Subject: [RFC] ns containers (v2): namespace entering
Posted by serue on Mon, 19 Feb 2007 22:14:46 GMT
View Forum Message <> Reply to Message

(Updated patchset addressing Paul's comments. Still looking more
for discussion on functionality, but barring much of that I guess
I'll do something with tsk->fs then send the set to lkml)

The following patchset uses the ns container subsystem to implement
namespace entering. It applies over the container patchset Paul
sent out earlier today.

= Usage =

Following is example usage:

```
        mkdir /container_ns
        mount -t container -ons nsproxy /container_ns
        (start screen with two windows)
        (screen one)
  (unshare uts namespace)
                hostname serge
        (screen two)
                hostname
                        (shows old hostname)
                echo $$ > /container_ns/node1/tasks (assuming node1 is screen one's new container)
                hostname
                        (shows 'serge')
```

Of course to get a useful result with the mounts namespace, we would
have to convert the fs_struct to a namespace, and the mounts and
fs_struct namespaces would be entered together.

If we follow this path, the next step would be to add files under
the container directory for each namespace, with ops->link defined
such that you could

```
 mkdir /container_ns/new_container
 ln /container_ns/vserver1/network /container_ns/new_container/network
 echo $$ > /container_ns/new_container/tasks
```

and be entered into a set of namespaces containing all of your defaults
except network, which would come from vserver1.

This is RFC not just on implementation, but also on whether to do
it at all. If so, then for all namespace, or only some? And if not,
how to facilitate virtual server management.

= Security =

Currently to enter a namespace, you must have CAP_SYS_ADMIN, and must
be entering a container which is an immediate child of your current
container. So from the root container you could enter container /vserver1,
but from container /vserver1 you could not enter /vserver2 or the root
container.

This may turn out to be sufficient. If not, then LSM hooks should be
added for namespace management. Four hooks for nsproxy management (create,
compose, may_enter, and enter), as well as some security_ns_clone hook for
each separate namespace, so that the nsproxy enter and compose hooks have
the information they need to properly authorize.

= Quick question =

Is it deemed ok to allow entering an existing namespace?

If so, the next section can be disregarded. Assuming not, the following
will need to be worked out.

= Management alternatives =

Mounts

Ram has suggested that for mounts, instead of implementing namespace
entering, the example from the OLS Shared Subtree paper could be
used, as follows (quoting from Ram):

> The overall idea is each container creates its own fs-namespace which
> has a mirror mount tree under /container/c1 in the original
> fs-namespace. So any changes in the container's fs-namespace reflect in
> the mount tree under /container/c1 in the original fs-namespace, and
> vice-versa. And all processes in the original fs-namespace can freely
> roam around in the mirror trees of all the containers, mounted
> under /container/cx, giving illusion that they have control over the
> mounts in all the containers. Whereas the processes in a container can
> just roam around its own fs-namespace and has no visibility to anything
> outside its own fs-namespace..
>
> As an example the way to accomplish this is:
>
>   in the original namespace,
>   1-1) mkdir /container
>   1-2) mount --bind /container /container
>   1-3) mount --make-unbindable /container
>
>

> when a new container c1 is created
>
> 2-1) mkdir /container/c1
> 2-2) mount --rbind / /container/c1
> 2-3) mount --make-rshared /container/c1
> 2-4) clone fs-namespace
> in the new namespace,
> 2-4-1) pivot_root /container/c1 /tmp
> 2-4-2) umount -l /tmp
>
>
> the mount at /container is made unbindable in steps 1-1 to 1-3, to
> prune-out the mounts of other containers from being visible in this
> container. The mount tree at /container/c1 is made shared in step
> 2-3, to ensure that the cloned mount tree in the new namespace shall
> be a exact mirror. The pivot_root in step 2-4-1 followed by umount in
> step 2-4-2 ensures that the container sees only what is needed and
> nothing else.

Now, currently a root process in container c1 could make his fs unshared,
but if that's deemed a problem presumably we can require an extra
priv for that operation which can be dropped for any vservers.

Herbert, and anyone else who wants mounts namespace entering, is the
above an acceptable alternative?

net+pid+uts

Not sure about uts, but I'm pretty sure the vserver folks want the ability
to enter another existing network namespace, and both vserver and openvz
have asked for the ability to enter pid namespaces.

The pid namespaces could be solved by always generating as many pids for
a process as it has parent pid_namespaces. So if I'm in /vserver1, with
one pid_namespace above me, not only my init process has an entry in the
root pid_namespace (as I think has been suggested), but all my children
will also continue to have pids in the root pid_namespace.

Or, if it is ok for the pid namespace operations to be as coarse as
"kill all processes in /vserver1", then that was going to be implemented
using the namespace container subsystem as:

 rm -rf /container_ns/vserver1

Any other (a) requirements, (b) ideas for alternate pid and network
ns management without allowing namespace enters?

-serge

Subject: [PATCH 1/4] namespaces: make get_nsproxy nonstatic
Posted by serue on Mon, 19 Feb 2007 22:15:36 GMT
View Forum Message <> Reply to Message

From: Serge E. Hallyn <serue@us.ibm.com>
Subject: [PATCH 1/4] namespaces: make get_nsproxy nonstatic

We'll need get_nsproxy accessible from containers code, but it is
statically defined in kernel/nsproxy.c. Make it inline in
include/linux/nsproxy.h.

Signed-off-by: Serge E. Hallyn <serue@us.ibm.com>

---

include/linux/nsproxy.h | 5 +++++
kernel/nsproxy.c | 5 -----
2 files changed, 5 insertions(+), 5 deletions(-)

```
ad2cf2568b8fe9cf094905ec7370149baafc6495
diff --git a/include/linux/nsproxy.h b/include/linux/nsproxy.h
index c55f20b..0255e27 100644
--- a/include/linux/nsproxy.h
+++ b/include/linux/nsproxy.h
@@ -31,6 +31,11 @@ struct nsproxy {
 };
 extern struct nsproxy init_nsproxy;

+static inline void get_nsproxy(struct nsproxy *ns)
+{
+ atomic_inc(&ns->count);
+}
+
 struct nsproxy *dup_namespaces(struct nsproxy *orig);
 int copy_namespaces(int flags, struct task_struct *tsk);
 void get_task_namespaces(struct task_struct *tsk);
diff --git a/kernel/nsproxy.c b/kernel/nsproxy.c
index d174d1f..1123ab2 100644
--- a/kernel/nsproxy.c
+++ b/kernel/nsproxy.c
@@ -23,11 +23,6 @@

 struct nsproxy init_nsproxy = INIT_NSPROXY(init_nsproxy);

-static inline void get_nsproxy(struct nsproxy *ns)
-{
- atomic_inc(&ns->count);
-}
-
```

```
 void get_task_namespaces(struct task_struct *tsk)
 {
  struct nsproxy *ns = tsk->nsproxy;
--
1.1.6
```

Subject: [PATCH 2/4] namespace container: move nsproxy setting code
Posted by serue on Mon, 19 Feb 2007 22:16:00 GMT
View Forum Message <> Reply to Message

From: Serge E. Hallyn <serue@us.ibm.com>
Subject: [PATCH 2/4] namespace container: move nsproxy setting code

Move nsproxy setting code from clone and unshare into container_clone.
Containers will need to do this for namespace entering functionality, so
go ahead and move all setting of tsk->nsproxy there for simplicity/
consistency.

