Subject: user namespace - introduction
Posted by serue on Mon, 04 Jun 2007 19:39:57 GMT
View Forum Message <> Reply to Message

[I've been sitting on this for some months, and am just dumping it so people can talk if they like, maybe even build on the patchset by adding support for more filesystems or implementing the keyring. Or tell me how much the approach sucks.]

First, I point out once more that the base user namespace patchset Cedric originally sent out really is sufficient. We just need for users to have different quotas, limits, and in-kernel key storage. Signal delivery, file controls, etc can be set up using pidspaces and separate mount trees, using selinux policy or other lsms, and even using ecryptfs.

But if it will be insisted upon that uid checks be enhanced, here is a new patchset which just might satisfy everyone, and which is based on user namespace discussions from the last year, particularly comments by Eric Biederman and David Howells.

Below is how I think the user namespace controls would work. The patches that follow only touch on parts of steps 1-4.

1. let filesystem tag inodes and superblocks with one user namespace
2. let generic_permission - and through inode->i_op->permission, the fs, if it wants to be smarter - enforce user namespaces
3. by default, inode->i_userns comes from sb->s_userns, just as is done in these patches.
4. By default, if inode->i_userns != task->userns, the process gets treated as 'nobody'. This is a change from my current patches and what is done in -lxc, where all permission is denied. I think it is a far preferable behavior. It allows read-only bind mount sharing among user namespaces without a silly MS_USER_NS flag.
5. Capabilities relating to actions on subjects or objects associated with a user namespace are only effective for targets in the same user namespace as the actor.
   This *could* be changed to also work for targets in decendant user namespaces, but that could slow things down.
6. Create a new keychain for user namespaces. Two types of entries.
   The first type of entry, (user_ns 5, uid 501) means that whichever user has that key will be recognized in user namespace 5 as uid 501. Presumably, uid 501 in user_ns 5 would have started a vserver with a new user namespace, say user_ns 7. He would likely want to give uid 0 in user_ns 7 a (user_ns 5, uid 501) key.
   The second type of key, (user_ns 5, CAP_FOWNER) gives the user

holding the key the ability to have CAP_FOWNER in userns 5. By default, uid 0 in userns 7 cannot have CAP_FOWNER in userns 5. (Only) a task with (userns 5, CAP_SETPCAP) can give that key to any user in userns 7. The key by itself does not grant the capability, but allows a task with that uid which has CAP_FOWNER in its P set to assert it for userns 5.

7. Eventually filesystems could begin storing global uids in inode xattrs on disk, and use these in inode->i_op->permission() along with data in the user's userns key to do global uid permission checking. Really this should almost trivial to implement once the above has been implemented. It could be done right in ext234 etc, or in a small stackable fs.

-serge

Note: step 1 has been complained about bc some think it should be done at the vfsmount level. If you read through the whole set of steps I think you'll see why it is not more limited. The fs gets to decide the real owner of a file, and despite there being one real owner, any number of users can be made to be treated as the owner, so there is no limitation in this approach.

Subject: [PATCH 1/6] user namespace : add the framework
Posted by serue on Mon, 04 Jun 2007 19:40:24 GMT
View Forum Message <> Reply to Message

>From nobody Mon Sep 17 00:00:00 2001
From: Cedric Le Goater <clg@fr.ibm.com>
Date: Thu, 5 Apr 2007 12:51:51 -0400
Subject: [PATCH 1/6] user namespace : add the framework

Add the user namespace struct and framework

Basically, it will allow a process to unshare its user_struct table, resetting at the same time its own user_struct and all the associated accounting.

A new root user (uid == 0) is added to the user namespace upon creation. Such root users have full privileges and it seems that theses privileges should be controlled through some means (process capabilities ?)

The unshare is not included in this patch.

Changes since [try #4]:
- Updated get_user_ns and put_user_ns to accept NULL, and get_user_ns to return the namespace.

Changes since [try #3]:
 - moved struct user_namespace to files user_namespace.{c,h}

Changes since [try #2]:
 - removed struct user_namespace* argument from find_user()

Changes since [try #1]:
 - removed struct user_namespace* argument from find_user()
 - added a root_user per user namespace

Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>

Signed-off-by: Serge E. Hallyn <serue@us.ibm.com>
Cc: Herbert Poetzl <herbert@13thfloor.at>
Cc: Kirill Korotaev <dev@sw.ru>
Cc: "Eric W. Biederman" <ebiederm@xmission.com>
Signed-off-by: Andrew Morton <akpm@osdl.org>

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

---

 include/linux/init_task.h | 2 +
 include/linux/nsproxy.h | 1 +
 include/linux/sched.h | 3 +-
 include/linux/user_namespace.h | 57 ++
 init/Kconfig | 9 ++++++
 kernel/Makefile | 2 +
 kernel/fork.c | 2 +
 kernel/nsproxy.c | 10 +++++++
 kernel/sys.c | 5 ++--
 kernel/user.c | 18 ++++++++----
 kernel/user_namespace.c | 43 ++++++++++++++++++++++++++++++
 11 files changed, 138 insertions(+), 14 deletions(-)
 create mode 100644 include/linux/user_namespace.h
 create mode 100644 kernel/user_namespace.c

