Subject: [PATCH 0/8] user namespace: Introduction
Posted by serue on Tue, 19 Dec 2006 22:59:02 GMT
View Forum Message <> Reply to Message

This patchset adds a user namespace, which allows a process to
unshare it's user_struct table, allowing for separate accounting
per user namespace. It appends a user namespace to vfsmounts and
fown_structs, so that uid1==uid2 checks can be extended to be
false if uid1 and uid2 are in different namespaces.

A vfsmount generally cannot be accessed by another user namespace
than that in which it was mounted. A vfsmount can be mounted
"shared-ns", in which case it can be accessed by any user namespace.
This is needed at least to bootstrap a container so it can get far
enough to create it's own private file system tree, and can be
used in conjunction with read-only bind mounts to provide shared
/usr trees, for instance. However, for more useful, more fine-grained
sharing accross user namespaces, it has been suggested that a new
filesystem specifying global userid's be used.

Patches are as follows:

1. make exit_task_namespaces extern to prevent future
compile failure.
2. add userns framework.
3. add userns pointer to vfsmount
4. hook permission to check current against vfsmount userns
5. prepare for copy_mnt and copy_tree to return -EPERM
6. implement shared mounts which can cross user namespaces
7. implement user ns checks for file sigio
8. implement user namespace unshare

```
#include <unistd.h>
#include <linux/unistd.h>
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>
#include <sys/mount.h>
#include <sys/syscall.h>
#include <errno.h>
#include <signal.h>

#ifndef MNT_DETACH
#define MNT_DETACH 2
#endif

#ifndef MS_SHARE_NS
```

```
#define MS_SHARE_NS 1<<22
#endif

#ifndef CLONE_NEWUSER
#define CLONE_NEWUSER 0x10000000
#endif

#define PIPEREAD 0
#define PIPEWRITE 1

static inline _syscall2(int, clone, int, flags, int, foo)
int to1[2], from1[1], to2[2], from2[2];

void do_test1(void)
{
 int ret;
 int fd;
 char buf[100];

 rmdir("/mnt1");
 ret = mkdir("/mnt1", 0666);
 if (ret == -1) {
  perror("mkdir mnt1");
  _exit(1);
 }
 ret = mount("/", "/mnt1", "none", MS_BIND, "");
 if (ret == -1) {
  perror("mount mnt1");
  _exit(1);
 }
 fd = open("/mnt1/testme", O_RDWR|O_CREAT);
 if (fd == -1) {
  printf("thread 1: unable to write /testme: ERROR\n");
 } else {
  write(fd, "a", 1);
  printf("thread 1: able to write /testme: GOOD\n");
  close(fd);
 }
 write(from1[PIPEWRITE], "mount", 6);

 read(to1[PIPEREAD], buf, 2);
 umount("/mnt1");
 printf("thread 1: exiting.\n");
 write(from1[PIPEWRITE], "done", 5);
 rmdir("/mnt1");
}

void do_test2(void)
```

```
{
 int ret;
 char buf[100];
 int fd;

 rmdir("/mnt2");
 ret = mkdir("/mnt2", 0666);
 if (ret == -1) {
  perror("mkdir mnt2");
  _exit(1);
 }
 ret = read(to2[PIPEREAD], buf, 10);
 fd = open("/testme", O_RDWR|O_CREAT);
 if (fd == -1) {
  printf("thread 2: unable to write /testme: ERROR\n");
 } else {
  write(fd, "b", 1);
  printf("thread 2: able to write /testme: GOOD\n");
  close(fd);
 }

 fd = open("/mnt1/testme", O_RDWR|O_CREAT);
 if (fd == -1) {
  printf("thread 2: unable to open /mnt1/testme: GOOD\n");
 } else {
  write(fd, "c", 1);
  printf("thread 2: able to open /mnt1/testme: ERROR\n");
  close(fd);
 }

 /* now try remounting /mnt1 to /mnt2 */
 ret = mount("/mnt1", "/mnt2", "none", MS_BIND, "");
 if (ret == -1) {
  printf("thread2: unable to remount /mnt1 to /mnt2: GOOD\n");
 } else {
  perror("thread2: able to remount /mnt1 to /mnt2: BAD\n");
  fd = open("/mnt2/testme", O_RDWR|O_CREAT);
  if (fd == -1) {
   printf("thread 2: unable to open /mnt2/testme: GOOD\n");
  } else {
   write(fd, "d", 1);
   printf("thread 2: able to open /mnt2/testme: ERROR\n");
   close(fd);
  }
  umount ("/mnt2");
 }

 write(from2[PIPEWRITE], "done", 5);
```

```
 printf("thread 2: exiting\n");
 rmdir("/mnt2");
}

int main(int argc, char *argv[])
{
 int childpid1, childpid2;
 int ret;
 char buf[100];

 ret = mount("/", "/mnt", "none", MS_SHARE_NS | MS_BIND, "");
 if (ret == -1) {
  perror("failed to mount /mnt shared_ns.\n");
  _exit(1);
 }

 chdir("/mnt");
 chroot("/mnt");

 ret = pipe(to1);
 if (ret == -1) {
  perror("failed to create child pipe 1\n");
  _exit(1);
 }
 ret = pipe(to2);
 if (ret == -1) {
  perror("failed to create child pipe 2\n");
  _exit(1);
 }
 ret = pipe(from1);
 if (ret == -1) {
  perror("failed to create child pipe 1\n");
  _exit(1);
 }
 ret = pipe(from2);
 if (ret == -1) {
  perror("failed to create child pipe 2\n");
  _exit(1);
 }

 childpid1 = clone(CLONE_NEWUSER | SIGCHLD, 0);
 if (childpid1 != 0) {
  do_test1();
  _exit(0);
 }
 childpid2 = clone(CLONE_NEWUSER | SIGCHLD, 0);
 if (childpid2 != 0) {
  do_test2();
```

```
 _exit(0);
}

 read(from1[PIPEREAD], buf, 5);
 write(to2[PIPEWRITE], "go", 3);
 read(from2[PIPEREAD], buf, 4);
 write(to1[PIPEWRITE], "go", 3);
 read(from1[PIPEREAD], buf, 4);
 return 0;
}
```

