Subject: [PATCH] containers: define a namespace container subsystem
Posted by serue on Wed, 31 Jan 2007 06:12:04 GMT
View Forum Message <> Reply to Message

Hi,

Here's my next, much more satisfying attempt at doing namespace
tracking through a container subsystem. The commit log pretty
much details where I would expect to go with this.

It behaves pretty differently from other subsystems implemented
so far, which could either be seen as evidence that it doesn't
belong as a subsystem, or, more likely, that the container
subsystem approach is quite flexible. I particularly like the
implications of being able to mount both the ns_container subsystem
and a resource usage subsystem under the same hierarchy to get
automatic resource control on vservers or checkpointable jobs.

-serge

From: "Serge E. Hallyn" <serue@us.ibm.com>
Subject: [PATCH] containers: define a namespace container subsystem

Define a container subsystem to track namespace unshares.
So long as the ns_container subsystem is mounted in a
container hierarchy, every unshare (or clone with a new
mounts, uts, etc namespace) will create a new child container
and move the process into it.

The purpose of this is to eventually allow operations on
virtual servers such as kill or enter, and checkpoint and
kill checkpointable namespaces.

Manual entering of an ns_container (that is, using
echo > /../container_dir/tasks) is only allowed into a
container which is a child of the current. Currently
there is also the restriction that the new container must
not yet have any tasks. This will change soon, but will
of course require that the task being transitioned switch
to the nsproxy associated with the new container.

Manual moving of an *other* process into a new container
also requires that the process being moved be in the same
container as or a subcontainer of that of the process
requesting the move.

Creation of an ns_container is restricted to subcontainers
of the creating task's container.

Creation of an ns_container, and moving another task into
a ns_container, both require CAP_SYS_ADMIN. The mkdir
control may seem overly harsh, but since currently CAP_SYS_ADMIN
is also required to unshare, it results in no additional
limitations.

The next steps are (not necessarily in order):

1. allow rm -rf to kill all processes under a
   ns_container - with the intent of killing all
   processes in a virtual server

2. implement transitioning into a populated container,
   with the effect of setting the task's nsproxy to
   the one represented by the container.

3. define a file for each type of namespace in each
   ns_container, with the i_op->symlink() defined to
   allow creation of a new ns_container which references
   only some of the namespace pointers of an existing
   (child) container. All other namespaces will be
   taken from the existing process. In this way it
   is possible to enter just a network namespace of
   some vserver.

4. probably make containers mac-aware, that is add a
   ->security pointer, and LSM hooks at appropriate
   points so that, for instance, SELinux can control
   vserver kill and enters.

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

---

 include/linux/container.h | 5 ++
 init/Kconfig | 9 +++
 kernel/Makefile | 1
 kernel/container.c | 114 ++
 kernel/fork.c | 5 ++
 kernel/ns_container.c | 117 ++
 kernel/nsproxy.c | 8 +++
 7 files changed, 259 insertions(+), 0 deletions(-)
 create mode 100644 kernel/ns_container.c

