

Re: [Keyrings] [PATCH] Keys: Add LSM hooks for key management

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2005-10/1628.html>

From: Chris Wright (chrisw_at_osdl.org)

Date: 10/07/05

Date: Thu, 6 Oct 2005 16:10:06 -0700
To: David Howells <dhowells@redhat.com>

* David Howells (dhowells@redhat.com) wrote:

> Chris Wright <chrisw@osdl.org> wrote:

>

>> So the hooks you added control search/read/write via permission

>

> Actually, that's search/read/write/view/link permissions.

Good point, thanks.

>> ->instantiate

>

> Either must be creating the key or must have a request-key-authorisation key
> for the key being instantiated and Write permission on that key.

>

>> ->duplicate

>

> Need to be able to create a new key. Don't need to be able to Read the
> original key because you're not actually trying to access it.

>

>> ->update

>

> Need Write permission on the key.

>

>> ->match

>

> Need Search permission on the key.

>

>> ->describe

>

> Need View permission on the key. I should probably make /proc/keys consult the
> LSM too.

BTW, /proc/keys should move, esp since it's a debugging interface.

Linux-Kernel: Re: [Keyrings] [PATCH] Keys: Add LSM hooks for key management

> > ->read
>
> Need Read permission on the key, or it must be Searchable from the accessing
> process's keyrings.
>
> > and then special case instantiate and update. Does this mean you expect the
> > security module to inspect the key data and set a label based on data?
>
> It seemed like a good idea to make it available, since I have it to hand.
> Whether the security module should use it to impose a lable, probably not, I
> suppose.

It means the security modules have to be able to parse the data. I think that'd be the rough analog to updating file label based on file contents, right? And we definitely don't want that.

> > Does duplicate need to update label to a new context, or should it get
> > identical lable, and does it need a hook for that?
>
> I think it should start off with identical labels, but the set_security keyctl
> op can then be called to change that.
>
> It might be better to implement what Kyle Moffett wants and make clonable key
> handles that contain the security information, but still refer to the same
> key.

So this security information is COW?

> > This move of key_ref_t handling is either unrelated or simple prep work,
> > yes? Just clutters the patch.
>
> The problem is that key_ref_t isn't available if CONFIG_KEYS is not defined,
> but it's still referenced in security.h. Would it be reasonable to make all
> the security_key_*(*) functions contingent on CONFIG_KEYS since they're only
> called from the key management code? That would mean I wouldn't need to do
> this.

I see. I thought they already were conditional on CONFIG_KEYS.

> > Again patch clutter, since this is just removing unused code AFAICT
> > (which should definitely be removed). I mention only because it makes it
> > harder to review since I don't know the key infrastructure as well as you.
>
> I can split the permissions function out into a .c file in a preliminary
> patch.

Proolly be nice. I think I've deciphered the patch, so it was just a nitpick.

> > Just curious, from quick look, appears that key_task_permission is
> > basically always called with current. I found one (which appears to be
> > a recursive call) that uses rka->context in search_process_keyrings.

Re: [Keyrings] [PATCH] Keys: Add LSM hooks for key management

Linux-Kernel: Re: [Keyrings] [PATCH] Keys: Add LSM hooks for key management

- > > *What case causes context != current?*
- >
- > *What happens is this:*
- >
- > *(1) Process A calls request_key().*
- >
- > *(2) request_key() sees that A doesn't have the desired key yet, so it creates:*
- >
- > *(a) An uninstantiated key U of appropriate type and description.*
- >
- > *(b) An authorisation key V that refers to U and notes that process A is the context in which V should be instantiated, and from which key requests may be satisfied.*

So this is where 'rka->context = current' is established. And since call_usermodehelper is called with wait flag set, you're sure current won't go away...OK scratch that worry of mine.

- > *(3) request_key() then forks and executes /sbin/request-key with a session keyring containing auth key V.*
- >
- > *(4) /sbin/request-key execs an appropriate program to instantiate key U.*
- >
- > *(5) The program may want to access another key from A's context (say a Kerberos TGT key). It just requests the appropriate key, and the keyring search notes that the session keyring has in its bottom level auth key V.*
- >
- > *This will permit it to then search the keyrings of process A with the UID, GID, groups and security info of process A as if it was process A, coming up with key W.*
- >
- > *(6) The program then does what it must to get the security data for key U, using key W as a reference (perhaps it contacts a Kerberos server using the TGT) and then instantiates key U.*
- >
- > *(7) Upon instantiating key U, auth key V is automatically revoked so that it may not be used again.*
- >
- > *(8) The program then exits 0 and request_key() deletes key V and returns key U to the caller.*
- >
- > *This also extends further. If key W (step 5 above) didn't exist, another auth key (X) will be created and another copy of /sbin/request-key spawned; but the context specified by key X will still be process A.*

Ah, I saw that code and didn't grok why that bit was needed, thanks.

