Subject: [PATCH 1/2] containers: implement subsys->post_clone()
Posted by serue on Wed, 13 Jun 2007 22:56:56 GMT
View Forum Message <> Reply to Message

>From aed04d506feac3a71896713a5d5aeded839fdd9e Mon Sep 17 00:00:00 2001
From: Serge E. Hallyn <serue@us.ibm.com>
Date: Tue, 12 Jun 2007 15:06:38 -0400
Subject: [PATCH 1/2] containers: implement subsys->post_clone()

container_clone() in one step creates a new container and moves
the current task into it. Since cpusets do not automatically
fill in the allowed cpus and mems, and do not allow a task to
be attached without these filled in, composing the ns subsystem,
which uses container_clone(), and the cpuset subsystem, results
in sys_unshare() (and clone(CLONE_NEWNS)) always being denied.

To allow the two subsystems to be meaningfully composed, implement
subsystem->post_clone(), called from container_clone() after
creating the new container.

Only the cpuset_post_clone() is currently implemented. If any
sibling containers have exclusive cpus or mems, then the cpus
and mems are not filled in for the new container, meaning that
unshare/clone(CLONE_NEWNS) will be denied. However so long as
no siblings have exclusive cpus or mems, the new container's
cpus and mems are inherited from the parent container.

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

 Documentation/containers.txt | 7 +++++++
 include/linux/container.h | 1 +
 kernel/container.c | 7 +++++++
 kernel/cpuset.c | 37 ++
 4 files changed, 52 insertions(+), 0 deletions(-)

diff --git a/Documentation/containers.txt b/Documentation/containers.txt
index ae159b9..9fdb808 100644
--- a/Documentation/containers.txt
+++ b/Documentation/containers.txt
@@ -514,6 +514,13 @@ include/linux/container.h for details). Note that although this
 method can return an error code, the error code is currently not
 always handled well.

+void post_clone(struct container_subsys *ss, struct container *cont)
+
+Called at the end of container_clone() to do any paramater
+initialization which might be required before a task could attach. For
+example in cpusets, no task may attach before 'cpus' and 'mems' are set

```
+up.
+
 void bind(struct container_subsys *ss, struct container *root)
 LL=callback_mutex

diff --git a/include/linux/container.h b/include/linux/container.h
index 3fb0b0f..cb0ff7b 100644
--- a/include/linux/container.h
+++ b/include/linux/container.h
@@ -213,6 +213,7 @@ struct container_subsys {
  void (*exit)(struct container_subsys *ss, struct task_struct *task);
  int (*populate)(struct container_subsys *ss,
    struct container *cont);
+ void (*post_clone)(struct container_subsys *ss, struct container *cont);
  void (*bind)(struct container_subsys *ss, struct container *root);
  int subsys_id;
  int active;
diff --git a/kernel/container.c b/kernel/container.c
index 6828af5..28badda 100644
--- a/kernel/container.c
+++ b/kernel/container.c
@@ -2315,6 +2315,7 @@ int container_clone(struct task_struct *tsk, struct container_subsys
*subsys)
  struct inode *inode;
  struct css_group *cg;
  struct containerfs_root *root;
+ struct container_subsys *ss;

  /* We shouldn't be called by an unregistered subsystem */
  BUG_ON(!subsys->active);
@@ -2394,6 +2395,12 @@ int container_clone(struct task_struct *tsk, struct container_subsys
*subsys)
   goto again;
  }

+ /* do any required auto-setup */
+ for_each_subsys(root, ss) {
+  if (ss->post_clone)
+   ss->post_clone(ss, child);
+ }
+
  /* All seems fine. Finish by moving the task into the new container */
  ret = attach_task(child, tsk);
  mutex_unlock(&container_mutex);
diff --git a/kernel/cpuset.c b/kernel/cpuset.c
index 0f9ce7d..ecefb1d 100644
--- a/kernel/cpuset.c
+++ b/kernel/cpuset.c
```

```
@@ -1190,6 +1190,42 @@ int cpuset_populate(struct container_subsys *ss, struct container
*cont)
 }

 /*
+ * post_clone() is called at the end of container_clone().
+ * 'container' was just created automatically as a result of
+ * a container_clone(), and the current task is about to
+ * be moved into 'container'.
+ *
+ * Currently we refuse to set up the container - thereby
+ * refusing the task to be entered, and as a result refusing
+ * the sys_unshare() or clone() which initiated it - if any
+ * sibling cpusets have exclusive cpus or mem.
+ *
+ * If this becomes a problem for some users who wish to
+ * allow that scenario, then cpuset_post_clone() could be
+ * changed to grant parent->cpus_allowed-sibling_cpus_exclusive
+ * (and likewise for mems) to the new container.
+ */
+void cpuset_post_clone(struct container_subsys *ss,
+  struct container *container)
+{
+ struct container *parent, *child;
+ struct cpuset *cs, *parent_cs;
+
+ parent = container->parent;
+ list_for_each_entry(child, &parent->children, sibling) {
+  cs = container_cs(child);
+  if (is_mem_exclusive(cs) || is_cpu_exclusive(cs))
+   return;
+ }
+ cs = container_cs(container);
+ parent_cs = container_cs(parent);
+
+ cs->mems_allowed = parent_cs->mems_allowed;
+ cs->cpus_allowed = parent_cs->cpus_allowed;
+ return;
+}
+
+/*
  * cpuset_create - create a cpuset
  * parent: cpuset that will be parent of the new cpuset.
  * name:  name of the new cpuset. Will be strcpy'ed.
@@ -1249,6 +1285,7 @@ struct container_subsys cpuset_subsys = {
  .can_attach = cpuset_can_attach,
  .attach = cpuset_attach,
  .populate = cpuset_populate,
```

```
+ .post_clone = cpuset_post_clone,
  .subsys_id = cpuset_subsys_id,
  .early_init = 1,
 };
--
1.5.1.1.GIT
```