Subject: [PATCH 1/4] user namespaces: add ns to user_struct
Posted by serue on Mon, 28 Jan 2008 19:09:20 GMT
View Forum Message <> Reply to Message

>From f032ac37281fb71a9e0fc5d71e1b268fd6a8cb3a Mon Sep 17 00:00:00 2001
From: sergeh@us.ibm.com <sergeh@us.ibm.com>
Date: Wed, 28 Nov 2007 14:50:54 -0800
Subject: [PATCH 1/4] user namespaces: add ns to user_struct

Add the user_namespace to user_struct.

Use ns to make sure we get right user with uid_hash_find().

Move /sys/kernel/uids/<uid> under
/sys/kernel/uids/<user_ns_address/<uid>

Signed-off-by: Serge Hallyn <serue@us.ibm.com>
---
 fs/dquot.c | 2 +-
 fs/ioprio.c | 2 +-
 include/linux/sched.h | 3 +-
 include/linux/types.h | 10 +++++
 include/linux/user_namespace.h | 3 ++
 kernel/user.c | 79 ++----
 kernel/user_namespace.c | 14 +++++++-
 security/keys/process_keys.c | 8 ++--
 8 files changed, 106 insertions(+), 15 deletions(-)

diff --git a/fs/dquot.c b/fs/dquot.c
index 9c7feb6..f413094 100644
--- a/fs/dquot.c
+++ b/fs/dquot.c
@@ -960,7 +960,7 @@ static void send_warning(const struct dquot *dquot, const char warntype)
   MINOR(dquot->dq_sb->s_dev));
  if (ret)
   goto attr_err_out;
- ret = nla_put_u64(skb, QUOTA_NL_A_CAUSED_ID, current->user->uid);
+ ret = nla_put_u64(skb, QUOTA_NL_A_CAUSED_ID, current->user->uid.uid);
  if (ret)
   goto attr_err_out;
  genlmsg_end(skb, msg_head);
diff --git a/fs/ioprio.c b/fs/ioprio.c
index e4e01bc..2ec1430 100644
--- a/fs/ioprio.c
+++ b/fs/ioprio.c
@@ -214,7 +214,7 @@ asmlinkage long sys_ioprio_get(int which, int who)
     break;

```
    do_each_thread(g, p) {
-    if (p->uid != user->uid)
+    if (!task_user_equiv(p, user))
      continue;
     tmpio = get_task_ioprio(p);
     if (tmpio < 0)
diff --git a/include/linux/sched.h b/include/linux/sched.h
index 198659b..1206486 100644
--- a/include/linux/sched.h
+++ b/include/linux/sched.h
@@ -584,7 +584,7 @@ struct user_struct {

  /* Hash table maintenance information */
  struct hlist_node uidhash_node;
- uid_t uid;
+ struct k_uid_t uid;

 #ifdef CONFIG_FAIR_USER_SCHED
  struct task_group *tg;
@@ -1634,6 +1634,7 @@ static inline struct user_struct *get_uid(struct user_struct *u)
 extern void free_uid(struct user_struct *);
 extern void switch_uid(struct user_struct *);
 extern void release_uids(struct user_namespace *ns);
+extern int task_user_equiv(struct task_struct *tsk, struct user_struct *u);

 #include <asm/current.h>

diff --git a/include/linux/types.h b/include/linux/types.h
index f4f8d19..a6a130e 100644
--- a/include/linux/types.h
+++ b/include/linux/types.h
@@ -37,6 +37,16 @@ typedef __kernel_gid32_t gid_t;
 typedef __kernel_uid16_t        uid16_t;
 typedef __kernel_gid16_t        gid16_t;

+struct k_uid_t {
+ uid_t uid;
+ struct user_namespace *ns;
+};
+
+struct k_gid_t {
+ gid_t gid;
+ struct user_namespace *ns;
+};
+
 typedef unsigned long  uintptr_t;

 #ifdef CONFIG_UID16
```

```
diff --git a/include/linux/user_namespace.h b/include/linux/user_namespace.h
index b5f41d4..bb5e88a 100644
--- a/include/linux/user_namespace.h
+++ b/include/linux/user_namespace.h
@@ -12,6 +12,9 @@
 struct user_namespace {
  struct kref  kref;
  struct hlist_head uidhash_table[UIDHASH_SZ];
+#if defined(CONFIG_FAIR_USER_SCHED) && defined(CONFIG_SYSFS)
+ struct kobject  kobject;
+#endif
  struct user_struct *root_user;
 };

diff --git a/kernel/user.c b/kernel/user.c
index 7d7900c..73dc8db 100644
--- a/kernel/user.c
+++ b/kernel/user.c
@@ -53,6 +53,10 @@ struct user_struct root_user = {
  .files  = ATOMIC_INIT(0),
  .sigpending = ATOMIC_INIT(0),
  .locked_shm     = 0,
+ .uid  = {
