Subject: semantics for namespace naming
Posted by serue on Wed, 13 Dec 2006 14:35:55 GMT
View Forum Message <> Reply to Message

Let's say we have a vserver, from which we start some jobs
which we want to checkpoint/restart/migrate. These are two
of the usages we currently foresee for the namespaces, though
I'd say it's safe to assume there will be more.

I'll want to be able to address the c/r jobs by some ID in
order to checkpoint and kill them. I'll also want to be
able to address the entire vserver by some ID, in order to
kill it. In that case the c/r jobs should also be killed.
So those jobs are known by at least two id's. Furthermore, I
may want two vservers on the same machine, both running a c/r
job called 'calculate_pi'.

So we can look at this as a filesystem. In the above scenario,
we've got /sergesvserver, /sergesvserver/calculate_pi,
/randomvserver, and /randomvserver/calculate_pi. And, if
user hallyn logs into /sergesvserver using pam_namespace.so,
unsharing his mounts namespace to get a private /tmp and /home,
then he ends up in /sergesvserver/unnamed1. So each nsproxy
has a node in the namespace id filesystem, with random names
unless/until it is renamed to a more meaningful name. This
allows us to switch to a vserver by specifying the vserver's
name (ln /sys/namespaces/vserver1 /proc/nsproxy or whatever
semantics we end up using), kill an entire vserver recursively
(rm -rf /sys/namespaces/vserver1), perhaps even checkpoint
(tar jcf /tarballs/vserver1 /sys/namespaces/vserver1) and
certainly rename (mv /sys/namespaces/unnamed1 /sys/namespaces/sergeprivhome).

One key observeration which I haven't made explicit is that you
never actually leave a nsid ("container"). If you start under
/vserver1, you will always be under /vserver1. I don't know of
any reason that would not be appropriate. If I start a nested
vserver from there, then to me it may be known as
'vserver_testme', while to the admin of the machine, it would be
known as /vserver1/vserver_testme.

This makes one possible implementation of the container struct:

```
 struct container {
  struct container *parent;
  char *name;
  struct nsproxy *nsproxy;
  struct list_head children;
 };
```

```
 struct nsproxy {
  ...
  struct container *container;
 };
```

Plus of course relevant sysfs stuff.

-serge

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: semantics for namespace naming
Posted by Dave Hansen on Wed, 13 Dec 2006 18:36:33 GMT
View Forum Message <> Reply to Message

On Wed, 2006-12-13 at 08:35 -0600, Serge E. Hallyn wrote:
>
> struct container {
> struct container *parent;
> char *name;
> struct nsproxy *nsproxy;
> struct list_head children;
> };
> struct nsproxy {
> ...
> struct container *container;
> };

Why does a container need a pointer to an nsproxy? I think I missed that. Is that the "default" set of namespaces for tasks in that container?

-- Dave

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: semantics for namespace naming
Posted by Herbert Poetzl on Wed, 13 Dec 2006 21:47:40 GMT
View Forum Message <> Reply to Message

On Wed, Dec 13, 2006 at 10:36:33AM -0800, Dave Hansen wrote:
> On Wed, 2006-12-13 at 08:35 -0600, Serge E. Hallyn wrote:
> >
> >	struct container {
> >		struct container *parent;
> >		char *name;
> >		struct nsproxy *nsproxy;
> >		struct list_head children;
> >	};
> >	struct nsproxy {
> >		...
> >		struct container *container;
> >	};
>
> Why does a container need a pointer to an nsproxy? I think I missed
> that. Is that the "default" set of namespaces for tasks in that
> container?

IMHO it is mainly required to _enter_ the container
in some controlled way (at least that is what we
use the nsproxy reference for)

best,
Herbert

> -- Dave
>
> _____
> Containers mailing list
> Containers@lists.osdl.org
> https://lists.osdl.org/mailman/listinfo/containers

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: semantics for namespace naming
Posted by serue on Wed, 13 Dec 2006 22:51:55 GMT
View Forum Message <> Reply to Message

Quoting Herbert Poetzl (herbert@13thfloor.at):
> On Wed, Dec 13, 2006 at 10:36:33AM -0800, Dave Hansen wrote:
> > On Wed, 2006-12-13 at 08:35 -0600, Serge E. Hallyn wrote:
> > >
> > >	struct container {
> > >		struct container *parent;
> > >		char *name;

```
> > >             struct nsproxy *nsproxy;
> > >             struct list_head children;
> > >       };
> > >       struct nsproxy {
> > >             ...
> > >             struct container *container;
> > >       };
> >
> > Why does a container need a pointer to an nsproxy?  I think I missed
> > that.  Is that the "default" set of namespaces for tasks in that
> > container?
>
> IMHO it is mainly required to _enter_ the container
> in some controlled way (at least that is what we
> use the nsproxy reference for)
```

Exactly, it is there to facilitate entering containers.