_____

Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

---

## Subject: [PATCH 1/8] nsproxy: externalizes exit_task_namespaces
Posted by serue on Tue, 19 Dec 2006 22:59:49 GMT
View Forum Message <> Reply to Message

From: Cedric Le Goater <clg@fr.ibm.com>
Subject: [PATCH 1/8] nsproxy: externalizes exit_task_namespaces

this is required to remove a header dependency in sched.h which breaks
next patches.

Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>
Signed-off-by: Serge E. Hallyn <serue@us.ibm.com>
---
 include/linux/nsproxy.h |   15 ++++----------
 kernel/nsproxy.c        |   11 ++++++++++
 2 files changed, 15 insertions(+), 11 deletions(-)

```
diff --git a/include/linux/nsproxy.h b/include/linux/nsproxy.h
index 0b9f0dc..602f3d2 100644
--- a/include/linux/nsproxy.h
+++ b/include/linux/nsproxy.h
@@ -2,7 +2,8 @@ #ifndef _LINUX_NSPROXY_H
 #define _LINUX_NSPROXY_H

 #include <linux/spinlock.h>
-#include <linux/sched.h>
+
+struct task_struct;

 struct mnt_namespace;
 struct uts_namespace;
@@ -43,14 +44,6 @@ static inline void put_nsproxy(struct ns
```

```
 }
}

-static inline void exit_task_namespaces(struct task_struct *p)
-{
- struct nsproxy *ns = p->nsproxy;
- if (ns) {
-  task_lock(p);
-  p->nsproxy = NULL;
-  task_unlock(p);
-  put_nsproxy(ns);
- }
-}
+extern void exit_task_namespaces(struct task_struct *p);
+
 #endif
diff --git a/kernel/nsproxy.c b/kernel/nsproxy.c
index f5b9ee6..f11bbbf 100644
--- a/kernel/nsproxy.c
+++ b/kernel/nsproxy.c
@@ -36,6 +36,17 @@ void get_task_namespaces(struct task_str
  }
}

+void exit_task_namespaces(struct task_struct *p)
+{
+ struct nsproxy *ns = p->nsproxy;
+ if (ns) {
+  task_lock(p);
+  p->nsproxy = NULL;
+  task_unlock(p);
+  put_nsproxy(ns);
+ }
+}
+
 /*
  * creates a copy of "orig" with refcount 1.
  * This does not grab references to the contained namespaces,
--
1.4.1
```

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: [PATCH 2/8] user ns: add the framework
Posted by serue on Tue, 19 Dec 2006 23:00:00 GMT
View Forum Message <> Reply to Message

From: Cedric Le Goater <clg@fr.ibm.com>
Subject: [PATCH 2/8] user ns: add the framework

This patch adds 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>
---
 include/linux/init_task.h      |    2 ++
 include/linux/nsproxy.h        |    1 +
 include/linux/sched.h          |    3 ++
 include/linux/user_namespace.h |   53 ++++++++++++++++++++++++++++++++++++++++
 init/Kconfig                   |    8 ++++++
 kernel/Makefile                |    2 +-
 kernel/fork.c                  |    2 +-
 kernel/nsproxy.c               |   12 +++++++++
 kernel/sys.c                   |    5 ++--
 kernel/user.c                  |   18 +++++++-------
 kernel/user_namespace.c        |   44 ++++++++++++++++++++++++++++++++++

```
 11 files changed, 136 insertions(+), 14 deletions(-)

diff --git a/include/linux/init_task.h b/include/linux/init_task.h
index a2d95ff..26dc24b 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 602f3d2..e57920e 100644
--- a/include/linux/nsproxy.h
+++ b/include/linux/nsproxy.h
@@ -29,6 +29,7 @@ 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 d7e4de1..5a3f630 100644
--- a/include/linux/sched.h
+++ b/include/linux/sched.h
@@ -252,6 +252,7 @@ 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
@@ -1285,7 +1286,7 @@ 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..4ad4c0d
--- /dev/null
+++ b/include/linux/user_namespace.h
@@ -0,0 +1,53 @@
+#ifndef _LINUX_USER_NAMESPACE_H
+#define _LINUX_USER_NAMESPACE_H
+
+#include <linux/kref.h>
+#include <linux/nsproxy.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 int copy_user_ns(int flags, struct task_struct *tsk);
+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 NULL;
+}
+
+static inline int copy_user_ns(int flags, struct task_struct *tsk)
+{
+ 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 d1fed9e..c930912 100644
--- a/init/Kconfig
+++ b/init/Kconfig
@@ -222,6 +222,14 @@ config UTS_NS
    vservers, to use uts namespaces to provide different
    uts info for different servers.  If unsure, say N.

+config USER_NS
+ bool "User Namespaces"
+ default n
+ 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 839a58b..7c39952 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 ptrace.o timer.o user.o \
+      sysctl.o capability.o ptrace.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 a6ad544..deafa6e 100644
--- a/kernel/fork.c
+++ b/kernel/fork.c
@@ -996,7 +996,7 @@ #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 f11bbbf..cf43e06 100644
--- a/kernel/nsproxy.c
+++ b/kernel/nsproxy.c
@@ -80,6 +80,8 @@ struct nsproxy *dup_namespaces(struct ns
    get_ipc_ns(ns->ipc_ns);
   if (ns->pid_ns)
    get_pid_ns(ns->pid_ns);
+  if (ns->user_ns)
+   get_user_ns(ns->user_ns);
  }

  return ns;
@@ -127,10 +129,18 @@ int copy_namespaces(int flags, struct ta
  if (err)
   goto out_pid;

+ err = copy_user_ns(flags, tsk);
+ if (err)
+  goto out_user;
+
 out:
  put_nsproxy(old_ns);
  return err;

+out_user:
+ if (new_ns->pid_ns)
+  put_pid_ns(new_ns->pid_ns);
+
 out_pid:
  if (new_ns->ipc_ns)
   put_ipc_ns(new_ns->ipc_ns);
@@ -156,5 +166,7 @@ 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);
  kfree(ns);
 }
diff --git a/kernel/sys.c b/kernel/sys.c
index 5590255..26bdd84 100644
--- a/kernel/sys.c
+++ b/kernel/sys.c
@@ -33,6 +33,7 @@ #include <linux/getcpu.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>
@@ -1070,13 +1071,13 @@ 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..368f8da
--- /dev/null
+++ b/kernel/user_namespace.c
@@ -0,0 +1,44 @@
+/*
+ *  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
+
+int copy_user_ns(int flags, struct task_struct *tsk)
+{
+ struct user_namespace *old_ns = tsk->nsproxy->user_ns;
+ int err = 0;
+
+ if (!old_ns)
+  return 0;
+
+ get_user_ns(old_ns);
+ return err;
+}
+
+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 */
--
1.4.1
```

