Subject: Re: [PATCH 5/9] Allow ptrace from non-init user namespaces
Posted by serge on Thu, 24 Feb 2011 00:43:37 GMT
View Forum Message <> Reply to Message

Quoting Andrew Morton (akpm@linux-foundation.org):
> On Thu, 17 Feb 2011 15:03:33 +0000
> "Serge E. Hallyn" <serge@hallyn.com> wrote:
>
> > ptrace is allowed to tasks in the same user namespace according to
> > the usual rules (i.e. the same rules as for two tasks in the init
> > user namespace). ptrace is also allowed to a user namespace to
> > which the current task the has CAP_SYS_PTRACE capability.
> >
> >
> > ...
> >
> > --- a/include/linux/capability.h
> > +++ b/include/linux/capability.h
> > @@ -546,6 +546,8 @@ extern const kernel_cap_t __cap_init_eff_set;
> > */
> > #define has_capability(t, cap) (security_real_capable((t), &init_user_ns, (cap)) == 0)
> >
> > +#define has_ns_capability(t, ns, cap) (security_real_capable((t), (ns), (cap)) == 0)
>
> macroitis.

Thanks for the review, Andrew. Unfortunately this one is hard to turn
into a function beecause it uses security_real_capable(), which is
sometimes defined in security/security.c as a real function, and
other times as a static inline in include/linux/security.h. So
I'd have to #include security.h in capability.h, but security.h
already #includes capability.h.

All the other comments affect same_or_ancestor_user_ns(), which
following Eric's feedback is going away.

> > /**
> > * has_capability_noaudit - Determine if a task has a superior capability available (unaudited)
> > * @t: The task in question
> > diff --git a/include/linux/user_namespace.h b/include/linux/user_namespace.h
> > index faf4679..862fc59 100644
> > --- a/include/linux/user_namespace.h
> > +++ b/include/linux/user_namespace.h
> > @@ -39,6 +39,9 @@ static inline void put_user_ns(struct user_namespace *ns)
> > uid_t user_ns_map_uid(struct user_namespace *to, const struct cred *cred, uid_t uid);
> > gid_t user_ns_map_gid(struct user_namespace *to, const struct cred *cred, gid_t gid);
> >
> > +int same_or_ancestor_user_ns(struct task_struct *task,

> > + struct task_struct *victim);
>
> bool.
>
> > #else
> >
> > static inline struct user_namespace *get_user_ns(struct user_namespace *ns)
> >
> > ...
> >
> > --- a/kernel/user_namespace.c
> > +++ b/kernel/user_namespace.c
> > @@ -129,6 +129,22 @@ gid_t user_ns_map_gid(struct user_namespace *to, const struct
cred *cred, gid_t
> > return overflowgid;
> > }
> >
> > +int same_or_ancestor_user_ns(struct task_struct *task,
> > + struct task_struct *victim)
> > +{
> > + struct user_namespace *u1 = task_cred_xxx(task, user)->user_ns;
> > + struct user_namespace *u2 = task_cred_xxx(victim, user)->user_ns;
> > + for (;;) {
> > + if (u1 == u2)
> > + return 1;
> > + if (u1 == &init_user_ns)
> > + return 0;
> > + u1 = u1->creator->user_ns;
> > + }
> > + /* We never get here */
> > + return 0;
>
> Remove?
>
> > +}
> > +
> > static __init int user_namespaces_init(void)
> > {
> > user_ns_cachep = KMEM_CACHE(user_namespace, SLAB_PANIC);
> >
> > ...
> >
> > int cap_ptrace_access_check(struct task_struct *child, unsigned int mode)
> > {
> > int ret = 0;
> > + const struct cred *cred, *tcred;
> >
> > rcu_read_lock();

```
> > - if (!cap_issubset(__task_cred(child)->cap_permitted,
> > -     current_cred()->cap_permitted) &&
> > -     !capable(CAP_SYS_PTRACE))
> > -  ret = -EPERM;
> > + cred = current_cred();
> > + tcred = __task_cred(child);
> > + /*
> > +  * The ancestor user_ns check may be gratuitous, as I think
> > +  * we've already guaranteed that in kernel/ptrace.c.
> > +  */
>
> ?
>
> > + if (same_or_ancestor_user_ns(current, child) &&
> > +     cap_issubset(tcred->cap_permitted, cred->cap_permitted))
> > +  goto out;
> > + if (ns_capable(tcred->user->user_ns, CAP_SYS_PTRACE))
> > +  goto out;
> > + ret = -EPERM;
> > +out:
> >   rcu_read_unlock();
> >   return ret;
> >  }
> >
> > ...
> >
>
```