-serge

Subject: Re: semantics for namespace naming
Posted by Sukadev Bhattiprolu on Thu, 14 Dec 2006 02:16:08 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn [serue@us.ibm.com] wrote:
| Let's say we have a vserver, from which we start some jobs
| which we want to checkpoint/restart/migrate. These are two
| of the usages we currently foresee for the namespaces, though
| I'd say it's safe to assume there will be more.
|
| I'll want to be able to address the c/r jobs by some ID in
| order to checkpoint and kill them. I'll also want to be
| able to address the entire vserver by some ID, in order to
| kill it. In that case the c/r jobs should also be killed.
| So those jobs are known by at least two id's.

For your calculate_pi example below, are the two ids "calculate_pi"
and "/sergevserver/calculate_pi" (ie. are the two plus ids basically
like relative and absolute pathnames or are they independent ?

And unrelated to the namespace naming - by "job" do you mean a single
process or can a job include multiple processes ? If it can include
multiple, can we checkpoint/restart/migrate just the job ? I am

thinking that we would need to migrate the entire vserver to
preserve process relationships - no ?

| Furthermore, I may want two vservers on the same machine, both
| running a c/r job called 'calculate_pi'.
|
| So we can look at this as a filesystem. In the above scenario,
| we've got /sergesvserver, /sergesvserver/calculate_pi,
| /randomvserver, and /randomvserver/calculate_pi. And, if
| user hallyn logs into /sergesvserver using pam_namespace.so,
| unsharing his mounts namespace to get a private /tmp and /home,
| then he ends up in /sergesvserver/unnamed1. So each nsproxy
| has a node in the namespace id filesystem, with random names
| unless/until it is renamed to a more meaningful name. This
| allows us to switch to a vserver by specifying the vserver's
| name (ln /sys/namespaces/vserver1 /proc/nsproxy or whatever
| semantics we end up using), kill an entire vserver recursively
| (rm -rf /sys/namespaces/vserver1), perhaps even checkpoint
| (tar jcf /tarballs/vserver1 /sys/namespaces/vserver1) and
| certainly rename (mv /sys/namespaces/unnamed1 /sys/namespaces/sergeprivhome).
|
| One key observeration which I haven't made explicit is that you
| never actually leave a nsid ("container"). If you start under
| /vserver1, you will always be under /vserver1. I don't know of
| any reason that would not be appropriate. If I start a nested
| vserver from there, then to me it may be known as
| 'vserver_testme', while to the admin of the machine, it would be
| known as /vserver1/vserver_testme.
|
| This makes one possible implementation of the container struct:
|
| struct container {
| struct container *parent;
| char *name;
| struct nsproxy *nsproxy;
| struct list_head children;
| };
| struct nsproxy {
| ...
| struct container *container;
| };
|
| Plus of course relevant sysfs stuff.
|
| -serge
| _____
| Containers mailing list
| Containers@lists.osdl.org

| https://lists.osdl.org/mailman/listinfo/containers

_____

Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: semantics for namespace naming
Posted by ebiederm on Thu, 14 Dec 2006 05:41:20 GMT
View Forum Message <> Reply to Message

Temporarily restricting myself to system containers because
they are well defined.

We have two things we need to name.
- Entire containers.
- Namespaces inside of a container.

So far Cedric's suggestion is a peculiar way of naming namespaces.
Which for the bind_ns is probably what we want, but it is not
what we want for container identification.

Entire container identification.

All process in unix are organized into a process tree.

Every system container has a unique init process that always
exists for the life of the container.

In the process tree the descendants of that init process are
the process in the specified container.

So if we remember the init process to container mapping we
can find the containers of any process merely by following
the parent process (ppid) up the process tree.

Which leads to the strong suggestion that for application containers
we somehow maintain the cohesiveness of the process tree.

To be clear, the unit of checkpoint/restart/migration is the
container. So far only our definitions of system containers have
managed the state properly for checkpoint/restart but ideally
application containers should be designed so we can do that as
well.

The basic operations on a whole container are pretty much:
suspend/restart, checkpoint/restart/migration, kill, accounting

and user display.

Namespace identification.

We also need a way to talk about individual namespaces.

We need this so we can clearly export to user space which process
share a namespace and which processes don't. Allowing us to talk
clearly about the group of process that share that namespace, as
well as give us the opportunity to debug reference counting problems.

For functionality like bind_ns per identifiers that clearly identify
a namespace are what we really want.

Debugging

Capturing a checkpoint of a set of processes and debugging a set of
processes is a very similar operation. Entering a namespace and
debugging processes in a namespace is a very similar operation.