+  .uid = 0,
+  .ns = &init_user_ns,
+ },
 #ifdef CONFIG_KEYS
  .uid_keyring = &root_user_keyring,
  .session_keyring = &root_session_keyring,
@@ -75,13 +79,30 @@ static void uid_hash_remove(struct user_struct *up)
  hlist_del_init(&up->uidhash_node);
 }

-static struct user_struct *uid_hash_find(uid_t uid, struct hlist_head *hashent)
+int k_uid_equiv(struct k_uid_t a, struct k_uid_t b)
+{
+ if (a.uid == b.uid && a.ns == b.ns)
+  return 1;
+ return 0;
+}
+
+int task_user_equiv(struct task_struct *tsk, struct user_struct *u)
+{
+ if (tsk->uid != u->uid.uid)
+  return 0;
+ if (tsk->nsproxy->user_ns != u->uid.ns)
+  return 0;
+ return 1;
```

```
+}
+
+static struct user_struct *uid_hash_find(struct k_uid_t uid,
+    struct hlist_head *hashent)
 {
  struct user_struct *user;
  struct hlist_node *h;

  hlist_for_each_entry(user, h, hashent, uidhash_node) {
-  if (user->uid == uid) {
+  if (k_uid_equiv(user->uid, uid)) {
    atomic_inc(&user->__count);
    return user;
   }
@@ -190,7 +211,8 @@ static int uids_user_create(struct user_struct *up)

  memset(kobj, 0, sizeof(struct kobject));
  kobj->kset = uids_kset;
- error = kobject_init_and_add(kobj, &uids_ktype, NULL, "%d", up->uid);
+ error = kobject_init_and_add(kobj, &uids_ktype, &up->uid.ns->kobject,
+         "%d", up->uid.uid);
  if (error) {
   kobject_put(kobj);
   goto done;
@@ -201,6 +223,9 @@ done:
  return error;
 }

+int register_user_ns_kobj(struct user_namespace *ns);
+void unregister_user_ns_kobj(struct user_namespace *ns);
+
 /* create these entries in sysfs:
  *  "/sys/kernel/uids" directory
  *  "/sys/kernel/uids/0" directory (for root user)
@@ -208,10 +233,16 @@ done:
  */
 int __init uids_sysfs_init(void)
 {
+ int error;
+
  uids_kset = kset_create_and_add("uids", NULL, kernel_kobj);
  if (!uids_kset)
   return -ENOMEM;

+ error = register_user_ns_kobj(&init_user_ns);
+ if (error)
+  return error;
+
```

```
  return uids_user_create(&root_user);
 }

@@ -269,6 +300,30 @@ static inline void free_user(struct user_struct *up, unsigned long flags)
  schedule_work(&up->work);
 }

+int register_user_ns_kobj(struct user_namespace *ns)
+{
+ struct kobject *obj = &ns->kobject;
+ int error;
+
+ obj->kset = uids_kset;
+
+ error = kobject_init_and_add(obj, &uids_ktype, &uids_kset->kobj,
+         "%lx", (unsigned long)ns);
+ if (error)
+  return error;
+
+ kobject_uevent(obj, KOBJ_ADD);
+
+ return 0;
+}
+
+void unregister_user_ns_kobj(struct user_namespace *ns)
+{
+ struct kobject *obj = &ns->kobject;
+ kobject_uevent(obj, KOBJ_REMOVE);
+ kobject_del(obj);
+}
+
 #else /* CONFIG_FAIR_USER_SCHED && CONFIG_SYSFS */

 int uids_sysfs_init(void) { return 0; }
@@ -290,6 +345,8 @@ static inline void free_user(struct user_struct *up, unsigned long flags)
  kmem_cache_free(uid_cachep, up);
 }

+int register_user_ns_kobj(struct user_namespace *ns) { return 0; }
+void unregister_user_ns_kobj(struct user_namespace *ns) {}
 #endif

 /*
@@ -303,9 +360,12 @@ struct user_struct *find_user(uid_t uid)
  struct user_struct *ret;
  unsigned long flags;
  struct user_namespace *ns = current->nsproxy->user_ns;
+ struct k_uid_t kuid;
```

```
+ kuid.ns = current->nsproxy->user_ns;
+ kuid.uid = uid;
  spin_lock_irqsave(&uidhash_lock, flags);
- ret = uid_hash_find(uid, uidhashentry(ns, uid));
+ ret = uid_hash_find(kuid, uidhashentry(ns, uid));
  spin_unlock_irqrestore(&uidhash_lock, flags);
  return ret;
 }
@@ -328,6 +388,10 @@ struct user_struct * alloc_uid(struct user_namespace *ns, uid_t uid)
 {
  struct hlist_head *hashent = uidhashentry(ns, uid);
  struct user_struct *up, *new;
