Subject: [PATCH 0/4] user namespaces: introduction
Posted by serue on Mon, 28 Jan 2008 19:08:28 GMT
View Forum Message <> Reply to Message

Here is a small patchset I've been sitting on for awhile
to make signaling mostly subject to user namespaces. In
particular,

1. store user_namespace in user struct
2. introduce CAP_NS_OVERRIDE
3. require CAP_NS_OVERRIDE to signal another user namespace

The first step should have been done all along. Else wouldn't
a hash collision on (ns1, uid) and (ns2, uid), however unlikely,
give us wrong results at uid_hash_find()?

The main remaining signaling+userns issue is of course the
siginfo. Tacking a userns onto siginfo is a pain due to
lifetime mgmt issues. I haven't decided whether to just
catch all the callers and fake uid=0 if user namespaces
aren't the same, introduce some unique non-refcounted id to
represent (user,user_ns), or find some other way to deal with
it.

thanks,
-serge

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

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

---

Subject: [PATCH 2/4] containers: add CAP_NS_OVERRIDE capability
Posted by serue on Mon, 28 Jan 2008 19:09:40 GMT
View Forum Message <> Reply to Message

```
>From ce1cf14000860f82ab59a5253bbe468da767e77f Mon Sep 17 00:00:00 2001
From: sergeh@us.ibm.com <sergeh@us.ibm.com>
Date: Wed, 28 Nov 2007 18:52:28 -0800
Subject: [PATCH 2/4] containers: add CAP_NS_OVERRIDE capability

containers: add CAP_NS_OVERRIDE capability

Signed-off-by: sergeh@us.ibm.com <hallyn@kernel.(none)>
---
 include/linux/capability.h |    9 ++++++++-
 1 files changed, 8 insertions(+), 1 deletions(-)

diff --git a/include/linux/capability.h b/include/linux/capability.h
index 7d50ff6..58bc24e 100644
--- a/include/linux/capability.h
+++ b/include/linux/capability.h
@@ -332,7 +332,14 @@ typedef struct kernel_cap_struct {

 #define CAP_MAC_ADMIN        33
```

```
-#define CAP_LAST_CAP         CAP_MAC_ADMIN
+/* Allow acting on resources in another namespace.  In
+   particular:
+      1. when combined with CAP_KILL, kill users in another
+         user namespace
+ */
+#define CAP_NS_OVERRIDE      34
+
+#define CAP_LAST_CAP         CAP_NS_OVERRIDE

 #define cap_valid(x) ((x) >= 0 && (x) <= CAP_LAST_CAP)

--
1.5.1
```

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

---

Subject: [PATCH 3/4] user namespaces: enforce CAP_NS_OVERRIDE for cross-namespace kill

Posted by serue on Mon, 28 Jan 2008 19:09:58 GMT

View Forum Message <> Reply to Message

>From 491e532e336a98b1b9d1ecda6f4160d0c0adde89 Mon Sep 17 00:00:00 2001
From: sergeh@us.ibm.com <sergeh@us.ibm.com>
Date: Thu, 29 Nov 2007 08:18:16 -0800
Subject: [PATCH 3/4] user namespaces: enforce CAP_NS_OVERRIDE for cross-namespace kill

Require CAP_NS_OVERRIDE to 'kill' across user namespaces.

If the signaling task is exiting, then current->nsproxy is
NULL. Since we are only notifying a parent of our death,
we permit the signal.

If the target task is exiting, our signal doesn't matter
anyway.

Signed-off-by: Serge Hallyn <serue@us.ibm.com>
---
 kernel/signal.c | 23 +++++++++++++++++++++++
 1 files changed, 23 insertions(+), 0 deletions(-)