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

---

## Subject: [PATCH 3/8] user ns: add user_namespace ptr to vfsmount
Posted by serue on Tue, 19 Dec 2006 23:00:34 GMT
View Forum Message <> Reply to Message

From: Serge E. Hallyn <serue@us.ibm.com>
Subject: [PATCH 3/8] user ns: add user_namespace ptr to vfsmount

Add user_namespace ptr to vfsmount, and define a helper to compare it
to the task's user_ns.

Signed-off-by: Serge E. Hallyn <serue@us.ibm.com>
---
```
 fs/namespace.c       |    3 +++
 include/linux/mount.h |    2 ++
 include/linux/sched.h |   20 ++++++++++++++++++++
 3 files changed, 25 insertions(+), 0 deletions(-)

diff --git a/fs/namespace.c b/fs/namespace.c
index 5ef336c..9f98a67 100644
--- a/fs/namespace.c
+++ b/fs/namespace.c
@@ -25,6 +25,7 @@ #include <linux/namei.h>
 #include <linux/security.h>
 #include <linux/mount.h>
 #include <linux/ramfs.h>
+#include <linux/user_namespace.h>
 #include <asm/uaccess.h>
 #include <asm/unistd.h>
 #include "pnode.h"
@@ -56,6 +57,7 @@ struct vfsmount *alloc_vfsmnt(const char
  struct vfsmount *mnt = kmem_cache_alloc(mnt_cache, GFP_KERNEL);
  if (mnt) {
```

```
   memset(mnt, 0, sizeof(struct vfsmount));
+  mnt->mnt_user_ns = get_user_ns(current->nsproxy->user_ns);
   atomic_set(&mnt->mnt_count, 1);
   INIT_LIST_HEAD(&mnt->mnt_hash);
   INIT_LIST_HEAD(&mnt->mnt_child);
@@ -88,6 +90,7 @@ EXPORT_SYMBOL(simple_set_mnt);

 void free_vfsmnt(struct vfsmount *mnt)
 {
+ put_user_ns(mnt->mnt_user_ns);
  kfree(mnt->mnt_devname);
  kmem_cache_free(mnt_cache, mnt);
 }
diff --git a/include/linux/mount.h b/include/linux/mount.h
index 1b7e178..acdeca7 100644
--- a/include/linux/mount.h
+++ b/include/linux/mount.h
@@ -21,6 +21,7 @@ struct super_block;
 struct vfsmount;
 struct dentry;
 struct mnt_namespace;
+struct user_namespace;

 #define MNT_NOSUID 0x01
 #define MNT_NODEV 0x02
@@ -54,6 +55,7 @@ struct vfsmount {
  struct list_head mnt_slave; /* slave list entry */
  struct vfsmount *mnt_master; /* slave is on master->mnt_slave_list */
  struct mnt_namespace *mnt_ns; /* containing namespace */
+ struct user_namespace *mnt_user_ns; /* namespace for uid interpretation */
  int mnt_pinned;
 };

diff --git a/include/linux/sched.h b/include/linux/sched.h
index 5a3f630..450fc39 100644
--- a/include/linux/sched.h
+++ b/include/linux/sched.h
@@ -83,6 +83,8 @@ #include <linux/resource.h>
 #include <linux/timer.h>
 #include <linux/hrtimer.h>
 #include <linux/task_io_accounting.h>
+#include <linux/nsproxy.h>
+#include <linux/mount.h>

 #include <asm/processor.h>

@@ -1586,6 +1588,24 @@ extern int cond_resched_lock(spinlock_t
 extern int cond_resched_softirq(void);
```

```
 /*
+ * Check whether a task and a vfsmnt belong to the same uidns.
+ * Since the initial namespace is exempt from these checks,
+ * return 1 if so.  Also return 1 if the vfsmnt is exempt from
+ * such checking.  Otherwise, if the uid namespaces are different,
+ * return 0.
+ */
+static inline int task_mnt_same_uidns(struct task_struct *tsk,
+     struct vfsmount *mnt)
+{
+ if (tsk->nsproxy == init_task.nsproxy)
+  return 1;
+ if (mnt->mnt_user_ns == tsk->nsproxy->user_ns)
+  return 1;
+ return 0;
+}
+
+
+/*
  * Does a critical section need to be broken due to another
  * task waiting?:
  */
--
1.4.1
```

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

---

## Subject: [PATCH 4/8] user ns: hook permission
Posted by serue on Tue, 19 Dec 2006 23:00:51 GMT
View Forum Message <> Reply to Message

From: Serge E. Hallyn <serue@us.ibm.com>
Subject: [PATCH 4/8] user ns: hook permission

Hook permission to check vfsmnt->user_ns against current.