+ struct k_uid_t kuid;
+
+ kuid.ns = ns;
+ kuid.uid = uid;

  /* Make uid_hash_find() + uids_user_create() + uid_hash_insert()
   * atomic.
@@ -335,7 +399,7 @@ struct user_struct * alloc_uid(struct user_namespace *ns, uid_t uid)
  uids_mutex_lock();

  spin_lock_irq(&uidhash_lock);
- up = uid_hash_find(uid, hashent);
+ up = uid_hash_find(kuid, hashent);
  spin_unlock_irq(&uidhash_lock);

  if (!up) {
@@ -343,7 +407,8 @@ struct user_struct * alloc_uid(struct user_namespace *ns, uid_t uid)
   if (!new)
    goto out_unlock;

-  new->uid = uid;
+  new->uid.uid = uid;
+  new->uid.ns = ns;
   atomic_set(&new->__count, 1);
   atomic_set(&new->processes, 0);
   atomic_set(&new->files, 0);
@@ -371,7 +436,7 @@ struct user_struct * alloc_uid(struct user_namespace *ns, uid_t uid)
    * on adding the same user already..
    */
   spin_lock_irq(&uidhash_lock);
-  up = uid_hash_find(uid, hashent);
+  up = uid_hash_find(kuid, hashent);
   if (up) {
    /* This case is not possible when CONFIG_FAIR_USER_SCHED
     * is defined, since we serialize alloc_uid() using
```

```
diff --git a/kernel/user_namespace.c b/kernel/user_namespace.c
index 4c90062..7dc6cc7 100644
--- a/kernel/user_namespace.c
+++ b/kernel/user_namespace.c
@@ -10,6 +10,9 @@
 #include <linux/nsproxy.h>
 #include <linux/user_namespace.h>

+int register_user_ns_kobj(struct user_namespace *ns);
+void unregister_user_ns_kobj(struct user_namespace *ns);
+
 /*
  * Clone a new ns copying an original user ns, setting refcount to 1
  * @old_ns: namespace to clone
@@ -19,12 +22,16 @@ static struct user_namespace *clone_user_ns(struct user_namespace
*old_ns)
 {
  struct user_namespace *ns;
  struct user_struct *new_user;
- int n;
+ int n, err;

  ns = kmalloc(sizeof(struct user_namespace), GFP_KERNEL);
  if (!ns)
   return ERR_PTR(-ENOMEM);

+ err = register_user_ns_kobj(ns);
+ if (err)
+  goto out_free_ns;
+
  kref_init(&ns->kref);

  for (n = 0; n < UIDHASH_SZ; ++n)
@@ -47,6 +54,10 @@ static struct user_namespace *clone_user_ns(struct user_namespace
*old_ns)

  switch_uid(new_user);
  return ns;
+
+out_free_ns:
+ kfree(ns);
+ return ERR_PTR(err);
 }

 struct user_namespace * copy_user_ns(int flags, struct user_namespace *old_ns)
@@ -71,5 +82,6 @@ void free_user_ns(struct kref *kref)

  ns = container_of(kref, struct user_namespace, kref);
```

```
  release_uids(ns);
+ unregister_user_ns_kobj(ns);
  kfree(ns);
 }
diff --git a/security/keys/process_keys.c b/security/keys/process_keys.c
index c886a2b..0343a52 100644
--- a/security/keys/process_keys.c
+++ b/security/keys/process_keys.c
@@ -75,9 +75,9 @@ int alloc_uid_keyring(struct user_struct *user,
  int ret;

  /* concoct a default session keyring */
- sprintf(buf, "_uid_ses.%u", user->uid);
+ sprintf(buf, "_uid_ses.%u", user->uid.uid);

- session_keyring = keyring_alloc(buf, user->uid, (gid_t) -1, ctx,
+ session_keyring = keyring_alloc(buf, user->uid.uid, (gid_t) -1, ctx,
      KEY_ALLOC_IN_QUOTA, NULL);
  if (IS_ERR(session_keyring)) {
   ret = PTR_ERR(session_keyring);
@@ -86,9 +86,9 @@ int alloc_uid_keyring(struct user_struct *user,

  /* and a UID specific keyring, pointed to by the default session
   * keyring */
- sprintf(buf, "_uid.%u", user->uid);
+ sprintf(buf, "_uid.%u", user->uid.uid);

- uid_keyring = keyring_alloc(buf, user->uid, (gid_t) -1, ctx,
+ uid_keyring = keyring_alloc(buf, user->uid.uid, (gid_t) -1, ctx,
        KEY_ALLOC_IN_QUOTA, session_keyring);
  if (IS_ERR(uid_keyring)) {
   key_put(session_keyring);
--
1.5.1
```

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers