Subject: Re: [RFD] L2 Network namespace infrastructure
Posted by ebiederm on Sat, 23 Jun 2007 21:41:16 GMT
View Forum Message <> Reply to Message

David Miller <davem@davemloft.net> writes:

> From: ebiederm@xmission.com (Eric W. Biederman)
> Date: Sat, 23 Jun 2007 11:19:34 -0600
>
>> Further and fundamentally all a global achieves is removing the need
>> for the noise patches where you pass the pointer into the various
>> functions. For long term maintenance it doesn't help anything.
>
> I don't accept that we have to add another function argument
> to a bunch of core routines just to support this crap,
> especially since you give no way to turn it off and get
> that function argument slot back.
>
> To be honest I think this form of virtualization is a complete
> waste of time, even the openvz approach.
>
> We're protecting the kernel from itself, and that's an endless
> uphill battle that you will never win. Let's do this kind of
> stuff properly with a real minimal hypervisor, hopefully with
> appropriate hardware level support and good virtualized device
> interfaces, instead of this namespace stuff.
>
> At least the hypervisor approach you have some chance to fully
> harden in some verifyable and truly protected way, with
> namespaces it's just a pipe dream and everyone who works on
> these namespace approaches knows that very well.
>
> The only positive thing that came out of this work is the
> great auditing that the openvz folks have done and the bugs
> they have found, but it basically ends right there.

Dave thank you for your candor, it looks like I have finally made
the pieces small enough that we can discuss them.

If you want the argument to compile out. That is not a problem at all.
I dropped that part from my patch because it makes infrastructure more
complicated and there appeared to be no gain. However having a type
that you can pass that the compiler can optimize away is not a
problem. Basically you just make the argument:

```
typedef struct {} you_can_compile_me_out;  /* when you don't want it. */
typedef void * you_can_compile_me_out;     /* when you do want it. */
```

And gcc will generate no code to pass the argument when you compile it out.

As far as the hardening goes. There is definitely a point there, short of a kernel proof subsystem that sounds correct to me.

There are some other factors that make a different tradeoff interesting. First hypervisors do not allow global optimizations (because of the better isolation) so have an inherent performance disadvantage. Something like a 10x scaling penalty from the figures I have seen.

Even more interesting for me is the possibility of unmodified application migration. Where the limiting factor is that you cannot reliably restore an application because the global identifiers are not available.

So yes monolithic kernels may have grown so complex that they cannot be verified and thus you cannot actually keep untrusted users from doing bad things to each other with any degree of certainty.

However the interesting cases for me are cases where the users are not aggressively hostile with each other but being stuck with one set of global identifiers are a problem.

Eric

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