Signed-off-by: Serge E. Hallyn <serue@us.ibm.com>
---
 fs/namei.c |    4 ++++
 1 files changed, 4 insertions(+), 0 deletions(-)

diff --git a/fs/namei.c b/fs/namei.c
index e4f108f..d6687af 100644

```
--- a/fs/namei.c
+++ b/fs/namei.c
@@ -246,6 +246,8 @@ int permission(struct inode *inode, int
    return -EACCES;
  }

+ if (nd && !task_mnt_same_uidns(current, nd->mnt))
+  return -EACCES;

  /*
   * MAY_EXEC on regular files requires special handling: We override
@@ -433,6 +435,8 @@ static int exec_permission_lite(struct i
 {
  umode_t mode = inode->i_mode;

+ if (!task_mnt_same_uidns(current, nd->mnt))
+  return -EACCES;
  if (inode->i_op && inode->i_op->permission)
   return -EAGAIN;

--
1.4.1
```

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

---

## Subject: [PATCH 5/8] user ns: prepare copy_tree, copy_mnt, and their callers to handle errs

Posted by serue on Tue, 19 Dec 2006 23:01:06 GMT

View Forum Message <> Reply to Message

From: Serge E. Hallyn <serue@us.ibm.com>
Subject: [PATCH 5/8] user ns: prepare copy_tree, copy_mnt, and their callers to handle errs

With shareduserns and non-shareduserns mounts, it will be possible
for clone_mnt to return -EPERM if a namespace tries to bind
mount a non-shareduserns vfsmnt from another user namespace.
But currently they only return NULL, which is interpreted as
-ENOMEM.  Update the callers to handle other errors.

Signed-off-by: Serge E. Hallyn <serue@us.ibm.com>
---
 fs/namespace.c |   20 ++++++++++++--------
 fs/pnode.c     |    5 ++---
 2 files changed, 15 insertions(+), 10 deletions(-)

```
diff --git a/fs/namespace.c b/fs/namespace.c
index 9f98a67..f85dd73 100644
--- a/fs/namespace.c
+++ b/fs/namespace.c
@@ -709,8 +709,9 @@ struct vfsmount *copy_tree(struct vfsmou
   return NULL;

  res = q = clone_mnt(mnt, dentry, flag);
- if (!q)
-  goto Enomem;
+ if (!q || IS_ERR(q)) {
+  return q;
+ }
  q->mnt_mountpoint = mnt->mnt_mountpoint;

  p = mnt;
@@ -731,8 +732,9 @@ struct vfsmount *copy_tree(struct vfsmou
    nd.mnt = q;
    nd.dentry = p->mnt_mountpoint;
    q = clone_mnt(p, p->mnt_root, flag);
-   if (!q)
-    goto Enomem;
+   if (!q || IS_ERR(q)) {
+    goto Error;
+   }
    spin_lock(&vfsmount_lock);
    list_add_tail(&q->mnt_list, &res->mnt_list);
    attach_mnt(q, &nd);
@@ -740,7 +742,7 @@ struct vfsmount *copy_tree(struct vfsmou
   }
  }
  return res;
-Enomem:
+Error:
  if (res) {
   LIST_HEAD(umount_list);
   spin_lock(&vfsmount_lock);
@@ -748,7 +750,7 @@ Enomem:
   spin_unlock(&vfsmount_lock);
   release_mounts(&umount_list);
  }
- return NULL;
+ return q;
 }

 /*
@@ -928,8 +930,10 @@ static int do_loopback(struct nameidata
```

```
  else
   mnt = clone_mnt(old_nd.mnt, old_nd.dentry, 0);

- if (!mnt)
+ if (!mnt || IS_ERR(mnt)) {
+  err = mnt ? PTR_ERR(mnt) : -ENOMEM;
   goto out;
+ }

  err = graft_tree(mnt, nd);
  if (err) {
@@ -1466,7 +1470,7 @@ struct mnt_namespace *dup_mnt_ns(struct
  /* First pass: copy the tree topology */
  new_ns->root = copy_tree(mnt_ns->root, mnt_ns->root->mnt_root,
      CL_COPY_ALL | CL_EXPIRE);
- if (!new_ns->root) {
+ if (!new_ns->root || IS_ERR(new_ns->root)) {
   up_write(&namespace_sem);
   kfree(new_ns);
   return NULL;
diff --git a/fs/pnode.c b/fs/pnode.c
index 56aacea..1821c95 100644
--- a/fs/pnode.c
+++ b/fs/pnode.c
@@ -187,8 +187,9 @@ int propagate_mnt(struct vfsmount *dest_

   source =  get_source(m, prev_dest_mnt, prev_src_mnt, &type);

-  if (!(child = copy_tree(source, source->mnt_root, type))) {
-   ret = -ENOMEM;
+  child = copy_tree(source, source->mnt_root, type);
+  if (!child || IS_ERR(child)) {
+   ret = child ? PTR_ERR(child) : -ENOMEM;
    list_splice(tree_list, tmp_list.prev);
    goto out;
   }
--
1.4.1
```

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

---

Subject: [PATCH 6/8] user ns: implement shared mounts
Posted by serue on Tue, 19 Dec 2006 23:01:16 GMT

From: Serge E. Hallyn <serue@us.ibm.com>
Subject: [PATCH 6/8] user ns: implement shared mounts

Implement shared-ns mounts, which allow containers in different user
namespaces to share mounts.  Without this, containers can obviously
never even be started.