I can currently manipulate processes in namespace, and by
manipulating those processes create new processes in a namespace
with sys_ptrace, the standard debugging facility.

I have yet to look at the possibilities in great detail but it looks
to me that what we want for containers is an enhancement of our
debugging mechanisms. So we can do the inspection and manipulation
we find desirable.

The classic enter implementation seems weak and error prone
when compared to what the current sys_ptrace can do and what we would
like to do in terms of checkpoint restart.

One of the issues is for good long term support is that we want
interfaces that are either absolutely trivial to implement or
interfaces that large numbers of people will use. The more people
using an interface the more free testers and fixers we get and
the higher the priority of keeping our code working.

"Serge E. Hallyn" <serue@us.ibm.com> writes:

> Let's say we have a vserver, from which we start some jobs
> which we want to checkpoint/restart/migrate. These are two
> of the usages we currently foresee for the namespaces, though
> I'd say it's safe to assume there will be more.

>
> I'll want to be able to address the c/r jobs by some ID in
> order to checkpoint and kill them. I'll also want to be
> able to address the entire vserver by some ID, in order to
> kill it. In that case the c/r jobs should also be killed.
> So those jobs are known by at least two id's. Furthermore, I
> may want two vservers on the same machine, both running a c/r
> job called 'calculate_pi'.
>
> So we can look at this as a filesystem. In the above scenario,
> we've got /sergesvserver, /sergesvserver/calculate_pi,
> /randomvserver, and /randomvserver/calculate_pi. And, if
> user hallyn logs into /sergesvserver using pam_namespace.so,
> unsharing his mounts namespace to get a private /tmp and /home,
> then he ends up in /sergesvserver/unnamed1. So each nsproxy
> has a node in the namespace id filesystem, with random names
> unless/until it is renamed to a more meaningful name. This
> allows us to switch to a vserver by specifying the vserver's
> name (ln /sys/namespaces/vserver1 /proc/nsproxy or whatever
> semantics we end up using), kill an entire vserver recursively
> (rm -rf /sys/namespaces/vserver1), perhaps even checkpoint
> (tar jcf /tarballs/vserver1 /sys/namespaces/vserver1) and
> certainly rename (mv /sys/namespaces/unnamed1
> /sys/namespaces/sergeprivhome).

I certainly see merit in using a file system interface for some
aspects of namespace manipulation. As much as possible we want
to keep to the old interfaces but that should not be a big deal.

> One key observeration which I haven't made explicit is that you
> never actually leave a nsid ("container"). If you start under
> /vserver1, you will always be under /vserver1. I don't know of
> any reason that would not be appropriate. If I start a nested
> vserver from there, then to me it may be known as
> 'vserver_testme', while to the admin of the machine, it would be
> known as /vserver1/vserver_testme.

Yes. Although on the crazy suggestion from I have heard
pivot_container suggested... Which may have some merit for the
software suspend story but otherwise doesn't seem useful...

> This makes one possible implementation of the container struct:
>
> struct container {
>   struct container *parent;
>   char *name;
>   struct nsproxy *nsproxy;
>   struct list_head children;

> };
> struct nsproxy {
> ...
> struct container *container;
> };

For your chosen struct container I guess if the hierarchy are
struct containers that will work. Going from your struct container
to anything interesting is currently a walk through the process list
which is painful. So I would suggest putting a pointer to the
init process of the container, that is probably better than the
nsproxy.

I'm not quite convinced we need the struct container. But I have
no fundamental objects to it either.

> Plus of course relevant sysfs stuff.

/proc is actually the appropriate filesystem for this sort of
information not sysfs. Handling the network information that
is in sysfs is going to be hard enough.

Eric

Subject: Re: semantics for namespace naming
Posted by Cedric Le Goater on Thu, 14 Dec 2006 13:58:29 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Let's say we have a vserver, from which we start some jobs
> which we want to checkpoint/restart/migrate. These are two
> of the usages we currently foresee for the namespaces, though
> I'd say it's safe to assume there will be more.
>
> I'll want to be able to address the c/r jobs by some ID in
> order to checkpoint and kill them. I'll also want to be
> able to address the entire vserver by some ID, in order to
> kill it. In that case the c/r jobs should also be killed.
> So those jobs are known by at least two id's. Furthermore, I
> may want two vservers on the same machine, both running a c/r
> job called 'calculate_pi'.
>
> So we can look at this as a filesystem. In the above scenario,

