Subject: [PATCH 1/2][RFC] containers: improve automatic container naming
Posted by serue on Fri, 01 Jun 2007 21:48:09 GMT
View Forum Message <> Reply to Message

This compiles and boots, but is not intended for inclusion in -mm (yet),
just as an RFC for the naming scheme to fix the bug Andrew pointed out.

Seem ok overall?

thanks,
-serge

>From 8e9b972f7482415777e982d3bc9a0d55cbaf862b Mon Sep 17 00:00:00 2001
From: Serge E. Hallyn <serue@us.ibm.com>
Date: Fri, 1 Jun 2007 15:32:15 -0400
Subject: [PATCH 1/2] containers: improve automatic container naming

The automatic naming of containers created using container_clone()
is currently broken (not protected from wraparound) and inconvenient.

Add a per-container counter for use in naming children of the container.
Before two unshares in a row by one process, and a third in another,
would result in

 /node1/node2
 /node3

The current scheme should result in

 /node1/node1
 /node2

Also, keep a hash table populated with used names, to protect
against counter wrap-around.

Signed-off-by: Serge E. Hallyn <serue@us.ibm.com>
---
 include/linux/container.h | 8 +++
 kernel/container.c | 116 ++--
 2 files changed, 120 insertions(+), 4 deletions(-)

