Subject: Re: [PATCH 1/1] capabilities: introduce per-process capability bounding set (v8)
Posted by serue on Tue, 20 Nov 2007 18:14:52 GMT
View Forum Message <> Reply to Message

Quoting Andrew Morgan (morgan@kernel.org):
> -----BEGIN PGP SIGNED MESSAGE-----
> Hash: SHA1
>
> Serge E. Hallyn wrote:
> > Andrew, this version follows all of your suggestions. Definately nicer
> > userspace interface. thanks
> [...]
> >
> > /* Allow ioperm/iopl access */
> > @@ -314,6 +314,10 @@ typedef struct kernel_cap_struct {
> >
> > #define CAP_SETFCAP   31
> >
> > +#define CAP_NUM_CAPS   32
> > +
> > +#define cap_valid(x) ((x) >= 0 && (x) < CAP_NUM_CAPS)
> > +
>
> Could you change the name of CAP_NUM_CAPS? There is some libcap building
> code that does the following to automatically build the "cap_*" names
> for libcap, and this new define above messes that up! :-(
>
> sed -ne '/^#define[\t]CAP[_A-Z]\+[\t]\+[0-9]\+/{s/^#define \([^
> \t]*\)[\t]*\([^ \t]*\)/ \{ \2, \"\1\"
> \},/;y/ABCDEFGHIJKLMNOPQRSTUVWXYZ/abcdefghijklmnopqrstuvwxyz/;p;}' <
> $(KERNEL_HEADERS)/linux/capability.h | fgrep -v 0x > cap_names.sed
>
> Something like:
>
>   #define CAP_NUM_CAPS (CAP_SETFCAP+1)
>
> will save me some hassle. :-)

Gotcha. Will change that.

I worry that what you have is just a *touch* too busy so whoever adds
capability #32 might forget to update CAP_NUM_CAPS, but it looks like

#define CAP_LAST_CAP CAP_SETFCAP

#define cap_valid(x) ((x) >= 0 && (x) <= CAP_LAST_CAP)

should also be ok for libcap.

> [...]
>
> > /*
> > * Bit location of each capability (used by user-space library and kernel)
> > */
> > @@ -350,6 +354,17 @@ typedef struct kernel_cap_struct {
> >
> > #define CAP_INIT_INH_SET CAP_EMPTY_SET
> >
>
> Its kind of a pity to put a kernel config ifdef in a header file. Could
> you put the ifdef code in the c-files that uses these definitions?

Hmm, now that you mention it, I notice that the exact same block of
code is still in commoncap.c. I must have lost the patch hunk dropping
that some time ago...

But at this point CAP_INIT_BSET is only used in
include/linux/init_task.h. And I'd really rather not put the definition
in there.

Note that the conditional is under a #ifdef __KERNEL__, so applications
shouldn't be looking at it anyway. Does that help?

> > +#ifdef CONFIG_SECURITY_FILE_CAPABILITIES
>
> In my experience when headers define things differently based on
> configuration #defines, other users of header files (apps, kernel
> modules etc.), never quite know what the current define is. If we can
> avoid conditional code like this in this header file, I'd be happier.
>
> > +#ifdef CONFIG_SECURITY_FILE_CAPABILITIES
>
> ditto.

For this I really can't, because that is the recommended way to handle
functions with different behavior per CONFIG_ variables. #ifdefs are to
be kept out of .c files to improve their readability, and helper
functions called in .c files are to have their definition in .h files
depend on the CONFIG_ variables.

> [...]
> > +extern long cap_prctl_drop(unsigned long cap);
> > +#else
> > +#include <linux/errno.h>
> > +static inline long cap_prctl_drop(unsigned long cap)

> > +{ return -EINVAL; }
> > +#endif
> > +
> > +long cap_prctl_drop(unsigned long cap)
> > +{
> > + if (!capable(CAP_SETPCAP))
> > + return -EPERM;
> > + if (!cap_valid(cap))
> > + return -EINVAL;
> > + cap_lower(current->cap_bset, cap);
>
> I think the following lines are overkill. Basically, the next exec()
> will perform the pP/pE clipping, and cap_bset should only interact with
> fP (and not fI).
>
> We already have a mechanism to manipulate pI, which in turn gates fI.
> And this same mechanism (libcap) can clip pE, pP if it is needed pre-exec().
>
> So, if you want to drop a capability irrevocably, you drop it in bset,
> and separately in pI. The current process may continue to have the
> capability, but post-exec the working process tree has lost it. For
> things like login, this is desirable.

Ok...

I think this makes sense. It seems pretty subtle and complicated, and
therefore I'm a little worried that it will be fragile against future
code changes. Someone will think it's a good idea to slightly change
the capset() semantics and only a year later will we realize that the
bounding set is no longer working...

So this will all have to be very well documented (and tested).

(Actually I notice that capabilities(7) manpage isn't in the libcap
sources. So an update to that is probably long overdue...)

> This also makes it possible for you to allow pI to have a capability
> otherwise banned in cap_bset which is useful with limited role accounts.

Yeah... so the way you'd see this happening, I assume, is that

1. login would keep some capset in pI for user hallyn,
2. so if /bin/foo had some nonempty fI, hallyn could run
   /bin/foo with cap_intersect(pI|fI)?

So now the bounding set would place a restriction on what /bin/login in
some container could leave in hallyn's pI.

> > + current->cap_effective = cap_intersect(current->cap_effective,
> > + current->cap_bset);
> > + current->cap_permitted = cap_intersect(current->cap_permitted,
> > + current->cap_bset);
> > + current->cap_inheritable = cap_intersect(current->cap_inheritable,
> > + current->cap_bset);
>
> You might want to replace the above three lines with a restriction
> elsewhere on what CAP_SETPCAP can newly set in
> commoncap.c:cap_capset_check().
>
> That is, CAP_SETPCAP permits the current process to raise 'any' pI
> capability. I suspect that you'll want to prevent raising any bits not
> masked by this:
>
>   pI' & ~(pI | (pP & cap_bset)).
>
> Cheers
>
> Andrew

Ok, I'll try this and see where I get.

thanks,
-serge

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