> we've got /sergesvserver, /sergesvserver/calculate_pi,
> /randomvserver, and /randomvserver/calculate_pi. And, if
> user hallyn logs into /sergesvserver using pam_namespace.so,
> unsharing his mounts namespace to get a private /tmp and /home,
> then he ends up in /sergesvserver/unnamed1. So each nsproxy
> has a node in the namespace id filesystem, with random names
> unless/until it is renamed to a more meaningful name. This
> allows us to switch to a vserver by specifying the vserver's
> name (ln /sys/namespaces/vserver1 /proc/nsproxy or whatever
> semantics we end up using), kill an entire vserver recursively
> (rm -rf /sys/namespaces/vserver1), perhaps even checkpoint
> (tar jcf /tarballs/vserver1 /sys/namespaces/vserver1) and
> certainly rename (mv /sys/namespaces/unnamed1 /sys/namespaces/sergeprivhome).
>
> One key observeration which I haven't made explicit is that you
> never actually leave a nsid ("container"). If you start under
> /vserver1, you will always be under /vserver1. I don't know of
> any reason that would not be appropriate. If I start a nested
> vserver from there, then to me it may be known as
> 'vserver_testme', while to the admin of the machine, it would be
> known as /vserver1/vserver_testme.
>
> This makes one possible implementation of the container struct:
>
> struct container {
>   struct container *parent;
>   char *name;
>   struct nsproxy *nsproxy;
>   struct list_head children;
> };
> struct nsproxy {
>   ...
>   struct container *container;
> };
>
> Plus of course relevant sysfs stuff.

I like the naming model. a few questions :

how do you enter only a subset of namespaces of a nsproxy/container
and not all of it ?

what flexibility the struct container is giving us ? why not have
container == nsproxy ?

the recursivity model looks like extra overhead. it could be flat.

C.

Subject: Re: semantics for namespace naming
Posted by serue on Thu, 14 Dec 2006 15:36:31 GMT
View Forum Message <> Reply to Message

Quoting Cedric Le Goater (clg@fr.ibm.com):
> Serge E. Hallyn wrote:
> > Let's say we have a vserver, from which we start some jobs
> > which we want to checkpoint/restart/migrate. These are two
> > of the usages we currently foresee for the namespaces, though
> > I'd say it's safe to assume there will be more.
> >
> > I'll want to be able to address the c/r jobs by some ID in
> > order to checkpoint and kill them. I'll also want to be
> > able to address the entire vserver by some ID, in order to
> > kill it. In that case the c/r jobs should also be killed.
> > So those jobs are known by at least two id's. Furthermore, I
> > may want two vservers on the same machine, both running a c/r
> > job called 'calculate_pi'.
> >
> > So we can look at this as a filesystem. In the above scenario,
> > we've got /sergesvserver, /sergesvserver/calculate_pi,
> > /randomvserver, and /randomvserver/calculate_pi. And, if
> > user hallyn logs into /sergesvserver using pam_namespace.so,
> > unsharing his mounts namespace to get a private /tmp and /home,
> > then he ends up in /sergesvserver/unnamed1. So each nsproxy
> > has a node in the namespace id filesystem, with random names
> > unless/until it is renamed to a more meaningful name. This
> > allows us to switch to a vserver by specifying the vserver's
> > name (ln /sys/namespaces/vserver1 /proc/nsproxy or whatever
> > semantics we end up using), kill an entire vserver recursively
> > (rm -rf /sys/namespaces/vserver1), perhaps even checkpoint
> > (tar jcf /tarballs/vserver1 /sys/namespaces/vserver1) and
> > certainly rename (mv /sys/namespaces/unnamed1 /sys/namespaces/sergeprivhome).
> >
> > One key observeration which I haven't made explicit is that you
> > never actually leave a nsid ("container"). If you start under
> > /vserver1, you will always be under /vserver1. I don't know of
> > any reason that would not be appropriate. If I start a nested
> > vserver from there, then to me it may be known as
> > 'vserver_testme', while to the admin of the machine, it would be
> > known as /vserver1/vserver_testme.

> >
> > This makes one possible implementation of the container struct:
> >
> > struct container {
> >   struct container *parent;
> >   char *name;
> >   struct nsproxy *nsproxy;
> >   struct list_head children;
> > };
> > struct nsproxy {
> >   ...
> >   struct container *container;
> > };
> >
> > Plus of course relevant sysfs stuff.
>
> I like the naming model. a few questions :
>
> how do you enter only a subset of namespaces of a nsproxy/container
> and not all of it ?

one container corresponds to one nsproxy which is one set of namespaces.
Are you asking how you would only switch your pid namespace but keep
your network namespace as the original?

I'm not sure that's something we want/need to support. (Can you cite a
use case?) But since I haven't specified how to ask for the nsproxy
switch anyway, it's too early to ask how we add a flag to specify a
namespace subset :)

I'm calling it a filesystem because we all understand the naming
semantics then, but that doesn't mean there'll be an actual
filesystem. It may all be hidden behind syscalls, in which case
we could do bind_ns(container_id, flags), where flags specifies which
namespaces to switch over.

