Subject: user namespace - introduction
Posted by serue on Mon, 04 Jun 2007 19:39:57 GMT
View Forum Message <> Reply to Message

[I've been sitting on this for some months, and am just dumping it so people can talk if they like, maybe even build on the patchset by adding support for more filesystems or implementing the keyring. Or tell me how much the approach sucks.]

First, I point out once more that the base user namespace patchset Cedric originally sent out really is sufficient. We just need for users to have different quotas, limits, and in-kernel key storage. Signal delivery, file controls, etc can be set up using pidspaces and separate mount trees, using selinux policy or other lsms, and even using ecryptfs.

But if it will be insisted upon that uid checks be enhanced, here is a new patchset which just might satisfy everyone, and which is based on user namespace discussions from the last year, particularly comments by Eric Biederman and David Howells.

Below is how I think the user namespace controls would work. The patches that follow only touch on parts of steps 1-4.

1. let filesystem tag inodes and superblocks with one user namespace
2. let generic_permission - and through inode->i_op->permission, the fs, if it wants to be smarter - enforce user namespaces
3. by default, inode->i_userns comes from sb->s_userns, just as is done in these patches.
4. By default, if inode->i_userns != task->userns, the process gets treated as 'nobody'. This is a change from my current patches and what is done in -lxc, where all permission is denied. I think it is a far preferable behavior. It allows read-only bind mount sharing among user namespaces without a silly MS_USER_NS flag.
5. Capabilities relating to actions on subjects or objects associated with a user namespace are only effective for targets in the same user namespace as the actor.
   This *could* be changed to also work for targets in decendant user namespaces, but that could slow things down.
6. Create a new keychain for user namespaces. Two types of entries.
   The first type of entry, (user_ns 5, uid 501) means that whichever user has that key will be recognized in user namespace 5 as uid 501. Presumably, uid 501 in user_ns 5 would have started a vserver with a new user namespace, say user_ns 7. He would likely want to give uid 0 in user_ns 7 a (user_ns 5, uid 501) key.
   The second type of key, (user_ns 5, CAP_FOWNER) gives the user

holding the key the ability to have CAP_FOWNER in userns 5. By default, uid 0 in userns 7 cannot have CAP_FOWNER in userns 5. (Only) a task with (userns 5, CAP_SETPCAP) can give that key to any user in userns 7. The key by itself does not grant the capability, but allows a task with that uid which has CAP_FOWNER in its P set to assert it for userns 5.

7. Eventually filesystems could begin storing global uids in inode xattrs on disk, and use these in inode->i_op->permission() along with data in the user's userns key to do global uid permission checking. Really this should almost trivial to implement once the above has been implemented. It could be done right in ext234 etc, or in a small stackable fs.

-serge

Note: step 1 has been complained about bc some think it should be done at the vfsmount level. If you read through the whole set of steps I think you'll see why it is not more limited. The fs gets to decide the real owner of a file, and despite there being one real owner, any number of users can be made to be treated as the owner, so there is no limitation in this approach.