The clone path (at kernel/nsproxy.c:copy_namespaces()) should be
cleaned up:

1. The kfree(new_ns) on error at bottom may not be safe,
   if the nscont->nsproxy has already been set to it.
   However if it has been set, then container_clone() should
   have succeeded, so this *should* not be possible.

2. This path is taking a few extra copies - it sets the
   tsk->nsproxy to the new nsproxy early, then the
   swap_nsproxies() function copies it again. This should
   be cleaned up, but at least it is currently correct.

Best thing would be to create a common helper for the unshare
and clone cases.

Changelog:
 Feb 14: move swap_nsproxies call into ns_container.c so as to leave nsproxy
  knowledge out of container.c

Signed-off-by: Serge E. Hallyn <serue@us.ibm.com>

```
 include/linux/nsproxy.h |   20 ++++++++++++++++++--
 kernel/fork.c           |    7 +------
 kernel/ns_container.c   |    4 +++-
 kernel/nsproxy.c        |    2 +-
 4 files changed, 23 insertions(+), 10 deletions(-)
```

```
225efc9fea0771d283d22c393d948492162c84d4
diff --git a/include/linux/nsproxy.h b/include/linux/nsproxy.h
index 0255e27..d11eb09 100644
--- a/include/linux/nsproxy.h
+++ b/include/linux/nsproxy.h
@@ -58,10 +58,26 @@ static inline void exit_task_namespaces(
   put_nsproxy(ns);
  }
 }
+
+static inline void swap_nsproxies(struct task_struct *tsk, struct nsproxy *nsproxy)
+{
+ struct nsproxy *oldnsp;
+
+ task_lock(tsk);
+ oldnsp = tsk->nsproxy;
+ tsk->nsproxy = nsproxy;
+ get_nsproxy(nsproxy);
+ task_unlock(tsk);
+ put_nsproxy(oldnsp);
+}
+
 #ifdef CONFIG_CONTAINER_NS
-int ns_container_clone(struct task_struct *tsk);
+int ns_container_clone(struct task_struct *tsk, struct nsproxy *nsproxy);
 #else
-static inline int ns_container_clone(struct task_struct *tsk) { return 0; }
+static inline int ns_container_clone(struct task_struct *tsk, struct nsproxy *nsproxy) {
+ swap_nsproxies(tsk, nsproxy);
+ return 0;
+}
 #endif

 #endif
diff --git a/kernel/fork.c b/kernel/fork.c
index b1a3d6c..4ebdd53 100644
--- a/kernel/fork.c
+++ b/kernel/fork.c
@@ -1663,7 +1663,7 @@ asmlinkage long sys_unshare(unsigned lon
    err = -ENOMEM;
    goto bad_unshare_cleanup_ipc;
   }
-  err = ns_container_clone(current);
+  err = ns_container_clone(current, new_nsproxy);
   if (err)
    goto bad_unshare_cleanup_dupns;
  }
```

```
@@ -1673,11 +1673,6 @@ asmlinkage long sys_unshare(unsigned lon

   task_lock(current);

-  if (new_nsproxy) {
-   current->nsproxy = new_nsproxy;
-   new_nsproxy = old_nsproxy;
-  }
-
   if (new_fs) {
    fs = current->fs;
    current->fs = new_fs;
diff --git a/kernel/ns_container.c b/kernel/ns_container.c
index c90485d..23fac0e 100644
--- a/kernel/ns_container.c
+++ b/kernel/ns_container.c
@@ -7,6 +7,7 @@
 #include <linux/module.h>
 #include <linux/container.h>
 #include <linux/fs.h>
+#include <linux/nsproxy.h>

 struct nscont {
  struct container_subsys_state css;
@@ -21,8 +22,9 @@ static inline struct nscont *container_n
        struct nscont, css);
 }

-int ns_container_clone(struct task_struct *tsk)
+int ns_container_clone(struct task_struct *tsk, struct nsproxy *nsproxy)
 {
+ swap_nsproxies(tsk, nsproxy);
  return container_clone(tsk, &ns_subsys);
 }

diff --git a/kernel/nsproxy.c b/kernel/nsproxy.c
index 1123ab2..6312ef8 100644
--- a/kernel/nsproxy.c
+++ b/kernel/nsproxy.c
@@ -111,7 +111,7 @@ int copy_namespaces(int flags, struct ta
  if (err)
   goto out_pid;

- err = ns_container_clone(tsk);
+ err = ns_container_clone(tsk, new_ns);
  if (err)
   goto out_container;
 out:
```

--
1.1.6

Subject: [PATCH 3/4] namespace containers: add nsproxy to nscont struct
Posted by serue on Mon, 19 Feb 2007 22:16:20 GMT
View Forum Message <> Reply to Message

From: Serge E. Hallyn <serue@us.ibm.com>
Subject: [PATCH 3/4] namespace containers: add nsproxy to nscont struct

Each ns container is associated with an nsproxy. Add that
nsproxy to the nscont struct, set it when a container is auto-
created on clone/unshare, and inc/dec the nsproxy to account for
each container referencing it.

Note that once the nscont->nsproxy is set, it will never change
for the duration of the container's lifetime.

Changelog:
Feb 14: added ss->init_from_task() hook so ns_container
can initialize a container's private data from a task
on clone().

Signed-off-by: Serge E. Hallyn <serue@us.ibm.com>

Documentation/containers.txt | 9 +++++++++
include/linux/container.h | 1 +
include/linux/nsproxy.h | 1 +
kernel/container.c | 16 ++++++++++++++++
kernel/ns_container.c | 11 +++++++++++
5 files changed, 38 insertions(+), 0 deletions(-)

f863632142517f79ef885c238a8e5df238e8420c
diff --git a/Documentation/containers.txt b/Documentation/containers.txt
index 7918827..0001191 100644
--- a/Documentation/containers.txt
+++ b/Documentation/containers.txt
@@ -466,6 +466,15 @@ LL=manage_mutex
The container system is about to destroy the passed container; the
subsystem should do any necessary cleanup