But boy, then we need a new nsproxy to refcount those namespaces :)
Yuck.

> what flexibility the struct container is giving us ? why not have
> container == nsproxy ?

One reason is: once a process is in a container c1, if it unshares
and enters c2, it is still in c1. So if it was the last process out
of c1 to unshare, we need c1 to stick around, but the nsproxy will
be deleted. We could change the nsproxy semantics to match these
container semantics, but that makes the nsproxy implementation more
"magical".

And, of course, it's still under debate whether we'll keep the nsproxy.
This container model doesn't depend on the nsproxy, it's just exploiting
it. It can also work if all the namespaces are in the task_struct
explicitly.

> the recursivity model looks like extra overhead. it could be flat.

Absolutely not - it allows us to know with basically no accounting
overhead which namespaces a process is a part of. It naturally fits
our needs here. A process which is a part of /vserver1/calculate_pi
is also a part of /vserver1, and so if we kill /vserver1, we should
kill /vserver1/calculate_pi as well. Not doing this simple hierarchical
model means we have to explicitly keep track of all the containers
which a process is a part of.

As I said, once a process is in a container, it never leaves that
container. It only enters additional ones. That model fits everyone's
needs, without needing some funky API. If you consider the kinds of
things users would need to specify otherwise - "Unshare, creating a new
container, but leaving me in the old container", "unshare, creating a
new container, and leaving the old container" (for which there is no
need), and "unshare, keeping me inthe old container", the almost
entirely implicit approach I just outlined is far far preferable IMO.

-serge

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: semantics for namespace naming
Posted by serue on Thu, 14 Dec 2006 15:47:22 GMT
View Forum Message <> Reply to Message

Quoting Sukadev Bhattiprolu (sukadev@us.ibm.com):
> Serge E. Hallyn [serue@us.ibm.com] wrote:
> | Let's say we have a vserver, from which we start some jobs
> | which we want to checkpoint/restart/migrate. These are two
> | of the usages we currently foresee for the namespaces, though
> | I'd say it's safe to assume there will be more.
> |
> | I'll want to be able to address the c/r jobs by some ID in
> | order to checkpoint and kill them. I'll also want to be
> | able to address the entire vserver by some ID, in order to
> | kill it. In that case the c/r jobs should also be killed.
> | So those jobs are known by at least two id's.

>
> For your calculate_pi example below, are the two ids "calculate_pi"
> and "/sergevserver/calculate_pi" (ie. are the two plus ids basically
> like relative and absolute pathnames or are they independent ?

Right, so if you're inside /servervserver, you will just know the
namespace as 'calculate_pi', but if you're in the initial namespace,
'/', then you know it as /servervserver/calculate_pi.

> And unrelated to the namespace naming - by "job" do you mean a single
> process or can a job include multiple processes ? If it can include

Multiple processes.

> multiple, can we checkpoint/restart/migrate just the job ? I am
> thinking that we would need to migrate the entire vserver to
> preserve process relationships - no ?

So presumably when you did your

 /bin/checkpointable_exec -nsid calculate_pi /bin/calculate_pi

then /bin/checkpointable_exec would have unshared the pid namespace.
So /servervserver and /servervserver/calculate_pi would have
different pidspaces. If they don't, then presumably you knew what
you were doing.

Whether you do have to checkpoint/restart an entire pidspace I'm
not sure, though I guess it mainly depends on what the applications
do. So long as they only talk to each other, I don't see why
the restore couldn't just recognize that pid 245, which was the
one to start the job, but which itself is not in the container
and therefore not checkpointed. So it can tweak the children of
pid 245 so their ->parent pointers point to the container init task.
But maybe not. I haven't really thought through that all.

thanks,
-serge

Subject: Re: semantics for namespace naming
Posted by serue on Thu, 14 Dec 2006 16:28:33 GMT
View Forum Message <> Reply to Message

Quoting Eric W. Biederman (ebiederm@xmission.com):

(still digesting the earlier part of your email, will respond
to that later. I'm not sure whether you're laying out the
purely pid-addressed approach as an alternative, or just
extensively arguing that the container should point to an
init_pid and not an nsproxy, or whether you are saying
something differently entirely which I still need to process.)

> > struct container {
> >   struct container *parent;
> >   char *name;
> >   struct nsproxy *nsproxy;
> >   struct list_head children;
> > };
> > struct nsproxy {
> >   ...
> >   struct container *container;
> > };
>
> For your chosen struct container I guess if the hierarchy are
> struct containers that will work. Going from your struct container
> to anything interesting is currently a walk through the process list
> which is painful. So I would suggest putting a pointer to the
> init process of the container, that is probably better than the
> nsproxy.