cfeebe74a11622089791ecb14070f4d620f55535
diff --git a/include/linux/init_task.h b/include/linux/init_task.h
index cca7bb1..1fdc46c 100644
--- a/include/linux/init_task.h
+++ b/include/linux/init_task.h
@@ -8,6 +8,7 @@ #include <linux/utsname.h>
 #include <linux/lockdep.h>
 #include <linux/ipc.h>
 #include <linux/pid_namespace.h>
+#include <linux/user_namespace.h>

```
 #define INIT_FDTABLE \
 {      \
@@ -78,6 +79,7 @@ #define INIT_NSPROXY(nsproxy) {      \
  .uts_ns  = &init_uts_ns,     \
  .mnt_ns  = NULL,      \
  INIT_IPC_NS(ipc_ns)      \
+ .user_ns = &init_user_ns,    \
 }

 #define INIT_SIGHAND(sighand) {      \
diff --git a/include/linux/nsproxy.h b/include/linux/nsproxy.h
index 616a710..36e84e5 100644
--- a/include/linux/nsproxy.h
+++ b/include/linux/nsproxy.h
@@ -28,6 +28,7 @@ struct nsproxy {
  struct ipc_namespace *ipc_ns;
  struct mnt_namespace *mnt_ns;
  struct pid_namespace *pid_ns;
+ struct user_namespace *user_ns;
 };
 extern struct nsproxy init_nsproxy;

diff --git a/include/linux/sched.h b/include/linux/sched.h
index 8016ec8..259d0a5 100644
--- a/include/linux/sched.h
+++ b/include/linux/sched.h
@@ -263,6 +263,7 @@ extern signed long schedule_timeout_unin
 asmlinkage void schedule(void);

 struct nsproxy;
+struct user_namespace;

 /* Maximum number of active map areas.. This is a random (large) number */
 #define DEFAULT_MAX_MAP_COUNT 65536
@@ -1311,7 +1312,7 @@ extern struct task_struct *find_task_by_
 extern void __set_special_pids(pid_t session, pid_t pgrp);

 /* per-UID process charging. */
-extern struct user_struct * alloc_uid(uid_t);
+extern struct user_struct * alloc_uid(struct user_namespace *, uid_t);
 static inline struct user_struct *get_uid(struct user_struct *u)
 {
  atomic_inc(&u->__count);
diff --git a/include/linux/user_namespace.h b/include/linux/user_namespace.h
new file mode 100644
index 0000000..c2debd8
--- /dev/null
+++ b/include/linux/user_namespace.h
```

```
@@ -0,0 +1,57 @@
+#ifndef _LINUX_USER_NAMESPACE_H
+#define _LINUX_USER_NAMESPACE_H
+
+#include <linux/kref.h>
+#include <linux/nsproxy.h>
+#include <linux/sched.h>
+
+#define UIDHASH_BITS (CONFIG_BASE_SMALL ? 3 : 8)
+#define UIDHASH_SZ (1 << UIDHASH_BITS)
+
+struct user_namespace {
+ struct kref  kref;
+ struct list_head uidhash_table[UIDHASH_SZ];
+ struct user_struct *root_user;
+};
+
+extern struct user_namespace init_user_ns;
+
+#ifdef CONFIG_USER_NS
+
+static inline struct user_namespace *get_user_ns(struct user_namespace *ns)
+{
+ if (ns)
+  kref_get(&ns->kref);
+ return ns;
+}
+
+extern struct user_namespace *copy_user_ns(int flags,
+       struct user_namespace *old_ns);
+extern void free_user_ns(struct kref *kref);
+
+static inline void put_user_ns(struct user_namespace *ns)
+{
+ if (ns)
+  kref_put(&ns->kref, free_user_ns);
+}
+
+#else
+
+static inline struct user_namespace *get_user_ns(struct user_namespace *ns)
+{
+ return &init_user_ns;
+}
+
+static inline struct user_namespace *copy_user_ns(int flags,
+       struct user_namespace *old_ns)
+{
```

```
+ return 0;
+}
+
+static inline void put_user_ns(struct user_namespace *ns)
+{
+}
+
+#endif
+
+#endif /* _LINUX_USER_H */
diff --git a/init/Kconfig b/init/Kconfig
index 99b0b32..0bbf957 100644
--- a/init/Kconfig
+++ b/init/Kconfig
@@ -229,6 +229,15 @@ config TASK_IO_ACCOUNTING

    Say N if unsure.

+config USER_NS
+ bool "User Namespaces (EXPERIMENTAL)"
+ default n
+ depends on EXPERIMENTAL
+ help
+   Support user namespaces.  This allows containers, i.e.
+   vservers, to use user namespaces to provide different
+   user info for different servers.  If unsure, say N.
+
 config AUDIT
  bool "Auditing support"
  depends on NET
diff --git a/kernel/Makefile b/kernel/Makefile
index 6101981..b5cef19 100644
--- a/kernel/Makefile
+++ b/kernel/Makefile
@@ -4,7 +4,7 @@ #

 obj-y     = sched.o fork.o exec_domain.o panic.o printk.o profile.o \
       exit.o itimer.o time.o softirq.o resource.o \
-      sysctl.o capability.o timer.o user.o \
+      sysctl.o capability.o timer.o user.o user_namespace.o \
       signal.o sys.o kmod.o workqueue.o pid.o \
       rcupdate.o extable.o params.o posix-timers.o \
       kthread.o wait.o kfifo.o sys_ni.o posix-cpu-timers.o mutex.o \
diff --git a/kernel/fork.c b/kernel/fork.c
index 6ebd35b..c20137e 100644
--- a/kernel/fork.c
+++ b/kernel/fork.c
@@ -999,7 +999,7 @@ #endif
```

```
  if (atomic_read(&p->user->processes) >=
     p->signal->rlim[RLIMIT_NPROC].rlim_cur) {
    if (!capable(CAP_SYS_ADMIN) && !capable(CAP_SYS_RESOURCE) &&
-     p->user != &root_user)
+       p->user != current->nsproxy->user_ns->root_user)
     goto bad_fork_free;
  }

diff --git a/kernel/nsproxy.c b/kernel/nsproxy.c
index 3f3b7ad..1ea8a73 100644
--- a/kernel/nsproxy.c
+++ b/kernel/nsproxy.c
@@ -83,8 +83,15 @@ static struct nsproxy *create_new_namesp
  if (IS_ERR(new_nsp->pid_ns))
   goto out_pid;

+ new_nsp->user_ns = copy_user_ns(flags, tsk->nsproxy->user_ns);
+ if (IS_ERR(new_nsp->user_ns))
+  goto out_user;
+
  return new_nsp;

+out_user:
+ if (new_nsp->pid_ns)
+  put_pid_ns(new_nsp->pid_ns);
 out_pid:
  if (new_nsp->ipc_ns)
   put_ipc_ns(new_nsp->ipc_ns);
@@ -144,6 +151,9 @@ void free_nsproxy(struct nsproxy *ns)
   put_ipc_ns(ns->ipc_ns);
  if (ns->pid_ns)
   put_pid_ns(ns->pid_ns);
+ if (ns->user_ns)
+  put_user_ns(ns->user_ns);
+
  kmem_cache_free(nsproxy_cachep, ns);
 }

diff --git a/kernel/sys.c b/kernel/sys.c
index 7d79146..bb37fcb 100644
--- a/kernel/sys.c
+++ b/kernel/sys.c
@@ -34,6 +34,7 @@ #include <linux/task_io_accounting_ops.h
 #include <linux/compat.h>
 #include <linux/syscalls.h>
 #include <linux/kprobes.h>
+#include <linux/user_namespace.h>
```

```
 #include <asm/uaccess.h>
 #include <asm/io.h>
@@ -1077,13 +1078,13 @@ static int set_user(uid_t new_ruid, int
 {
  struct user_struct *new_user;

- new_user = alloc_uid(new_ruid);
+ new_user = alloc_uid(current->nsproxy->user_ns, new_ruid);
  if (!new_user)
   return -EAGAIN;

  if (atomic_read(&new_user->processes) >=
     current->signal->rlim[RLIMIT_NPROC].rlim_cur &&
-   new_user != &root_user) {
+   new_user != current->nsproxy->user_ns->root_user) {
   free_uid(new_user);
   return -EAGAIN;
  }
diff --git a/kernel/user.c b/kernel/user.c
index 4869563..98b8250 100644
--- a/kernel/user.c
+++ b/kernel/user.c
@@ -14,20 +14,19 @@ #include <linux/slab.h>
 #include <linux/bitops.h>
 #include <linux/key.h>
 #include <linux/interrupt.h>
+#include <linux/module.h>
+#include <linux/user_namespace.h>

 /*
  * UID task count cache, to get fast user lookup in "alloc_uid"
  * when changing user ID's (ie setuid() and friends).
  */

-#define UIDHASH_BITS (CONFIG_BASE_SMALL ? 3 : 8)
-#define UIDHASH_SZ  (1 << UIDHASH_BITS)
 #define UIDHASH_MASK  (UIDHASH_SZ - 1)
 #define __uidhashfn(uid) (((uid >> UIDHASH_BITS) + uid) & UIDHASH_MASK)
-#define uidhashentry(uid) (uidhash_table + __uidhashfn((uid)))
+#define uidhashentry(ns, uid) ((ns)->uidhash_table + __uidhashfn((uid)))

 static struct kmem_cache *uid_cachep;
-static struct list_head uidhash_table[UIDHASH_SZ];

 /*
  * The uidhash_lock is mostly taken from process context, but it is
@@ -94,9 +93,10 @@ struct user_struct *find_user(uid_t uid)
 {
```

```
  struct user_struct *ret;
  unsigned long flags;
+ struct user_namespace *ns = current->nsproxy->user_ns;

  spin_lock_irqsave(&uidhash_lock, flags);
- ret = uid_hash_find(uid, uidhashentry(uid));
+ ret = uid_hash_find(uid, uidhashentry(ns, uid));
  spin_unlock_irqrestore(&uidhash_lock, flags);
  return ret;
 }
@@ -120,9 +120,9 @@ void free_uid(struct user_struct *up)
  }
 }

-struct user_struct * alloc_uid(uid_t uid)
+struct user_struct * alloc_uid(struct user_namespace *ns, uid_t uid)
 {
- struct list_head *hashent = uidhashentry(uid);
+ struct list_head *hashent = uidhashentry(ns, uid);
  struct user_struct *up;

  spin_lock_irq(&uidhash_lock);
@@ -211,11 +211,11 @@ static int __init uid_cache_init(void)
    0, SLAB_HWCACHE_ALIGN|SLAB_PANIC, NULL, NULL);

  for(n = 0; n < UIDHASH_SZ; ++n)
-  INIT_LIST_HEAD(uidhash_table + n);
+  INIT_LIST_HEAD(init_user_ns.uidhash_table + n);

  /* Insert the root user immediately (init already runs as root) */
  spin_lock_irq(&uidhash_lock);
- uid_hash_insert(&root_user, uidhashentry(0));
+ uid_hash_insert(&root_user, uidhashentry(&init_user_ns, 0));
  spin_unlock_irq(&uidhash_lock);

  return 0;
diff --git a/kernel/user_namespace.c b/kernel/user_namespace.c
new file mode 100644
index 0000000..3d79642
--- /dev/null
+++ b/kernel/user_namespace.c
@@ -0,0 +1,43 @@
+/*
+ *  This program is free software; you can redistribute it and/or
+ *  modify it under the terms of the GNU General Public License as
+ *  published by the Free Software Foundation, version 2 of the
+ *  License.
+ */
```

```
+
+#include <linux/module.h>
+#include <linux/version.h>
+#include <linux/nsproxy.h>
+#include <linux/user_namespace.h>
+
+struct user_namespace init_user_ns = {
+ .kref = {
+  .refcount = ATOMIC_INIT(2),
+ },
+ .root_user = &root_user,
+};
+
+EXPORT_SYMBOL_GPL(init_user_ns);
+
+#ifdef CONFIG_USER_NS
+
+struct user_namespace * copy_user_ns(int flags, struct user_namespace *old_ns)
+{
+ struct user_namespace *new_ns;
+
+ BUG_ON(!old_ns);
+ get_user_ns(old_ns);
+
+ new_ns = old_ns;
+ return new_ns;
+}
+
+void free_user_ns(struct kref *kref)
+{
+ struct user_namespace *ns;
+
+ ns = container_of(kref, struct user_namespace, kref);
+ kfree(ns);
+}
+
+#endif /* CONFIG_USER_NS */
--
1.3.2
```

Subject: [PATCH 5/6] userns strict: hook ext2

Posted by serue on Mon, 04 Jun 2007 19:42:00 GMT

View Forum Message <> Reply to Message

>From nobody Mon Sep 17 00:00:00 2001
From: Serge Hallyn <serue@us.ibm.com>
Date: Wed, 28 Mar 2007 15:06:47 -0500

Subject: [PATCH 5/6] userns strict: hook ext2

Add a user namespace pointer to the ext2 superblock and inode.