diff --git a/kernel/signal.c b/kernel/signal.c
index 280bccb..de19433 100644

```
--- a/kernel/signal.c
+++ b/kernel/signal.c
@@ -531,9 +531,32 @@ static int check_kill_permission(int sig, struct siginfo *info,
   return error;

  if (info == SEND_SIG_NOINFO || (!is_si_special(info) && SI_FROMUSER(info))) {
+  struct nsproxy *nsproxy;
+  struct user_namespace *my_user_ns = NULL, *t_user_ns = NULL;
+
   error = audit_signal_info(sig, t); /* Let audit system see the signal */
   if (error)
    return error;
+
+  /*
+   * if current->nsproxy is NULL, then we are exiting and
+   * are just sending an exit signal to our parent.
+   * Uid may be wrong under certain circumstances, but
+   * global init shouldn't care, and a container creation
+   * program should know what it is doing.
+   * If target is exiting then it doesn't matter anyway.
+   */
+  rcu_read_lock();
+  nsproxy = task_nsproxy(t);
+  if (nsproxy)
+   t_user_ns = nsproxy->user_ns;
+  rcu_read_unlock();
+  if (current->nsproxy)
+   my_user_ns = current->nsproxy->user_ns;
+  if (my_user_ns && t_user_ns && my_user_ns != t_user_ns
+   && !(capable(CAP_KILL) && capable(CAP_NS_OVERRIDE)))
+   return -EPERM;
+
   error = -EPERM;
   if (((sig != SIGCONT) ||
    (task_session_nr(current) != task_session_nr(t)))
--
1.5.1
```

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

Subject: [PATCH 4/4] user namespaces: handle user namespaces in file sigio
Posted by serue on Mon, 28 Jan 2008 19:10:13 GMT
View Forum Message <> Reply to Message

>From 1fa55246fc67f44e9c77829cb88a0df59e559bb5 Mon Sep 17 00:00:00 2001
From: sergeh@us.ibm.com <sergeh@us.ibm.com>
Date: Tue, 4 Dec 2007 15:37:48 -0800
Subject: [PATCH 4/4] user namespaces: handle user namespaces in file sigio

Store user namespaces in fown_struct. _f_modown() sets the
user_ns to current's, or to NULL if current has
capable(CAP_NS_OVERRIDE).

Only allow a signal to be sent through sigio if the recipient
is in the same user namespace or the sender (meaning the user
who did the F_SETOWN, not the one triggering io) has
CAP_NS_OVERRIDE.

Test as follows:

 1. log in as user hallyn in two windows
 2. in window 1, run vim
 3. in window 2, get pid for vim, i.e. PID=`pidof vim`
 4. mkfifo ab
 5. set_sigio ab $PID

If both logins are in same userns, the vim is killed.
if the logins are in separate user namespaces, vim is not
 killed.

****
set_sigio.c
****
```
 #include <stdio.h>
 #include <unistd.h>
 #include <stdlib.h>
 #include <sys/types.h>
 #include <sys/stat.h>
 #define __USE_GNU
 #include <fcntl.h>

int main(int argc, char *argv[])
{
 int err;
 int pid;
 int fd = open(argv[1], O_RDWR);

 if (!fd) {
  perror("open");
  exit(1);
 }
 printf("asked to set owner to %s\n", argv[2]);
```

```
 pid = atoi(argv[2]);
 printf("setting owner to %d\n", pid);
 err = fcntl(fd, F_SETOWN, pid);
 if (err == -1) {
  perror("fcntl 1\n");
  exit(1);
 }
 err = fcntl(fd, F_SETSIG, 9);
 if (err == -1) {
  perror("fcntl 3\n");
  exit(1);
 }
 err = fcntl(fd, F_SETFL, O_ASYNC);
 if (err == -1) {
  perror("fcntl 2\n");
  exit(1);
 }

 write(fd, "ab", 2);

 close(fd);
 printf("done\n");
}
```
****

Changelog:
 jan 28: always NULL userns when freeing it in f_modown().
 jan 28: fix sigio_userns_check_ok nsproxy access.

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

```
---
 fs/fcntl.c         |   67 +++++++++++++++++++++++++++++++++++++++++++++++----
 include/linux/fs.h |    2 +-
 2 files changed, 62 insertions(+), 7 deletions(-)

diff --git a/fs/fcntl.c b/fs/fcntl.c
index 603732a..b2565b8 100644
--- a/fs/fcntl.c
+++ b/fs/fcntl.c
@@ -19,6 +19,7 @@
 #include <linux/signal.h>
 #include <linux/rcupdate.h>
 #include <linux/pid_namespace.h>
+#include <linux/user_namespace.h>

 #include <asm/poll.h>
 #include <asm/siginfo.h>
@@ -259,10 +260,20 @@ static void f_modown(struct file *filp, struct pid *pid, enum pid_type
```

```
type,
  write_lock_irq(&filp->f_owner.lock);
  if (force || !filp->f_owner.pid) {
   put_pid(filp->f_owner.pid);
+  put_user_ns(filp->f_owner.uid.ns);
+  filp->f_owner.uid.ns = NULL;
+  filp->f_owner.euid.ns = NULL;
   filp->f_owner.pid = get_pid(pid);
   filp->f_owner.pid_type = type;
-  filp->f_owner.uid = uid;
-  filp->f_owner.euid = euid;
+  filp->f_owner.uid.uid = uid;
+  filp->f_owner.euid.uid = euid;
+  if (pid) {
+   if (!capable(CAP_NS_OVERRIDE)) {
+    filp->f_owner.uid.ns = current->nsproxy->user_ns;
+    filp->f_owner.euid.ns = current->nsproxy->user_ns;
+    get_user_ns(filp->f_owner.uid.ns);
+   }
+  }
  }
  write_unlock_irq(&filp->f_owner.lock);
 }
@@ -457,13 +468,57 @@ static const long band_table[NSIGPOLL] = {
  POLLHUP | POLLERR   /* POLL_HUP */
 };

+/*
+ * If task doing fcntl(F_SETOWN) had CAP_NS_OVERRIDE then
+ * the signal can cross user namespaces.  Otherwise, ensure
+ * that receiving task is in the same user namespace as the
+ * task which did the fcntl().
+ */
+static inline int sigio_userns_check_ok(struct task_struct *p,
+           struct fown_struct *fown)
+{
+ struct nsproxy *p_nsproxy;
+ struct user_namespace *p_userns = NULL;
+
+ if (!fown->euid.ns) /* sender had CAP_NS_OVERRIDE */
+  return 1;
+
+ rcu_read_lock();
+ p_nsproxy = task_nsproxy(p);
+ if (p_nsproxy)
+  p_userns = p_nsproxy->user_ns;
+ rcu_read_unlock();
+
```

```
+ /* if p_userns==NULL then p is exiting anyway */
+ if (!p_userns || p_userns == fown->euid.ns)
+  return 1;
+ return 0;
+}
+
+static inline int sigio_uids_check_ok(struct task_struct *p,
+          struct fown_struct *fown)
+{
+ return ((fown->euid.uid == 0) ||
+  (fown->euid.uid == p->suid) || (fown->euid.uid == p->uid) ||
+  (fown->uid.uid == p->suid) || (fown->uid.uid == p->uid));
+}
+
+#define lsm_sigiotask_ok(p, fown, sig) \
+ (!security_file_send_sigiotask(p, fown, sig))
+
 static inline int sigio_perm(struct task_struct *p,
                              struct fown_struct *fown, int sig)
 {
- return (((fown->euid == 0) ||
-   (fown->euid == p->suid) || (fown->euid == p->uid) ||
-   (fown->uid == p->suid) || (fown->uid == p->uid)) &&
-  !security_file_send_sigiotask(p, fown, sig));
+ if (!sigio_userns_check_ok(p, fown))
+  return 0;
+
+ if (!sigio_uids_check_ok(p, fown))
+  return 0;
+
+ if (!lsm_sigiotask_ok(p, fown, sig))
+  return 0;
+
+ return 1;
 }

 static void send_sigio_to_task(struct task_struct *p,
diff --git a/include/linux/fs.h b/include/linux/fs.h
index a456afe..6659081 100644
--- a/include/linux/fs.h
+++ b/include/linux/fs.h
@@ -749,7 +749,7 @@ struct fown_struct {
  rwlock_t lock;          /* protects pid, uid, euid fields */
  struct pid *pid; /* pid or -pgrp where SIGIO should be sent */
  enum pid_type pid_type; /* Kind of process group SIGIO should be sent to */
- uid_t uid, euid; /* uid/euid of process setting the owner */
+ struct k_uid_t uid, euid; /* uid/euid of process setting the owner */
  int signum;  /* posix.1b rt signal to be delivered on IO */
```

};

--
1.5.1

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

---

Subject: Re: [PATCH 0/4] user namespaces: introduction
Posted by Daniel Hokka Zakrisso on Mon, 28 Jan 2008 19:24:24 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Here is a small patchset I've been sitting on for awhile
> to make signaling mostly subject to user namespaces. In
> particular,
>
> 1. store user_namespace in user struct
> 2. introduce CAP_NS_OVERRIDE
> 3. require CAP_NS_OVERRIDE to signal another user namespace
>
> The first step should have been done all along. Else wouldn't
> a hash collision on (ns1, uid) and (ns2, uid), however unlikely,
> give us wrong results at uid_hash_find()?

Unless I've completely misunderstood the code, each namespace has a
separate hash. Please correct me if I'm wrong.

> The main remaining signaling+userns issue is of course the
> siginfo. Tacking a userns onto siginfo is a pain due to
> lifetime mgmt issues. I haven't decided whether to just
> catch all the callers and fake uid=0 if user namespaces
> aren't the same, introduce some unique non-refcounted id to
> represent (user,user_ns), or find some other way to deal with
> it.
>
> thanks,
> -serge

--
Daniel Hokka Zakrisson

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

---

Subject: Re: [PATCH 0/4] user namespaces: introduction
Posted by serue on Mon, 28 Jan 2008 19:44:50 GMT
View Forum Message <> Reply to Message

Quoting Daniel Hokka Zakrisson (daniel@hozac.com):
> Serge E. Hallyn wrote:
> > Here is a small patchset I've been sitting on for awhile
> > to make signaling mostly subject to user namespaces. In
> > particular,
> >
> > 1. store user_namespace in user struct
> > 2. introduce CAP_NS_OVERRIDE
> > 3. require CAP_NS_OVERRIDE to signal another user namespace
> >
> > The first step should have been done all along. Else wouldn't
> > a hash collision on (ns1, uid) and (ns2, uid), however unlikely,
> > give us wrong results at uid_hash_find()?
>
> Unless I've completely misunderstood the code, each namespace has a
> separate hash. Please correct me if I'm wrong.

So it does!

Yikes, how did I misread that so badly?

That means the find_user() code may be simplified a bit after all.
Well, or we could keep this and go to a single hash to save some
memory...

Thanks Daniel.

-serge

>
> > The main remaining signaling+userns issue is of course the
> > siginfo. Tacking a userns onto siginfo is a pain due to
> > lifetime mgmt issues. I haven't decided whether to just
> > catch all the callers and fake uid=0 if user namespaces
> > aren't the same, introduce some unique non-refcounted id to
> > represent (user,user_ns), or find some other way to deal with
> > it.
> >
> > thanks,
> > -serge
>
> --
> Daniel Hokka Zakrisson
> -
> To unsubscribe from this list: send the line "unsubscribe linux-security-module" in

> the body of a message to majordomo@vger.kernel.org
> More majordomo info at http://vger.kernel.org/majordomo-info.html

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

---

Subject: Re: [PATCH 0/4] user namespaces: introduction
Posted by serue on Mon, 04 Feb 2008 21:45:44 GMT
View Forum Message <> Reply to Message

Quoting Serge E. Hallyn (serue@us.ibm.com):
> Quoting Daniel Hokka Zakrisson (daniel@hozac.com):
> > Serge E. Hallyn wrote:
> > > Here is a small patchset I've been sitting on for awhile
> > > to make signaling mostly subject to user namespaces. In
> > > particular,
> > >
> > > 1. store user_namespace in user struct
> > > 2. introduce CAP_NS_OVERRIDE
> > > 3. require CAP_NS_OVERRIDE to signal another user namespace
> > >
> > > The first step should have been done all along. Else wouldn't
> > > a hash collision on (ns1, uid) and (ns2, uid), however unlikely,
> > > give us wrong results at uid_hash_find()?
> >
> > Unless I've completely misunderstood the code, each namespace has a
> > separate hash. Please correct me if I'm wrong.
>
> So it does!
>
> Yikes, how did I misread that so badly?
>
> That means the find_user() code may be simplified a bit after all.
> Well, or we could keep this and go to a single hash to save some
> memory...
>
> Thanks Daniel.
>
> -serge

Oh, here is the slightly simplified patch exploiting the fact
that each user namespace has its own user hashlist.

The rest of the patches remain unchanged.

-serge

>From 5c22bd668795c069c595b17bcfcce7db0e2c5423 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>

Changelog: feb 4: user hash tables are separate by namespace,
 so remove ns checks from uid_hash_find().

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 | 63 ++--
 kernel/user_namespace.c | 14 ++++++++-
 security/keys/process_keys.c | 8 ++---
 8 files changed, 94 insertions(+), 11 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..5cc9d77 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 +79,29 @@ static void uid_hash_remove(struct user_struct *up)
  hlist_del_init(&up->uidhash_node);
 }

+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(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 (user->uid.uid == uid) {
    atomic_inc(&user->__count);
    return user;
   }
@@ -190,7 +210,8 @@ 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 +222,9 @@ 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 +232,16 @@ 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 +299,30 @@ 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 +344,8 @@ 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

 /*
@@ -343,7 +399,8 @@ 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);
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