Main downside of that is that we then again expect the init
process to stick around.

And since we can have a new container without having a new pidspace,
it's not even limited to one "reserved" process per pidspace.
Imagine a system with 300 users logging in, each with a
polyinstantiated /tmp directory. So now you have 300 implicit
containers due to their sys_unshare(CLONE_NS), and each of
these containers points to and reserves the PAM process which
did the unshare?

> I'm not quite convinced we need the struct container. But I have
> no fundamental objects to it either.

I'm not convinced either. I stuck it in there mainly for
description of the idea.

Though as I mentioned in my response to Suka, there is the
issue of keeping container 'vserver1' around even if both
the original nsproxy and the init process for that vserver
are gone. Because so long as one of it's decendents still

exists, we should still be able to say "kill vserver1", and
kill all it's decendents.

(And here we may want to talk about unsharing the container
namespace so that /vserver1/vserver3 can become independent
of /vserver1, but I don't like the security implications of
that)

> > Plus of course relevant sysfs stuff.
>
> /proc is actually the appropriate filesystem for this sort of
> information not sysfs. Handling the network information that
> is in sysfs is going to be hard enough.

Ok, good point. In addition to actually being process info, that'll
also make it trivial (compared to sysfs) to present different
information depending on a process' container.

thanks,
-serge

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: semantics for namespace naming
Posted by Dave Hansen on Thu, 14 Dec 2006 18:48:59 GMT
View Forum Message <> Reply to Message

On Thu, 2006-12-14 at 09:36 -0600, Serge E. Hallyn wrote:
> one container corresponds to one nsproxy which is one set of namespaces.

On container has at least one nsproxy associated with it. Did you mean
to say here that each container has one and only one nsproxy?

> As I said, once a process is in a container, it never leaves that
> container. It only enters additional ones. That model fits everyone's
> needs, without needing some funky API.

This makes logical sense to me. In practice this has the feel of
ptracing where the ptracer becomes a temporary parent of the tracee.

The process entering a container temporarily becomes a member of that
container, but it doesn't completely _stop_ being a member of its
container. The real_parent of a process being ptraced may not be doing
all of the parental duties during a ptrace, but it doesn't _stop_ being
the real_parent.

Maybe I'm stretching the analogy too far :)

-- Dave

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: semantics for namespace naming
Posted by serue on Thu, 14 Dec 2006 18:59:47 GMT
View Forum Message <> Reply to Message

Quoting Dave Hansen (haveblue@us.ibm.com):
> On Thu, 2006-12-14 at 09:36 -0600, Serge E. Hallyn wrote:
> > one container corresponds to one nsproxy which is one set of namespaces.
>
> On container has at least one nsproxy associated with it. Did you mean
> to say here that each container has one and only one nsproxy?

Sorry, that was struct container, not container.

Since the containers are hierarchical, you can say that a "container"
"has" all the nsproxies of all it's child containers.

So when there is a container for /vserver1, and from that vserver:

1. serge logs in and unshares the mount namespace for his
private /tmp and /home
2. dave starts a checkpointable job in a private container

The struct container for /vserver1 has just one nsproxy, but it has a
child container for serge's login, and a child container for dave's
checkpointable job, so you can say the vserver1 container has three
nsproxies.

> > As I said, once a process is in a container, it never leaves that
> > container. It only enters additional ones. That model fits everyone's
> > needs, without needing some funky API.
>
> This makes logical sense to me. In practice this has the feel of
> ptracing where the ptracer becomes a temporary parent of the tracee.
>
> The process entering a container temporarily becomes a member of that
> container, but it doesn't completely _stop_ being a member of its

> container. The real_parent of a process being ptraced may not be doing
> all of the parental duties during a ptrace, but it doesn't _stop_ being
> the real_parent.
>
> Maybe I'm stretching the analogy too far :)

Maybe, or maybe you're showing me a kink in my reasoning. I was in
fact thinking that entering a new container would be the one way
to fully disengage from the old container. Meaning it would be best
if it were forced to be done on a clone+exec. But even so, is that
reasonable?

-serge

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: semantics for namespace naming
Posted by ebiederm on Thu, 14 Dec 2006 21:24:44 GMT
View Forum Message <> Reply to Message

"Serge E. Hallyn" <serue@us.ibm.com> writes:

> Quoting Eric W. Biederman (ebiederm@xmission.com):
>
> (still digesting the earlier part of your email, will respond
> to that later. I'm not sure whether you're laying out the
> purely pid-addressed approach as an alternative, or just
> extensively arguing that the container should point to an
> init_pid and not an nsproxy, or whether you are saying
> something differently entirely which I still need to process.)

Mostly I was thinking aloud, and trying to define the problem
we are trying to solve.