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

```
 fs/ext2/acl.c                |    4 +++-
 fs/ext2/balloc.c             |   13 ++++++++++---
 fs/ext2/ialloc.c             |    5 +++++
 fs/ext2/inode.c              |    2 ++
 fs/ext2/ioctl.c              |   12 ++++++++----
 fs/ext2/super.c              |    4 ++++
 fs/ext2/xattr_trusted.c      |    3 ++-
 include/linux/ext2_fs_sb.h   |    1 +
 include/linux/user_namespace.h |   16 ++++++++++++++++
 9 files changed, 51 insertions(+), 9 deletions(-)

04b91190fbcaf388100bc748062724e384f1d868
diff --git a/fs/ext2/acl.c b/fs/ext2/acl.c
index 7c420b8..7bb1ccd 100644
--- a/fs/ext2/acl.c
+++ b/fs/ext2/acl.c
@@ -9,6 +9,7 @@ #include <linux/init.h>
 #include <linux/sched.h>
 #include <linux/slab.h>
 #include <linux/fs.h>
+#include <linux/user_namespace.h>
 #include "ext2.h"
 #include "xattr.h"
 #include "acl.h"
@@ -464,7 +465,8 @@ ext2_xattr_set_acl(struct inode *inode,

  if (!test_opt(inode->i_sb, POSIX_ACL))
   return -EOPNOTSUPP;
- if ((current->fsuid != inode->i_uid) && !capable(CAP_FOWNER))
+ if (!task_inode_same_fsuid(current, inode) &&
+   !task_ino_capable(inode, CAP_FOWNER))
   return -EPERM;

  if (value) {
diff --git a/fs/ext2/balloc.c b/fs/ext2/balloc.c
index 538c221..2924f0e 100644
--- a/fs/ext2/balloc.c
+++ b/fs/ext2/balloc.c
@@ -16,6 +16,7 @@ #include <linux/quotaops.h>
 #include <linux/sched.h>
```

```
 #include <linux/buffer_head.h>
 #include <linux/capability.h>
+#include <linux/user_namespace.h>

 /*
  * balloc.c contains the blocks allocation and deallocation routines
@@ -1127,9 +1128,15 @@ static int ext2_has_free_blocks(struct e

  free_blocks = percpu_counter_read_positive(&sbi->s_freeblocks_counter);
  root_blocks = le32_to_cpu(sbi->s_es->s_r_blocks_count);
- if (free_blocks < root_blocks + 1 && !capable(CAP_SYS_RESOURCE) &&
-  sbi->s_resuid != current->fsuid &&
-  (sbi->s_resgid == 0 || !in_group_p (sbi->s_resgid))) {
+ if (free_blocks < root_blocks + 1) {
+  if (sbi->s_resuidns != task_user_ns(current))
+   return 0;
+  if (capable(CAP_SYS_RESOURCE))
+   return 1;
+  if (sbi->s_resuid == current->fsuid)
+   return 1;
+  if (sbi->s_resgid != 0 && in_group_p (sbi->s_resgid))
+   return 1;
   return 0;
  }
  return 1;
diff --git a/fs/ext2/ialloc.c b/fs/ext2/ialloc.c
index 86a2f3b..34f16ab 100644
--- a/fs/ext2/ialloc.c
+++ b/fs/ext2/ialloc.c
@@ -17,6 +17,7 @@ #include <linux/sched.h>
 #include <linux/backing-dev.h>
 #include <linux/buffer_head.h>
 #include <linux/random.h>
+#include <linux/user_namespace.h>
 #include "ext2.h"
 #include "xattr.h"
 #include "acl.h"
@@ -133,6 +134,9 @@ void ext2_free_inode (struct inode * ino
  /* Do this BEFORE marking the inode not in use or returning an error */
  clear_inode (inode);

+ put_user_ns(inode->i_userns);
+ inode->i_userns = NULL;
+
  if (ino < EXT2_FIRST_INO(sb) ||
      ino > le32_to_cpu(es->s_inodes_count)) {
   ext2_error (sb, "ext2_free_inode",
@@ -563,6 +567,7 @@ got:
```

```
  sb->s_dirt = 1;
  mark_buffer_dirty(bh2);
  inode->i_uid = current->fsuid;
+ inode->i_userns = get_task_user_ns(current);
  if (test_opt (sb, GRPID))
   inode->i_gid = dir->i_gid;
  else if (dir->i_mode & S_ISGID) {
diff --git a/fs/ext2/inode.c b/fs/ext2/inode.c
index 5deb23b..dcca474 100644
--- a/fs/ext2/inode.c
+++ b/fs/ext2/inode.c
@@ -31,6 +31,7 @@ #include <linux/module.h>
 #include <linux/writeback.h>
 #include <linux/buffer_head.h>
 #include <linux/mpage.h>
+#include <linux/user_namespace.h>
 #include "ext2.h"
 #include "acl.h"
 #include "xip.h"
@@ -1168,6 +1169,7 @@ #endif
  inode->i_mode = le16_to_cpu(raw_inode->i_mode);
  inode->i_uid = (uid_t)le16_to_cpu(raw_inode->i_uid_low);
  inode->i_gid = (gid_t)le16_to_cpu(raw_inode->i_gid_low);
+ inode->i_userns = get_task_user_ns(current);
  if (!(test_opt (inode->i_sb, NO_UID32))) {
   inode->i_uid |= le16_to_cpu(raw_inode->i_uid_high) << 16;
   inode->i_gid |= le16_to_cpu(raw_inode->i_gid_high) << 16;
diff --git a/fs/ext2/ioctl.c b/fs/ext2/ioctl.c
index 315a98b..2326804 100644
--- a/fs/ext2/ioctl.c
+++ b/fs/ext2/ioctl.c
@@ -13,6 +13,7 @@ #include <linux/time.h>
 #include <linux/sched.h>
 #include <linux/compat.h>
 #include <linux/smp_lock.h>
+#include <linux/user_namespace.h>
 #include <asm/current.h>
 #include <asm/uaccess.h>

@@ -36,7 +37,8 @@ int ext2_ioctl (struct inode * inode, st
   if (IS_RDONLY(inode))
    return -EROFS;

-  if ((current->fsuid != inode->i_uid) && !capable(CAP_FOWNER))
+   if (!task_inode_same_fsuid(current, inode) &&
+    !task_ino_capable(inode, CAP_FOWNER))
    return -EACCES;
```

```
   if (get_user(flags, (int __user *) arg))
@@ -55,7 +57,7 @@ int ext2_ioctl (struct inode * inode, st
    * This test looks nicer. Thanks to Pauline Middelink
    */
   if ((flags ^ oldflags) & (EXT2_APPEND_FL | EXT2_IMMUTABLE_FL)) {
-   if (!capable(CAP_LINUX_IMMUTABLE)) {
+   if (!task_ino_capable(inode, CAP_LINUX_IMMUTABLE)) {
     mutex_unlock(&inode->i_mutex);
     return -EPERM;
    }
@@ -74,7 +76,8 @@ int ext2_ioctl (struct inode * inode, st
  case EXT2_IOC_GETVERSION:
   return put_user(inode->i_generation, (int __user *) arg);
  case EXT2_IOC_SETVERSION:
-  if ((current->fsuid != inode->i_uid) && !capable(CAP_FOWNER))
+  if (!task_inode_same_fsuid(current, inode) &&
+      !task_ino_capable(inode, CAP_FOWNER))
    return -EPERM;
   if (IS_RDONLY(inode))
    return -EROFS;
@@ -99,7 +102,8 @@ int ext2_ioctl (struct inode * inode, st
   if (IS_RDONLY(inode))
    return -EROFS;

-  if ((current->fsuid != inode->i_uid) && !capable(CAP_FOWNER))
+  if (!task_inode_same_fsuid(current, inode) &&
+      !task_ino_capable(inode, CAP_FOWNER))
    return -EACCES;

   if (get_user(rsv_window_size, (int __user *)arg))
diff --git a/fs/ext2/super.c b/fs/ext2/super.c
index 932579b..75ce9e8 100644
--- a/fs/ext2/super.c
+++ b/fs/ext2/super.c
@@ -29,6 +29,7 @@ #include <linux/smp_lock.h>
 #include <linux/vfs.h>
 #include <linux/seq_file.h>
 #include <linux/mount.h>
+#include <linux/user_namespace.h>
 #include <asm/uaccess.h>
 #include "ext2.h"
 #include "xattr.h"
@@ -125,6 +126,7 @@ static void ext2_put_super (struct super
    brelse (sbi->s_group_desc[i]);
  kfree(sbi->s_group_desc);
  kfree(sbi->s_debts);
+ put_user_ns(sbi->s_resuidns);
  percpu_counter_destroy(&sbi->s_freeblocks_counter);
```

```
  percpu_counter_destroy(&sbi->s_freeinodes_counter);
  percpu_counter_destroy(&sbi->s_dirs_counter);
@@ -742,6 +744,7 @@ #endif

  sbi->s_resuid = le16_to_cpu(es->s_def_resuid);
  sbi->s_resgid = le16_to_cpu(es->s_def_resgid);
+ sbi->s_resuidns = get_task_user_ns(current);

  set_opt(sbi->s_mount_opt, RESERVATION);

@@ -990,6 +993,7 @@ failed_mount_group_desc:
  kfree(sbi->s_group_desc);
  kfree(sbi->s_debts);
 failed_mount:
+ put_user_ns(sbi->s_resuidns);
  brelse(bh);
 failed_sbi:
  sb->s_fs_info = NULL;
diff --git a/fs/ext2/xattr_trusted.c b/fs/ext2/xattr_trusted.c
index f28a6a4..d24729a 100644
--- a/fs/ext2/xattr_trusted.c
+++ b/fs/ext2/xattr_trusted.c
@@ -11,6 +11,7 @@ #include <linux/capability.h>
 #include <linux/fs.h>
 #include <linux/smp_lock.h>
 #include <linux/ext2_fs.h>
+#include <linux/user_namespace.h>
 #include "xattr.h"

 #define XATTR_TRUSTED_PREFIX "trusted."
@@ -22,7 +23,7 @@ ext2_xattr_trusted_list(struct inode *in
  const int prefix_len = sizeof(XATTR_TRUSTED_PREFIX)-1;
  const size_t total_len = prefix_len + name_len + 1;

- if (!capable(CAP_SYS_ADMIN))
+ if (!task_ino_capable(inode, CAP_SYS_ADMIN))
   return 0;

  if (list && total_len <= list_size) {
diff --git a/include/linux/ext2_fs_sb.h b/include/linux/ext2_fs_sb.h
index b1b69bc..bdbc7cd 100644
--- a/include/linux/ext2_fs_sb.h
+++ b/include/linux/ext2_fs_sb.h
@@ -85,6 +85,7 @@ struct ext2_sb_info {
  unsigned long  s_mount_opt;
  uid_t s_resuid;
  gid_t s_resgid;
+ struct user_namespace *s_resuidns;
```

```
  unsigned short s_mount_state;
  unsigned short s_pad;
  int s_addr_per_block_bits;
diff --git a/include/linux/user_namespace.h b/include/linux/user_namespace.h
index 073f3e0..e87d6f5 100644
--- a/include/linux/user_namespace.h
+++ b/include/linux/user_namespace.h
@@ -27,6 +27,8 @@ static inline struct user_namespace *get
  return ns;
 }

+extern struct user_namespace *get_task_user_ns(struct task_struct *tsk);
+
 extern struct user_namespace *copy_user_ns(int flags,
         struct user_namespace *old_ns);
 extern void free_user_ns(struct kref *kref);
@@ -76,6 +78,10 @@ task_inode_same_fsuid(struct task_struct
  return 1;
 }

+/* the set of helpers is really getting out of hand.  consolidate
+ * in thr next release
+ */
+
 /*
  * task_ino_capable:
  * Again, when userns keys exist, we will need to check for a
@@ -95,6 +101,16 @@ static inline struct user_namespace *get
  return &init_user_ns;
 }

+static inline struct user_namespace *task_user_ns(struct task_struct *tsk)
+{
+ return &init_user_ns;
+}
+
+static inline struct user_namespace *get_task_user_ns(struct task_struct *tsk)
+{
+ return &init_user_ns;
+}
+
 static inline struct user_namespace *copy_user_ns(int flags,
         struct user_namespace *old_ns)
 {
--
1.3.2
```