f4f9d31fa238b2ff804d35c6e1b5a0776f2bf303
diff --git a/include/linux/container.h b/include/linux/container.h
index 45db753..15b0446 100644

```
--- a/include/linux/container.h
+++ b/include/linux/container.h
@@ -207,6 +207,10 @@ static inline struct container_subsys_st
 }

 int container_path(const struct container *cont, char *buf, int buflen);
+int task_container_is_ancestor(struct task_struct *tsk, int subsys,
+       struct container *cont);
+
+int container_switch(struct task_struct *tsk);

 #else /* !CONFIG_CONTAINERS */

@@ -218,6 +222,7 @@ static inline void container_exit(struct

 static inline void container_lock(void) {}
 static inline void container_unlock(void) {}
+static inline void container_switch(struct task_struct *tsk) { }

 #endif /* !CONFIG_CONTAINERS */

diff --git a/init/Kconfig b/init/Kconfig
index ebaec57..c00b19c 100644
--- a/init/Kconfig
+++ b/init/Kconfig
@@ -297,6 +297,15 @@ config CONTAINER_CPUACCT
    Provides a simple Resource Controller for monitoring the
    total CPU consumed by the tasks in a container

+config CONTAINER_NS
+ bool "Namespace container subsystem"
+ select CONTAINERS
+ help
+   Provides a simple namespace container subsystem to
+   provide hierarchical naming of sets of namespaces,
+   for instance virtual servers and checkpoint/restart
+   jobs.
+
 config RELAY
  bool "Kernel->user space relay support (formerly relayfs)"
  help
diff --git a/kernel/Makefile b/kernel/Makefile
index feba860..6c73a5e 100644
--- a/kernel/Makefile
+++ b/kernel/Makefile
@@ -39,6 +39,7 @@ obj-$(CONFIG_COMPAT) += compat.o
 obj-$(CONFIG_CONTAINERS) += container.o
 obj-$(CONFIG_CPUSETS) += cpuset.o
```

```
 obj-$(CONFIG_CONTAINER_CPUACCT) += cpu_acct.o
+obj-$(CONFIG_CONTAINER_NS) += ns_container.o
 obj-$(CONFIG_IKCONFIG) += configs.o
 obj-$(CONFIG_STOP_MACHINE) += stop_machine.o
 obj-$(CONFIG_AUDIT) += audit.o auditfilter.o
diff --git a/kernel/container.c b/kernel/container.c
index 1f44351..97de43b 100644
--- a/kernel/container.c
+++ b/kernel/container.c
@@ -1558,6 +1558,120 @@ static int container_rmdir(struct inode
  return 0;
 }

+static int namecnt = 0;
+static DEFINE_SPINLOCK(namecnt_lock);
+
+static void get_unused_name(char *buf)
+{
+
+ spin_lock(&namecnt_lock);
+ memset(buf, 0, 20);
+ snprintf(buf, 20, "node%d", namecnt++);
+ spin_unlock(&namecnt_lock);
+}
+
+#ifndef CONFIG_CONTAINER_NS
+static int ns_container_subsys_idx = -1;
+#else
+extern int ns_container_subsys_idx;
+#endif
+
+static inline struct container *find_child_container(struct container *parent,
+      char *name)
+{
+ struct dentry *d = container_get_dentry(parent->dentry, name);
+ return d ? d->d_fsdata : NULL;
+}
+
+#define NS_CONT_MODE (S_IFDIR | S_IRUGO | S_IXUGO | S_IWUSR)
+int container_switch(struct task_struct *tsk)
+{
+ int h;
+ struct container *cur_cont, *new_cont;
+ char path[20];
+ struct qstr name;
+ struct dentry *dentry;
+ int ret;
+ char *pathbuf = NULL;
```

```
+ char buffer[20];
+
+ /* check if nsproxy subsys is registered */
+ if (ns_container_subsys_idx == -1)
+  return 0;
+
+ printk(KERN_NOTICE "%s: ns_container subsys registered\n", __FUNCTION__);
+ /* check if nsproxy subsys is mounted in some hierarchy */
+ rcu_read_lock();
+ h = rcu_dereference(subsys[ns_container_subsys_idx]->hierarchy);
+ rcu_read_unlock();
+ if (h == 0) {
+  /* do we mount the nsproxy subsys, or just skip
+   * creating a container?  I think we just skip
+   * it.
+   */
+  printk(KERN_NOTICE "%s: hierarchy (for idx %d) was 0\n", __FUNCTION__,
+   ns_container_subsys_idx);
+   return 0;
+ }
+ printk(KERN_NOTICE "%s: ns_container subsys not mounted.\n", __FUNCTION__);
+ cur_cont = tsk->container[h];
+ get_unused_name(path);
+
+ name.name = path;
+ name.len = strlen(name.name);
+ dentry = d_alloc(cur_cont->dentry, &name);
+ if (IS_ERR(dentry)) {
+  printk(KERN_NOTICE "%s: couldn't get dentry for current container\n", __FUNCTION__);
+  return PTR_ERR(dentry);
+ }
+ printk(KERN_NOTICE "%s: got dentry for current container\n", __FUNCTION__);
+ ret = vfs_mkdir(cur_cont->dentry->d_inode, dentry, NS_CONT_MODE);
+ dput(dentry);
+ if (ret) {
+  printk(KERN_NOTICE "%s: Couldn't make new directory (%d)\n", __FUNCTION__, ret);
+  return ret;
+ }
+ printk(KERN_NOTICE "%s: made new dir\n", __FUNCTION__);
+
+ new_cont = find_child_container(cur_cont, path);
+ if (!new_cont) {
+  printk(KERN_NOTICE "%s: Couldn't find new container\n", __FUNCTION__);
+  return -ENOMEM;
+ }
+ printk(KERN_NOTICE "%s: found new container\n", __FUNCTION__);
+
+ snprintf(buffer, 20, "%lu\n", (unsigned long)tsk->pid);
```

```
+ mutex_lock(&manage_mutex);
+ ret = attach_task(new_cont, buffer, &pathbuf);
+ mutex_unlock(&manage_mutex);
+
+ /* I dunno - do I *need* to call container_release_agent? */
+ // container_release_agent(cont->hierarchy, pathbuf);
