Subject: Re: [PATCH 8/9] signal: Drop signals before sending them to init.
Posted by Oleg Nesterov on Tue, 18 Dec 2007 15:30:07 GMT
View Forum Message <> Reply to Message

On 12/18, Eric W. Biederman wrote:
>
> Oleg Nesterov <oleg@tv-sign.ru> writes:
>
> > On 12/17, Eric W. Biederman wrote:
> >>
> >> So I would have no problem with a definition said signals
> >> will be dropped when sent to init if at the time they are
> >> sent the signal is SIG_DFL and unblocked.
> >
> > Great!
>
> My only issue is that with including the blocked signals in
> the definition is that it makes things a little harder to
> explain and understand.

Perhaps yes.

> >> > But this can happen with
> >> > your patch as well. sig_init_drop() returns false if we have a handler,
> >> > but this races with sys_rt_sigaction() which can set SIG_DFL, so init
> >> > could be killed.
> >>
> >> I am checking under the sighand lock so we should not race,
> >> at least not internally to the kernel.
> >
> > Yes, but as soon as we drop ->siglock /sbin/init can set SIG_DFL before
> > noticing the signal.
>
> I guess I don't see this as a race. The definition in my head is
> all about permission to send a signal to init. That permission happens
> if init shows interest in the incoming signal. So it is an
> instant in time decision.

Agreed.

> You seem to be implying that something is wrong. If something
> is wrong if something can be wrong either we have the definition wrong
> or the implementation wrong.

I think that the resulting behaviour is not right. /sbin/init should not
die after signal(SIG_DFL) if it has a pending !sig_kernel_ignore() signal.

Of course this doesn't depend on "should we look at the blocked state or not"

issue.

> >> My fundamental problem with that patch is that it drops signals
> >> after we have started processing them, and it modifies the code
> >> of an optimization.
> >>
> >> To have a clean definition and clean semantics I think we need
> >> to drop the signal earlier in the path. Which is what I
> >> really object to in your patch.
> >
> > Hmm. Could you look at this patch again? I'm attaching it at the end.
> > (re-diffed against the current code)
> >
> > It modifies sig_ignored(), so we drop the signal before we started
> > processing. And in fact it is more "optimized", because we don't need
> > to check sa_handler twice.
>
> Yes. It is more "optimized" but from what I can tell less correct.
> It makes it really easy to get the definition wrong. The big
> problem is you allow all signals through in the case of ptrace.
> Which is so totally wrong.

Well yes, but I don't think this is "totally wrong". The global init can't
be ptraced, so this doesn't matter. Ptracing of the sub-namespace init from
the parent namespace is special anyway, and currently is not fully supported.

> > --- t/kernel/signal.c~INITSIGS 2007-08-19 14:39:35.000000000 +0400
> > +++ t/kernel/signal.c 2007-08-19 19:00:27.000000000 +0400
> > @@ -39,11 +39,33 @@
> >
> > static struct kmem_cache *sigqueue_cachep;
> >
> > +static int sig_init_ignore(struct task_struct *tsk)
> > +{
> > + if (likely(!is_container_init(tsk->group_leader)))
> > + return 0;
> > +
> > + // ---------------- Multiple pid namespaces ----------------
> > + // if (current is from tsk's parent pid_ns && !in_interrupt())
> > + // return 0;
>
> We need to test siginfo to see if the signal is a signal from
> the kernel not in_interrupt(). So (before handling namespaces
> this would be)

We can siginfo to the list of arguments. But this is another issue.
in_interrupt() just means we should not check namespace.

> static int sig_init_ignrore(struct task_struct *tsk, siginfo_t *info,
> struct pid *sender)
> {
> /* Grumble we should look at the TGID and not need to
> * pass in group_leader.
> */
> if (likely(!is_container_init(tsk->group_leader)))
> return 0;
>
>
> /* Ignore signals from the kernel */
> if ((!is_si_special(info) && SI_FROM_KERNEL(info)) ||
> (info != SEND_SIG_NOINFO))
> return 1;
>
> /* If the kernel didn't send the signal figure out who did */
> if (!sender)
> sender = task_tgid(current);

I don't really understand why do we need this "sender". It is always
current, unless in_interrupt(). do_notify_parent() is special, yes,
but I don't see any problems here, we still can use current, no?

> > static int sig_ignored(struct task_struct *t, int sig)
> > {
> > - void __user * handler;
> > -
> > /*
> > * Tracers always want to know about signals..
> > */
>
> Since we are dropping the signal before it is sent, tracers
> should never see the signal.

please see above.

> > @@ -2303,8 +2316,7 @@ int do_sigaction(int sig, struct k_sigac
> > * (for example, SIGCHLD), shall cause the pending signal to
> > * be discarded, whether or not it is blocked"
> > */
> > - if (act->sa.sa_handler == SIG_IGN ||
> > - (act->sa.sa_handler == SIG_DFL && sig_kernel_ignore(sig))) {
> > + if (sig_task_ignore(current, sig)) {
> > struct task_struct *t = current;
> > sigemptyset(&mask);
> > sigaddset(&mask, sig);
>
> Any time you start ignoring the signal here you are not thinking

> in terms of never sending the signal or not. Once a signal is sent
> we must treat it like normal (to have a clean definition).

We don't agree here.

> In the namespace case we can not look at a pending signal and decide
> if we should drop it or not. So changing sigaction is impossible.

You mean that it is possible that this signal has come from the parent
namespace, and so we should die but not just discard the signal.

I think we can ignore this problem. If we had a handler before (when
the signal was sent), this is - imho - the correct behaviour. If not
then yes, /sbin/init can "accidentally" survive. But the parent namespace
can always use SIGKILL to really kill us.

But yes I agree, this changes one corner case to another. And let me
repeat, I don't claim that "I am right and you are not", and I can't
really prove that my approach is "technically" better. Just a personal
feeling about the "better" tradeoff. And I already said my taste is
perverted ;)

Oleg.