Subject: Re: [PATCH 5/6] userns strict: hook ext2
Posted by xemul on Tue, 05 Jun 2007 10:16:37 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
>>From nobody Mon Sep 17 00:00:00 2001
> From: Serge Hallyn <serue@us.ibm.com>
> Date: Wed, 28 Mar 2007 15:06:47 -0500
> Subject: [PATCH 5/6] userns strict: hook ext2
>
> Add a user namespace pointer to the ext2 superblock and inode.
>
> Signed-off-by: Serge E. Hallyn <serue@us.ibm.com>
>
> ---
>
> fs/ext2/acl.c | 4 +++-
> fs/ext2/balloc.c | 13 ++++++++++---
> fs/ext2/ialloc.c | 5 +++++
> fs/ext2/inode.c | 2 ++
> fs/ext2/ioctl.c | 12 ++++++++----
> fs/ext2/super.c | 4 ++++
> fs/ext2/xattr_trusted.c | 3 ++-
> include/linux/ext2_fs_sb.h | 1 +
> include/linux/user_namespace.h | 16 ++++++++++++++++
> 9 files changed, 51 insertions(+), 9 deletions(-)
>

[snip]

> diff --git a/fs/ext2/ialloc.c b/fs/ext2/ialloc.c
> index 86a2f3b..34f16ab 100644
> --- a/fs/ext2/ialloc.c
> +++ b/fs/ext2/ialloc.c
> @@ -17,6 +17,7 @@ #include <linux/sched.h>
> #include <linux/backing-dev.h>
> #include <linux/buffer_head.h>
> #include <linux/random.h>
> +#include <linux/user_namespace.h>
> #include "ext2.h"
> #include "xattr.h"
> #include "acl.h"
> @@ -133,6 +134,9 @@ void ext2_free_inode (struct inode * ino
>   /* Do this BEFORE marking the inode not in use or returning an error */
>   clear_inode (inode);
>
> + put_user_ns(inode->i_userns);
> + inode->i_userns = NULL;
> +

>   if (ino < EXT2_FIRST_INO(sb) ||
>   ino > le32_to_cpu(es->s_inodes_count)) {
>   ext2_error (sb, "ext2_free_inode",
> @@ -563,6 +567,7 @@ got:
>   sb->s_dirt = 1;
>   mark_buffer_dirty(bh2);
>   inode->i_uid = current->fsuid;
> + inode->i_userns = get_task_user_ns(current);

We have all the ext2 (and in the next patch - the ext3) inodes
attached to a particular user and prohibit access to the inodes
belonging to other tasks' namespaces, don't we?

If so how can we allow the admin of the node to configure the
root of a virtual server on the fly?

[snip]

> diff --git a/fs/ext2/super.c b/fs/ext2/super.c
> index 932579b..75ce9e8 100644
> --- a/fs/ext2/super.c
> +++ b/fs/ext2/super.c
> @@ -29,6 +29,7 @@ #include <linux/smp_lock.h>
> #include <linux/vfs.h>
> #include <linux/seq_file.h>
> #include <linux/mount.h>
> +#include <linux/user_namespace.h>
> #include <asm/uaccess.h>
> #include "ext2.h"
> #include "xattr.h"
> @@ -125,6 +126,7 @@ static void ext2_put_super (struct super
>   brelse (sbi->s_group_desc[i]);
>   kfree(sbi->s_group_desc);
>   kfree(sbi->s_debts);
> + put_user_ns(sbi->s_resuidns);
>   percpu_counter_destroy(&sbi->s_freeblocks_counter);
>   percpu_counter_destroy(&sbi->s_freeinodes_counter);
>   percpu_counter_destroy(&sbi->s_dirs_counter);
> @@ -742,6 +744,7 @@ #endif
>
>   sbi->s_resuid = le16_to_cpu(es->s_def_resuid);
>   sbi->s_resgid = le16_to_cpu(es->s_def_resgid);
> + sbi->s_resuidns = get_task_user_ns(current);
>
>   set_opt(sbi->s_mount_opt, RESERVATION);
>
> @@ -990,6 +993,7 @@ failed_mount_group_desc:
>   kfree(sbi->s_group_desc);

> kfree(sbi->s_debts);
> failed_mount:
> + put_user_ns(sbi->s_resuidns);
> brelse(bh);
> failed_sbi:
> sb->s_fs_info = NULL;

If we have a super block attached to a namespace, why not attach
the inodes to the same namespace, not to the task opening the inode?

Thanks,
Pavel

Subject: Re: [PATCH 5/6] userns strict: hook ext2
Posted by serue on Tue, 05 Jun 2007 12:37:36 GMT
View Forum Message <> Reply to Message

Quoting Pavel Emelianov (xemul@sw.ru):
> Serge E. Hallyn wrote:
> >>From nobody Mon Sep 17 00:00:00 2001
> > From: Serge Hallyn <serue@us.ibm.com>
> > Date: Wed, 28 Mar 2007 15:06:47 -0500
> > Subject: [PATCH 5/6] userns strict: hook ext2
> >
> > Add a user namespace pointer to the ext2 superblock and inode.
> >
> > Signed-off-by: Serge E. Hallyn <serue@us.ibm.com>
> >
> > ---
> >
> > fs/ext2/acl.c | 4 +++-
> > fs/ext2/balloc.c | 13 ++++++++++---
> > fs/ext2/ialloc.c | 5 +++++
> > fs/ext2/inode.c | 2 ++
> > fs/ext2/ioctl.c | 12 ++++++++----
> > fs/ext2/super.c | 4 ++++
> > fs/ext2/xattr_trusted.c | 3 ++-
> > include/linux/ext2_fs_sb.h | 1 +
> > include/linux/user_namespace.h | 16 ++++++++++++++++
> > 9 files changed, 51 insertions(+), 9 deletions(-)
> >
>
> [snip]
>
> > diff --git a/fs/ext2/ialloc.c b/fs/ext2/ialloc.c
> > index 86a2f3b..34f16ab 100644
> > --- a/fs/ext2/ialloc.c

> > +++ b/fs/ext2/ialloc.c
> > @@ -17,6 +17,7 @@ #include <linux/sched.h>
> > #include <linux/backing-dev.h>
> > #include <linux/buffer_head.h>
> > #include <linux/random.h>
> > +#include <linux/user_namespace.h>
> > #include "ext2.h"
> > #include "xattr.h"
> > #include "acl.h"
> > @@ -133,6 +134,9 @@ void ext2_free_inode (struct inode * ino
> >   /* Do this BEFORE marking the inode not in use or returning an error */
> >   clear_inode (inode);
> >
> > + put_user_ns(inode->i_userns);
> > + inode->i_userns = NULL;
> > +
> >   if (ino < EXT2_FIRST_INO(sb) ||
> >   ino > le32_to_cpu(es->s_inodes_count)) {
> >   ext2_error (sb, "ext2_free_inode",
> > @@ -563,6 +567,7 @@ got:
> >   sb->s_dirt = 1;
> >   mark_buffer_dirty(bh2);
> >   inode->i_uid = current->fsuid;
> > + inode->i_userns = get_task_user_ns(current);
>
> We have all the ext2 (and in the next patch - the ext3) inodes
> attached to a particular user and prohibit access to the inodes
> belonging to other tasks' namespaces, don't we?

No. We provide access as though they were nobody/other. So for a large chunk of the fs that would usually be read-only access.

> If so how can we allow the admin of the node to configure the
> root of a virtual server on the fly?

See below regarding using keys to set userns on an inode, but really the idea here is that we would use the in-kernel keyring to store access rights to other user namespaces, rather than try to do much with the owners of a filesystem.

So the initial userns might be the owner of all of /usr, which has 755 perms, so all namespaces get read and execute permission. Root in one user namespace might have a key saying (init_user_ns, uid 500), meaning that root in that user namespace would have full access in the initial user namespace as though it were uid 500. (Which presumably is who started the user namespace)

> [snip]
>
> > diff --git a/fs/ext2/super.c b/fs/ext2/super.c
> > index 932579b..75ce9e8 100644
> > --- a/fs/ext2/super.c
> > +++ b/fs/ext2/super.c
> > @@ -29,6 +29,7 @@ #include <linux/smp_lock.h>
> > #include <linux/vfs.h>
> > #include <linux/seq_file.h>
> > #include <linux/mount.h>
> > +#include <linux/user_namespace.h>
> > #include <asm/uaccess.h>
> > #include "ext2.h"
> > #include "xattr.h"
> > @@ -125,6 +126,7 @@ static void ext2_put_super (struct super
> >   brelse (sbi->s_group_desc[i]);
> >   kfree(sbi->s_group_desc);
> >   kfree(sbi->s_debts);
> > + put_user_ns(sbi->s_resuidns);
> >   percpu_counter_destroy(&sbi->s_freeblocks_counter);
> >   percpu_counter_destroy(&sbi->s_freeinodes_counter);
> >   percpu_counter_destroy(&sbi->s_dirs_counter);
> > @@ -742,6 +744,7 @@ #endif
> >
> >   sbi->s_resuid = le16_to_cpu(es->s_def_resuid);
> >   sbi->s_resgid = le16_to_cpu(es->s_def_resgid);
> > + sbi->s_resuidns = get_task_user_ns(current);
> >
> >   set_opt(sbi->s_mount_opt, RESERVATION);
> >
> > @@ -990,6 +993,7 @@ failed_mount_group_desc:
> >   kfree(sbi->s_group_desc);
> >   kfree(sbi->s_debts);
> > failed_mount:
> > + put_user_ns(sbi->s_resuidns);
> >   brelse(bh);
> > failed_sbi:
> >   sb->s_fs_info = NULL;
>
> If we have a super block attached to a namespace, why not attach
> the inodes to the same namespace, not to the task opening the inode?

Well shoot - that is what I intended to do, and had done in some
version. Must have gotten patches mixed up.

Yes, like you suggest, for "mostly userns-ignorant" filesystems that was
my intent. userns-savvy filesystems could of course parse some userns
xattr to set a userns. Not 100% sure how we'd do that, but I suppose

when a userns started up it could register some secret key which is then
placed in a table next to the userns *, and that key is what is stored
in the xattr.