Here is a sample smount.c (based on Miklos' version) which only
does a bind mount of arg1 onto arg2, but making the destination
a shared-ns mount.

```
int main(int argc, char *argv[])
{
 int type;
 if(argc != 3) {
  fprintf(stderr, "usage: %s src dest", argv[0]);
  return 1;
 }

 fprintf(stdout, "%s %s %s\n", argv[0], argv[1], argv[2]);

 type = MS_SHARE_NS | MS_BIND;
 setfsuid(getuid());

 if(mount(argv[1], argv[2], "none", type, "") == -1) {
  perror("mount");
  return 1;
 }
 return 0;
}
```

Signed-off-by: Serge E. Hallyn <serue@us.ibm.com>
---
```
 fs/namespace.c        |   30 ++++++++++++++++++++++++------
 fs/pnode.h            |    1 +
 include/linux/fs.h    |    1 +
 include/linux/mount.h |    1 +
 include/linux/sched.h |    2 ++
 5 files changed, 29 insertions(+), 6 deletions(-)

diff --git a/fs/namespace.c b/fs/namespace.c
index f85dd73..9cf8463 100644
--- a/fs/namespace.c
+++ b/fs/namespace.c
@@ -235,7 +235,14 @@ static struct vfsmount *clone_mnt(struct
      int flag)
 {
```

```
  struct super_block *sb = old->mnt_sb;
- struct vfsmount *mnt = alloc_vfsmnt(old->mnt_devname);
+ struct vfsmount *mnt;
+
+ if (!(old->mnt_flags & MNT_SHARE_NS)) {
+  if (old->mnt_user_ns != current->nsproxy->user_ns)
+   return ERR_PTR(-EPERM);
+ }
+
+ mnt = alloc_vfsmnt(old->mnt_devname);

  if (mnt) {
   mnt->mnt_flags = old->mnt_flags;
@@ -258,6 +265,10 @@ static struct vfsmount *clone_mnt(struct
   }
   if (flag & CL_MAKE_SHARED)
    set_mnt_shared(mnt);
+  if (flag & CL_SHARE_NS)
+   mnt->mnt_flags |= MNT_SHARE_NS;
+  else
+   mnt->mnt_flags &= ~MNT_SHARE_NS;

   /* stick the duplicate mount on the same expiry list
    * as the original if that was on one */
@@ -369,6 +380,7 @@ static int show_vfsmnt(struct seq_file *
   { MNT_NOSUID, ",nosuid" },
   { MNT_NODEV, ",nodev" },
   { MNT_NOEXEC, ",noexec" },
+  { MNT_SHARE_NS, ",share_userns" },
   { MNT_NOATIME, ",noatime" },
   { MNT_NODIRATIME, ",nodiratime" },
   { MNT_RELATIME, ",relatime" },
@@ -903,11 +915,14 @@ static int do_change_type(struct nameida
 /*
  * do loopback mount.
  */
-static int do_loopback(struct nameidata *nd, char *old_name, int recurse)
+static int do_loopback(struct nameidata *nd, char *old_name, int recurse,
+       int uidns_share)
 {
  struct nameidata old_nd;
  struct vfsmount *mnt = NULL;
  int err = mount_is_safe(nd);
+ int flag = (uidns_share ? CL_SHARE_NS : 0);
+
  if (err)
   return err;
  if (!old_name || !*old_name)
```

```
@@ -926,9 +941,9 @@ static int do_loopback(struct nameidata

  err = -ENOMEM;
  if (recurse)
-  mnt = copy_tree(old_nd.mnt, old_nd.dentry, 0);
+  mnt = copy_tree(old_nd.mnt, old_nd.dentry, flag);
  else
-  mnt = clone_mnt(old_nd.mnt, old_nd.dentry, 0);
+  mnt = clone_mnt(old_nd.mnt, old_nd.dentry, flag);

  if (!mnt || IS_ERR(mnt)) {
   err = mnt ? PTR_ERR(mnt) : -ENOMEM;
@@ -1415,9 +1430,11 @@ long do_mount(char *dev_name, char *dir_
   mnt_flags |= MNT_NODIRATIME;
  if (flags & MS_RELATIME)
   mnt_flags |= MNT_RELATIME;
+ if (flags & MS_SHARE_NS)
+  mnt_flags |= MNT_SHARE_NS;

  flags &= ~(MS_NOSUID | MS_NOEXEC | MS_NODEV | MS_ACTIVE |
-     MS_NOATIME | MS_NODIRATIME | MS_RELATIME);
+     MS_NOATIME | MS_NODIRATIME | MS_RELATIME | MS_SHARE_NS);

  /* ... and get the mountpoint */
  retval = path_lookup(dir_name, LOOKUP_FOLLOW, &nd);
@@ -1432,7 +1449,8 @@ long do_mount(char *dev_name, char *dir_
   retval = do_remount(&nd, flags & ~MS_REMOUNT, mnt_flags,
         data_page);
  else if (flags & MS_BIND)
-  retval = do_loopback(&nd, dev_name, flags & MS_REC);
+  retval = do_loopback(&nd, dev_name, flags & MS_REC,
+       mnt_flags & MNT_SHARE_NS);
  else if (flags & (MS_SHARED | MS_PRIVATE | MS_SLAVE | MS_UNBINDABLE))
   retval = do_change_type(&nd, flags);
  else if (flags & MS_MOVE)
diff --git a/fs/pnode.h b/fs/pnode.h
index d45bd8e..eb62f4c 100644
--- a/fs/pnode.h
+++ b/fs/pnode.h
@@ -22,6 +22,7 @@ #define CL_SLAVE       0x02
 #define CL_COPY_ALL   0x04
 #define CL_MAKE_SHARED   0x08
 #define CL_PROPAGATION   0x10
+#define CL_SHARE_NS   0x20

 static inline void set_mnt_shared(struct vfsmount *mnt)
 {
diff --git a/include/linux/fs.h b/include/linux/fs.h
```

```
index e7268d2..bb801cb 100644
--- a/include/linux/fs.h
+++ b/include/linux/fs.h
@@ -121,6 +121,7 @@ #define MS_PRIVATE (1<<18) /* change to
 #define MS_SLAVE (1<<19) /* change to slave */
 #define MS_SHARED (1<<20) /* change to shared */
 #define MS_RELATIME (1<<21) /* Update atime relative to mtime/ctime. */
+#define MS_SHARE_NS (1<<22) /* ignore user namespaces for permission */
 #define MS_ACTIVE (1<<30)
 #define MS_NOUSER (1<<31)

diff --git a/include/linux/mount.h b/include/linux/mount.h
index acdeca7..28e0964 100644
--- a/include/linux/mount.h
+++ b/include/linux/mount.h
@@ -35,6 +35,7 @@ #define MNT_SHRINKABLE 0x100
 #define MNT_SHARED 0x1000 /* if the vfsmount is a shared mount */
 #define MNT_UNBINDABLE 0x2000 /* if the vfsmount is a unbindable mount */
 #define MNT_PNODE_MASK 0x3000 /* propogation flag mask */
+#define MNT_SHARE_NS 0x4000  /* ignore user namespaces for permission */

 struct vfsmount {
   struct list_head mnt_hash;
diff --git a/include/linux/sched.h b/include/linux/sched.h
index 450fc39..73df38c 100644
--- a/include/linux/sched.h
+++ b/include/linux/sched.h
@@ -1599,6 +1599,8 @@ static inline int task_mnt_same_uidns(st
 {
   if (tsk->nsproxy == init_task.nsproxy)
     return 1;
+  if (mnt->mnt_flags & MNT_SHARE_NS)
+    return 1;
   if (mnt->mnt_user_ns == tsk->nsproxy->user_ns)
     return 1;
   return 0;
--
1.4.1
```