+
+ return ret;
+}
+
+/*
+ * XXX - should probably be called with a lock
+ * isn't right now
+ */
+int task_container_is_ancestor(struct task_struct *tsk, int subsys,
+      struct container *cont)
+{
+ struct container *q = tsk->container[subsys];
+
+ if (!q)
+  return 1;
+
+ do {
+  cont = cont->parent;
+ } while (cont != cont->top_container && cont != q);
+
+ if (cont == q)
+  return 1;
+ return 0;
+}
+

 /**
  * container_init_early - initialize containers at system boot
diff --git a/kernel/fork.c b/kernel/fork.c
index 984fe31..cf06ff5 100644
--- a/kernel/fork.c
+++ b/kernel/fork.c
@@ -1661,6 +1661,10 @@ asmlinkage long sys_unshare(unsigned lon
    err = -ENOMEM;
    goto bad_unshare_cleanup_ipc;
   }
+
+  err = container_switch(current);
+  if (err)
+   goto bad_unshare_cleanup_dupns;
  }
```

```
  if (new_fs || new_ns || new_mm || new_fd || new_ulist ||
@@ -1715,6 +1719,7 @@ asmlinkage long sys_unshare(unsigned lon
   task_unlock(current);
  }

+bad_unshare_cleanup_dupns:
  if (new_nsproxy)
   put_nsproxy(new_nsproxy);

diff --git a/kernel/ns_container.c b/kernel/ns_container.c
new file mode 100644
index 0000000..fd2e92b
--- /dev/null
+++ b/kernel/ns_container.c
@@ -0,0 +1,117 @@
+/*
+ * ns_container.c - namespace container subsystem
+ *
+ * Copyright IBM, 2006
+ */
+
+#include <linux/module.h>
+#include <linux/container.h>
+#include <linux/fs.h>
+
+int ns_container_subsys_idx = -1;
+
+struct nscont {
+ struct container_subsys_state css;
+ spinlock_t lock;
+};
+
+static struct container_subsys ns_subsys;
+
+static inline struct nscont *container_nscont(struct container *cont)
+{
+ return container_of(container_subsys_state(cont, &ns_subsys),
+       struct nscont, css);
+}
+
+/*
+ * Rules:
+ *   1. you can only enter a container which is a child of your current
+ * container
+ *   2. you can only place another process into a container if
+ * a. you have CAP_SYS_ADMIN
+ * b. your container is an ancestor of tsk's destination container
+ *   (hence either you are in the same container as tsk, or in an
```

```
+ *    ancestor container thereof)
+ */
+int ns_can_attach(struct container_subsys *ss,
+        struct container *cont, struct task_struct *tsk)
+{
+ struct container *c;
+
+ if (current != tsk) {
+  int hierarchy;
+  if (!capable(CAP_SYS_ADMIN))
+   return -EPERM;
+
+  rcu_read_lock();
+  hierarchy = rcu_dereference(ns_subsys.hierarchy);
+  rcu_read_unlock();
+
+  if (!task_container_is_ancestor(current, hierarchy, cont))
+   return -EPERM;
+ }
+
+ if (atomic_read(&cont->count) != 0)
+  return -EPERM;
+
+ c = task_container(tsk, &ns_subsys);
+ if (c && c != cont->parent)
+  return -EPERM;
+
+ return 0;
+}
+
+/*
+ * Rules: you can only create a container if
+ * 1. you are capable(CAP_SYS_ADMIN)
+ * 2. the target container is a decendent of your own container
+ */
+static int ns_create(struct container_subsys *ss, struct container *cont)
+{
+ struct nscont *ns;
+ int hierarchy;
+
+ if (!capable(CAP_SYS_ADMIN))
+  return -EPERM;
+ rcu_read_lock();
+ hierarchy = rcu_dereference(ns_subsys.hierarchy);
+ rcu_read_unlock();
+ if (hierarchy && !task_container_is_ancestor(current, hierarchy, cont))
+  return -EPERM;
+
```

```
+ ns = kzalloc(sizeof(*ns), GFP_KERNEL);
+ if (!ns) return -ENOMEM;
+ spin_lock_init(&ns->lock);
+ cont->subsys[ns_subsys.subsys_id] = &ns->css;
+ return 0;
+}
+
+static void ns_destroy(struct container_subsys *ss,
+       struct container *cont)
+{
+ struct nscont *ns = container_nscont(cont);
+ kfree(ns);
+}
+
+static struct container_subsys ns_subsys = {
+ .name = "ns_container",
+ .create = ns_create,
+ .destroy  = ns_destroy,
+ .can_attach = ns_can_attach,
+ //.attach = ns_attach,
+ //.post_attach = ns_post_attach,
+ //.populate = ns_populate,
+ .subsys_id = -1,
+};
+
+int __init ns_init(void)
+{
+ int ret;
+
+ ret = container_register_subsys(&ns_subsys);
+ ns_container_subsys_idx = ns_subsys.subsys_id;
+
+ return ret < 0 ? ret : 0;
+}
+
+module_init(ns_init)
diff --git a/kernel/nsproxy.c b/kernel/nsproxy.c
index f5b9ee6..c4f41de 100644
--- a/kernel/nsproxy.c
+++ b/kernel/nsproxy.c
@@ -20,6 +20,7 @@
 #include <linux/mnt_namespace.h>
 #include <linux/utsname.h>
 #include <linux/pid_namespace.h>
+#include <linux/container.h>

 struct nsproxy init_nsproxy = INIT_NSPROXY(init_nsproxy);
```

```
@@ -115,11 +116,18 @@ int copy_namespaces(int flags, struct ta
  err = copy_pid_ns(flags, tsk);
  if (err)
   goto out_pid;
+
+ err = container_switch(tsk);
+ if (err)
+  goto out_container;

 out:
  put_nsproxy(old_ns);
  return err;

+out_container:
+ if (new_ns->pid_ns)
+  put_pid_ns(new_ns->pid_ns);
 out_pid:
  if (new_ns->ipc_ns)
   put_ipc_ns(new_ns->ipc_ns);
--
1.1.6
```

_____

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