Subject: user namespace semantics (may 16)
Posted by serue on Fri, 16 May 2008 23:48:26 GMT
View Forum Message <> Reply to Message

The following is a brainstorm dump after a conversation with Mike Halcrow about user namespaces. Basically, we can use xattrs to allow an fs to support universal user namespace ids (unsids), and place stricter controls otherwise.

Based on that, here is where I'm at right now. It probably looks more complicated here than it really is. I'm just trying to cover as many points here as I can think of right now.

```
struct user {
 ...
 int uid;
 struct user_namespace *ns;
 struct list_head child_user_ns;
};

struct unsid {
 int length;
 long id[];
};

struct user_namespace {
 ...
 struct user *creator;
 struct list_head siblings;
 struct unsid *unsid;
};
```

When a user creates a user_namespace, the new userns gets tacked to his user_struct->child_user_ns list, and his user struct is marked as the new_user_ns->creator. This supports kill+ptrace controls across usernamespaces. (*1) A new user struct, with uid 0 and in the new userns, is created and current->user is set to that.

The unsid is the 'Universal NameSpace IDentifier', and ties a userns to an fs. In other words the unsid (*2) lets us sanely correlate a userns, which is ephemeral, to a disk store, which is persistant.

The unsid is initially NULL, and gets set through prctl.

Setting unsid through prctl requires privilege (CAP_NS_OVERRIDE?). Perhaps we should pick some 'unsafe' unsid which unprivileged

tasks can set their unsid to? (*3)

```
struct vfsmount {
 ...
 struct user_namespace *userns;
};
```

At mount, the mnt->userns gets set to current->userns. Except:

If the fs is not FS_USER_NS, and this is a bind mount,
then new_mnt->users = orig_mnt->userns.

if (current->userns == mnt->userns), then use current behavior
for inode->getattr and inode->permission.

If the fs is not FS_USER_NS, then any access by a task with
current->userns!=mnt->userns will get the permissions of 'other',
and any file creation will be refused (we don't know what uid
to safely store in the inode).

If the fs is not FS_USER_NS, then any access by a privileged
(root or capable(CAP_DAC_OVERRIDE)) task with
current->userns!=mnt->userns will not get privilege.

If the fs is FS_USER_NS, then it must be able to base
access decisions on (current->uid, current->userns->unsid).

nfsv4, 9p, ext4, and ecryptfs can make the access decisions
however they like by defining inode_getattr and inode_permission.

Legacy filesystems can be compiled with CONFIG_X_FS_USERNS (akin to
CONFIG_EXT2_FS_SECURITY but using a different xattr namespace) to support
universal user ids being stored as a pair (unsid, id), in the xattr named
userns.unsid. When so compiled, then FS_USER_NS gets set in their FS_TYPE, and
the following access rules are used:

As mentioned above, if current->userns == mnt->userns, use normal
access rules. Otherwise, go below.

inode_getattr:
 1. if current->user->userns->unsid is not set, return '0'
 2. if current->user->userns->unsid is not listed in the
 xattr named userns.unsid, then return '0'.
 3. if current->user->userns->unsid is listed, then return the
 associated uid.
inode_permission:
 1. if current->user->userns->unsid is not set, use 'user other' perms
 2. if current->user->userns->unsid is not listed in the

xattr named userns.unsid, then use 'user other' perms
3. if current->user->userns->unsid is listed, then use the associated
uid as the owner to make access decisions.

When a file is created in a FS_USER_NS fs, the owner is automatically marked as the full user chain which created the file. So using 500:0 to mean user 500 in unsid 0, let us say that 500:0 created a userns with unsid 1, 400:1 created userns unsid 2, and 2:2 creates a file. That file will be tagged with (2:2,400:1,500:0). I suspect we'll do the same for gids.

-serge

*1: I do not in this email address the problem of siginfo lifetypes vs user namespace lifetimes, but I think the unsid will allow us to solve that)

*2: unsids unfortunately need to be variably sized
consider # machines to which you might migrate, and # unique
isolatable userns's will exist across all
Consider that you have 512-byte limit on ext2 xattrs
So make unsids 8-1024 bits

*3: An alternative is to make unsids hierarchical. Any admin can then set the unsid of a newly created userns to any unsid which is a child of the parent unsid. Unfortunately that requires textual unsids, and that unfortunately will take up a heckuvalota space in xattrs. It's a design option to keep in mind though, as it fully facilitates safe unprivileged unsharing of user namespaces.

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