-serge

Subject: Re: [PATCH 1/6] user namespace : add the framework
Posted by Pavel Emelianov on Fri, 08 Jun 2007 08:35:04 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
>>From nobody Mon Sep 17 00:00:00 2001
> From: Cedric Le Goater <clg@fr.ibm.com>
> Date: Thu, 5 Apr 2007 12:51:51 -0400
> Subject: [PATCH 1/6] user namespace : add the framework
>
> Add the user namespace struct and framework
>
> Basically, it will allow a process to unshare its user_struct table, resetting
> at the same time its own user_struct and all the associated accounting.
>
> A new root user (uid == 0) is added to the user namespace upon creation. Such
> root users have full privileges and it seems that theses privileges should be
> controlled through some means (process capabilities ?)
>
> The unshare is not included in this patch.
>
> Changes since [try #4]:
> - Updated get_user_ns and put_user_ns to accept NULL, and
>   get_user_ns to return the namespace.
>
> Changes since [try #3]:
> - moved struct user_namespace to files user_namespace.{c,h}
>
> Changes since [try #2]:
> - removed struct user_namespace* argument from find_user()
>
> Changes since [try #1]:
> - removed struct user_namespace* argument from find_user()
> - added a root_user per user namespace
>
> Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>
>
> Signed-off-by: Serge E. Hallyn <serue@us.ibm.com>
> Cc: Herbert Poetzl <herbert@13thfloor.at>
> Cc: Kirill Korotaev <dev@sw.ru>
> Cc: "Eric W. Biederman" <ebiederm@xmission.com>

> Signed-off-by: Andrew Morton <akpm@osdl.org>
>
> Signed-off-by: Serge E. Hallyn <serue@us.ibm.com>

Acked-by: Pavel Emelianov <xemul@openvz.org>

Yup. This patch is good :)

> ---
>
> include/linux/init_task.h | 2 +
> include/linux/nsproxy.h | 1 +
> include/linux/sched.h | 3 +-
> include/linux/user_namespace.h | 57 ++
> init/Kconfig | 9 ++++++
> kernel/Makefile | 2 +
> kernel/fork.c | 2 +
> kernel/nsproxy.c | 10 +++++++
> kernel/sys.c | 5 ++--
> kernel/user.c | 18 ++++++++-------
> kernel/user_namespace.c | 43 ++++++++++++++++++++++++++++++++
> 11 files changed, 138 insertions(+), 14 deletions(-)
> create mode 100644 include/linux/user_namespace.h
> create mode 100644 kernel/user_namespace.c
>
> cfeebe74a11622089791ecb14070f4d620f55535
> diff --git a/include/linux/init_task.h b/include/linux/init_task.h
> index cca7bb1..1fdc46c 100644
> --- a/include/linux/init_task.h
> +++ b/include/linux/init_task.h
> @@ -8,6 +8,7 @@ #include <linux/utsname.h>
> #include <linux/lockdep.h>
> #include <linux/ipc.h>
> #include <linux/pid_namespace.h>
> +#include <linux/user_namespace.h>
>
> #define INIT_FDTABLE \
> {   \
> @@ -78,6 +79,7 @@ #define INIT_NSPROXY(nsproxy) {   \
>   .uts_ns = &init_uts_ns,   \
>   .mnt_ns = NULL,   \
>   INIT_IPC_NS(ipc_ns)   \
> + .user_ns = &init_user_ns,   \
> }
>
> #define INIT_SIGHAND(sighand) {   \
> diff --git a/include/linux/nsproxy.h b/include/linux/nsproxy.h
> index 616a710..36e84e5 100644

> --- a/include/linux/nsproxy.h
> +++ b/include/linux/nsproxy.h
> @@ -28,6 +28,7 @@ struct nsproxy {
> struct ipc_namespace *ipc_ns;
> struct mnt_namespace *mnt_ns;
> struct pid_namespace *pid_ns;
> + struct user_namespace *user_ns;
> };
> extern struct nsproxy init_nsproxy;
>
> diff --git a/include/linux/sched.h b/include/linux/sched.h
> index 8016ec8..259d0a5 100644
> --- a/include/linux/sched.h
> +++ b/include/linux/sched.h
> @@ -263,6 +263,7 @@ extern signed long schedule_timeout_unin
> asmlinkage void schedule(void);
>
> struct nsproxy;
> +struct user_namespace;
>
> /* Maximum number of active map areas.. This is a random (large) number */
> #define DEFAULT_MAX_MAP_COUNT 65536
> @@ -1311,7 +1312,7 @@ extern struct task_struct *find_task_by_
> extern void __set_special_pids(pid_t session, pid_t pgrp);
>
> /* per-UID process charging. */
> -extern struct user_struct * alloc_uid(uid_t);
> +extern struct user_struct * alloc_uid(struct user_namespace *, uid_t);
> static inline struct user_struct *get_uid(struct user_struct *u)
> {
> atomic_inc(&u->__count);
> diff --git a/include/linux/user_namespace.h b/include/linux/user_namespace.h
> new file mode 100644
> index 0000000..c2debd8
> --- /dev/null
> +++ b/include/linux/user_namespace.h
> @@ -0,0 +1,57 @@
> +#ifndef _LINUX_USER_NAMESPACE_H
> +#define _LINUX_USER_NAMESPACE_H
> +
> +#include <linux/kref.h>
> +#include <linux/nsproxy.h>
> +#include <linux/sched.h>
> +
> +#define UIDHASH_BITS (CONFIG_BASE_SMALL ? 3 : 8)
> +#define UIDHASH_SZ (1 << UIDHASH_BITS)
> +
> +struct user_namespace {

