. The root in the what is the container but not the root
of the entire system.

So I want to minimize the changes needed to existing programs.
Now if all we have to do is specify MS_SETUSER when root a
user with CAP_SETUID is setting up a mount as a user other
then himself then I don't much care. If we have to call MS_SETUSER
as unprivileged users I will have to modify mount binaries to work
differently inside and outside of containers.

Further there is only one or two versions of mount in widespread
use on linux, and unless you do something special fsuid == euid.
So the chance of fsuid set to some random value is pretty low.
So yes I think we can be 99.9% certain that existing callers of
mount(2) don't have fsuid set to some random value just by
inspecting the code of mount(1).

>> > #define MNT_SHRINKABLE 0x100
>> > +#define MNT_USER 0x200
>>
>> If we assign a user to all mount points and root gets to own the
>> initial set of mounts then we don't need the internal MNT_USER
>> flag.
>
> I think we do want to treat "owned" mounts special, rather than
> treating user=0 mounts special.

I don't think we should treat any mount special and all mounts
should be owned.

>> > +
>> > + uid_t mnt_uid;   /* owner of the mount */
>>
>> Can we please make this a user struct.   That requires a bit of
>> reference counting but it has uid namespace benefits as well
>> as making it easy to implement per user mount rlimits.
>
> OK, can you ellaborate, what the uid namespace benifits are?

In the uid namespace the comparison is simpler as are the propagations
rules. Basically if you use a struct user you will never need to
care about a uid namespace. If you don't we will have to tear through
this code another time.

Plus like I was mentioning earlier. If we do have a struct user there
implementing per user mount rlimits becomes trivial.

Eric

_____

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