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: [PATCH 7/8] user ns: handle file sigio
Posted by serue on Tue, 19 Dec 2006 23:01:27 GMT

From: Serge E. Hallyn <serue@us.ibm.com>
Subject: [PATCH 7/8] user ns: handle file sigio

A process in one user namespace could set a fowner and sigio on a file in a
shared vfsmount, ending up killing a task in another user namespace.

Prevent this by adding a user namespace pointer to the fown_struct, and
enforcing that a process causing a signal to be sent be in the same
user namespace as the file owner.

Signed-off-by: Serge E. Hallyn <serue@us.ibm.com>
---
 fs/fcntl.c         |   14 ++++++++++---
 fs/file_table.c    |    2 ++
 include/linux/fs.h |    1 +
 3 files changed, 14 insertions(+), 3 deletions(-)

```
diff --git a/fs/fcntl.c b/fs/fcntl.c
index 8e382a5..6a774c1 100644
--- a/fs/fcntl.c
+++ b/fs/fcntl.c
@@ -18,6 +18,7 @@ #include <linux/security.h>
 #include <linux/ptrace.h>
 #include <linux/signal.h>
 #include <linux/rcupdate.h>
+#include <linux/user_namespace.h>

 #include <asm/poll.h>
 #include <asm/siginfo.h>
@@ -250,15 +251,18 @@ static int setfl(int fd, struct file * f
 }

 static void f_modown(struct file *filp, struct pid *pid, enum pid_type type,
-                     uid_t uid, uid_t euid, int force)
+       uid_t uid, uid_t euid, struct user_namespace *user_ns,
+         int force)
 {
  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.user_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.user_ns = get_user_ns(user_ns);
  }
```

```
  write_unlock_irq(&filp->f_owner.lock);
 }
@@ -272,7 +276,8 @@ int __f_setown(struct file *filp, struct
  if (err)
   return err;

- f_modown(filp, pid, type, current->uid, current->euid, force);
+ f_modown(filp, pid, type, current->uid, current->euid,
+     current->nsproxy->user_ns, force);
  return 0;
 }
 EXPORT_SYMBOL(__f_setown);
@@ -298,7 +303,7 @@ EXPORT_SYMBOL(f_setown);

 void f_delown(struct file *filp)
 {
- f_modown(filp, NULL, PIDTYPE_PID, 0, 0, 1);
+ f_modown(filp, NULL, PIDTYPE_PID, 0, 0, NULL, 1);
 }

 pid_t f_getown(struct file *filp)
@@ -455,6 +460,9 @@ static const long band_table[NSIGPOLL] =
 static inline int sigio_perm(struct task_struct *p,
                              struct fown_struct *fown, int sig)
 {
+ if (fown->user_ns != init_task.nsproxy->user_ns &&
+     fown->user_ns != p->nsproxy->user_ns)
+  return 0;
  return (((fown->euid == 0) ||
    (fown->euid == p->suid) || (fown->euid == p->uid) ||
    (fown->uid == p->suid) || (fown->uid == p->uid)) &&
diff --git a/fs/file_table.c b/fs/file_table.c
index 4c17a18..6e6632c 100644
--- a/fs/file_table.c
+++ b/fs/file_table.c
@@ -21,6 +21,7 @@ #include <linux/cdev.h>
 #include <linux/fsnotify.h>
 #include <linux/sysctl.h>
 #include <linux/percpu_counter.h>
+#include <linux/user_namespace.h>

 #include <asm/atomic.h>

@@ -175,6 +176,7 @@ void fastcall __fput(struct file *file)
  if (file->f_mode & FMODE_WRITE)
   put_write_access(inode);
  put_pid(file->f_owner.pid);
+ put_user_ns(file->f_owner.user_ns);
```

```
  file_kill(file);
  file->f_path.dentry = NULL;
  file->f_path.mnt = NULL;
diff --git a/include/linux/fs.h b/include/linux/fs.h
index bb801cb..a5532e1 100644
--- a/include/linux/fs.h
+++ b/include/linux/fs.h
@@ -684,6 +684,7 @@ struct fown_struct {
  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 user_namespace *user_ns; /* namespace to which uid belongs */
  int signum;  /* posix.1b rt signal to be delivered on IO */
 };

--
1.4.1
```