> + struct kref kref;
> + struct list_head uidhash_table[UIDHASH_SZ];
> + struct user_struct *root_user;
> +};
> +
> +extern struct user_namespace init_user_ns;
> +
> +#ifdef CONFIG_USER_NS
> +
> +static inline struct user_namespace *get_user_ns(struct user_namespace *ns)
> +{
> + if (ns)
> + kref_get(&ns->kref);
> + return ns;
> +}
> +
> +extern struct user_namespace *copy_user_ns(int flags,
> +   struct user_namespace *old_ns);
> +extern void free_user_ns(struct kref *kref);
> +
> +static inline void put_user_ns(struct user_namespace *ns)
> +{
> + if (ns)
> + kref_put(&ns->kref, free_user_ns);
> +}
> +
> +#else
> +
> +static inline struct user_namespace *get_user_ns(struct user_namespace *ns)
> +{
> + return &init_user_ns;
> +}
> +
> +static inline struct user_namespace *copy_user_ns(int flags,
> +   struct user_namespace *old_ns)
> +{
> + return 0;
> +}
> +
> +static inline void put_user_ns(struct user_namespace *ns)
> +{
> +}
> +
> +#endif
> +
> +#endif /* _LINUX_USER_H */
> diff --git a/init/Kconfig b/init/Kconfig
> index 99b0b32..0bbf957 100644

```
> --- a/init/Kconfig
> +++ b/init/Kconfig
> @@ -229,6 +229,15 @@ config TASK_IO_ACCOUNTING
>
>     Say N if unsure.
>
> +config USER_NS
> + bool "User Namespaces (EXPERIMENTAL)"
> + default n
> + depends on EXPERIMENTAL
> + help
> +   Support user namespaces.  This allows containers, i.e.
> +   vservers, to use user namespaces to provide different
> +   user info for different servers.  If unsure, say N.
> +
>  config AUDIT
>   bool "Auditing support"
>   depends on NET
> diff --git a/kernel/Makefile b/kernel/Makefile
> index 6101981..b5cef19 100644
> --- a/kernel/Makefile
> +++ b/kernel/Makefile
> @@ -4,7 +4,7 @@ #
>
>  obj-y     = sched.o fork.o exec_domain.o panic.o printk.o profile.o \
>       exit.o itimer.o time.o softirq.o resource.o \
> -     sysctl.o capability.o timer.o user.o \
> +     sysctl.o capability.o timer.o user.o user_namespace.o \
>       signal.o sys.o kmod.o workqueue.o pid.o \
>       rcupdate.o extable.o params.o posix-timers.o \
>       kthread.o wait.o kfifo.o sys_ni.o posix-cpu-timers.o mutex.o \
> diff --git a/kernel/fork.c b/kernel/fork.c
> index 6ebd35b..c20137e 100644
> --- a/kernel/fork.c
> +++ b/kernel/fork.c
> @@ -999,7 +999,7 @@ #endif
>   if (atomic_read(&p->user->processes) >=
>     p->signal->rlim[RLIMIT_NPROC].rlim_cur) {
>    if (!capable(CAP_SYS_ADMIN) && !capable(CAP_SYS_RESOURCE) &&
> -    p->user != &root_user)
> +     p->user != current->nsproxy->user_ns->root_user)
>     goto bad_fork_free;
>   }
>
> diff --git a/kernel/nsproxy.c b/kernel/nsproxy.c
> index 3f3b7ad..1ea8a73 100644
> --- a/kernel/nsproxy.c
> +++ b/kernel/nsproxy.c
```

```
> @@ -83,8 +83,15 @@ static struct nsproxy *create_new_namesp
>   if (IS_ERR(new_nsp->pid_ns))
>    goto out_pid;
>
> + new_nsp->user_ns = copy_user_ns(flags, tsk->nsproxy->user_ns);
> + if (IS_ERR(new_nsp->user_ns))
> +  goto out_user;
> +
>   return new_nsp;
>
> +out_user:
> + if (new_nsp->pid_ns)
> +  put_pid_ns(new_nsp->pid_ns);
>  out_pid:
>   if (new_nsp->ipc_ns)
>    put_ipc_ns(new_nsp->ipc_ns);
> @@ -144,6 +151,9 @@ void free_nsproxy(struct nsproxy *ns)
>    put_ipc_ns(ns->ipc_ns);
>   if (ns->pid_ns)
>    put_pid_ns(ns->pid_ns);
> + if (ns->user_ns)
> +  put_user_ns(ns->user_ns);
> +
>   kmem_cache_free(nsproxy_cachep, ns);
>  }
>
> diff --git a/kernel/sys.c b/kernel/sys.c
> index 7d79146..bb37fcb 100644
> --- a/kernel/sys.c
> +++ b/kernel/sys.c
> @@ -34,6 +34,7 @@ #include <linux/task_io_accounting_ops.h
>  #include <linux/compat.h>
>  #include <linux/syscalls.h>
>  #include <linux/kprobes.h>
> +#include <linux/user_namespace.h>
>
>  #include <asm/uaccess.h>
>  #include <asm/io.h>
> @@ -1077,13 +1078,13 @@ static int set_user(uid_t new_ruid, int
>  {
>   struct user_struct *new_user;
>
> - new_user = alloc_uid(new_ruid);
> + new_user = alloc_uid(current->nsproxy->user_ns, new_ruid);
>   if (!new_user)
>    return -EAGAIN;
>
>   if (atomic_read(&new_user->processes) >=
```

```
>      current->signal->rlim[RLIMIT_NPROC].rlim_cur &&
> -   new_user != &root_user) {
> +   new_user != current->nsproxy->user_ns->root_user) {
>     free_uid(new_user);
>     return -EAGAIN;
>   }
> diff --git a/kernel/user.c b/kernel/user.c
> index 4869563..98b8250 100644
> --- a/kernel/user.c
> +++ b/kernel/user.c
> @@ -14,20 +14,19 @@ #include <linux/slab.h>
>  #include <linux/bitops.h>
>  #include <linux/key.h>
>  #include <linux/interrupt.h>
> +#include <linux/module.h>
> +#include <linux/user_namespace.h>
>
>  /*
>   * UID task count cache, to get fast user lookup in "alloc_uid"
>   * when changing user ID's (ie setuid() and friends).
>   */
>
> -#define UIDHASH_BITS (CONFIG_BASE_SMALL ? 3 : 8)
> -#define UIDHASH_SZ  (1 << UIDHASH_BITS)
>  #define UIDHASH_MASK  (UIDHASH_SZ - 1)
>  #define __uidhashfn(uid) (((uid >> UIDHASH_BITS) + uid) & UIDHASH_MASK)
> -#define uidhashentry(uid) (uidhash_table + __uidhashfn((uid)))
> +#define uidhashentry(ns, uid) ((ns)->uidhash_table + __uidhashfn((uid)))
>
>  static struct kmem_cache *uid_cachep;
> -static struct list_head uidhash_table[UIDHASH_SZ];
>
>  /*
>   * The uidhash_lock is mostly taken from process context, but it is
> @@ -94,9 +93,10 @@ struct user_struct *find_user(uid_t uid)
>  {
>   struct user_struct *ret;
>   unsigned long flags;
> + struct user_namespace *ns = current->nsproxy->user_ns;
>
>   spin_lock_irqsave(&uidhash_lock, flags);
> - ret = uid_hash_find(uid, uidhashentry(uid));
> + ret = uid_hash_find(uid, uidhashentry(ns, uid));
>   spin_unlock_irqrestore(&uidhash_lock, flags);
>   return ret;
>  }
> @@ -120,9 +120,9 @@ void free_uid(struct user_struct *up)
>   }
```

```
>  }
>
> -struct user_struct * alloc_uid(uid_t uid)
> +struct user_struct * alloc_uid(struct user_namespace *ns, uid_t uid)
>  {
> - struct list_head *hashent = uidhashentry(uid);
> + struct list_head *hashent = uidhashentry(ns, uid);
>   struct user_struct *up;
>
>   spin_lock_irq(&uidhash_lock);
> @@ -211,11 +211,11 @@ static int __init uid_cache_init(void)
>     0, SLAB_HWCACHE_ALIGN|SLAB_PANIC, NULL, NULL);
>
>   for(n = 0; n < UIDHASH_SZ; ++n)
> -  INIT_LIST_HEAD(uidhash_table + n);
> +  INIT_LIST_HEAD(init_user_ns.uidhash_table + n);
>
>   /* Insert the root user immediately (init already runs as root) */
>   spin_lock_irq(&uidhash_lock);
> - uid_hash_insert(&root_user, uidhashentry(0));
> + uid_hash_insert(&root_user, uidhashentry(&init_user_ns, 0));
>   spin_unlock_irq(&uidhash_lock);
>
>   return 0;
> diff --git a/kernel/user_namespace.c b/kernel/user_namespace.c
> new file mode 100644
> index 0000000..3d79642
> --- /dev/null
> +++ b/kernel/user_namespace.c
> @@ -0,0 +1,43 @@
> +/*
> + *  This program is free software; you can redistribute it and/or
> + *  modify it under the terms of the GNU General Public License as
> + *  published by the Free Software Foundation, version 2 of the
> + *  License.
> + */
> +
> +#include <linux/module.h>
> +#include <linux/version.h>
> +#include <linux/nsproxy.h>
> +#include <linux/user_namespace.h>
> +
> +struct user_namespace init_user_ns = {
> + .kref = {
> +  .refcount = ATOMIC_INIT(2),
> + },
> + .root_user = &root_user,
> +};
```

```
> +
> +EXPORT_SYMBOL_GPL(init_user_ns);
> +
> +#ifdef CONFIG_USER_NS
> +
> +struct user_namespace * copy_user_ns(int flags, struct user_namespace *old_ns)
> +{
> + struct user_namespace *new_ns;
> +
> + BUG_ON(!old_ns);
> + get_user_ns(old_ns);
> +
> + new_ns = old_ns;
> + return new_ns;
> +}
> +
> +void free_user_ns(struct kref *kref)
> +{
> + struct user_namespace *ns;
> +
> + ns = container_of(kref, struct user_namespace, kref);
> + kfree(ns);
> +}
> +
> +#endif /* CONFIG_USER_NS */
```

Subject: Re: [PATCH 1/6] user namespace : add the framework
Posted by akpm on Mon, 16 Jul 2007 01:31:32 GMT
View Forum Message <> Reply to Message

On Mon, 4 Jun 2007 14:40:24 -0500 "Serge E. Hallyn" <serue@us.ibm.com> wrote:

> Add the user namespace struct and framework
>
> Basically, it will allow a process to unshare its user_struct table, resetting
> at the same time its own user_struct and all the associated accounting.
>
> A new root user (uid == 0) is added to the user namespace upon creation. Such
> root users have full privileges and it seems that theses privileges should be
> controlled through some means (process capabilities ?)

The whole magical-uid-0-user thing in this patch seem just wrong to
me.

I'll merge it anyway, mainly because I want to merge _something_ (why oh
why do the git-tree guys leave everything to the last minute?) but it strikes
me that there's something fundamentally wrong whenever the kernel starts

"knowing" about the significance of UIDs in this fashion.

It worries me.

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

---

Subject: Re: [PATCH 1/6] user namespace : add the framework
Posted by serue on Mon, 16 Jul 2007 14:34:43 GMT
View Forum Message <> Reply to Message

Quoting Andrew Morton (akpm@linux-foundation.org):
> On Mon, 4 Jun 2007 14:40:24 -0500 "Serge E. Hallyn" <serue@us.ibm.com> wrote:
>
> > > Add the user namespace struct and framework
> > >
> > > Basically, it will allow a process to unshare its user_struct table, resetting
> > > at the same time its own user_struct and all the associated accounting.
> > >
> > > A new root user (uid == 0) is added to the user namespace upon creation. Such
> > > root users have full privileges and it seems that theses privileges should be
> > > controlled through some means (process capabilities ?)
>
> The whole magical-uid-0-user thing in this patch seem just wrong to
> me.
>
> I'll merge it anyway, mainly because I want to merge _something_ (why oh
> why do the git-tree guys leave everything to the last minute?) but it strikes
> me that there's something fundamentally wrong whenever the kernel starts
> "knowing" about the significance of UIDs in this fashion.

$(&(%

I thought I disagreed, but now I'm pretty sure I completely agree.

'root_user' exists in the kernel right now, but the root_user checks
which exist (in fork.c and sys.c) shouldn't actually be applied for root
in a container, since the container may not be trusted.

There is the root_user_keyring stuff - David, is that only special cased
so that root's keys can be statically allocated?

> It worries me.

Cedric, you've probably mentioned this before, but what is wrong with
the following patch? User namespaces still seem to work for me with

this, but maybe I'm testing wrong. Can you give it a shot?

thanks,
-serge

>From 743e4f5c15ff9ec4110adc9d06e39a3fb0541512 Mon Sep 17 00:00:00 2001
From: Serge E. Hallyn <serue@us.ibm.com>
Date: Mon, 16 Jul 2007 10:26:25 -0400
Subject: [PATCH 1/1] userns: remove uid0 logic

While 'root_user' is hard_coded in the kernel, as are uid 0
checks, removing the 'root_user' from a user namespace
does not appear to break user namespaces.

Signed-off-by: Serge E. Hallyn <serue@us.ibm.com>
---
 include/linux/user_namespace.h | 1 -
 kernel/fork.c | 2 +-
 kernel/sys.c | 2 +-
 kernel/user_namespace.c | 9 ---------
 4 files changed, 2 insertions(+), 12 deletions(-)

diff --git a/include/linux/user_namespace.h b/include/linux/user_namespace.h
index bb32057..fcb4f30 100644
--- a/include/linux/user_namespace.h
+++ b/include/linux/user_namespace.h
@@ -12,7 +12,6 @@
 struct user_namespace {
  struct kref kref;
  struct list_head uidhash_table[UIDHASH_SZ];
- struct user_struct *root_user;
 };

 extern struct user_namespace init_user_ns;
diff --git a/kernel/fork.c b/kernel/fork.c
index 8b6ba70..f5c7f49 100644
--- a/kernel/fork.c
+++ b/kernel/fork.c
@@ -1004,7 +1004,7 @@ static struct task_struct *copy_process(unsigned long clone_flags,
  if (atomic_read(&p->user->processes) >=
    p->signal->rlim[RLIMIT_NPROC].rlim_cur) {
   if (!capable(CAP_SYS_ADMIN) && !capable(CAP_SYS_RESOURCE) &&
-      p->user != current->nsproxy->user_ns->root_user)
+   p->user != &root_user)
    goto bad_fork_free;
  }

diff --git a/kernel/sys.c b/kernel/sys.c

```
index e01b5d1..809e416 100644
--- a/kernel/sys.c
+++ b/kernel/sys.c
@@ -1085,7 +1085,7 @@ static int set_user(uid_t new_ruid, int dumpclear)

  if (atomic_read(&new_user->processes) >=
      current->signal->rlim[RLIMIT_NPROC].rlim_cur &&
-    new_user != current->nsproxy->user_ns->root_user) {
+    new_user != &root_user) {
   free_uid(new_user);
   return -EAGAIN;
  }
diff --git a/kernel/user_namespace.c b/kernel/user_namespace.c
index 89a27e8..df11a27 100644
--- a/kernel/user_namespace.c
+++ b/kernel/user_namespace.c
@@ -14,7 +14,6 @@ struct user_namespace init_user_ns = {
  .kref = {
   .refcount = ATOMIC_INIT(2),
  },
- .root_user = &root_user,
 };

 EXPORT_SYMBOL_GPL(init_user_ns);
@@ -41,17 +40,9 @@ static struct user_namespace *clone_user_ns(struct user_namespace
*old_ns)
  for (n = 0; n < UIDHASH_SZ; ++n)
   INIT_LIST_HEAD(ns->uidhash_table + n);

- /* Insert new root user.  */
- ns->root_user = alloc_uid(ns, 0);
- if (!ns->root_user) {
-  kfree(ns);
-  return NULL;
- }
-
  /* 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 NULL;
  }
--
1.5.1.1.GIT
```

