
Subject: [PATCH 1/1] containers: implement nsproxy containers subsystem

Posted by [serue](#) on Mon, 04 Jun 2007 18:50:34 GMT

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

>From 190ea72d213393dd1440643b2b87b5b2128dff87 Mon Sep 17 00:00:00 2001

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

Date: Mon, 4 Jun 2007 14:18:52 -0400

Subject: [PATCH 1/1] containers: implement nsproxy containers subsystem

When a task enters a new namespace via a clone() or unshare(), a new container is created and the task moves into it. This enables tracking - and applying resource controls - to virtual servers, resource jobs, often logins (using per-user namespaces), and in general any application of per-process namespaces.

This version uses an ida to generate a unique id for auto-created containers, i.e. those created through container_clone(). After a container is created, it can be renamed to a more useful name if so desired. The original id isn't released until the container is destroyed. (This could be fixed at container_rename() if it was deemed desirable)

There is just one ida for all containers, because an ida per container to track ids used for child containers would take a great deal of space, and probably isn't very useful.

Changelog:

Use an ida to generate unique ids for auto-created containers.

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

```
include/linux/container.h      | 2 +
include/linux/container_subsys.h | 6 ++
include/linux/nsproxy.h        | 7 +++
init/Kconfig                   | 9 ++++
kernel/Makefile                 | 1 +
kernel/container.c              | 52 ++++++++-----
kernel/ns_container.c           | 99 ++++++++++++++++++++++++++++++++++++++
kernel/nsproxy.c                | 16 ++++++
8 files changed, 183 insertions(+), 9 deletions(-)
create mode 100644 kernel/ns_container.c
```

diff --git a/include/linux/container.h b/include/linux/container.h

index 37c0bdf..2f8ee47 100644

--- a/include/linux/container.h

+++ b/include/linux/container.h

@@ -87,6 +87,8 @@ struct container {

/* Private pointers for each registered subsystem */

```

struct container_subsys_state *subsys[CONTAINER_SUBSYS_COUNT];

+ int unique_id; /* usually -1, set to >=0 in container_clone() */
+
+ struct containerfs_root *root;
+ struct container *top_container;

diff --git a/include/linux/container_subsys.h b/include/linux/container_subsys.h
index 8fea7cf..9861751 100644
--- a/include/linux/container_subsys.h
+++ b/include/linux/container_subsys.h
@@ -24,3 +24,9 @@ SUBSYS(debug)
#endif

/* */
+
+ #ifdef CONFIG_CONTAINER_NS
+ SUBSYS(ns)
+ #endif
+
+ /* */
diff --git a/include/linux/nsproxy.h b/include/linux/nsproxy.h
index 189e0dc..8be975b 100644
--- a/include/linux/nsproxy.h
+++ b/include/linux/nsproxy.h
@@ -54,4 +54,11 @@ static inline void exit_task_namespaces(struct task_struct *p)
    put_nsproxy(ns);
}
}
+
+ #ifdef CONFIG_CONTAINER_NS
+ int ns_container_clone(struct task_struct *tsk);
+ #else
+ static inline int ns_container_clone(struct task_struct *tsk) { return 0; }
+ #endif
+
+ #endif
diff --git a/init/Kconfig b/init/Kconfig
index 5861ad9..d79c505 100644
--- a/init/Kconfig
+++ b/init/Kconfig
@@ -355,6 +355,15 @@ 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 PROC_PID_CPUSET
    bool "Include legacy /proc/<pid>/cpuset file"
    depends on CPUSETS
diff --git a/kernel/Makefile b/kernel/Makefile
index f73b3d3..34f2345 100644
--- a/kernel/Makefile
+++ b/kernel/Makefile
@@ -40,6 +40,7 @@ obj-$(CONFIG_CONTAINERS) += container.o
obj-$(CONFIG_CONTAINER_DEBUG) += container_debug.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 6f80487..0d46d66 100644
--- a/kernel/container.c
+++ b/kernel/container.c
@@ -55,6 +55,7 @@
#include <linux/time.h>
#include <linux/backing-dev.h>
#include <linux/sort.h>
+#include <linux/idr.h>

#include <asm/uaccess.h>
#include <asm/atomic.h>
@@ -64,6 +65,8 @@

static DEFINE_MUTEX(container_mutex);

+static struct ida container_ida;
+
/* Generate an array of container subsystem pointers */
#define SUBSYS(_x) &_x ## _subsys,

@@ -1818,6 +1821,7 @@ static long container_create(struct container *parent, struct dentry
*dentry,
    INIT_LIST_HEAD(&cont->children);
    INIT_LIST_HEAD(&cont->css_groups);
    INIT_LIST_HEAD(&cont->release_list);
+    cont->unique_id = -1;

