
Subject: Re: [PATCH 1/9] Add a user_namespace as creator/owner of uts_namespace

Posted by [David Howells](#) on Wed, 23 Feb 2011 17:16:33 GMT

[View Forum Message](#) <> [Reply to Message](#)

Serge E. Hallyn <serge@hallyn.com> wrote:

```
> struct uts_namespace {
>     struct kref kref;
>     struct new_utsname name;
> + struct user_namespace *user_ns;
> };
```

If a uts_namespace belongs to a user_namespace, should CLONE_NEWUSER then imply CLONE_NEWUTS?

Or is uts_namespace::user_ns more an implication that the set of users in user_namespace are the only ones authorised to alter the uts data.

I presume that the uts_namespace of a process must be owned by one of the user_namespaces in the alternating inheritance chain of namespaces and their creators leading from current_user_ns() to init_user_ns.

With that in mind, looking at patch 3:

```
- if (!capable(CAP_SYS_ADMIN))
+ if (!ns_capable(current->nsproxy->uts_ns->user_ns, CAP_SYS_ADMIN))
```

what is it you're actually asking? I presume it's 'does this user have CAP_SYS_ADMIN capability over objects belonging to the uts_namespace's user_namespace?'

So, to look at the important bit of patch 2:

```
-int cap_capable(struct task_struct *tsk, const struct cred *cred, int cap,
- int audit)
+int cap_capable(struct task_struct *tsk, const struct cred *cred,
+ struct user_namespace *targ_ns, int cap, int audit)
{
- return cap_raised(cred->cap_effective, cap) ? 0 : -EPERM;
+ for (;;) {
+     /* The creator of the user namespace has all caps. */
+     if (targ_ns != &init_user_ns && targ_ns->creator == cred->user)
+         return 0;
+
+     /* Do we have the necessary capabilities? */
+     if (targ_ns == cred->user->user_ns)
+         return cap_raised(cred->cap_effective, cap) ? 0 : -EPERM;
```

```

+
+ /* Have we tried all of the parent namespaces? */
+ if (targ_ns == &init_user_ns)
+   return -EPERM;
+
+ /* If you have the capability in a parent user ns you have it
+  * in the over all children user namespaces as well, so see
+  * if this process has the capability in the parent user
+  * namespace.
+  */
+ targ_ns = targ_ns->creator->user_ns;
+ }
+
+ /* We never get here */
+ return -EPERM;
+ }

```

On entry, as we're called from `ns_capable()`, `cred->user` is the user that the current process is running as, and, as such, may be in a separate namespace from `uts_namespace` - which may itself be in a separate namespace from `init_user_ns`.

So, assume for the sake of argument that there are three `user_namespaces` along the chain from the calling process to the root, and that the `uts_namespace` belongs to the middle one.

```

if (targ_ns != &init_user_ns && targ_ns->creator == cred->user)
    return 0;

```

Can never match because `targ_ns->creator` cannot be `cred->user`; even if the `uts_namespace` belongs to our namespace, given that the creator lies outside our namespace.

```

if (targ_ns == cred->user->user_ns)
    return cap_raised(cred->cap_effective, cap) ? 0 : -EPERM;

```

Can only match if we are in the target `user_namespace` (ie. the one to which `uts_namespace` belongs), whether or not we have `CAP_SYS_ADMIN`.

Which means that unless the `uts_namespace` belongs to our `user_namespace`, we cannot change it. Is that correct?

So `ns_capable()` restricts you to only doing interesting things to objects that belong to a `user_namespace` if they are in your own `user_namespace`. Is that correct?

If that is so, is the loop required for `ns_capable()`?

Looking further at patch 2:

```
#define nsown_capable(cap) (ns_capable(current_user_ns(), (cap)))
```

Given what I've said above, I presume the loop isn't necessary here either.

I think you're using `ns_capable()` in two different ways:

- (1) You're using it to see if a process has power over its descendents in a `user_namespace` that can be traced back to a `clone()` that it did with `CLONE_NEWUSER`.

For example, automatically granting a process permission to kill descendents in a namespace it created.

- (2) You're using it to see if a process can access objects that might be outside its own `user_namespace`.

For example, setting the hostname.

Is it worth giving two different interfaces to make this clearer (even if they actually do the same thing)?

Sorry if this seems rambling, but I'm trying to get my head round your code.

David

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