_____
Containers mailing list

Subject: Re: [PATCH 1/6] user namespace : add the framework
Posted by serue on Mon, 16 Jul 2007 14:38:48 GMT
View Forum Message <> Reply to Message

Quoting Serge E. Hallyn (serue@us.ibm.com):
> Quoting Andrew Morton (akpm@linux-foundation.org):
> > On Mon, 4 Jun 2007 14:40:24 -0500 "Serge E. Hallyn" <serue@us.ibm.com> wrote:
> >
> > > Add the user namespace struct and framework
> > >
> > > Basically, it will allow a process to unshare its user_struct table, resetting
> > > at the same time its own user_struct and all the associated accounting.
> > >
> > > A new root user (uid == 0) is added to the user namespace upon creation. Such
> > > root users have full privileges and it seems that theses privileges should be
> > > controlled through some means (process capabilities ?)
> >
> > The whole magical-uid-0-user thing in this patch seem just wrong to
> > me.
> >
> > I'll merge it anyway, mainly because I want to merge _something_ (why oh
> > why do the git-tree guys leave everything to the last minute?) but it strikes
> > me that there's something fundamentally wrong whenever the kernel starts
> > "knowing" about the significance of UIDs in this fashion.
>
> $(&(%
>
> I thought I disagreed, but now I'm pretty sure I completely agree.
>
> 'root_user' exists in the kernel right now, but the root_user checks
> which exist (in fork.c and sys.c) shouldn't actually be applied for root
> in a container, since the container may not be trusted.

By the way, I don't think these two uses of 'user == &root_user' are
legitimate. CAP_SYS_RESOURCE should probably be checked for both. Hah,
it already is in fork.c, in addition to root_user check.

But I guess I should take that up elsewhere.

thanks,
-serge

_____
Containers mailing list
Containers@lists.linux-foundation.org

Subject: Re: [PATCH 1/6] user namespace : add the framework
Posted by dev on Mon, 16 Jul 2007 14:54:28 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting Andrew Morton (akpm@linux-foundation.org):
>
>>On Mon, 4 Jun 2007 14:40:24 -0500 "Serge E. Hallyn" <serue@us.ibm.com> wrote:
>>
>>
>>>Add the user namespace struct and framework
>>>
>>>Basically, it will allow a process to unshare its user_struct table, resetting
>>>at the same time its own user_struct and all the associated accounting.
>>>
>>>A new root user (uid == 0) is added to the user namespace upon creation. Such
>>>root users have full privileges and it seems that theses privileges should be
>>>controlled through some means (process capabilities ?)
>>
>>The whole magical-uid-0-user thing in this patch seem just wrong to
>>me.
>>
>>I'll merge it anyway, mainly because I want to merge _something_ (why oh
>>why do the git-tree guys leave everything to the last minute?) but it strikes
>>me that there's something fundamentally wrong whenever the kernel starts
>>"knowing" about the significance of UIDs in this fashion.
>
>
> $(&(%
>
> I thought I disagreed, but now I'm pretty sure I completely agree.
>
> 'root_user' exists in the kernel right now, but the root_user checks
> which exist (in fork.c and sys.c) shouldn't actually be applied for root
> in a container, since the container may not be trusted.

This rlimit check doesn't help *untrusted* containers, so your logic is wrong here.
Instead, it allows root of the container to operate in any situation.
E.g. consider root user hit the limit. After that you won't be able to login/ssh to fix anything.

NOTE: container root can have no CAP_SYS_RESOURCE and CAP_SYS_ADMIN as it is in OpenVZ.

But in general I'm not against the patch, since in OpenVZ we can replace the check
with another capability we use for VE admin - CAP_VE_SYS_ADMIN.

Kirill

> There is the root_user_keyring stuff - David, is that only special cased
> so that root's keys can be statically allocated?
>
>
>>It worries me.
>
>
> Cedric, you've probably mentioned this before, but what is wrong with
> the following patch? User namespaces still seem to work for me with
> this, but maybe I'm testing wrong. Can you give it a shot?
>
> thanks,
> -serge
>
>>From 743e4f5c15ff9ec4110adc9d06e39a3fb0541512 Mon Sep 17 00:00:00 2001
> From: Serge E. Hallyn <serue@us.ibm.com>
> Date: Mon, 16 Jul 2007 10:26:25 -0400
> Subject: [PATCH 1/1] userns: remove uid0 logic
>
> While 'root_user' is hard_coded in the kernel, as are uid 0
> checks, removing the 'root_user' from a user namespace
> does not appear to break user namespaces.
>
> Signed-off-by: Serge E. Hallyn <serue@us.ibm.com>
> ---
> include/linux/user_namespace.h | 1 -
> kernel/fork.c | 2 +-
> kernel/sys.c | 2 +-
> kernel/user_namespace.c | 9 ---------
> 4 files changed, 2 insertions(+), 12 deletions(-)
>
> diff --git a/include/linux/user_namespace.h b/include/linux/user_namespace.h
> index bb32057..fcb4f30 100644
> --- a/include/linux/user_namespace.h
> +++ b/include/linux/user_namespace.h
> @@ -12,7 +12,6 @@
> struct user_namespace {
>   struct kref kref;
>   struct list_head uidhash_table[UIDHASH_SZ];
> - struct user_struct *root_user;
> };
>
> extern struct user_namespace init_user_ns;
> diff --git a/kernel/fork.c b/kernel/fork.c
> index 8b6ba70..f5c7f49 100644

```
> --- a/kernel/fork.c
> +++ b/kernel/fork.c
> @@ -1004,7 +1004,7 @@ static struct task_struct *copy_process(unsigned long clone_flags,
>   if (atomic_read(&p->user->processes) >=
>     p->signal->rlim[RLIMIT_NPROC].rlim_cur) {
>    if (!capable(CAP_SYS_ADMIN) && !capable(CAP_SYS_RESOURCE) &&
> -      p->user != current->nsproxy->user_ns->root_user)
> +      p->user != &root_user)
>     goto bad_fork_free;
>   }
>
> diff --git a/kernel/sys.c b/kernel/sys.c
> index e01b5d1..809e416 100644
> --- a/kernel/sys.c
> +++ b/kernel/sys.c
> @@ -1085,7 +1085,7 @@ static int set_user(uid_t new_ruid, int dumpclear)
>
>   if (atomic_read(&new_user->processes) >=
>       current->signal->rlim[RLIMIT_NPROC].rlim_cur &&
> -   new_user != current->nsproxy->user_ns->root_user) {
> +   new_user != &root_user) {
>    free_uid(new_user);
>    return -EAGAIN;
>   }
> diff --git a/kernel/user_namespace.c b/kernel/user_namespace.c
> index 89a27e8..df11a27 100644ldn't actually be applied for root
in a container, since the container may not be trus
> --- a/kernel/user_namespace.c
> +++ b/kernel/user_namespace.c
> @@ -14,7 +14,6 @@ struct user_namespace init_user_ns = {
>   .kref = {
>    .refcount = ATOMIC_INIT(2),
>   },
> - .root_user = &root_user,
>  };
>
>  EXPORT_SYMBOL_GPL(init_user_ns);
> @@ -41,17 +40,9 @@ static struct user_namespace *clone_user_ns(struct user_namespace
*old_ns)
>   for (n = 0; n < UIDHASH_SZ; ++n)
>    INIT_LIST_HEAD(ns->uidhash_table + n);
>
> - /* Insert new root user.  */
> - ns->root_user = alloc_uid(ns, 0);
> - if (!ns->root_user) {
> -  kfree(ns);
> -  return NULL;
> - }
```

> -
> /* 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 NULL;
> }

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

---

Subject: Re: [PATCH 1/6] user namespace : add the framework
Posted by serue on Mon, 16 Jul 2007 15:08:00 GMT
View Forum Message <> Reply to Message

Quoting Kirill Korotaev (dev@sw.ru):
> Serge E. Hallyn wrote:
> > Quoting Andrew Morton (akpm@linux-foundation.org):
> >
> >>On Mon, 4 Jun 2007 14:40:24 -0500 "Serge E. Hallyn" <serue@us.ibm.com> wrote:
> >>
> >>
> >>>Add the user namespace struct and framework
> >>>
> >>>Basically, it will allow a process to unshare its user_struct table, resetting
> >>>at the same time its own user_struct and all the associated accounting.
> >>>
> >>>A new root user (uid == 0) is added to the user namespace upon creation. Such
> >>>root users have full privileges and it seems that theses privileges should be
> >>>controlled through some means (process capabilities ?)
> >>
> >>The whole magical-uid-0-user thing in this patch seem just wrong to
> >>me.
> >>
> >>I'll merge it anyway, mainly because I want to merge _something_ (why oh
> >>why do the git-tree guys leave everything to the last minute?) but it strikes
> >>me that there's something fundamentally wrong whenever the kernel starts
> >>"knowing" about the significance of UIDs in this fashion.
> >
> >
> > $(&(%
> >
> > I thought I disagreed, but now I'm pretty sure I completely agree.
> >

> > 'root_user' exists in the kernel right now, but the root_user checks
> > which exist (in fork.c and sys.c) shouldn't actually be applied for root
> > in a container, since the container may not be trusted.
>
> This rlimit check doesn't help *untrusted* containers, so your logic is wrong here.
> Instead, it allows root of the container to operate in any situation.