```

```

cont->parent = parent;
cont->root = parent->root;
@@ -1941,6 +1945,9 @@ static int container_rmdir(struct inode *unused_dir, struct dentry
*dentry)
    dput(d);
    root->number_of_containers--;

+ if (cont->unique_id != -1)
+ ida_remove(&container_ida, cont->unique_id);
+
    if (!list_empty(&cont->release_list))
        list_del(&cont->release_list);
    set_bit(CONT_RELEASABLE, &parent->flags);
@@ -2040,6 +2047,8 @@ int __init container_init_early(void)
    if (ss->early_init)
        container_init_subsys(ss);
}
+
+ ida_init(&container_ida);
    return 0;
}

@@ -2302,12 +2311,6 @@ void container_exit(struct task_struct *tsk, int run_callbacks)
    put_css_group_taskexit(cg);
}

-static atomic_t namecnt;
-static void get_unused_name(char *buf)
-{
- sprintf(buf, "node%d", atomic_inc_return(&namecnt));
-}
-
/**
 * container_clone - duplicate the current container in the hierarchy
 * that the given subsystem is attached to, and move this task into
@@ -2322,6 +2325,7 @@ int container_clone(struct task_struct *tsk, struct container_subsys
*subsys)
    struct inode *inode;
    struct css_group *cg;
    struct containerfs_root *root;
+ int new_id;

    /* We shouldn't be called by an unregistered subsystem */
    BUG_ON(!subsys->active);
@@ -2340,6 +2344,19 @@ int container_clone(struct task_struct *tsk, struct container_subsys
*subsys)
}

```

```

    cg = tsk->containers;
    parent = task_container(tsk, subsys->subsys_id);
+
+ ret = ida_get_new(&container_ida, &new_id);
+ if (ret == -EAGAIN) {
+   if (!ida_pre_get(&container_ida, GFP_KERNEL))
+     return -ENOMEM;
+   ret = ida_get_new(&container_ida, &new_id);
+ }
+ if (ret) {
+   ret = -EBUSY;
+   goto out_no_name;
+ }
+ sprintf(nodename, "node%d", new_id);
+
+ /* Pin the hierarchy */
+ atomic_inc(&parent->root->sb->s_active);

@@ -2347,8 +2364,8 @@ int container_clone(struct task_struct *tsk, struct container_subsys
*subsys)
    get_css_group(cg);
    mutex_unlock(&container_mutex);

+
+ /* Now do the VFS work to create a container */
- get_unused_name(nodename);
inode = parent->dentry->d_inode;

/* Hold the parent directory mutex across this operation to
@@ -2403,6 +2420,10 @@ int container_clone(struct task_struct *tsk, struct container_subsys
*subsys)

/* All seems fine. Finish by moving the task into the new container */
ret = attach_task(child, tsk);
+
+ if (!ret)
+   child->unique_id = new_id;
+
+ mutex_unlock(&container_mutex);

out_release:
@@ -2410,19 +2431,32 @@ int container_clone(struct task_struct *tsk, struct container_subsys
*subsys)

    mutex_lock(&container_mutex);
    put_css_group(cg);
+
+ out_no_name:

```

```

mutex_unlock(&container_mutex);
deactivate_super(parent->root->sb);
return ret;
}

```

```

-/* See if "cont" is a descendant of the current task's container in
- * the appropriate hierarchy */
+/*
+ * See if "cont" is a descendant of the current task's container in
+ * the appropriate hierarchy
+ *
+ * If we are sending in dummytop, then presumably we are creating
+ * the top container in the subsystem.
+ *
+ * Called only by the ns (nsproxy) container.
+ */

```

```

int container_is_descendant(const struct container *cont)
{
    int ret;
    struct container *target;
    int subsys_id;
+
+ if (cont == dummytop)
+     return 1;
+
    get_first_subsys(cont, NULL, &subsys_id);
    target = task_container(current, subsys_id);
    while (cont != target && cont != cont->top_container) {
diff --git a/kernel/ns_container.c b/kernel/ns_container.c
new file mode 100644
index 0000000..3465716
--- /dev/null
+++ b/kernel/ns_container.c
@@ -0,0 +1,99 @@
+/*
+ * ns_container.c - namespace container subsystem
+ *
+ * Copyright 2006, 2007 IBM Corp
+ */
+
+#include <linux/module.h>
+#include <linux/container.h>
+#include <linux/fs.h>
+
+struct ns_container {
+    struct container_subsys_state css;
+    spinlock_t lock;

```

```

+};
+
+struct container_subsys ns_subsys;
+
+static inline struct ns_container *container_to_ns(
+ struct container *container)
+{
+ return container_of(container_subsys_state(container, ns_subsys_id),
+ struct ns_container, css);
+}
+
+int ns_container_clone(struct task_struct *task)
+{
+ return container_clone(task, &ns_subsys);
+}
+
+/*
+ * 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 task's destination container
+ *       (hence either you are in the same container as task, or in an
+ *       ancestor container thereof)
+ */
+static int ns_can_attach(struct container_subsys *ss,
+ struct container *new_container, struct task_struct *task)
+{
+ struct container *orig;
+
+ if (current != task) {
+ if (!capable(CAP_SYS_ADMIN))
+ return -EPERM;
+
+ if (!container_is_descendant(new_container))
+ return -EPERM;
+ }
+
+ if (atomic_read(&new_container->count) != 0)
+ return -EPERM;
+
+ orig = task_container(task, ns_subsys_id);
+ if (orig && orig != new_container->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 descendant of your own container
+ */
+static int ns_create(struct container_subsys *ss, struct container *container)
+{
+ struct ns_container *ns_container;
+
+ if (!capable(CAP_SYS_ADMIN))
+ return -EPERM;
+ if (!container_is_descendant(container))
+ return -EPERM;
+
+ ns_container = kzalloc(sizeof(*ns_container), GFP_KERNEL);
+ if (!ns_container) return -ENOMEM;
+ spin_lock_init(&ns_container->lock);
+ container->subsys[ns_subsys.subsys_id] = &ns_container->css;
+ return 0;
+}
+
+static void ns_destroy(struct container_subsys *ss,
+ struct container *container)
+{
+ struct ns_container *ns_container;
+
+ ns_container = container_to_ns(container);
+ kfree(ns_container);
+}
+
+struct container_subsys ns_subsys = {
+ .name = "ns",
+ .can_attach = ns_can_attach,
+ .create = ns_create,
+ .destroy = ns_destroy,
+ .subsys_id = ns_subsys_id,
+};
diff --git a/kernel/nsproxy.c b/kernel/nsproxy.c
index 1bc4b55..afce808 100644
--- a/kernel/nsproxy.c
+++ b/kernel/nsproxy.c
@@ -124,7 +124,14 @@ int copy_namespaces(int flags, struct task_struct *tsk)
     goto out;
 }

+ err = ns_container_clone(tsk);
+ if (err) {

```

```

+ put_nsproxy(new_ns);
+ goto out;
+ }
+
+ tsk->nsproxy = new_ns;
+
+ out:
+ put_nsproxy(old_ns);
+ return err;
@@ -177,6 +184,15 @@ int unshare_nsproxy_namespaces(unsigned long unshare_flags,
+ if (IS_ERR(*new_nsp)) {
+ err = PTR_ERR(*new_nsp);
+ put_nsproxy(old_ns);
+ goto out;
+ }
+
+ err = ns_container_clone(current);
+ if (err) {
+ put_nsproxy(*new_nsp);
+ put_nsproxy(old_ns);
+ }
+
+out:
+ return err;
+ }
+
+
1.5.1.1.GIT

```