> Main downside of that is that we then again expect the init
> process to stick around.
>
> And since we can have a new container without having a new pidspace,
> it's not even limited to one "reserved" process per pidspace.
> Imagine a system with 300 users logging in, each with a
> polyinstantiated /tmp directory. So now you have 300 implicit
> containers due to their sys_unshare(CLONE_NS), and each of
> these containers points to and reserves the PAM process which
> did the unshare?

I guess if we want to think of those uses of namespaces as containers
there is a clear problem.

>> I'm not quite convinced we need the struct container. But I have
>> no fundamental objects to it either.
>
> I'm not convinced either. I stuck it in there mainly for
> description of the idea.
>
> Though as I mentioned in my response to Suka, there is the
> issue of keeping container 'vserver1' around even if both
> the original nsproxy and the init process for that vserver
> are gone. Because so long as one of it's decendents still
> exists, we should still be able to say "kill vserver1", and
> kill all it's decendents.
>
> (And here we may want to talk about unsharing the container
> namespace so that /vserver1/vserver3 can become independent
> of /vserver1, but I don't like the security implications of
> that)

Digesting things a little more.
If we are assigning names, there does seem to be value in
hierarchical names. My biggest objection is that if the
operation we really want to perform is kill vserver1 that
does not map very naturally to the current kill command.

As for making /vserver1/vserver3 independent of /vserver1 you need to
be outside of /vserver1 to do it and it probably just a move on your
magic filesystem. But having the ability for a process to leave
a container even if it is pulled out has all kinds of interesting
consequences, especially if you inadvertently remove a security
restriction by doing so.

>> > Plus of course relevant sysfs stuff.
>>
>> /proc is actually the appropriate filesystem for this sort of
>> information not sysfs. Handling the network information that
>> is in sysfs is going to be hard enough.
>
> Ok, good point. In addition to actually being process info, that'll
> also make it trivial (compared to sysfs) to present different
> information depending on a process' container.

Exactly.

Eric

Subject: Re: semantics for namespace naming
Posted by ebiederm on Thu, 14 Dec 2006 21:56:34 GMT
View Forum Message <> Reply to Message

"Serge E. Hallyn" <serue@us.ibm.com> writes:

> Quoting Dave Hansen (haveblue@us.ibm.com):
>> On Thu, 2006-12-14 at 09:36 -0600, Serge E. Hallyn wrote:
>> > As I said, once a process is in a container, it never leaves that
>> > container. It only enters additional ones. That model fits everyone's
>> > needs, without needing some funky API.
>>
>> This makes logical sense to me. In practice this has the feel of
>> ptracing where the ptracer becomes a temporary parent of the tracee.
>>
>> The process entering a container temporarily becomes a member of that
>> container, but it doesn't completely _stop_ being a member of its
>> container. The real_parent of a process being ptraced may not be doing
>> all of the parental duties during a ptrace, but it doesn't _stop_ being
>> the real_parent.
>>
>> Maybe I'm stretching the analogy too far :)
>
> Maybe, or maybe you're showing me a kink in my reasoning. I was in
> fact thinking that entering a new container would be the one way
> to fully disengage from the old container. Meaning it would be best
> if it were forced to be done on a clone+exec. But even so, is that
> reasonable?

What I am doing today is using ptrace to force a process in
the container to fork. Then I can remote control that forked
process using ptrace to do whatever I need to do.

Because that model fundamentally keeps every process in it's own
container and never allows it to leave, nor does it allow things
from one container to cross into another container in an uncontrolled
fashion this feels to me like a very safe model.

We probably want to enhance ptrace so that it is easier to get
a remote process to execute a system call for us so without having
to jump through hoops, to find a syscall instruction etc, but I
think it is a model that makes a lot of sense.

The only nasty case I have is how do you handle a login daemon
outside of your container wanting to spawn new processes inside of your
container.

A- login daemon B - container init
+ C - login child ---> + D - process spawned by login child (child of container init)

If your login daemon (A) spawns a child process (C) that you
want to place in a container you create process (D) a process
of peculiar heritage, living entirely in the container. In normal
situations the remote login does not terminate until D performs
the desired work.

Since C has done all it needs to do ideally it would then exit.
The problem is that if C exits the login daemon will normally
think that the work is done and terminate the login session.
Despite D holding open it's connection to the login session.

So in practice I have to keep C around doing the ptrace parent
think until D exits, so I can forward the exit code back to A. Not
bad but a little annoying.

Eric

Subject: Re: semantics for namespace naming
Posted by Dave Hansen on Fri, 15 Dec 2006 17:08:55 GMT
View Forum Message <> Reply to Message

On Thu, 2006-12-14 at 14:56 -0700, Eric W. Biederman wrote:
> Because that model fundamentally keeps every process in it's own
> container and never allows it to leave, nor does it allow things
> from one container to cross into another container in an uncontrolled
> fashion this feels to me like a very safe model.