And I'm not sure that should be the case.

In my view, root of a container is equivalent to a normal user on the
host system, just like root in a qemu process.

> E.g. consider root user hit the limit. After that you won't be able to login/ssh to fix anything.

That's fine in the container.

> NOTE: container root can have no CAP_SYS_RESOURCE and CAP_SYS_ADMIN as it is in OpenVZ.

And eventually we'll want that to be the default in upstream containers.
But it's not the case upstream right now. Before we can do that, we
need an answer to per-container capabilities.

Do you (either you specifically, or anyone at openvz) have plans to
address the per-container capabilities problem? Herbert? Eric?

I'm interested, but would like to get the file capabilites squared away
before I consider coding on it.

> But in general I'm not against the patch, since in OpenVZ we can replace the check
> with another capability we use for VE admin - CAP_VE_SYS_ADMIN.

If that truly sufficies then great. If not, then in order to support
openvz in the meantime I say we drop my patch, but we remember that when
we straighten out the security issues this will need to be addressed.

thanks,
-serge

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

---

Subject: Re: [PATCH 1/6] user namespace : add the framework
Posted by serue on Mon, 16 Jul 2007 15:27:49 GMT
View Forum Message <> Reply to Message

Quoting David Howells (dhowells@redhat.com):
> Serge E. Hallyn <serue@us.ibm.com> wrote:
>
> > There is the root_user_keyring stuff - David, is that only special cased
> > so that root's keys can be statically allocated?
>
> It's because the boot processes start up with UID 0, and the user_struct for
> UID 0 is statically allocated (root_user).

Right - thanks David.

-serge

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

---

Subject: Re: [PATCH 1/6] user namespace : add the framework
Posted by Herbert Poetzl on Wed, 18 Jul 2007 00:11:35 GMT
View Forum Message <> Reply to Message

On Mon, Jul 16, 2007 at 10:08:00AM -0500, Serge E. Hallyn wrote:
> Quoting Kirill Korotaev (dev@sw.ru):
> > Serge E. Hallyn wrote:
> > > Quoting Andrew Morton (akpm@linux-foundation.org):
> > >
> > >>On Mon, 4 Jun 2007 14:40:24 -0500 "Serge E. Hallyn" <serue@us.ibm.com> wrote:
> > >>
> > >>
> > >>>Add the user namespace struct and framework
> > >>>
> > >>>Basically, it will allow a process to unshare its user_struct
> > >>>table, resetting at the same time its own user_struct and all the
> > >>>associated accounting.
> > >>>
> > >>>A new root user (uid == 0) is added to the user namespace upon
> > >>>creation. Such root users have full privileges and it seems
> > >>>that theses privileges should be controlled through some means
> > >>>(process capabilities ?)
> > >>
> > >>The whole magical-uid-0-user thing in this patch seem just wrong
> > >>to me.
> > >>
> > >>I'll merge it anyway, mainly because I want to merge _something_
> > >>(why oh why do the git-tree guys leave everything to the last
> > >>minute?) but it strikes me that there's something fundamentally
> > >>wrong whenever the kernel starts "knowing" about the significance

> > >>of UIDs in this fashion.
> > >
> > >
> > > $(&(%
> > >
> > > I thought I disagreed, but now I'm pretty sure I completely agree.
> > >
> > > 'root_user' exists in the kernel right now, but the root_user
> > > checks which exist (in fork.c and sys.c) shouldn't actually be
> > > applied for root in a container, since the container may not be
> > > trusted.
> >
> > This rlimit check doesn't help *untrusted* containers, so your logic
> > is wrong here. Instead, it allows root of the container to operate
> > in any situation.
>
> And I'm not sure that should be the case.
>
> In my view, root of a container is equivalent to a normal user on the
> host system, just like root in a qemu process.
>
> > E.g. consider root user hit the limit. After that you won't be able
> > to login/ssh to fix anything.
>
> That's fine in the container.
>
> > NOTE: container root can have no CAP_SYS_RESOURCE and CAP_SYS_ADMIN
> > as it is in OpenVZ.

> And eventually we'll want that to be the default in upstream containers.
> But it's not the case upstream right now. Before we can do that, we
> need an answer to per-container capabilities.
>
> Do you (either you specifically, or anyone at openvz) have plans to
> address the per-container capabilities problem? Herbert? Eric?

it is already addressed in Linux-VServer and OpenVZ
Linux-VServer adds a so called 'capabilitiy mask',
which is applied to the 'normal' capability system,
thus a guest cannot utilize/exercise capabilities
not included in that mask (which makes the guest
root 'secure')

> I'm interested, but would like to get the file capabilites squared
> away before I consider coding on it.
>
> > But in general I'm not against the patch, since in OpenVZ we can
> > replace the check with another capability we use for VE admin -

> > CAP_VE_SYS_ADMIN.
>
> If that truly sufficies then great. If not, then in order to support
> openvz in the meantime I say we drop my patch, but we remember that
> when we straighten out the security issues this will need to be
> addressed.

I'm not very fond of handling guest or host root special
and I think the capability system was designed to exactly
handle the guest root case properly ...

will look through the patches shortly and comment ...

best,
Herbert

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

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

---

Subject: Re: [PATCH 1/6] user namespace : add the framework
Posted by serue on Wed, 18 Jul 2007 14:21:51 GMT
View Forum Message <> Reply to Message

Quoting Herbert Poetzl (herbert@13thfloor.at):
> On Mon, Jul 16, 2007 at 10:08:00AM -0500, Serge E. Hallyn wrote:
> > Quoting Kirill Korotaev (dev@sw.ru):
> > > Serge E. Hallyn wrote:
> > > > Quoting Andrew Morton (akpm@linux-foundation.org):
> > > >
> > > >On Mon, 4 Jun 2007 14:40:24 -0500 "Serge E. Hallyn" <serue@us.ibm.com> wrote:
> > > >>
> > > >>
> > > >>Add the user namespace struct and framework
> > > >>
> > > >>Basically, it will allow a process to unshare its user_struct
> > > >>table, resetting at the same time its own user_struct and all the
> > > >>associated accounting.
> > > >>
> > > >>A new root user (uid == 0) is added to the user namespace upon

> > > >>>creation. Such root users have full privileges and it seems
> > > >>>that theses privileges should be controlled through some means
> > > >>>(process capabilities ?)
> > > >>
> > > >>The whole magical-uid-0-user thing in this patch seem just wrong
> > > >>to me.
> > > >>
> > > >>I'll merge it anyway, mainly because I want to merge _something_
> > > >>(why oh why do the git-tree guys leave everything to the last
> > > >>minute?) but it strikes me that there's something fundamentally
> > > >>wrong whenever the kernel starts "knowing" about the significance
> > > >>of UIDs in this fashion.
> > > >
> > > >
> > > > $(&(%
> > > >
> > > > I thought I disagreed, but now I'm pretty sure I completely agree.
> > > >
> > > > 'root_user' exists in the kernel right now, but the root_user
> > > > checks which exist (in fork.c and sys.c) shouldn't actually be
> > > > applied for root in a container, since the container may not be
> > > > trusted.
> > >
> > > This rlimit check doesn't help *untrusted* containers, so your logic
> > > is wrong here. Instead, it allows root of the container to operate
> > > in any situation.
> >
> > And I'm not sure that should be the case.
> >
> > In my view, root of a container is equivalent to a normal user on the
> > host system, just like root in a qemu process.
> >
> > > E.g. consider root user hit the limit. After that you won't be able
> > > to login/ssh to fix anything.
> >
> > That's fine in the container.
> >
> > > NOTE: container root can have no CAP_SYS_RESOURCE and CAP_SYS_ADMIN
> > > as it is in OpenVZ.
>
> > And eventually we'll want that to be the default in upstream containers.
> > But it's not the case upstream right now. Before we can do that, we
> > need an answer to per-container capabilities.
> >
> > Do you (either you specifically, or anyone at openvz) have plans to
> > address the per-container capabilities problem? Herbert? Eric?
>
> it is already addressed in Linux-VServer and OpenVZ

> Linux-VServer adds a so called 'capabilitiy mask',
> which is applied to the 'normal' capability system,
> thus a guest cannot utilize/exercise capabilities
> not included in that mask (which makes the guest
> root 'secure')

Are you special-casing some capabilities? For instance, cap_sys_admin in a container obviously shouldn't be fully granted, but some of it's abilities (setting hostname, setting enc key on loopback, etc) should be allowed.

Anyway, there unfortunately are several problems with a plain capability mask now.

First, there is the fact that we don't have just 'the host' and 'virtual hosts'. We have a set of namespaces, with usually no notion of one master namespace. So do we just mask out the masked capabilities as soon as any process does an unshare with any of the key flags? Or do we have a separate way to 'unshare capabilities'?

We could make cap-bset a per-process thing, where any process can take flags out (but not add them back in).

We could make a cap-bset container, where when creating a new container, we can take capabilities out of cap-bset.

Back to shortcomings,

Second, we wanted to implement user equivalence across namespaces. And to do that for root likely requires cross-namespace capabilities.

For instance, let's say I, user hallyn, unshare all my namespaces to create a virtual host vh.

Maybe I want user serge in vh to be equivalent to user hallyn in the real host. So I want to give it a key (host-uidns, hallyn). Now it can read all my files.

Maybe I want user hallyn in the host to have acess to all files in the vh, whether owned by root or any user. So I want to give user hallyn in the host uidns to have a key (vh-uidns, root).

That second example is nice and simple, but of course we prefer to think about capabilities. So I might just want to be able to give user hallyn in the host uid namespace a key (vh-uidns, CAP_DAC_OVERRIDE|CAP_DAC_READ_SEARCH|CAP_PTRACE).

Maybe all of this can be addressed with a posix capability container,
a custom ns_capability LSM, and a uidns keyring.

thanks,
-serge

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