Subject: [RFC] [PATCH 2/3] container: create containerfs
Posted by serue on Wed, 20 Dec 2006 06:02:13 GMT
View Forum Message <> Reply to Message

From: Serge E. Hallyn <serue@us.ibm.com>
Subject: [RFC] [PATCH 2/3] container: create containerfs

Create containerfs as a view of the hierarchy of containers.

Signed-off-by: Serge E. Hallyn <serue@us.ibm.com>
---
 include/linux/container.h | 1
 kernel/container.c | 175 ++
 2 files changed, 176 insertions(+), 0 deletions(-)

```
diff --git a/include/linux/container.h b/include/linux/container.h
index fcd85f3..c224a53 100644
--- a/include/linux/container.h
+++ b/include/linux/container.h
@@ -13,6 +13,7 @@ struct container {
  struct nsproxy *nsproxy;
  struct list_head children;
  struct list_head peers;
+ struct dentry *dentry;
  struct kref ref;
 };
 extern struct container init_container;
diff --git a/kernel/container.c b/kernel/container.c
index ed3269f..6206e72 100644
--- a/kernel/container.c
+++ b/kernel/container.c
@@ -17,10 +17,18 @@ #include <linux/module.h>
 #include <linux/version.h>
 #include <linux/container.h>
 #include <linux/init_task.h>
+#include <linux/fsnotify.h>
+#include <linux/fs.h>
+#include <linux/mount.h>
+
+#define CONTAINERFS_MAGIC 0xb6663caf

 struct nsproxy;

 struct container init_container = INIT_CONTAINER(init_container);
+static struct vfsmount *containerfs_mount;
+static void containerfs_remove(struct dentry *dentry);
+static struct dentry *containerfs_create_dir(struct container *container);
```

```
 /*
  * free_container: called from rcu_call when all references
@@ -32,6 +40,7 @@ static void free_container(struct kref *
 {
  struct container *c = container_of(ref, struct container, ref);

+ containerfs_remove(c->dentry);
  if (c->parent != c)
   kfree(c->name);
  if (!list_empty(&c->peers))
@@ -103,6 +112,172 @@ struct container *new_container(struct c
  INIT_LIST_HEAD(&c->children);
  c->nsproxy = nsproxy;
  kref_init(&c->ref);
+ containerfs_create_dir(c);

  return c;
 }
+
+static struct inode *containerfs_get_inode(struct super_block *sb, int mode, dev_t dev)
+{
+ struct inode *inode = new_inode(sb);
+
+ if (inode) {
+  inode->i_mode = mode;
+  inode->i_uid = 0;
+  inode->i_gid = 0;
+  inode->i_blocks = 0;
+  inode->i_atime = inode->i_mtime = inode->i_ctime = CURRENT_TIME;
+  switch (mode & S_IFMT) {
+  default:
+   printk("%s: whoa: non-dirs for containerfs should not exist\n",
+       __FUNCTION__);
+  case S_IFDIR:
+   inode->i_op = &simple_dir_inode_operations;
+   inode->i_fop = &simple_dir_operations;
+
+   /* directory inodes start off with i_nlink == 2
+    * (for "." entry) */
+   inc_nlink(inode);
+   break;
+  }
+ }
+ return inode;
+}
+
+static int containerfs_mknod(struct inode *dir, struct dentry *dentry,
+    int mode, dev_t dev)
```

```
+{
+ struct inode *inode;
+ int error = -EPERM;
+
+ if (dentry->d_inode)
+  return -EEXIST;
+
+ inode = containerfs_get_inode(dir->i_sb, mode, dev);
+ if (inode) {
+  d_instantiate(dentry, inode);
+  dget(dentry);
+  error = 0;
+ }
+ return error;
+}
+
+#define IS_ROOT_CONTAINER(x) (x->parent == x)
+
+static int containerfs_mkdir(struct inode *dir, struct dentry *dentry, int mode)
+{
+ int res;
+
+ mode = (mode & (S_IRWXUGO | S_ISVTX)) | S_IFDIR;
+ res = containerfs_mknod(dir, dentry, mode, 0);
+ if (!res) {
+  inc_nlink(dir);
+  fsnotify_mkdir(dir, dentry);
+ }
+ return res;
+}
+
+static struct dentry *containerfs_create_dir(struct container *container)
+{
+ struct dentry *dentry = NULL;
+ struct dentry *parent;
+ int error;
+
+ if (IS_ROOT_CONTAINER(container))
+  parent = containerfs_mount->mnt_root;
+ else
+  parent = container->parent->dentry;
+
+ mutex_lock(&parent->d_inode->i_mutex);
+ dentry = lookup_one_len(container->name, parent,
+     strlen(container->name));
+ if (!IS_ERR(dentry)) {
+  error = containerfs_mkdir(parent->d_inode, dentry, S_IRUGO|S_IXUGO);
+  dput(dentry);
```

```
+ } else
+  error = PTR_ERR(dentry);
+ mutex_unlock(&parent->d_inode->i_mutex);
+
+ if (error) {
+  dentry = NULL;
+  goto exit;
+ }
+
+ container->dentry = dentry;
+exit:
+ return dentry;
+}
+
+static inline int containerfs_positive(struct dentry *dentry)
+{
+ return dentry->d_inode && !d_unhashed(dentry);
+}
+
+static void containerfs_remove(struct dentry *dentry)
+{
+ struct dentry *parent;
+ int ret = 0;
+
+ if (!dentry)
+  return;
+
+ parent = dentry->d_parent;
+ if (!parent || !parent->d_inode)
+  return;
+
+ mutex_lock(&parent->d_inode->i_mutex);
+ if (containerfs_positive(dentry)) {
+  if (dentry->d_inode) {
+   dget(dentry);
+   ret = simple_rmdir(parent->d_inode, dentry);
+   if (ret)
+    printk(KERN_ERR
+     "ContainerFS rmdir on %s failed : "
+     "directory not empty.\n",
+     dentry->d_name.name);
+   else
+    d_delete(dentry);
+   dput(dentry);
+  }
+ }
+ mutex_unlock(&parent->d_inode->i_mutex);
+}
```

```
+
+static int container_fill_super(struct super_block *sb, void *data, int silent)
+{
+ static struct tree_descr container_files[] = {{""}};
+
+ return simple_fill_super(sb, CONTAINERFS_MAGIC, container_files);
+}
+
+static int container_get_sb(struct file_system_type *fs_type,
+   int flags, const char *dev_name,
+   void *data, struct vfsmount *mnt)
+{
+ return get_sb_single(fs_type, flags, data, container_fill_super, mnt);
+}
+
+static struct file_system_type containerfs_type = {
+ .owner = THIS_MODULE,
+ .name =  "containerfs",
+ .get_sb = container_get_sb,
+ .kill_sb = kill_litter_super,
+};
+
+static int __init containerfs_init(void)
+{
+ int retval;
+
+ retval = register_filesystem(&containerfs_type);
+
+ if (retval)
+  return retval;
+ containerfs_mount =  kern_mount(&containerfs_type);
+ if (IS_ERR(containerfs_mount))
+  return PTR_ERR(containerfs_mount);
+ containerfs_create_dir(&init_container);
+
+ return 0;
+}
+
+core_initcall(containerfs_init);
--
1.4.1
```

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