Subject: Re: [RFC PATCH 0/5] Resend - Use procfs to change a syscall behavior
Posted by Oren Laadan on Thu, 17 Jul 2008 22:26:43 GMT
View Forum Message <> Reply to Message

Wow ... the volume of messages in this thread is overwhelming.
I guess it had to happen when I was off a month long vacation !

Ok .. lemme see if I can catch up:

Serge E. Hallyn wrote:
> Quoting Pavel Machek (pavel@ucw.cz):
>>>>>>> An alternative to this solution consists in defining a new field in the
>>>>>>> task structure (let's call it next_syscall_data) that, if set, would change
>>>>>>> the behavior of next syscall to be called. The sys_fork_with_id() previously
>>>>>>> cited can be replaced by
>>>>>>> 1) set next_syscall_data to a target upid nr
>>>>>>> 2) call fork().
>>>>>> ...bloat task struct and
>>>>>>
>>>>>>> A new file is created in procfs: /proc/self/task/<my_tid>/next_syscall_data.
>>>>>>> This makes it possible to avoid races between several threads belonging to
>>>>>>> the same process.
>>>>>> ...introducing this kind of uglyness.
>>>>>>
>>>>>> Actually, there were proposals for sys_indirect(), which is slightly
>>>>>> less ugly, but IIRC we ended up with adding syscalls, too.
>>>>> Silly question...
>>>>>
>>>>> Oren, would you object to defining sys_fork_with_id(),
>>>>> sys_msgget_with_id(), and sys_semget_with_id()?

I don't object, in particular given the backward-compatibility issue
that was discussed later in this thread. However, see more below.

>>>>>
>>>>> Eric, Pavel (Emelyanov), Dave, do you have preferences?
>>>>>
>>>>> For the cases Nadia has implemented here I'd be tempted to side with
>>>>> Pavel Machek, but once we get to things like open() and socket(), (a)
>>>>> the # new syscalls starts to jump, and (b) the per-syscall api starts to
>>>>> seem a lot more cumbersome.
>>>> You should not need to modify open/socket. You can already select fd
>>>> by creatively using open/dup/close...
>>> That's what we do right now in cryo. And if we end up patching up every
>>> API with separate syscalls, then we wouldn't create open_with_id(). But
>>> so long as the next_id were to exist, exploiting it in open is nigh on
>>> trivial and much nicer.
>> Ok, so ignore previous email. You know how unix works.

>>
>> I believe you should just introduce syscalls you need. Yes,
>> introducing new syscalls is hard/expensive, but changing existing
>> syscalls is simply bad idea.
>
> Ok, thanks, Pavel. I'm really far more inclined to agree with you than
> it probably sounds like. I'll go ahead and implement a clone_with_id()
> syscall for starters later this week just as a comparison.
>
> Unless, Nadia, you have already started that?
>
>> So what new syscalls do you _really_ need? Not open_this_fd, nor
>> socket_this_fd.
>
> Oren, do you have a list of the syscalls which were modified to use the
> next_id in zap?

Good question :)

In zap, all of the checkpoint, and most of the restart is performed in kernel space. The user space component of the restart takes care of the creation of the process tree correctly with the desired SID for each process. Thus, the _only_ syscall that requires this hack from userland is clone(). I'm ok with adding a new syscall to do this job.

Everything else is created from within the kernel, usually by invoking the appropriate syscall inside. I use a similar trick (but in this case, only visible from within the kernel, not settable from user space, so there is never an issue with backward compatibility) for SysV IPC (select virtual ID) and PTY (select slave number when opening /dev/ptmx).

As mentioned by others, FD's are adjusted with dup2() if necessary.

Lastly, I can envision a need for a similar trick with certain devices if they are to be supported (e.g., if /dev/rtc is modified to work per namespace etc). But I wouldn't bother about that now.

Oren.