+int init_from_task(struct container *cont, struct task_struct *task)
+LL=manage_mutex
+
+Called during a container_clone() call to allow differentiation

```
+between a container created automatically and one created by hand.
+A container created by hand inherits the nsproxy from the parent
+container.  A container created automatically inherits the nsproxy
+from the task entering, which may have already done some unsharing.
+
 int can_attach(struct container_subsys *ss, struct container *cont,
         struct task_struct *task)
 LL=manage_mutex
diff --git a/include/linux/container.h b/include/linux/container.h
index db2fc27..4c9c092 100644
--- a/include/linux/container.h
+++ b/include/linux/container.h
@@ -197,6 +197,7 @@ struct container_subsys {
  int (*create)(struct container_subsys *ss,
         struct container *cont);
  void (*destroy)(struct container_subsys *ss, struct container *cont);
+ void (*init_from_task)(struct container *cont, struct task_struct *task);
  int (*can_attach)(struct container_subsys *ss,
      struct container *cont, struct task_struct *tsk);
  void (*attach)(struct container_subsys *ss, struct container *cont,
diff --git a/include/linux/nsproxy.h b/include/linux/nsproxy.h
index d11eb09..43f5696 100644
--- a/include/linux/nsproxy.h
+++ b/include/linux/nsproxy.h
@@ -71,6 +71,7 @@ static inline void swap_nsproxies(struct
  put_nsproxy(oldnsp);
 }

+struct container;
 #ifdef CONFIG_CONTAINER_NS
 int ns_container_clone(struct task_struct *tsk, struct nsproxy *nsproxy);
 #else
diff --git a/kernel/container.c b/kernel/container.c
index 0606753..0352f84 100644
--- a/kernel/container.c
+++ b/kernel/container.c
@@ -920,6 +920,19 @@ static int attach_task(struct container
  return 0;
 }

+static void init_container_from_task(struct container *cont,
+      struct task_struct *tsk)
+{
+ struct container_subsys *ss;
+ int h = cont->hierarchy;
+
+ for_each_subsys(h, ss) {
+  if (ss->init_from_task) {
```

```
+   ss->init_from_task(cont, tsk);
+  }
+ }
+}
+
 /*
  * Attach task with pid 'pid' to container 'cont'. Call with
  * manage_mutex, may take callback_mutex and task_lock of task
@@ -1665,6 +1678,9 @@ int container_clone(struct task_struct *
   goto again;
  }

+ /* the new container needs private info initialized from the task */
+ init_container_from_task(child, tsk);
+
  /* All seems fine. Finish by moving the task into the new container */
  ret = attach_task(child, tsk);
  mutex_unlock(&manage_mutex);
diff --git a/kernel/ns_container.c b/kernel/ns_container.c
index 23fac0e..1cc9cea 100644
--- a/kernel/ns_container.c
+++ b/kernel/ns_container.c
@@ -11,6 +11,7 @@

 struct nscont {
  struct container_subsys_state css;
+ struct nsproxy *nsproxy; /* never changes once set */
  spinlock_t lock;
 };

@@ -82,10 +83,19 @@ static int ns_create(struct container_su
  return 0;
 }

+void ns_init_from_task(struct container *cont, struct task_struct *tsk)
+{
+ struct nscont *ns = container_nscont(cont);
+ ns->nsproxy = tsk->nsproxy;
+ get_nsproxy(ns->nsproxy);
+}
+
 static void ns_destroy(struct container_subsys *ss,
         struct container *cont)
 {
  struct nscont *ns = container_nscont(cont);
+ if (ns->nsproxy)
+  put_nsproxy(ns->nsproxy);
  kfree(ns);
```

```
 }

@@ -97,6 +107,7 @@ static struct container_subsys ns_subsys
  //.attach = ns_attach,
  //.post_attach = ns_post_attach,
  //.populate = ns_populate,
+ .init_from_task = ns_init_from_task,
  .subsys_id = -1,
 };

--
1.1.6
```

Subject: [PATCH 4/4] namespace containers: implement enter into existing container

Posted by serue on Mon, 19 Feb 2007 22:16:34 GMT

View Forum Message <> Reply to Message

From: Serge E. Hallyn <serue@us.ibm.com>
Subject: [PATCH 4/4] namespace containers: implement enter into existing container

Implement ns container subsys.can_attach(). Remove the constraint
in can_attach() that the destination container must be unpopulated.

When entering a container, if the container's nsproxy has not been
set, set it to the parent container's. This will eventually support
more complicated creation by composing namespaces from several
nsproxies.

Finally set the task's nsproxy to the container's to effect the
namespace transition.

Signed-off-by: Serge E. Hallyn <serue@us.ibm.com>

```
 kernel/ns_container.c |   30 ++++++++++++++++++++++++++----
 1 files changed, 26 insertions(+), 4 deletions(-)

fa70868cb500ac6e3319f6a5c04dac5e419cefbf
diff --git a/kernel/ns_container.c b/kernel/ns_container.c
index 1cc9cea..76044ad 100644
--- a/kernel/ns_container.c
+++ b/kernel/ns_container.c
@@ -23,6 +23,17 @@ static inline struct nscont *container_n
        struct nscont, css);
 }
```

```
+/* for now we just take the parnet container nsproxy.
+ * eventually we will construct them based on file link activity */
+static void ns_create_nsproxy(struct nscont *ns, struct container *cont)
+{
+ struct container *parent = cont->parent;
+ struct nscont *parentns;
+ parentns = container_nscont(parent);
+ ns->nsproxy = parentns->nsproxy;
+ get_nsproxy(ns->nsproxy);
+}
+
 int ns_container_clone(struct task_struct *tsk, struct nsproxy *nsproxy)
 {
  swap_nsproxies(tsk, nsproxy);
@@ -52,9 +63,6 @@ int ns_can_attach(struct container_subsy
    return -EPERM;
  }

- if (atomic_read(&cont->count) != 0)
-  return -EPERM;
-
  c = task_container(tsk, &ns_subsys);
  if (c && c != cont->parent)
   return -EPERM;
@@ -90,6 +98,20 @@ void ns_init_from_task(struct container
  get_nsproxy(ns->nsproxy);
 }

+void ns_attach(struct container_subsys *ss, struct container *cont,
+     struct container *old_cont, struct task_struct *tsk)
+{
+ struct nscont *ns = container_nscont(cont);
+ if (!ns->nsproxy) {
+  spin_lock(&ns->lock);
+  if (!ns->nsproxy)
+   ns_create_nsproxy(ns, cont);
+  spin_unlock(&ns->lock);
+ }
+ if (tsk->nsproxy != ns->nsproxy)
+  swap_nsproxies(tsk, ns->nsproxy);
+}
+
 static void ns_destroy(struct container_subsys *ss,
         struct container *cont)
 {
@@ -104,7 +126,7 @@ static struct container_subsys ns_subsys
  .create = ns_create,
```

```
  .destroy  = ns_destroy,
  .can_attach = ns_can_attach,
- //.attach = ns_attach,
+ .attach = ns_attach,
  //.post_attach = ns_post_attach,
  //.populate = ns_populate,
  .init_from_task = ns_init_from_task,
--
1.1.6
```

Subject: Re: [RFC] ns containers (v2): namespace entering
Posted by ebiederm on Tue, 20 Feb 2007 03:08:45 GMT
View Forum Message <> Reply to Message

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

> (Updated patchset addressing Paul's comments. Still looking more
> for discussion on functionality, but barring much of that I guess
> I'll do something with tsk->fs then send the set to lkml)
>
> The following patchset uses the ns container subsystem to implement
> namespace entering. It applies over the container patchset Paul
> sent out earlier today.
>

Let's describe the problem. There are two usage scenarios.
- Working with the parameters of an existing namespace.
  (For example: changing or reading the hostname of a container you are not
   in, or setting/reading various resource limits).
- Logging in to an existing namespace. I.e. You are the uber sysadmin
  and the person running a container has a problem or something similar.

Until we work through the details of the pid namespace there are going
to be details of this we cannot finalize, because we will not have finalized
how traditional process groups work. Something more for my todo guess.

I have given this some though and I can describe how far we can go without
implementing a traditional enter, which is a very long ways.

For the general management functions most things already have or can
have added a filesystem interface, and we just need to make that
filesystem interface per process...

That is we can do like what is currently done with /proc/mounts and
make /proc/sys a symlink to /proc/self/sys and /proc/<otherpid>/sys

can be that other processes view of all of the sysctls.

