Subject: [RFC] [PATCH 0/8] user namespaces: add ns to user_struct
Posted by serue on Fri, 07 Dec 2007 19:12:59 GMT
View Forum Message <> Reply to Message

I'm working toward fixing up some of the remaining uid==0 and
uid1==uid2 checks, and beginning to restrict capabilities within
namespaces.

This patchset starts to do that by

1. improving per-ns user_struct storing
2. introducing CAP_NS_OVERRIDE
3. requiring CAP_NS_OVERRIDE to signal another user namespace
4. remove a few uid==0 checks

Especially the last 3 patches are a definite security improvement in
the face of user namespaces.

The next steps would be

* add user_ns to siginfo
* signals delivered to another userns (like sigchld)
  send uid 0.
* fix up more uid and gid checks (sigh)
* convert struct key_user?
* introduce uid aliases
  * per-process keyring
  * stores (user_ns,uid) keys
  * allows process which is really (user_ns1, uid1)
    to act as though it were (user_ns2, uid2) on
    objects in user_ns2
* convert struct kstat (may have serious lifetime issues)

That should leave us in a reasonable shape to start considering how
to really handle file access.

I still have a set of patches which tag struct inode with user_ns
and patch ext2+ext3.  But it's at the end of my patch set for now.

Comments welcome, on these patches, on the outlined next steps,
or on anything I'm forgetting.

(Against 2.6.24-rc3-mm2)

thanks,
-serge

_____
Containers mailing list

Subject: [RFC] [PATCH 1/8] user namespaces: add ns to user_struct
Posted by serue on Fri, 07 Dec 2007 19:13:36 GMT
View Forum Message <> Reply to Message

>From 8a15d1ac1a24ec96847e17ca7f0ba69ae42ac75b 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: [RFC] [PATCH 1/8] 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:
 Dec 7: Fix user->uid access at fs/dquot.c (Mark Nelson)

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               |   80 +++++++++++++++++++++++++++++++++++++++---
 kernel/user_namespace.c     |   14 ++++++-
 security/keys/process_keys.c |    8 ++--
 8 files changed, 107 insertions(+), 15 deletions(-)

diff --git a/fs/dquot.c b/fs/dquot.c
index 50e7c2a..e86232b 100644
--- a/fs/dquot.c
+++ b/fs/dquot.c
@@ -948,7 +948,7 @@ 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 9efd7f2..4768051 100644
--- a/include/linux/sched.h
+++ b/include/linux/sched.h
@@ -565,7 +565,7 @@ 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;
@@ -1584,6 +1584,7 @@ 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 fb0a67e..766c640 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;
   }
@@ -191,8 +212,9 @@ static int uids_user_create(struct user_struct *up)
  memset(kobj, 0, sizeof(struct kobject));
  kobj->ktype = &uids_ktype;
  kobj->kset = uids_kset;
+ kobj->parent = &up->uid.ns->kobject;
  kobject_init(kobj);
- kobject_set_name(&up->kobj, "%d", up->uid);
+ kobject_set_name(kobj, "%d", up->uid.uid);
  error = kobject_add(kobj);
  if (error)
   goto done;
@@ -202,6 +224,9 @@ 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)
@@ -209,10 +234,16 @@ done:
  */
 int __init uids_sysfs_init(void)
 {
+ int error;
+
```

```
  uids_kset = kset_create_and_register("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);
 }

@@ -270,6 +301,31 @@ 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->parent = &uids_kset->kobj;
+ obj->kset = uids_kset;
+ kobject_set_name(obj, "%lx", (unsigned long)ns);
+ kobject_init(obj);
+ kobject_uevent(obj, KOBJ_ADD);
+
+ error = kobject_add(obj);
+ if (error)
+  return error;
+
+ 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; }
@@ -291,6 +347,8 @@ 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

 /*
@@ -304,9 +362,12 @@ 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;
 }
@@ -329,6 +390,10 @@ 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.
@@ -336,7 +401,7 @@ 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) {
@@ -344,7 +409,8 @@ 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);
```

