Subject: Re: [RFC PATCH 0/5] Resend -v2 - Use procfs to change a syscall behavior
Posted by Matt Helsley on Fri, 18 Jul 2008 01:09:25 GMT
View Forum Message <> Reply to Message

On Thu, 2008-07-17 at 18:42 -0400, Oren Laadan wrote:
>
> Eric W. Biederman wrote:
> > Alexey Dobriyan <adobriyan@gmail.com> writes:
>
> I seem to not have received any of Alexey's emails... ?
>
> >
> >> On Tue, Jul 08, 2008 at 01:24:22PM +0200, Nadia.Derbey@bull.net wrote:
> >>> # echo "LONG1 XX" > /proc/self/task/<my_tid>/next_syscall_data
> >> Same stuff.
> >>
> >> There is struct task_struct::did_exec , what about it?
> >>
> >> Also, patches are about de-serializing, how serializing from userspace looks
> >> like?
> >> You freezed group of processes, then what?
> >>
> >> How, for example, dump all VMAs correctly?
> >> [prepares counter-example]
> >
> > Alexey userspace vs a kernel space implementation is the wrong argument.
> >
> > It is clearly established that the current user space interfaces are
> > insufficient to do the job. So we need to implement something in the kernel.
> >
> > Further I have heard of no one suggesting running a single kernel on multiple
> > machines. Therefore there no one seems to be doing this entirely in the kernel
> > and so we need a user space component.
>
> I'm not sure I understand this argument ?
>
> In a kernel implementation, the component will merely open a file descriptor
> (to which the data will be streamed), freeze the container and invoke a
> system call. In a userland implementation, the component will do most of
> the work by continuously probing the kernel for information about the
> processes that are being checkpointed.
>
> So, of course we need a "component" - but what does that component do ?
>
> > So the question should not be user space vs. kernel space but can we build clean
> > interfaces for checkpoint/restart? What will those interfaces be?
>

> My question is why build a set of interfaces to export this and that from
> the kernel to user space ? if a kernel implementation (with minimal user
> space support) is chosen, then information extraction (and restoration) is
> straightforward and we don't get ourselves tied until the end of times to
> API exported to userland.

That still seems like an API exported to userland. It just combines the data into one block rather than distributing it amongst a bunch of pseudo-filesystems. Does this form of API really free us from always supporting it in the future?

> The output of the module will be a binary (like a core dump) that can be
> used by the same module to restart. User utilities will be available to
> inspect the contents (for whatever reason - like a debugger can inspect a
> core dump), and moreover to convert between old and new formats when moving
> from older to newer kernels.
>
> By doing so, we avoid many API issues - design, complexity, contents, and
> the amount of interfaces to be added.

Userspace is expected to inspect or convert the binary data. How does that truly avoid many of the API issues mentioned above? If it's really supposed to be a minimal API then the binary should be considered opaque and userspace tools which inspect or convert these binaries should be considered unreliable hacks at best. Otherwise it seems to me that it has most of the familiar problems associated with a kernel/userspace API -- including an obligation to support it.

Cheers,
-Matt Helsley

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