---

Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

---

## Subject: [PATCH 8/8] user ns: implement user ns unshare
Posted by serue on Tue, 19 Dec 2006 23:01:47 GMT
View Forum Message <> Reply to Message

From: Serge E. Hallyn <serue@us.ibm.com>
Subject: [PATCH 8/8] user ns: implement user ns unshare

Implement CLONE_NEWUSER flag useable at clone and unshare.

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

```
---
 include/linux/sched.h         |    1 +
 include/linux/user_namespace.h |   10 +++++
 kernel/fork.c                 |   22 ++++++++++--
 kernel/nsproxy.c              |    2 +
 kernel/user_namespace.c       |   74 ++++++++++++++++++++++++++++++++++++++++++-
 5 files changed, 102 insertions(+), 7 deletions(-)

diff --git a/include/linux/sched.h b/include/linux/sched.h
index 73df38c..55ecf81 100644
--- a/include/linux/sched.h
+++ b/include/linux/sched.h
@@ -26,6 +26,7 @@ #define CLONE_CHILD_SETTID 0x01000000 /*
 #define CLONE_STOPPED  0x02000000 /* Start in stopped state */
```

```
 #define CLONE_NEWUTS  0x04000000 /* New utsname group? */
 #define CLONE_NEWIPC  0x08000000 /* New ipcs */
+#define CLONE_NEWUSER  0x10000000 /* New user namespace */

 /*
  * Scheduling policies
diff --git a/include/linux/user_namespace.h b/include/linux/user_namespace.h
index 4ad4c0d..d577ede 100644
--- a/include/linux/user_namespace.h
+++ b/include/linux/user_namespace.h
@@ -25,6 +25,7 @@ static inline struct user_namespace *get
 }

 extern int copy_user_ns(int flags, struct task_struct *tsk);
+extern int unshare_user_ns(unsigned long flags, struct user_namespace **new_user);
 extern void free_user_ns(struct kref *kref);

 static inline void put_user_ns(struct user_namespace *ns)
@@ -40,6 +41,15 @@ static inline struct user_namespace *get
  return NULL;
 }

+static inline int unshare_user_ns(unsigned long flags,
+    struct user_namespace **new_user)
+{
+ if (flags & CLONE_NEWUSER)
+  return -EINVAL;
+
+ return 0;
+}
+
 static inline int copy_user_ns(int flags, struct task_struct *tsk)
 {
  return 0;
diff --git a/kernel/fork.c b/kernel/fork.c
index deafa6e..eead517 100644
--- a/kernel/fork.c
+++ b/kernel/fork.c
@@ -49,6 +49,7 @@ #include <linux/cn_proc.h>
 #include <linux/delayacct.h>
 #include <linux/taskstats_kern.h>
 #include <linux/random.h>
+#include <linux/user_namespace.h>

 #include <asm/pgtable.h>
 #include <asm/pgalloc.h>
@@ -1620,6 +1621,7 @@ asmlinkage long sys_unshare(unsigned lon
  struct nsproxy *new_nsproxy = NULL, *old_nsproxy = NULL;
```

```
  struct uts_namespace *uts, *new_uts = NULL;
  struct ipc_namespace *ipc, *new_ipc = NULL;
+ struct user_namespace *user, *new_user = NULL;

  check_unshare_flags(&unshare_flags);

@@ -1627,7 +1629,7 @@ asmlinkage long sys_unshare(unsigned lon
  err = -EINVAL;
  if (unshare_flags & ~(CLONE_THREAD|CLONE_FS|CLONE_NEWNS|CLONE_SIGHAND|
     CLONE_VM|CLONE_FILES|CLONE_SYSVSEM|
-    CLONE_NEWUTS|CLONE_NEWIPC))
+    CLONE_NEWUTS|CLONE_NEWIPC|CLONE_NEWUSER))
   goto bad_unshare_out;

  if ((err = unshare_thread(unshare_flags)))
@@ -1648,18 +1650,20 @@ asmlinkage long sys_unshare(unsigned lon
   goto bad_unshare_cleanup_semundo;
  if ((err = unshare_ipcs(unshare_flags, &new_ipc)))
   goto bad_unshare_cleanup_uts;
+ if ((err = unshare_user_ns(unshare_flags, &new_user)))
+  goto bad_unshare_cleanup_ipc;

- if (new_ns || new_uts || new_ipc) {
+ if (new_ns || new_uts || new_ipc || new_user) {
   old_nsproxy = current->nsproxy;
   new_nsproxy = dup_namespaces(old_nsproxy);
   if (!new_nsproxy) {
    err = -ENOMEM;
-   goto bad_unshare_cleanup_ipc;
+   goto bad_unshare_cleanup_user;
   }
  }

  if (new_fs || new_ns || new_mm || new_fd || new_ulist ||
-    new_uts || new_ipc) {
+    new_uts || new_ipc || new_user) {

   task_lock(current);

@@ -1707,12 +1711,22 @@ asmlinkage long sys_unshare(unsigned lon
    new_ipc = ipc;
   }

+  if (new_user) {
+   user = current->nsproxy->user_ns;
+   current->nsproxy->user_ns = new_user;
+   new_user = user;
+  }
```

```
+
   task_unlock(current);
  }

  if (new_nsproxy)
   put_nsproxy(new_nsproxy);

+bad_unshare_cleanup_user:
+ if (new_user)
+  put_user_ns(new_user);
+
 bad_unshare_cleanup_ipc:
  if (new_ipc)
   put_ipc_ns(new_ipc);
diff --git a/kernel/nsproxy.c b/kernel/nsproxy.c
index cf43e06..2d9bafb 100644
--- a/kernel/nsproxy.c
+++ b/kernel/nsproxy.c
@@ -102,7 +102,7 @@ int copy_namespaces(int flags, struct ta

  get_nsproxy(old_ns);

- if (!(flags & (CLONE_NEWNS | CLONE_NEWUTS | CLONE_NEWIPC)))
+ if (!(flags & (CLONE_NEWNS | CLONE_NEWUTS | CLONE_NEWIPC | CLONE_NEWUSER)))
   return 0;

  new_ns = clone_namespaces(old_ns);
diff --git a/kernel/user_namespace.c b/kernel/user_namespace.c
index 368f8da..d1ce4c1 100644
--- a/kernel/user_namespace.c
+++ b/kernel/user_namespace.c
@@ -20,17 +20,87 @@ struct user_namespace init_user_ns = {
 EXPORT_SYMBOL_GPL(init_user_ns);

 #ifdef CONFIG_USER_NS
+/*
+ * Clone a new ns copying an original user ns, setting refcount to 1
+ * @old_ns: namespace to clone
+ * Return NULL on error (failure to kmalloc), new ns otherwise
+ */
+static struct user_namespace *clone_user_ns(struct user_namespace *old_ns)
+{
+       struct user_namespace *ns;
+       struct user_struct *new_user;
+       int n;
+
+       ns = kmalloc(sizeof(struct user_namespace), GFP_KERNEL);
+       if (!ns)
```

```
+		return NULL;
+
+	kref_init(&ns->kref);
+
+	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;
+	}
+
+	switch_uid(new_user);
+	return ns;
+}
+
+/*
+ * unshare the current process' user namespace.
+ */
+int unshare_user_ns(unsigned long flags,
+   struct user_namespace **new_user)
+{
+	if (flags & CLONE_NEWUSER) {
+		if (!capable(CAP_SYS_ADMIN))
+			return -EPERM;
+
+		*new_user = clone_user_ns(current->nsproxy->user_ns);
+		if (!*new_user)
+			return -ENOMEM;
+	}
+
+	return 0;
+}

 int copy_user_ns(int flags, struct task_struct *tsk)
 {
- struct user_namespace *old_ns = tsk->nsproxy->user_ns;
+ struct user_namespace *new_ns, *old_ns = tsk->nsproxy->user_ns;
```

```
  int err = 0;

  if (!old_ns)
   return 0;

  get_user_ns(old_ns);
- return err;
+ if (!(flags & CLONE_NEWUSER))
+  return 0;
+ err = -EPERM;
+ if (!capable(CAP_SYS_ADMIN))
+  goto out;
+ err = -ENOMEM;
+ new_ns = clone_user_ns(old_ns);
+ if (!new_ns)
+  goto out;
+
+ tsk->nsproxy->user_ns = new_ns;
+ err = 0;
+out:
+ put_user_ns(old_ns);
+ return 0;
 }

 void free_user_ns(struct kref *kref)
--
1.4.1
```