```
@@ -372,7 +438,7 @@ 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
```

Subject: [RFC] [PATCH 2/8] Bump the value of CAP_LAST_CAP to reflect the current last cap value.
Posted by serue on Fri, 07 Dec 2007 19:13:58 GMT
View Forum Message <> Reply to Message

>From d01c86d95bc6d59d7ca3689a9737a1aa9e8d3b59 Mon Sep 17 00:00:00 2001
From: Casey Schaufler <casey@schaufler-ca.com>
Date: Wed, 28 Nov 2007 18:48:59 -0800
Subject: [RFC] [PATCH 2/8] Bump the value of CAP_LAST_CAP to reflect the current last cap value.

It appears that the patch that introduced CAP_LAST_CAP and the patch
that introduced CAP_MAC_ADMIN came in more or less at the same time.

Signed-off-by: Casey Schaufler <casey@schaufler-ca.com>
---
 include/linux/capability.h |    8 ++++----
 1 files changed, 4 insertions(+), 4 deletions(-)

diff --git a/include/linux/capability.h b/include/linux/capability.h
index d0add24..7d50ff6 100644
--- a/include/linux/capability.h
+++ b/include/linux/capability.h
@@ -315,10 +315,6 @@ typedef struct kernel_cap_struct {

 #define CAP_SETFCAP      31

-#define CAP_LAST_CAP         CAP_SETFCAP
-
-#define cap_valid(x) ((x) >= 0 && (x) <= CAP_LAST_CAP)
-
 /* Override MAC access.
    The base kernel enforces no MAC policy.
    An LSM may enforce a MAC policy, and if it does and it chooses
@@ -336,6 +332,10 @@ typedef struct kernel_cap_struct {

 #define CAP_MAC_ADMIN        33

+#define CAP_LAST_CAP         CAP_MAC_ADMIN
+
+#define cap_valid(x) ((x) >= 0 && (x) <= CAP_LAST_CAP)
+
 /*
  * Bit location of each capability (used by user-space library and kernel)
  */
--
1.5.1

_____

Subject: [RFC] [PATCH 3/8] containers: add CAP_NS_OVERRIDE capability
Posted by serue on Fri, 07 Dec 2007 19:14:14 GMT
View Forum Message <> Reply to Message

>From a3a4950f8e9094aac1a9ccd6d453ea3dd68129be 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: [RFC] [PATCH 3/8] 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..36f9717 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: [RFC] [PATCH 4/8] user namespace: enforce CAP_NS_OVERRIDE for cross-namespace kill
Posted by serue on Fri, 07 Dec 2007 19:14:26 GMT
View Forum Message <> Reply to Message

>From 62e6efe435a24f430e28f2398f374cef197b4964 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: [RFC] [PATCH 4/8] user namespace: enforce CAP_NS_OVERRIDE for cross-namespace kill

Require CAP_NS_OVERRIDE to 'kill' across user namespaces.

Signed-off-by: Serge Hallyn <serue@us.ibm.com>
---
 kernel/signal.c | 5 +++++
 1 files changed, 5 insertions(+), 0 deletions(-)

diff --git a/kernel/signal.c b/kernel/signal.c
index 787521e..a06dcc2 100644
--- a/kernel/signal.c
+++ b/kernel/signal.c
@@ -534,6 +534,11 @@ static int check_kill_permission(int sig, struct siginfo *info,
  error = audit_signal_info(sig, t); /* Let audit system see the signal */
  if (error)
   return error;
+
+ if (current->nsproxy->user_ns != t->nsproxy->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: [RFC] [PATCH 5/8] userns: handle user namespaces in file sigio
Posted by serue on Fri, 07 Dec 2007 19:14:44 GMT
View Forum Message <> Reply to Message

>From 649ee38555a07f5ce052a8483a5f271fc31054d3 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: [RFC] [PATCH 5/8] userns: 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");
}
************************************************************

Signed-off-by: Serge Hallyn <serue@us.ibm.com>
---
 fs/fcntl.c         |   53 ++++++++++++++++++++++++++++++++++++++++++++++++++-----
 include/linux/fs.h |    2 +-
 2 files changed, 48 insertions(+), 7 deletions(-)

diff --git a/fs/fcntl.c b/fs/fcntl.c
index 8685263..124408a 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 +260,23 @@ 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.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) && capable(CAP_KILL)) {
+    filp->f_owner.uid.ns = NULL;
+    filp->f_owner.euid.ns = NULL;
+   } else {
+    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 +471,40 @@ static const long band_table[NSIGPOLL] = {
  POLLHUP | POLLERR   /* POLL_HUP */
 };

+static inline int sigio_userns_check_ok(struct task_struct *p,
+     struct fown_struct *fown)
+{
+ if (!fown->euid.ns)
+  return 1;
+ if (p->nsproxy->user_ns == 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 8323ebf..cb3894d 100644
--- a/include/linux/fs.h
+++ b/include/linux/fs.h
@@ -747,7 +747,7 @@ 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: [RFC] [PATCH 6/8] userns: agp: remove uid comparison as security check
Posted by serue on Fri, 07 Dec 2007 19:15:08 GMT
View Forum Message <> Reply to Message

>From d4ca1a9749c5b40325ec2db9fcde2f2cbd0e0978 Mon Sep 17 00:00:00 2001
From: sergeh@us.ibm.com <sergeh@us.ibm.com>
Date: Wed, 5 Dec 2007 13:55:36 -0800
Subject: [RFC] [PATCH 6/8] userns: agp: remove uid comparison as security check

In the face of user namespaces, a uid==0 check for security is
not safe.  Switch to a capability check.

I'm not sure I picked the right capability, but this being AGP
CAP_SYS_RAWIO seemed to make sense.

Signed-off-by: Serge Hallyn <serue@us.ibm.com>
---
 drivers/char/agp/frontend.c |    2 +-
 1 files changed, 1 insertions(+), 1 deletions(-)

diff --git a/drivers/char/agp/frontend.c b/drivers/char/agp/frontend.c
index 9bd5a95..55d7a82 100644
--- a/drivers/char/agp/frontend.c
+++ b/drivers/char/agp/frontend.c
@@ -689,7 +689,7 @@ static int agp_open(struct inode *inode, struct file *file)
  set_bit(AGP_FF_ALLOW_CLIENT, &priv->access_flags);
  priv->my_pid = current->pid;

- if ((current->uid == 0) || (current->suid == 0)) {
+ if (capable(CAP_SYS_RAWIO)) {
   /* Root priv, can be controller */
   set_bit(AGP_FF_ALLOW_CONTROLLER, &priv->access_flags);
  }
--
1.5.1

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

---

## Subject: [RFC] [PATCH 7/8] userns: reiser4: replace uid==0 check with capability
Posted by serue on Fri, 07 Dec 2007 19:15:28 GMT
View Forum Message <> Reply to Message

>From c257cb67ce00c8769730cfa92379a53009d99b28 Mon Sep 17 00:00:00 2001
From: sergeh@us.ibm.com <sergeh@us.ibm.com>
Date: Wed, 5 Dec 2007 14:02:45 -0800
Subject: [RFC] [PATCH 7/8] userns: reiser4: replace uid==0 check with capability

Reiser4 gives root some reserved blocks.  Replace the uid==0 check, which
is not safe in the face of user namespaces, with a CAP_SYS_RESOURCE check,
which seems appropriate.

The per-uid and per-guid reservations appear unimplemented so I'm ignoring

them.

Signed-off-by: Serge Hallyn <serue@us.ibm.com>
---
 fs/reiser4/super.c |    2 +-
 1 files changed, 1 insertions(+), 1 deletions(-)

diff --git a/fs/reiser4/super.c b/fs/reiser4/super.c
index bc4113e..50e3d09 100644
--- a/fs/reiser4/super.c
+++ b/fs/reiser4/super.c
@@ -144,7 +144,7 @@ long reiser4_reserved_blocks(const struct super_block *super /* super
block
   reserved += reserved_for_gid(super, gid);
  if (REISER4_SUPPORT_UID_SPACE_RESERVATION)
   reserved += reserved_for_uid(super, uid);
- if (REISER4_SUPPORT_ROOT_SPACE_RESERVATION && (uid == 0))
+ if (REISER4_SUPPORT_ROOT_SPACE_RESERVATION &&
capable(CAP_SYS_RESOURCE))
   reserved += reserved_for_root(super);
  return reserved;
 }
--
1.5.1

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

---

## Subject: [RFC] [PATCH 8/8] userns: oom_kill: remove uid==0 checks
Posted by serue on Fri, 07 Dec 2007 19:16:00 GMT
View Forum Message <> Reply to Message

>From a5fd2d7c75168076dc6b4b94ea8cda529fc506b1 Mon Sep 17 00:00:00 2001
From: sergeh@us.ibm.com <sergeh@us.ibm.com>
Date: Wed, 5 Dec 2007 14:07:40 -0800
Subject: [RFC] [PATCH 8/8] userns: oom_kill: remove uid==0 checks

Root processes are considered more important when out of memory
and killing proceses.  The check for CAP_SYS_ADMIN was augmented
with a check for uid==0 or euid==0.

There are several possible ways to look at this:

 1. uid comparisons are unnecessary, trust CAP_SYS_ADMIN
    alone.  However CAP_SYS_RESOURCE is the one that really

means "give me extra resources" so allow for that as
well.
2. Any privileged code should be protected, but uid is not
an indication of privilege. So we should check whether
any capabilities are active in pP.
3. uid==0 makes processes on the host as well as in containers
more important, so we should keep the existing checks.
4. uid==0 makes processes only on the host more important,
even without any capabilities. So we should be keeping
the (uid==0||euid==0) check but only when
userns==&init_user_ns.

I'm following number 1 here.

Signed-off-by: Serge Hallyn <serue@us.ibm.com>
---
mm/oom_kill.c | 2 +-
1 files changed, 1 insertions(+), 1 deletions(-)

diff --git a/mm/oom_kill.c b/mm/oom_kill.c
index 016127e..9fd8d5d 100644
--- a/mm/oom_kill.c
+++ b/mm/oom_kill.c
@@ -128,7 +128,7 @@ unsigned long badness(struct task_struct *p, unsigned long uptime,
* Superuser processes are usually more important, so we make it
* less likely that we kill those.
*/
- if (__capable(p, CAP_SYS_ADMIN) || p->uid == 0 || p->euid == 0)
+ if (__capable(p, CAP_SYS_ADMIN) || __capable(p, CAP_SYS_RESOURCE))
points /= 4;

/*
--
1.5.1

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

---

Subject: Re: [RFC] [PATCH 4/8] user namespace: enforce CAP_NS_OVERRIDE for cross-namespace kill
Posted by serue on Wed, 12 Dec 2007 19:22:49 GMT
View Forum Message <> Reply to Message

Quoting Serge E. Hallyn (serue@us.ibm.com):
> >From 62e6efe435a24f430e28f2398f374cef197b4964 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: [RFC] [PATCH 4/8] user namespace: enforce CAP_NS_OVERRIDE for
cross-namespace kill
>
> Require CAP_NS_OVERRIDE to 'kill' across user namespaces.
>
> Signed-off-by: Serge Hallyn <serue@us.ibm.com>
> ---
>  kernel/signal.c |    5 +++++
>  1 files changed, 5 insertions(+), 0 deletions(-)
>
> diff --git a/kernel/signal.c b/kernel/signal.c
> index 787521e..a06dcc2 100644
> --- a/kernel/signal.c
> +++ b/kernel/signal.c
> @@ -534,6 +534,11 @@ static int check_kill_permission(int sig, struct siginfo *info,
>    error = audit_signal_info(sig, t); /* Let audit system see the signal */
>    if (error)
>     return error;
> +
> +  if (current->nsproxy->user_ns != t->nsproxy->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

This patch has a sadly obvious, but (just as sadly) hard for me to
reproduce pair of bugs which Cedric found (thanks).

The following patch *should* be a correct fix, but as it's the fourth
version I'm going to keep testing for awhile.

In the meantime, the last three patches in this set are really
independent of the rest, so I plan to send them individually later today
with cc:s to the appropriate maintainers for their review.

thanks,
-serge

>From 558e37c30d2047e040e4e2d40ed87c96cd78183f 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 1/1] user namespace: enforce CAP_NS_OVERRIDE for cross-namespace kill

(v4)

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 ++++++++++++++++++++++

```
diff --git a/kernel/signal.c b/kernel/signal.c
index 06e663d..75c4d00 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
```