

---

Subject: Re: [PATCH 1/2] namespaces: introduce sys\_hijack (v10)

Posted by [serue](#) on Wed, 28 Nov 2007 15:23:59 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

Quoting Stephen Smalley (sds@tycho.nsa.gov):

> On Tue, 2007-11-27 at 16:38 -0600, Serge E. Hallyn wrote:

> > Quoting Stephen Smalley (sds@tycho.nsa.gov):

> > > On Tue, 2007-11-27 at 10:11 -0600, Serge E. Hallyn wrote:

> > > > Quoting Crispin Cowan (crispin@crispincowan.com):

> > > > > Just the name "sys\_hijack" makes me concerned.

> > > > >

> > > > > This post describes a bunch of "what", but doesn't tell us about "why"

> > > > > we would want this. What is it for?

> > > >

> > > > Please see my response to Casey's email.

> > > >

> > > > > And I second Casey's concern about careful management of the privilege

> > > > > required to "hijack" a process.

> > > >

> > > > Absolutely. We're definately still in RFC territory.

> > > >

> > > > Note that there are currently several proposed (but no upstream) ways to

> > > > accomplish entering a namespace:

> > > >

> > > > 1. bind\_ns() is a new pair of syscalls proposed by Cedric. An

> > > > nsproxy is given an integer id. The id can be used to enter

> > > > an nsproxy, basically a straight current->nsproxy = target\_nsproxy;

> > > >

> > > > 2. I had previously posted a patchset on top of the nsproxy

> > > > cgroup which allowed entering a nsproxy through the ns cgroup

> > > > interface.

> > > >

> > > > There are objections to both those patchsets because simply switching a

> > > > task's nsproxy using a syscall or file write in the middle of running a

> > > > binary is quite unsafe. Eric Biederman had suggested using ptrace or

> > > > something like it to accomplish the goal.

> > > >

> > > > Just using ptrace is however not safe either. You are inheriting \*all\*

> > > > of the target's context, so it shouldn't be difficult for a nefarious

> > > > container/vserver admin to trick the host admin into running something

> > > > which gives the container/vserver admin full access to the host.

> > > >

> > > > I don't follow the above - with ptrace, you are controlling a process

> > > > already within the container (hence in theory already limited to its

> > > > container), and it continues to execute within that container. What's

> > > > the issue there?

> >

> > Hmm, yeah, I may have overspoken - I'm not good at making up exploits

> > but while I see it possible to confuse the host admin by setting bogus  
> > environment, I guess there may not be an actual exploit.  
> >  
> > Still after the fork induced through ptrace, we'll have to execute a  
> > file out of the hijacked process' namespaces and path (unless we get  
> > \*really\* 'exotic'). With hijack, execution continues under the caller's  
> > control, which I do much prefer.  
> >  
> > The remaining advantages of hijack over ptrace (beside "using ptrace for  
> > that is crufty") are  
> >  
> > 1. not subject to pid wraparound (when doing hijack\_cgroup  
> > or hijack\_ns)  
> > 2. ability to enter a namespace which has no active processes  
>  
> So possibly I'm missing something, but the situation with hijack seems  
> more exploitable than ptrace to me - you've created a hybrid task with  
> one foot in current's world (open files, tty, connection to parent,  
> executable) and one foot in the target's world (namespaces, uid/gid)  
> which can then be leveraged by other tasks within the target's  
> world/container as a way of breaking out of the container. No?

I \*think\* the things coming out of the new container are well enough  
chosen to prevent that. I see where you're opening up to being killed  
by a task in the target container, though. But apart from setting a  
PF\_FLAG I'm not sure how to stop that anyway.

This actually reminds me that we need a valid uid in the target  
namespace in the HIJACK\_NS case. It's not a problem right now, but  
as I was just looking at fixing up kernel/signal.c in light of user  
namespaces, it is something to keep in mind.

> > These also highlight selinux issues. In the case of hijacking an  
> > empty cgroup, there is no security context (because there is no task) so  
> > the context of 'current' will be used. In the case of hijacking a  
> > populated cgroup, a task is chosen "at random" to be the hijack source.  
>  
> Seems like you might be better off with a single operation for creating  
> a new task within a given namespace set / cgroup rather than trying to  
> handle multiple situations with different semantics / inheritance  
> behavior. IOW, forget about hijacking a specific pid or picking a task  
> at random from a populated cgroup - just always initialize the state of  
> the newly created task in the same manner based solely on elements of  
> the caller's state and the cgroup's state.

So you're saying implement only the HIJACK\_NS?

I'm fine with that. Does anyone on the containers list object?

> > So there are two ways to look at deciding which context to use. Since  
> > control continues in the original acting process' context, we might  
> > want the child to continue in its context. However if the process  
> > creates any objects in the virtual server, we don't want them  
> > mislabeled, so we might want the task in the hijacked task's context.  
>  
> I suspect that we want to continue in the parent's context, and then the  
> program can always use `setfscreatecon()` or `exec` a helper in a different  
> context if it wants to create files with contexts tailored to the  
> target.

That sounds good to me...

So we're looking at:

1. drop `HIJACK_PID` and `HIJACK_CGROUP`
2. have `selinux_task_alloc_security()` always set `task->security` to `current->security` and allow the hijack case.

thanks,  
-serge

---

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

---