This is like saying that brain surgery is safe and controlled because
the surgeon never actually goes into the patient's brain! :)

I think of ptrace as a pretty wide-open interface. While ptrace itself
has well-defined semantics, I could hardly consider using it in
production, nor would I want to be the one to write the userspace apps
to do the syscall futzing for each of Linux's architectures.

-- Dave

Subject: Re: semantics for namespace naming
Posted by ebiederm on Fri, 15 Dec 2006 20:47:56 GMT
View Forum Message <> Reply to Message

Dave Hansen <haveblue@us.ibm.com> writes:

> On Thu, 2006-12-14 at 14:56 -0700, Eric W. Biederman wrote:
>> Because that model fundamentally keeps every process in it's own
>> container and never allows it to leave, nor does it allow things
>> from one container to cross into another container in an uncontrolled
>> fashion this feels to me like a very safe model.
>
> This is like saying that brain surgery is safe and controlled because
> the surgeon never actually goes into the patient's brain! :)

Can you think how dangerous brain surgery would be if the surgeon
actually physically went into the patients brain.

> I think of ptrace as a pretty wide-open interface. While ptrace itself
> has well-defined semantics, I could hardly consider using it in
> production, nor would I want to be the one to write the userspace apps
> to do the syscall futzing for each of Linux's architectures.

Well that isn't exactly what I am proposing. What I am proposing is
that we compare any interface to what you could do with ptrace. If
it allows for something ptrace doesn't allow you likely have a
problem.

So I think the concept of mapping the semantics of a new interface
to the semantics of ptrace is a very interesting review exercise.

Plus thinking about ptrace changes the question from what new
interface do we add to get the semantics we want, to how do we
optimize what we can do with ptrace, so it doesn't suck to use.

Eric

Subject: Re: semantics for namespace naming
Posted by Herbert Poetzl on Sun, 17 Dec 2006 19:18:01 GMT
View Forum Message <> Reply to Message

On Thu, Dec 14, 2006 at 09:36:31AM -0600, Serge E. Hallyn wrote:

[lot of stuff zapped here]

> Quoting Cedric Le Goater (clg@fr.ibm.com):
> > how do you enter only a subset of namespaces of a nsproxy/container
> > and not all of it ?
>
> one container corresponds to one nsproxy which is one set of
> namespaces. Are you asking how you would only switch your pid
> namespace but keep your network namespace as the original?
>
> I'm not sure that's something we want/need to support.

I can, we use this for several purposes, one is to
extend or modify the VFS namespace by mounting or
unmounting filesystems _into_ the guest

> (Can you cite a use case?)

> But since I haven't specified how to ask for the nsproxy switch
> anyway, it's too early to ask how we add a flag to specify a
> namespace subset :)

we (Linux-VServer) extended the set/enter_namespace
command to take a flag mask similar to unshare
(only 64bit wide :) to allow for 'selecting' the
proper spaces

note: the nsproxy a context is referring to can also
be changed this way, and a priviledged process can
enter each guest space separately ...

[more stuff zapped here]

best,
Herbert

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: semantics for namespace naming
Posted by Cedric Le Goater on Wed, 20 Dec 2006 09:36:56 GMT
View Forum Message <> Reply to Message

Herbert Poetzl wrote:
> On Thu, Dec 14, 2006 at 09:36:31AM -0600, Serge E. Hallyn wrote:
>
> [lot of stuff zapped here]
>
>> Quoting Cedric Le Goater (clg@fr.ibm.com):
>>> how do you enter only a subset of namespaces of a nsproxy/container
>>> and not all of it ?
>> one container corresponds to one nsproxy which is one set of
>> namespaces. Are you asking how you would only switch your pid
>> namespace but keep your network namespace as the original?
>>
>> I'm not sure that's something we want/need to support.
>
> I can, we use this for several purposes, one is to
> extend or modify the VFS namespace by mounting or
> unmounting filesystems _into_ the guest
>
>> (Can you cite a use case?)
>
>> But since I haven't specified how to ask for the nsproxy switch
>> anyway, it's too early to ask how we add a flag to specify a
>> namespace subset :)
>
> we (Linux-VServer) extended the set/enter_namespace
> command to take a flag mask similar to unshare
> (only 64bit wide :) to allow for 'selecting' the
> proper spaces
>
> note: the nsproxy a context is referring to can also
> be changed this way, and a priviledged process can
> enter each guest space separately ...

I'll work on a clone64 and unshare64 in January. But did you take
a look at the bind_ns syscall to see how it fits with vserver ?

C.

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers