Subject: Re: [PATCH 2/9] security: Make capabilities relative to the user namespace.
Posted by akpm on Fri, 18 Feb 2011 23:59:18 GMT
View Forum Message <> Reply to Message

On Thu, 17 Feb 2011 15:03:06 +0000
"Serge E. Hallyn" <serge@hallyn.com> wrote:

> - Introduce ns_capable to test for a capability in a non-default
>   user namespace.
> - Teach cap_capable to handle capabilities in a non-default
>   user namespace.
>
> The motivation is to get to the unprivileged creation of new
> namespaces. It looks like this gets us 90% of the way there, with
> only potential uid confusion issues left.
>
> I still need to handle getting all caps after creation but otherwise I
> think I have a good starter patch that achieves all of your goals.
>
>
> ...
>
> --- a/include/linux/capability.h
> +++ b/include/linux/capability.h
> @@ -544,7 +544,7 @@ extern const kernel_cap_t __cap_init_eff_set;
>   *
>   * Note that this does not set PF_SUPERPRIV on the task.
>   */
> -#define has_capability(t, cap) (security_real_capable((t), (cap)) == 0)
> +#define has_capability(t, cap) (security_real_capable((t), &init_user_ns, (cap)) == 0)
>
> /**
>   * has_capability_noaudit - Determine if a task has a superior capability available (unaudited)
> @@ -558,9 +558,15 @@ extern const kernel_cap_t __cap_init_eff_set;
>   * Note that this does not set PF_SUPERPRIV on the task.
>   */
> #define has_capability_noaudit(t, cap) \
> - (security_real_capable_noaudit((t), (cap)) == 0)
> + (security_real_capable_noaudit((t), &init_user_ns, (cap)) == 0)
>
> +struct user_namespace;
> +extern struct user_namespace init_user_ns;

Two icky-should-be-written-in-C macros which reference init_user_ns,
followed by the declaration of init_user_ns and its type. Declarations
which duplicate those in other header files. It's ripe for some
upcleaning, methinks?

Also, please ensure that the forward struct declarations are all at
top-of-file (as in include/linux/security.h). Otherwise we can end up
accumulating multiple forward declarations of the same thing in the one
file.

> extern int capable(int cap);
> +extern int ns_capable(struct user_namespace *ns, int cap);
> +extern int task_ns_capable(struct task_struct *t, int cap);
> +
> +#define nsown_capable(cap) (ns_capable(current_user_ns(), (cap)))

macroitis!

> @@ -301,15 +302,42 @@ error:
>   */
> int capable(int cap)
> {
> + return ns_capable(&init_user_ns, cap);
> +}
> +EXPORT_SYMBOL(capable);
> +
> +/**
> + * ns_capable - Determine if the current task has a superior capability in effect
> + * @ns: The usernamespace we want the capability in
> + * @cap: The capability to be tested for
> + *
> + * Return true if the current task has the given superior capability currently
> + * available for use, false if not.

Actually it doesn't return true or false - it returns 1 or 0. Using a
`bool' return type would fix the comment :)

> + * This sets PF_SUPERPRIV on the task if the capability is available on the
> + * assumption that it's about to be used.
> + */
> +int ns_capable(struct user_namespace *ns, int cap)
> +{
>   if (unlikely(!cap_valid(cap))) {
>   printk(KERN_CRIT "capable() called with invalid cap=%u\n", cap);
>   BUG();
>   }
>
> - if (security_capable(current_cred(), cap) == 0) {
> + if (security_capable(ns, current_cred(), cap) == 0) {
>   current->flags |= PF_SUPERPRIV;
>   return 1;
>   }

>   return 0;
> }
> -EXPORT_SYMBOL(capable);
> +EXPORT_SYMBOL(ns_capable);
> +
> +/*
> + * does current have capability 'cap' to the user namespace of task
> + * 't'. Return true if it does, false otherwise.
> + */

Other comments were kerneldocified.

> +int task_ns_capable(struct task_struct *t, int cap)
> +{
> + return ns_capable(task_cred_xxx(t, user)->user_ns, cap);
> +}
> +EXPORT_SYMBOL(task_ns_capable);

Could return bool.

>
> ...
>
> +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;

So delete the code? Or does the compiler warn? If so, it's pretty busted.

> }
>
> /**
>
> ...
>

_____

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