
Subject: Re: design of user namespaces

Posted by [serue](#) on Fri, 20 Jun 2008 21:47:46 GMT

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

Quoting Serge E. Hallyn (serue@us.ibm.com):

> > Just skimming through your patch I don't expect we will need the list

> > of children, and not having should reduce our locking burden.

>

> Hmm, that's true. I can't see a reason for that. Thanks!

BTW here is the new, slightly smaller patch:

>From d17fbd87d97f64a0e879a7efbe5e1835fc573eae Mon Sep 17 00:00:00 2001

From: Serge Hallyn <serge@us.ibm.com>

Date: Thu, 19 Jun 2008 20:18:17 -0500

Subject: [PATCH 1/1] user namespaces: introduce user_struct->user_namespace relationship

When a task does clone(CLONE_NEWNS), the task's user is the 'creator' of the new user_namespace, and the user_namespace is tacked onto a list of those created by this user.

When we create or put a user in a namespace, we also do so for all creator users up the creator chain.

Changelog:

Jun 20: Eric Biederman pointed out the sibling/child_user_ns list is unnecessary!

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

```
include/linux/sched.h      | 1 +
include/linux/user_namespace.h | 1 +
kernel/user.c              | 66 ++++++-----
kernel/user_namespace.c     | 15 +++-----
4 files changed, 72 insertions(+), 11 deletions(-)
```

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

index 799bbdd..da1bcc6 100644

--- a/include/linux/sched.h

+++ b/include/linux/sched.h

```
@@ -604,6 +604,7 @@ struct user_struct {
/* Hash table maintenance information */
struct hlist_node uidhash_node;
uid_t uid;
+ struct user_namespace *user_namespace;
```

```
#ifdef CONFIG_USER_SCHED
```

```
struct task_group *tg;
```

```

diff --git a/include/linux/user_namespace.h b/include/linux/user_namespace.h
index b5f41d4..f9477c3 100644
--- a/include/linux/user_namespace.h
+++ b/include/linux/user_namespace.h
@@ -13,6 +13,7 @@ struct user_namespace {
    struct kref kref;
    struct hlist_head uidhash_table[UIDHASH_SZ];
    struct user_struct *root_user;
+ struct user_struct *creator;
};

extern struct user_namespace init_user_ns;
diff --git a/kernel/user.c b/kernel/user.c
index 865ecf5..e583be4 100644
--- a/kernel/user.c
+++ b/kernel/user.c
@@ -21,6 +21,7 @@ struct user_namespace init_user_ns = {
    .kref = {
        .refcount = ATOMIC_INIT(2),
    },
+ .creator = &root_user,
    .root_user = &root_user,
};
EXPORT_SYMBOL_GPL(init_user_ns);
@@ -53,6 +54,7 @@ struct user_struct root_user = {
    .files = ATOMIC_INIT(0),
    .sigpending = ATOMIC_INIT(0),
    .locked_shm = 0,
+ .user_namespace = &init_user_ns,
#ifdef CONFIG_USER_SCHED
    .tg = &init_task_group,
#endif
@@ -71,6 +73,18 @@ static void uid_hash_remove(struct user_struct *up)
    hlist_del_init(&up->uidhash_node);
}

+void inc_user_and_creators(struct user_struct *user)
+{
+ struct user_namespace *ns = user->user_namespace;
+ while (user) {
+ atomic_inc(&user->__count);
+ if (ns == ns->creator->user_namespace)
+ return;
+ user = ns->creator;
+ ns = user->user_namespace;
+ }
+}
+

```

```

static struct user_struct *uid_hash_find(uid_t uid, struct hlist_head *hashent)
{
    struct user_struct *user;
@@ -78,7 +92,7 @@ static struct user_struct *uid_hash_find(uid_t uid, struct hlist_head
*hashent)

    hlist_for_each_entry(user, h, hashent, uidhash_node) {
        if (user->uid == uid) {
- atomic_inc(&user->__count);
+ inc_user_and_creators(user);
            return user;
        }
    }
@@ -315,12 +329,30 @@ done:
    uids_mutex_unlock();
}

+/*
+ * Decrement use counts for all namespace ancestors of the user
+ * being freed. The user itself has already been dec'ed, so
+ * we only start at its creator.
+ */
+void dec_creators(struct user_struct *user)
+{
+ struct user_namespace *ns = user->user_namespace;
+ while (ns != ns->creator->user_namespace) {
+     user = ns->creator;
+     atomic_dec(&user->__count);
+     ns = user->user_namespace;
+ }
+}
+
+/* IRQs are disabled and uidhash_lock is held upon function entry.
+ * IRQ state (as stored in flags) is restored and uidhash_lock released
+ * upon function exit.
+ */
static inline void free_user(struct user_struct *up, unsigned long flags)
{
+ /* decrement all creator counts */
+ dec_creators(up);
+
+ /* restore back the count */
    atomic_inc(&up->__count);
    spin_unlock_irqrestore(&uidhash_lock, flags);
@@ -409,6 +441,8 @@ struct user_struct *alloc_uid(struct user_namespace *ns, uid_t uid)
    if (sched_create_user(new) < 0)
        goto out_free_user;

```

```

+ new->user_namespace = ns;
+
+ if (uids_user_create(new))
+   goto out_destroy_sched;

@@ -429,6 +463,7 @@ struct user_struct *alloc_uid(struct user_namespace *ns, uid_t uid)
+   kmem_cache_free(uid_cachep, new);
+ } else {
+   uid_hash_insert(new, hashent);
+ inc_user_and_creators(new);
+   up = new;
+ }
+ spin_unlock_irq(&uidhash_lock);
@@ -448,6 +483,35 @@ out_unlock:
+   return NULL;
+ }

+/*
+ * After doing clone(CLONE_NEWUSER), the new task continues to hold
+ * a refcount on ancestor users, but just switches to the new
+ * root user in the child namespace
+ */
+void switch_uid_for_created_root(struct user_struct *new_user)
+{
+ struct user_struct *old_user;
+
+   old_user = current->user;
+   atomic_inc(&new_user->processes);
+   switch_uid_keyring(new_user);
+   current->user = new_user;
+   sched_switch_user(current);
+
+   /*
+    * We need to synchronize with __sigqueue_alloc()
+    * doing a get_uid(p->user).. If that saw the old
+    * user value, we need to wait until it has exited
+    * its critical region before we can free the old
+    * structure.
+    */
+   smp_mb();
+   spin_unlock_wait(&current->sigband->siglock);
+
+   free_uid(old_user);
+   suid_keys(current);
+}
+
+void switch_uid(struct user_struct *new_user)
+{

```

```

    struct user_struct *old_user;
diff --git a/kernel/user_namespace.c b/kernel/user_namespace.c
index a9ab059..45ba675 100644
--- a/kernel/user_namespace.c
+++ b/kernel/user_namespace.c
@@ -11,6 +11,7 @@
#include <linux/slab.h>
#include <linux/user_namespace.h>

+extern void switch_uid_for_created_root(struct user_struct *new_user);
/*
 * Clone a new ns copying an original user ns, setting refcount to 1
 * @old_ns: namespace to clone
@@ -19,7 +20,6 @@
static struct user_namespace *clone_user_ns(struct user_namespace *old_ns)
{
    struct user_namespace *ns;
- struct user_struct *new_user;
    int n;

    ns = kmalloc(sizeof(struct user_namespace), GFP_KERNEL);
@@ -31,6 +31,9 @@ static struct user_namespace *clone_user_ns(struct user_namespace
*old_ns)
    for (n = 0; n < UIDHASH_SZ; ++n)
        INIT_HLIST_HEAD(ns->uidhash_table + n);

+ /* set up owner/parent relationship */
+ ns->creator = current->user;
+
    /* Insert new root user. */
    ns->root_user = alloc_uid(ns, 0);
    if (!ns->root_user) {
@@ -38,15 +41,7 @@ static struct user_namespace *clone_user_ns(struct user_namespace
*old_ns)
        return ERR_PTR(-ENOMEM);
    }

- /* Reset current->user with a new one */
- new_user = alloc_uid(ns, current->uid);
- if (!new_user) {
-     free_uid(ns->root_user);
-     kfree(ns);
-     return ERR_PTR(-ENOMEM);
- }
-
- switch_uid(new_user);
+ switch_uid_for_created_root(ns->root_user);
    return ns;

```

}

--

1.5.4.3

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