We can do the same thing with /proc/net, and /proc/sysvipc and
possibly /sys/net although that is a more difficult. sysfs is a pain
to work with.

For the actual enter functionality what is currently possible is ugly
to implement but extremely close to something useful. You can use
sys_ptrace and pick a random process and with a little care cause
it to fork and exec the executable of your choice. With these
manipulations you can create things like unix domain sockets and
pipes and that can be used to talk with the parent process. It has
been a while since I have done this so I don't remember the exact
limits are but I have gotten a fully functional login shell this way.
The big practical problem was who is the parent process.
CAP_SYS_PTRACE is the governing capability here.
I've got the code around someplace for doing this if you are curious.

If we optimize/cleanup this case it looks a lot like what (I think it
was Cedric) was proposing about a year ago. A magic fork that does
the enter for you. Since the caller controls the binary we don't
have the usual concerns about the magic fork becoming spawn, because
we can program the binary to do anything else it needs after the fork.

> This is RFC not just on implementation, but also on whether to do
> it at all. If so, then for all namespace, or only some? And if not,
> how to facilitate virtual server management.

My gut feeling is the best way to go is something that is a refinement
of the two techniques I have listed above.

>
> = Security =
>
> Currently to enter a namespace, you must have CAP_SYS_ADMIN, and must
> be entering a container which is an immediate child of your current
> container. So from the root container you could enter container /vserver1,
> but from container /vserver1 you could not enter /vserver2 or the root
> container.
>
> This may turn out to be sufficient. If not, then LSM hooks should be
> added for namespace management. Four hooks for nsproxy management (create,
> compose, may_enter, and enter), as well as some security_ns_clone hook for
> each separate namespace, so that the nsproxy enter and compose hooks have
> the information they need to properly authorize.

You miss an issue here. One of the dangers of enter is leaking
capabilities into a contained set of processes. Once you show up in
/proc processes can change into your working directory which is
outside of the container for example.

> = Quick question =
>
> Is it deemed ok to allow entering an existing namespace?
>
> If so, the next section can be disregarded. Assuming not, the following
> will need to be worked out.

I think we need to take a good hard look at the alternatives, because
the are very functional.

> = Management alternatives =
>
> Mounts
>
> Ram has suggested that for mounts, instead of implementing namespace
> entering, the example from the OLS Shared Subtree paper could be
> used, as follows (quoting from Ram):

Does anyone know why we have the shared subtree entering instead of
a enter on a fs namespace? I'm curious about the reasoning for that
design decision.

> Herbert, and anyone else who wants mounts namespace entering, is the
> above an acceptable alternative?
>
> net+pid+uts
>
> Not sure about uts, but I'm pretty sure the vserver folks want the ability
> to enter another existing network namespace, and both vserver and openvz
> have asked for the ability to enter pid namespaces.
>
> The pid namespaces could be solved by always generating as many pids for
> a process as it has parent pid_namespaces. So if I'm in /vserver1, with
> one pid_namespace above me, not only my init process has an entry in the
> root pid_namespace (as I think has been suggested), but all my children
> will also continue to have pids in the root pid_namespace.

Yes. This looks to be the most sensible thing and now that we have
struct pid we don't have to special case anything to implement a
pid showing up in multiple pid namespaces. So it is my expectation
that each process will show up in the pid namespace of all of it's
parents up to init.

> Or, if it is ok for the pid namespace operations to be as coarse as
> "kill all processes in /vserver1", then that was going to be implemented
> using the namespace container subsystem as:
>
> rm -rf /container_ns/vserver1

To some extent I have an issue with this since we have kill and
signals and other mechanisms already existing for most of this
the duplication at the very least seems silly.

> Any other (a) requirements, (b) ideas for alternate pid and network
> ns management without allowing namespace enters?

See above.

I'm stretched pretty thin at the moment, so this will have to do for
a first set of comments.

Eric

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

Subject: Re: [RFC] ns containers (v2): namespace entering
Posted by serue on Tue, 20 Feb 2007 16:27:08 GMT
View Forum Message <> Reply to Message

Quoting Eric W. Biederman (ebiederm@xmission.com):
> "Serge E. Hallyn" <serue@us.ibm.com> writes:
>
> > (Updated patchset addressing Paul's comments. Still looking more
> > for discussion on functionality, but barring much of that I guess
> > I'll do something with tsk->fs then send the set to lkml)
> >
> > The following patchset uses the ns container subsystem to implement
> > namespace entering. It applies over the container patchset Paul
> > sent out earlier today.
> >
>
> Let's describe the problem. There are two usage scenarios.
> - Working with the parameters of an existing namespace.
>   (For example: changing or reading the hostname of a container you are not
>   in, or setting/reading various resource limits).
> - Logging in to an existing namespace. I.e. You are the uber sysadmin
>   and the person running a container has a problem or something similar.

>
> Until we work through the details of the pid namespace there are going
> to be details of this we cannot finalize, because we will not have finalized
> how traditional process groups work. Something more for my todo guess.
>
> I have given this some though and I can describe how far we can go without
> implementing a traditional enter, which is a very long ways.
>
>
>
> For the general management functions most things already have or can
> have added a filesystem interface, and we just need to make that
> filesystem interface per process...
>
> That is we can do like what is currently done with /proc/mounts and
> make /proc/sys a symlink to /proc/self/sys and /proc/<otherpid>/sys
> can be that other processes view of all of the sysctls.
>
> We can do the same thing with /proc/net, and /proc/sysvipc and
> possibly /sys/net although that is a more difficult. sysfs is a pain
> to work with.
>
>
>
> For the actual enter functionality what is currently possible is ugly
> to implement but extremely close to something useful. You can use
> sys_ptrace and pick a random process and with a little care cause
> it to fork and exec the executable of your choice. With these
> manipulations you can create things like unix domain sockets and
> pipes and that can be used to talk with the parent process. It has
> been a while since I have done this so I don't remember the exact
> limits are but I have gotten a fully functional login shell this way.
> The big practical problem was who is the parent process.
> CAP_SYS_PTRACE is the governing capability here.
> I've got the code around someplace for doing this if you are curious.
>
> If we optimize/cleanup this case it looks a lot like what (I think it
> was Cedric) was proposing about a year ago. A magic fork that does
> the enter for you. Since the caller controls the binary we don't
> have the usual concerns about the magic fork becoming spawn, because
> we can program the binary to do anything else it needs after the fork.
>
> > This is RFC not just on implementation, but also on whether to do
> > it at all. If so, then for all namespace, or only some? And if not,
> > how to facilitate virtual server management.
>
> My gut feeling is the best way to go is something that is a refinement
> of the two techniques I have listed above.

>
> >
> > = Security =
> >
> > Currently to enter a namespace, you must have CAP_SYS_ADMIN, and must
> > be entering a container which is an immediate child of your current
> > container. So from the root container you could enter container /vserver1,
> > but from container /vserver1 you could not enter /vserver2 or the root
> > container.
> >
> > This may turn out to be sufficient. If not, then LSM hooks should be
> > added for namespace management. Four hooks for nsproxy management (create,
> > compose, may_enter, and enter), as well as some security_ns_clone hook for
> > each separate namespace, so that the nsproxy enter and compose hooks have
> > the information they need to properly authorize.
>
> You miss an issue here. One of the dangers of enter is leaking
> capabilities into a contained set of processes. Once you show up in

Good point. As wrong as it feels to me to use ptrace for this, the
advantage is that none of my task attributes leak into the target
namespace, and that's a very good thing.

I do wonder how you specify what the forced clone should run.
Presumably you want to run something not in the target container.
I suppose we can pass the fd over a socket or something.

Herbert, does this sound like it suffices to you? Of course, it does
mean that you cannot switch just a few namespaces. How does that limit
you?

> /proc processes can change into your working directory which is
> outside of the container for example.

I suppose I could get into specific ways that this could be prevented
from being exploited, but for now I'll just stick to agreeing that there
would be a whole bunch of issues like this, and we'd likely miss more
than one.

> > = Quick question =
> >
> > Is it deemed ok to allow entering an existing namespace?
> >
> > If so, the next section can be disregarded. Assuming not, the following
> > will need to be worked out.
>
> I think we need to take a good hard look at the alternatives, because
> the are very functional.

>
> > = Management alternatives =
> >
> > Mounts
> >
> > Ram has suggested that for mounts, instead of implementing namespace
> > entering, the example from the OLS Shared Subtree paper could be
> > used, as follows (quoting from Ram):
>
> Does anyone know why we have the shared subtree entering instead of
> a enter on a fs namespace? I'm curious about the reasoning for that
> design decision.
>
> > Herbert, and anyone else who wants mounts namespace entering, is the
> > above an acceptable alternative?
> >
> > net+pid+uts
> >
> > Not sure about uts, but I'm pretty sure the vserver folks want the ability
> > to enter another existing network namespace, and both vserver and openvz
> > have asked for the ability to enter pid namespaces.
> >
> > The pid namespaces could be solved by always generating as many pids for
> > a process as it has parent pid_namespaces. So if I'm in /vserver1, with
> > one pid_namespace above me, not only my init process has an entry in the
> > root pid_namespace (as I think has been suggested), but all my children
> > will also continue to have pids in the root pid_namespace.
>
> Yes. This looks to be the most sensible thing and now that we have
> struct pid we don't have to special case anything to implement a
> pid showing up in multiple pid namespaces. So it is my expectation
> that each process will show up in the pid namespace of all of it's
> parents up to init.

In fact we were thinking there could be an additional clone flag when
doing CLONE_PIDNS to determine whether all processes get pids in all
ancestor pid_namespaces or not, since really the only change should be
an overhead at clone() for allocing and linking the additional struct
pid_nr's.

> > Or, if it is ok for the pid namespace operations to be as coarse as
> > "kill all processes in /vserver1", then that was going to be implemented
> > using the namespace container subsystem as:
> >
> > rm -rf /container_ns/vserver1
>
> To some extent I have an issue with this since we have kill and
> signals and other mechanisms already existing for most of this

> the duplication at the very least seems silly.

But you don't really. If I want to kill all processes in a child container, I don't have an easy way to list only the processes in the child container using, say, ps. Those processes are just lumped in with my own, and with those from all my other child containers.

So we'd end up looping over all pids, checking their ->container, then killing them, and hoping that nothing funky happens meanwhile like that process dies and pids wrap around while i'm sleeping (ok, unlikely, but not impossible).

I'm not saying I'm hung up on doing it with an rm /container_ns/vs1. If the only use for the ns container subsystem ends up being to tie resource management to containers, I'll be very happy.

> > Any other (a) requirements, (b) ideas for alternate pid and network
> > ns management without allowing namespace enters?
>
>
> See above.
>
> I'm stretched pretty thin at the moment, so this will have to do for
> a first set of comments.

Thanks for your time, Eric.

-serge

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

Subject: Re: [RFC] ns containers (v2): namespace entering
Posted by ebiederm on Tue, 20 Feb 2007 17:38:12 GMT
View Forum Message <> Reply to Message

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

> Quoting Eric W. Biederman (ebiederm@xmission.com):
>>
>> You miss an issue here. One of the dangers of enter is leaking
>> capabilities into a contained set of processes. Once you show up in
>
> Good point. As wrong as it feels to me to use ptrace for this, the
> advantage is that none of my task attributes leak into the target
> namespace, and that's a very good thing.

>
> I do wonder how you specify what the forced clone should run.
> Presumably you want to run something not in the target container.
> I suppose we can pass the fd over a socket or something.

Yes. At least in the case without a network namespace I can setup
a unix domain socket and pass file descriptors around. I think my solution
to the network namespace case was to just setup a unix domain socket in
the parent namespace and leave it open in init. Not a real solution :(

> Herbert, does this sound like it suffices to you? Of course, it does
> mean that you cannot switch just a few namespaces. How does that limit
> you?
>
>> /proc processes can change into your working directory which is
>> outside of the container for example.
>
> I suppose I could get into specific ways that this could be prevented
> from being exploited, but for now I'll just stick to agreeing that there
> would be a whole bunch of issues like this, and we'd likely miss more
> than one.

Thanks. I think what we want is an API where what is leaked is an
explicit decision rather then everything leaking implicitly and the
user space programmer has to run around and close all of the holes.

>> Yes. This looks to be the most sensible thing and now that we have
>> struct pid we don't have to special case anything to implement a
>> pid showing up in multiple pid namespaces. So it is my expectation
>> that each process will show up in the pid namespace of all of it's
>> parents up to init.
>
> In fact we were thinking there could be an additional clone flag when
> doing CLONE_PIDNS to determine whether all processes get pids in all
> ancestor pid_namespaces or not, since really the only change should be
> an overhead at clone() for allocing and linking the additional struct
> pid_nr's.

I doubt it is worth the hassle of making the code conditional.

>> > Or, if it is ok for the pid namespace operations to be as coarse as
>> > "kill all processes in /vserver1", then that was going to be implemented
>> > using the namespace container subsystem as:
>> >
>> > rm -rf /container_ns/vserver1
>>
>> To some extent I have an issue with this since we have kill and
>> signals and other mechanisms already existing for most of this

>> the duplication at the very least seems silly.
>
> But you don't really. If I want to kill all processes in a child
> container, I don't have an easy way to list only the processes in the
> child container using, say, ps. Those processes are just lumped in with
> my own, and with those from all my other child containers.

True. Mostly because that is the interface that makes the most
sense most of the time.

> So we'd end up looping over all pids, checking their ->container, then
> killing them, and hoping that nothing funky happens meanwhile like that
> process dies and pids wrap around while i'm sleeping (ok, unlikely, but
> not impossible).
>
> I'm not saying I'm hung up on doing it with an rm /container_ns/vs1. If
> the only use for the ns container subsystem ends up being to tie
> resource management to containers, I'll be very happy.

We've got to get a better name for those things....

Regardless one of the requirements of the pid namespace is that we have
a process grouping of the entire namespace that kill(-1 can signal
and there is no reason not to have that process grouping available on
the outside of the pid namespace as well.

Eric

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

Subject: Re: [RFC] ns containers (v2): namespace entering
Posted by serue on Wed, 21 Feb 2007 21:04:01 GMT
View Forum Message <> Reply to Message

Quoting Eric W. Biederman (ebiederm@xmission.com):
> "Serge E. Hallyn" <serue@us.ibm.com> writes:
>
> > Quoting Eric W. Biederman (ebiederm@xmission.com):
> >>
> >> You miss an issue here. One of the dangers of enter is leaking
> >> capabilities into a contained set of processes. Once you show up in
> >
> > Good point. As wrong as it feels to me to use ptrace for this, the
> > advantage is that none of my task attributes leak into the target
> > namespace, and that's a very good thing.

> >
> > I do wonder how you specify what the forced clone should run.
> > Presumably you want to run something not in the target container.
> > I suppose we can pass the fd over a socket or something.
>
> Yes. At least in the case without a network namespace I can setup
> a unix domain socket and pass file descriptors around. I think my solution
> to the network namespace case was to just setup a unix domain socket in
> the parent namespace and leave it open in init. Not a real solution :(

How about we solve both this and the general ugliness of using ptrace
with a new

 hijack_and_clone(struct task_struct *tsk, int fd)

Which takes tsk, clones it, and execs the contents of fd?

-serge

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

Subject: Re: [RFC] ns containers (v2): namespace entering
Posted by ebiederm on Thu, 22 Feb 2007 08:55:27 GMT
View Forum Message <> Reply to Message

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

> How about we solve both this and the general ugliness of using ptrace
> with a new
>
> hijack_and_clone(struct task_struct *tsk, int fd)
>
> Which takes tsk, clones it, and execs the contents of fd?

That is what roughly what I was thinking. Although that is an ugly
beast to implement. Getting stdin and stdout should still be doable
using a tty. Getting the semantics and the implementation right is
a tough challenge.

After thinking about it a normal exec (unless you want your binary to
come from inside the namespace) is nearly useless because it requires
a static binary. With glibc not actually going static, static
binaries are nearly impossible to write. Although that might be
a good argument for minimalism, and security.

The really important use of the ptrace case is that it works using
existing mechanisms without leaks. So it is very useful yardstick.

The other important yardstick is arranging it so that when you login
to a machine all of the user code runs in your target environment.
How you get there is irrelevant.

One of the cases I have been worrying about in looking at the
semantics of enter is what do you do with the parent pid. Supporting
ptrace from outside the pid namespace of a process inside a pid
namespace requires supporting a parent process outside of the pid
namespace for processes other than init.

I'm not convinced setting up a non-ptrace parent that is outside the
pid namespace makes sense, but it looks like the mechanism is going
to be there. If we did support a foreign parent it would go a long
way towards supporting the login and be redirected into a container
case, as well as Herbert's pid namespace without an init case.

I have finally worked through all of the reasonable irq handling
alternatives and unless something goes wrong will I will be submitting
that code tomorrow. I really want to pull some pid namespace patches
together so we can bring those into the conversation but I don't
think I will be able to get there before I head out first of March
to Nebraska to spend some time with my brother.

Eric

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

Subject: Re: [RFC] ns containers (v2): namespace entering
Posted by Paul Menage on Thu, 22 Feb 2007 16:53:10 GMT
View Forum Message <> Reply to Message

On 2/22/07, Eric W. Biederman <ebiederm@xmission.com> wrote:
>
> The really important use of the ptrace case is that it works using
> existing mechanisms without leaks. So it is very useful yardstick.
>
> The other important yardstick is arranging it so that when you login
> to a machine all of the user code runs in your target environment.
> How you get there is irrelevant.
>
> One of the cases I have been worrying about in looking at the
> semantics of enter is what do you do with the parent pid. Supporting

> ptrace from outside the pid namespace of a process inside a pid
> namespace requires supporting a parent process outside of the pid
> namespace for processes other than init.
>

When I implemented a virtual server solution at my previous job, we solved the problem of leaking capabilities into the virtualized environment by allowing a process to enter the virtual server in a "privileged" mode, in which it didn't appear in the virtualized /proc, and hence none of its resources were accessible to processes in the VS. Children of a privileged process were by default regular members of the virtual server.

With the nsproxy model, maybe you could implement that by having two pid namespaces. When you create the VS, you create an entire set of new namespaces; then before forking init you create a new pid_ns. So by entering the outer nsproxy you'd get access to the same environment that the VS sees, but your process wouldn't be visible to processes in the VS. If you wanted to create a regular process, you'd just enter the inner pid_ns too. I think that doing the appropriate process "sanitization" to avoid leaking capabilities would be easier from the "twilight zone" intermediate nsproxy than from the machine top-level.

Paul

Subject: Re: [RFC] ns containers (v2): namespace entering
Posted by ebiederm on Thu, 22 Feb 2007 20:42:46 GMT
View Forum Message <> Reply to Message

"Paul Menage" <menage@google.com> writes:

> When I implemented a virtual server solution at my previous job, we
> solved the problem of leaking capabilities into the virtualized
> environment by allowing a process to enter the virtual server in a
> "privileged" mode, in which it didn't appear in the virtualized /proc,
> and hence none of its resources were accessible to processes in the
> VS. Children of a privileged process were by default regular members
> of the virtual server.
>
> With the nsproxy model, maybe you could implement that by having two
> pid namespaces. When you create the VS, you create an entire set of
> new namespaces; then before forking init you create a new pid_ns. So
> by entering the outer nsproxy you'd get access to the same environment

> that the VS sees, but your process wouldn't be visible to processes in
> the VS. If you wanted to create a regular process, you'd just enter
> the inner pid_ns too. I think that doing the appropriate process
> "sanitization" to avoid leaking capabilities would be easier from the
> "twilight zone" intermediate nsproxy than from the machine top-level.

And this is why I keep coming back to ptrace. It doesn't leak and
it does seem to get the job done. If you have to because you can
run system calls you can implement exec by hand. ptrace also allows
many of the intermediate enter scenarios.

ptrace is clumsy and there does have to be a better way but until I find
something that isn't clumsy I will stick with ptrace.

Now it is at least worth investigating if you can leak things if you don't
enter the pid namespace. If you can not leak things that potentially
simplifies big chunks of the problem, and we probably don't need the
intermediate pid namespace, of your suggestion.

Still I'm drawn back to the simplicity of the proof of the ptrace
situation.

On another note Serge there is another possibility for dealing with
fs_struct. You can chdir(/proc/<someotherpidinthenamespace>/) and
get your working directory changed after the unshare.

All I know for certain is that it really sucks designing things with
new semantics that interact closely with all of the old existing
rules. It is really easy to come up with something that doesn't
work well, or is a major hazard when people use it in unexpected ways.

So I'm generally in favor of not rushing things and taking the big
things like enter in as small a step as we can. Because whatever
we merge we will likely need to be supported for the rest our lives.

Eric

Subject: Re: [RFC] ns containers (v2): namespace entering
Posted by Paul Menage on Thu, 22 Feb 2007 20:49:34 GMT
View Forum Message <> Reply to Message

On 2/22/07, Eric W. Biederman <ebiederm@xmission.com> wrote:

>
> Now it is at least worth investigating if you can leak things if you don't
> enter the pid namespace. If you can not leak things that potentially
> simplifies big chunks of the problem, and we probably don't need the
> intermediate pid namespace, of your suggestion.

If you're happy to have your partially-entered process be viewing the
system pid namespace rather than (container pid namespace) + (self)
then yes, you don't need the intermediate namespace.

Paul

Subject: Re: [RFC] ns containers (v2): namespace entering
Posted by Herbert Poetzl on Sat, 10 Mar 2007 01:36:29 GMT
View Forum Message <> Reply to Message

On Mon, Feb 19, 2007 at 04:14:46PM -0600, Serge E. Hallyn wrote:
> (Updated patchset addressing Paul's comments. Still looking more
> for discussion on functionality, but barring much of that I guess
> I'll do something with tsk->fs then send the set to lkml)
>
> The following patchset uses the ns container subsystem to implement
> namespace entering. It applies over the container patchset Paul
> sent out earlier today.
>
> = Usage =
>
> Following is example usage:
>
> mkdir /container_ns
> mount -t container -ons nsproxy /container_ns
> (start screen with two windows)
> (screen one)
> (unshare uts namespace)
> hostname serge
> (screen two)
> hostname
> (shows old hostname)
> echo $$ > /container_ns/node1/tasks (assuming node1 is screen one's new container)
> hostname
> (shows 'serge')
>
> Of course to get a useful result with the mounts namespace, we would

> have to convert the fs_struct to a namespace, and the mounts and
> fs_struct namespaces would be entered together.
>
> If we follow this path, the next step would be to add files under
> the container directory for each namespace, with ops->link defined
> such that you could
>
> mkdir /container_ns/new_container
> ln /container_ns/vserver1/network /container_ns/new_container/network
> echo $$ > /container_ns/new_container/tasks
>
> and be entered into a set of namespaces containing all of your defaults
> except network, which would come from vserver1.
>
> This is RFC not just on implementation, but also on whether to do
> it at all. If so, then for all namespace, or only some? And if not,
> how to facilitate virtual server management.
>
> = Security =
>
> Currently to enter a namespace, you must have CAP_SYS_ADMIN, and must
> be entering a container which is an immediate child of your current
> container. So from the root container you could enter container /vserver1,
> but from container /vserver1 you could not enter /vserver2 or the root
> container.
>
> This may turn out to be sufficient. If not, then LSM hooks should be
> added for namespace management. Four hooks for nsproxy management (create,
> compose, may_enter, and enter), as well as some security_ns_clone hook for
> each separate namespace, so that the nsproxy enter and compose hooks have
> the information they need to properly authorize.
>
> = Quick question =
>
> Is it deemed ok to allow entering an existing namespace?
>
> If so, the next section can be disregarded. Assuming not, the following
> will need to be worked out.
>
> = Management alternatives =
>
> Mounts
>
> Ram has suggested that for mounts, instead of implementing namespace
> entering, the example from the OLS Shared Subtree paper could be
> used, as follows (quoting from Ram):
>
> > The overall idea is each container creates its own fs-namespace which

> > has a mirror mount tree under /container/c1 in the original
> > fs-namespace. So any changes in the container's fs-namespace reflect in
> > the mount tree under /container/c1 in the original fs-namespace, and
> > vice-versa. And all processes in the original fs-namespace can freely
> > roam around in the mirror trees of all the containers, mounted
> > under /container/cx, giving illusion that they have control over the
> > mounts in all the containers. Whereas the processes in a container can
> > just roam around its own fs-namespace and has no visibility to anything
> > outside its own fs-namespace..
> >
> > As an example the way to accomplish this is:
> >
> >   in the original namespace,
> >   1-1) mkdir /container
> >   1-2) mount --bind /container /container
> >   1-3) mount --make-unbindable /container
> >
> >
> > when a new container c1 is created
> >
> >   2-1) mkdir /container/c1
> >   2-2) mount --rbind / /container/c1
> >   2-3) mount --make-rshared /container/c1
> >   2-4) clone fs-namespace
> >   in the new namespace,
> >   2-4-1) pivot_root /container/c1 /tmp
> >   2-4-2) umount -l /tmp
> >
> >
> > the mount at /container is made unbindable in steps 1-1 to 1-3, to
> > prune-out the mounts of other containers from being visible in this
> > container. The mount tree at /container/c1 is made shared in step
> > 2-3, to ensure that the cloned mount tree in the new namespace shall
> > be a exact mirror. The pivot_root in step 2-4-1 followed by umount in
> > step 2-4-2 ensures that the container sees only what is needed and
> > nothing else.
>
> Now, currently a root process in container c1 could make his fs unshared,
> but if that's deemed a problem presumably we can require an extra
> priv for that operation which can be dropped for any vservers.
>
> Herbert, and anyone else who wants mounts namespace entering, is the
> above an acceptable alternative?

sorry for the late answer, I almost missed that one ...

yes, that sounds like an acceptable alternative, but
it might give some interesting issues with references

to devices ... for example:

you mount a filesystem inside a namespace, so that
only the guest will see it (in theory) now you somehow
show that in the namespace copy too (on the host system)
and if some task decides to go camping there (cd into
that) it might keep the guest from unmounting that
device without ever knowing why ... or do you have some
smart solution to that?

> net+pid+uts
>
> Not sure about uts, but I'm pretty sure the vserver folks want the
> ability to enter another existing network namespace, and both vserver
> and openvz have asked for the ability to enter pid namespaces.

yes, definitely, pid and network namespaces have to
be accessible somehow, most administrative work is
done this way, when the administrator also maintains
the guests (i.e. doesn't want to bother accessing the
guest via special console/ssh/logon/whatever)

> The pid namespaces could be solved by always generating as many pids for
> a process as it has parent pid_namespaces. So if I'm in /vserver1, with
> one pid_namespace above me, not only my init process has an entry in the
> root pid_namespace (as I think has been suggested), but all my children
> will also continue to have pids in the root pid_namespace.

yep, sounds okay to me ...
note, our lightweight guests do not have an init
process, which is perfectly fine with the above, as
long as the init process is not considered a special
handle to the pid namespace :)

> Or, if it is ok for the pid namespace operations to be as coarse as
> "kill all processes in /vserver1", then that was going to be implemented
> using the namespace container subsystem as:
>
> rm -rf /container_ns/vserver1

that is definitely something you do not want to make
the general signalling solution, because typically
we have the following scenarios:

 - init less (lightweight) guest
   + a bunch of shutdown scripts are executed
   + term/kill is sent to the processes
   + the context is disposed

- init based guest
  + a signal is sent to init
  + init executes the shutdown and kills off
    the 'other' processes
  + init finally calls reboot/halt
  + init and the context are disposed

> Any other (a) requirements, (b) ideas for alternate pid and network
> ns management without allowing namespace enters?

entering the spaces seems most natural and quite
essential to me, especially for administration and
debugging purposes ...

best,
Herbert

> -serge

Subject: Re: [RFC] ns containers (v2): namespace entering
Posted by ebiederm on Sun, 11 Mar 2007 19:41:03 GMT
View Forum Message <> Reply to Message

Herbert Poetzl <herbert@13thfloor.at> writes:

>
> sorry for the late answer, I almost missed that one ...
>
> yes, that sounds like an acceptable alternative, but
> it might give some interesting issues with references
> to devices ... for example:
>
> you mount a filesystem inside a namespace, so that
> only the guest will see it (in theory) now you somehow
> show that in the namespace copy too (on the host system)
> and if some task decides to go camping there (cd into
> that) it might keep the guest from unmounting that
> device without ever knowing why ... or do you have some
> smart solution to that?

lazy unmount.