- > *The problem I've tried to solve is that I can't just attach process A's keyrings to /sbin/request-key at the appropriate places because (a) execve will discard two of them, and (b) it requires the same UID/GID/Groups all the*

Linux-Kernel: Re: [Keyrings] [PATCH] Keys: Add LSM hooks for key management

> way through.
>
> At some point, I will have to make it so that I don't have to use
> /sbin/request-key, but can instead request an already running daemon assume
> the context from an auth key specified to it, say by passing the key serial
> number over a socket.

I can see the appeal, but actually current architecture makes it easier to do checks against the caller that initiated the request.

> > > +#include <linux/security.h>
> >
> > This looks unnecessary. I'd include it where it's used.
>
> I can do that.
>
> > > + /* do a final security check before publishing the key */
> > > + ret = security_key_alloc(key);
> >
> > This may simply be allocating space for the label (and possibly labelling)
> > not necessarily a security check.
>
> I don't know that. The LSM can always refuse permission to allocate a key if
> it wants to.

Typically this type of hook is used for reserving label space. Removing the comment is sufficient AFAIC.

> > > + security_key_post_alloc(key);
> >
> > This is odd, esp since nothing could have failed between alloc and
> > publish. Only state change is serial number. Would you expect the
> > security module to update a label based on serial number?
>
> I was thinking that it may want to be able to log it for auditing purposes.

Perhaps we could just consider that later, and focus on the access control for now.

> > > ret = key->type->instantiate(key, data, datalen);
> >
> > I assume you don't check key_permission here because it's in context
> > creating the key?
>
> The caller has to do that because it's done differently depending on how it's
> invoked. It may be called either by direct key creation (no permissions
> required, only quota and keyring Write), or by later instantiation (requires
> auth key and Write permission).

Yes, thanks, I missed the rka instantiation at first.

Linux-Kernel: Re: [Keyrings] [PATCH] Keys: Add LSM hooks for key management

```
>>> + /* if we're not the sysadmin, we can only change a key that we own */
>>> + if (capable(CAP_SYS_ADMIN) || key->uid == current->fsuid)
>>> + ret = security_key_set_security(key, name, data, dlen);
>>
>> Are you sure this is right? Normally I'd expect users can _not_ set the
>> security labels of their own keys. But perhaps I've missed the point
>> of this one, could you give a use case?
>
> I haven't said that they can; I've said that they must own the key to be able
> to request setting security data, or that they must be the superuser. I can
> drop that check if you'd prefer and just leave it to the LSM.
```

I simply don't see the point. I'd expect policy to mandate key labels, not users.

```
>>> + ret = security_key_get_security(key, name, buffer, blen);
>>
>> This would be a whole lot easier if keys were available in keyfs ;-)
>> /me ducks and runs
>
> It might be, but keyfs makes things so much bigger and more complicated.
>
>> Again, maybe I've lost you on the keyctl interface, but what's this for?
>> How will users benefit from read/write of the security label on a key?
>
> I was thinking of xattr equivalent. I can drop one or both functions if you'd
> prefer.
```

It's more that I don't understand the use.

```
>>> + ret = security_key_permission(key_ref, context, perm);
>>> + if (ret >= 0)
>>> + return ret;
>>
>> This is not right. Do normal permissions check first. If they pass,
>> then allow security module to check labels. And simply return 0 on
>> success and -ERR on error.
>
> Don't you want to be able to override that completely? I'd've thought you
> would have wanted to.
```

Heh, yes seems logical, but we actually want the basic DAC permission check to be done first, and only consult the security module if that passes. This keeps the interface restrictive as opposed to authoritative, which is required at present.

```
>>> +EXPORT_SYMBOL(key_task_permission);
>>
>> EXPORT_SYMBOL_GPL looks more appropriate here.
>
> Actually, it doesn't. The functions was publicly available in a header file
```

Linux-Kernel: Re: [Keyrings] [PATCH] Keys: Add LSM hooks for key management

> *before*.

You're right, somehow I thought it was newly introduced.

thanks,
-chris

-

To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
the body of a message to majordomo@vger.kernel.org

More majordomo info at <http://vger.kernel.org/majordomo-info.html>

Please read the FAQ at <http://www.tux.org/lkml/>