diff --git a/include/linux/container.h b/include/linux/container.h
index 37c0bdf..61c750d 100644
--- a/include/linux/container.h
+++ b/include/linux/container.h
@@ -75,6 +75,14 @@ struct container {
  atomic_t count;

```
  /*
+  * these are to support automatic naming of automatically
+  * created containers
+  */
+ int auto_cnt_set;  /* 1 if thie container is using up a auto_cnt */
+ unsigned long auto_cnt; /* the auto_cnt it's using up */
+ unsigned long next_childcnt; /* next avail cnt for child containers */
+
+ /*
   * We link our 'sibling' struct into our parent's 'children'.
   * Our children link their 'sibling' into our 'children'.
   */
diff --git a/kernel/container.c b/kernel/container.c
index 6f80487..2b2f6f2 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/hash.h>

 #include <asm/uaccess.h>
 #include <asm/atomic.h>
@@ -1819,6 +1820,9 @@ static long container_create(struct container *parent, struct dentry
*dentry,
  INIT_LIST_HEAD(&cont->css_groups);
  INIT_LIST_HEAD(&cont->release_list);

+ cont->auto_cnt_set = 0;
+ cont->next_childcnt = 0;
+
  cont->parent = parent;
  cont->root = parent->root;
  cont->top_container = parent->top_container;
@@ -1894,6 +1898,71 @@ static inline int container_has_css_refs(struct container *cont)
  return 0;
 }

+#define cnthash_shift 3
+#define childcnt_hash(ptr, nr) hash_long(nr+(unsigned long)ptr, cnthash_shift)
+static DEFINE_SPINLOCK(childcntlock);
+
+struct container_child_hash_elem {
+ struct container *parent;
+ unsigned long cnt;
+ struct hlist_node list;
+};
```

```
+
+static struct hlist_head cntnum_hash_list[1 << cnthash_shift];
+
+static void init_cntnum_hash(void)
+{
+ int i;
+
+ for (i=0; i<1 << cnthash_shift; i++)
+  INIT_HLIST_HEAD(&cntnum_hash_list[i]);
+}
+
+static inline struct container_child_hash_elem *find_cntnum_hash(
+  struct container *parent, unsigned long count)
+{
+ struct container_child_hash_elem *he;
+ struct hlist_head *h;
+ struct hlist_node *tmp;
+
+ h = &cntnum_hash_list[childcnt_hash(parent, count)];
+ hlist_for_each_entry(he, tmp, h, list) {
+  if (he->parent == parent && he->cnt == count)
+   return he;
+ }
+ return NULL;
+}
+
+#define UNSET_HASH 0
+#define SET_HASH 1
+static inline int set_cntnum_hash(struct container *parent,
+        unsigned long count, int set)
+{
+ struct container_child_hash_elem *he;
+ int idx;
+
+ he = find_cntnum_hash(parent, count);
+ if (he) {
+  if (!set) {
+   hlist_del(&he->list);
+   kfree(he);
+   return 0;
+  }
+  return -ENOENT;
+ }
+ if (!set)
+  return -ENOENT;
+ he = kmalloc(sizeof(*he), GFP_KERNEL);
+ if (!he)
+  return -ENOMEM;
```

```
+ he->parent = parent;
+ he->cnt = count;
+ INIT_HLIST_NODE(&he->list);
+ idx = childcnt_hash(parent, count);
+ hlist_add_head(&he->list, &cntnum_hash_list[idx]);
+ return 0;
+}
+
 static int container_rmdir(struct inode *unused_dir, struct dentry *dentry)
 {
  struct container *cont = dentry->d_fsdata;
@@ -1941,6 +2010,15 @@ static int container_rmdir(struct inode *unused_dir, struct dentry
*dentry)
  dput(d);
  root->number_of_containers--;

+ if (cont->auto_cnt_set) {
+  spin_lock(&childcntlock);
+  if (set_cntnum_hash(parent, cont->auto_cnt, UNSET_HASH))
+   printk(KERN_NOTICE "%s: could not unset %lu\n",
+    __FUNCTION__, cont->auto_cnt);
+  cont->auto_cnt_set = 0;
+  spin_unlock(&childcntlock);
+ }
+
  if (!list_empty(&cont->release_list))
   list_del(&cont->release_list);
  set_bit(CONT_RELEASABLE, &parent->flags);
@@ -2063,6 +2141,8 @@ int __init container_init(void)
  err = register_filesystem(&container_fs_type);
  if (err < 0)
   goto out;
+
+ init_cntnum_hash();

  entry = create_proc_entry("containers", 0, NULL);
  if (entry)
@@ -2302,10 +2382,31 @@ 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)
+/*
+ * the while (find_cntnum_hash) loop will only be hit after
+ * we wrap around on next_childnct, which should never happen.
+ */
+static int get_unused_name(struct container *parent, unsigned long *out_cnt,
```

```
+  char *buf)
 {
- sprintf(buf, "node%d", atomic_inc_return(&namecnt));
+ unsigned long cnt;
+ int err = 0;
+
+ spin_lock(&childcntlock);
+ cnt = parent->next_childcnt;
+ while (find_cntnum_hash(parent, cnt))
+  cnt++;
+ err = set_cntnum_hash(parent, cnt, SET_HASH);
+ if (err)
+  goto out_unlock;
+
+ sprintf(buf, "node%lu", cnt);
+ parent->next_childcnt = cnt+1;
+ *out_cnt = cnt;
+
+out_unlock:
+ spin_unlock(&childcntlock);
+ return err;
 }

 /**
@@ -2318,6 +2419,7 @@ int container_clone(struct task_struct *tsk, struct container_subsys
*subsys)
  struct dentry *dentry;
  int ret = 0;
  char nodename[32];
+ unsigned long auto_cntnum;
  struct container *parent, *child;
  struct inode *inode;
  struct css_group *cg;
@@ -2348,7 +2450,10 @@ int container_clone(struct task_struct *tsk, struct container_subsys
*subsys)
  mutex_unlock(&container_mutex);

  /* Now do the VFS work to create a container */
- get_unused_name(nodename);
+ ret = get_unused_name(parent, &auto_cntnum, nodename);
+ if (ret)
+  goto out_container_mutex;
+
  inode = parent->dentry->d_inode;

  /* Hold the parent directory mutex across this operation to
@@ -2380,6 +2485,8 @@ int container_clone(struct task_struct *tsk, struct container_subsys
*subsys)
```

```
   ret = -ENOMEM;
   goto out_release;
  }
+ child->auto_cnt_set = 1;
+ child->auto_cnt = auto_cntnum;

  /* The container now exists. Retake container_mutex and check
   * that we're still in the same state that we thought we
@@ -2408,6 +2515,7 @@ int container_clone(struct task_struct *tsk, struct container_subsys
*subsys)
  out_release:
  mutex_unlock(&inode->i_mutex);

+ out_container_mutex:
  mutex_lock(&container_mutex);
  put_css_group(cg);
  mutex_unlock(&container_mutex);
--
1.5.1.1.GIT
```