>> net+pid+uts
>>
>> Not sure about uts, but I'm pretty sure the vserver folks want the
>> ability to enter another existing network namespace, and both vserver

>> and openvz have asked for the ability to enter pid namespaces.
>
> yes, definitely, pid and network namespaces have to
> be accessible somehow, most administrative work is
> done this way, when the administrator also maintains
> the guests (i.e. doesn't want to bother accessing the
> guest via special console/ssh/logon/whatever)
>
>> The pid namespaces could be solved by always generating as many pids for
>> a process as it has parent pid_namespaces. So if I'm in /vserver1, with
>> one pid_namespace above me, not only my init process has an entry in the
>> root pid_namespace (as I think has been suggested), but all my children
>> will also continue to have pids in the root pid_namespace.
>
> yep, sounds okay to me ...
> note, our lightweight guests do not have an init
> process, which is perfectly fine with the above, as
> long as the init process is not considered a special
> handle to the pid namespace :)
>
>> Or, if it is ok for the pid namespace operations to be as coarse as
>> "kill all processes in /vserver1", then that was going to be implemented
>> using the namespace container subsystem as:
>>
>> rm -rf /container_ns/vserver1
>
> that is definitely something you do not want to make
> the general signalling solution, because typically
> we have the following scenarios:
>
> - init less (lightweight) guest
>   + a bunch of shutdown scripts are executed
>   + term/kill is sent to the processes
>   + the context is disposed
>
> - init based guest
>   + a signal is sent to init
>   + init executes the shutdown and kills off
>   the 'other' processes
>   + init finally calls reboot/halt
>   + init and the context are disposed

I have seen the same thing invented in a different context so this
sounds like a common pattern.

>> Any other (a) requirements, (b) ideas for alternate pid and network
>> ns management without allowing namespace enters?
>

> entering the spaces seems most natural and quite
> essential to me, especially for administration and
> debugging purposes ...

Yes. But how you implement the enter need not be modifying
the namespace pointer in a task_struct/nsproxy. You can get the same
user effect in other ways, which are potentially more secure.

Eric

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

Subject: Re: [RFC] ns containers (v2): namespace entering
Posted by serue on Mon, 12 Mar 2007 16:15:30 GMT
View Forum Message <> Reply to Message

Quoting Eric W. Biederman (ebiederm@xmission.com):
> Herbert Poetzl <herbert@13thfloor.at> writes:
>
> > >
> > > sorry for the late answer, I almost missed that one ...
> > >
> > > yes, that sounds like an acceptable alternative, but
> > > it might give some interesting issues with references
> > > to devices ... for example:
> > >
> > > you mount a filesystem inside a namespace, so that
> > > only the guest will see it (in theory) now you somehow
> > > show that in the namespace copy too (on the host system)
> > > and if some task decides to go camping there (cd into
> > > that) it might keep the guest from unmounting that
> > > device without ever knowing why ... or do you have some
> > > smart solution to that?
>
> lazy unmount.
>
> >> net+pid+uts
> >>
> >> Not sure about uts, but I'm pretty sure the vserver folks want the
> >> ability to enter another existing network namespace, and both vserver
> >> and openvz have asked for the ability to enter pid namespaces.
> > >
> > > yes, definitely, pid and network namespaces have to
> > > be accessible somehow, most administrative work is
> > > done this way, when the administrator also maintains

> > the guests (i.e. doesn't want to bother accessing the
> > guest via special console/ssh/logon/whatever)
> >
> >> The pid namespaces could be solved by always generating as many pids for
> >> a process as it has parent pid_namespaces. So if I'm in /vserver1, with
> >> one pid_namespace above me, not only my init process has an entry in the
> >> root pid_namespace (as I think has been suggested), but all my children
> >> will also continue to have pids in the root pid_namespace.
> >
> > yep, sounds okay to me ...
> > note, our lightweight guests do not have an init
> > process, which is perfectly fine with the above, as
> > long as the init process is not considered a special
> > handle to the pid namespace :)
> >
> >> Or, if it is ok for the pid namespace operations to be as coarse as
> >> "kill all processes in /vserver1", then that was going to be implemented
> >> using the namespace container subsystem as:
> >>
> >> rm -rf /container_ns/vserver1
> >
> > that is definitely something you do not want to make
> > the general signalling solution, because typically
> > we have the following scenarios:
> >
> > - init less (lightweight) guest
> >   + a bunch of shutdown scripts are executed
> >   + term/kill is sent to the processes
> >   + the context is disposed
> >
> > - init based guest
> >   + a signal is sent to init
> >   + init executes the shutdown and kills off
> >   the 'other' processes
> >   + init finally calls reboot/halt
> >   + init and the context are disposed
>
> I have seen the same thing invented in a different context so this
> sounds like a common pattern.
>
> >> Any other (a) requirements, (b) ideas for alternate pid and network
> >> ns management without allowing namespace enters?
> >
> > entering the spaces seems most natural and quite
> > essential to me, especially for administration and
> > debugging purposes ...
>
> Yes. But how you implement the enter need not be modifying

> the namespace pointer in a task_struct/nsproxy. You can get the same
> user effect in other ways, which are potentially more secure.

Thanks guys.

I think the main people interested in the namespace entering will be
Herbert, Kirill, and Cedric seems interested. Is someone planning to
try doing a non-enter virtual server management implementation, to see
what shortcomings there are? Or is it more likely that you'll keep
using your own namespace entering patches for a long time to come?

(The latter seems likely given that you still need to patch anyway at
the moment, so continuing to use your existing work is cheaper for you.
So i expect that experimenting with other approaches will be up to
someone doing it purely out of curiosity)

-serge

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

Subject: Re: [RFC] ns containers (v2): namespace entering
Posted by Dave Hansen on Mon, 12 Mar 2007 17:00:34 GMT
View Forum Message <> Reply to Message

On Sat, 2007-03-10 at 02:36 +0100, Herbert Poetzl wrote:
> you mount a filesystem inside a namespace, so that
> only the guest will see it (in theory) now you somehow
> show that in the namespace copy too (on the host system)
> and if some task decides to go camping there (cd into
> that) it might keep the guest from unmounting that
> device without ever knowing why ... or do you have some
> smart solution to that?

What is the actual issue here? That an underlying device might still be
in use, or that the container user has a directory they don't want
mounted sitting in their fs tree?

-- Dave

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

Subject: Re: [RFC] ns containers (v2): namespace entering
Posted by Herbert Poetzl on Tue, 13 Mar 2007 13:16:54 GMT
View Forum Message <> Reply to Message

On Mon, Mar 12, 2007 at 10:00:34AM -0700, Dave Hansen wrote:
> On Sat, 2007-03-10 at 02:36 +0100, Herbert Poetzl wrote:
> > you mount a filesystem inside a namespace, so that
> > only the guest will see it (in theory) now you somehow
> > show that in the namespace copy too (on the host system)
> > and if some task decides to go camping there (cd into
> > that) it might keep the guest from unmounting that
> > device without ever knowing why ... or do you have some
> > smart solution to that?
>
> What is the actual issue here?

> That an underlying device might still be in use,

yes, after thinking about it, it might not be such
an issue after all, because in 95% of all cases,
this is only a problem for the host admin, and can
be prevented by simply _not_ doing that ...

> or that the container user has a directory they don't want
> mounted sitting in their fs tree?

that shouldn't actually happen no? if the guest
is allowed to do unmounts, then the mount can be
removed from inside, if not, then the mount has to
be part of the guest configuration, so no problem
there IMHO

thanks,
Herbert

> -- Dave

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