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 4/8] user ns: hook permission
Posted by ebiederm on Wed, 24 Jan 2007 17:06:03 GMT
View Forum Message <> Reply to Message

"Serge E. Hallyn" <serue@us.ibm.com> writes:

> From: Serge E. Hallyn <serue@us.ibm.com>
> Subject: [PATCH 4/8] user ns: hook permission
>
> Hook permission to check vfsmnt->user_ns against current.

This looks wrong on several levels.
- This should ultimately be inside generic_permission instead of
  permission as there are some distributed filesystems that know how to cope with

multiple mount namespaces simultaneous.

- As implemented the test is not what I would expect. I would
expect comparisons of uid X == uid Y and gid X == gid Y to
be replaced by comparing the tuples of uid namesspace and uid.
Which would allow access to world readable/writeable files,
and it would allow users with CAP_DAC_OVERRIDE to be able to access
everything.

All we are really saying as I understand a user namespace is that
instead of uid's uniquely identifying a user the pair the pair uidns,
uid is uniquely identifies a user.

Because you didn't pick what I would consider the obvious choice
you now need an extra mount flag to disable the uid namespace all
together, so you can transition through the intermediate uid namespace
state. That really feels wrong.

All mounts should have an associated uid namespace and the only
way you should be able to ignore that is to access filesystems
that can cope with multiple uid namespaces simultaneously.

Eric

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 7/8] user ns: handle file sigio
Posted by ebiederm on Wed, 24 Jan 2007 17:23:31 GMT
View Forum Message <> Reply to Message

"Serge E. Hallyn" <serue@us.ibm.com> writes:

> From: Serge E. Hallyn <serue@us.ibm.com>
> Subject: [PATCH 7/8] user ns: handle file sigio
>
> A process in one user namespace could set a fowner and sigio on a file in a
> shared vfsmount, ending up killing a task in another user namespace.
>
> Prevent this by adding a user namespace pointer to the fown_struct, and
> enforcing that a process causing a signal to be sent be in the same
> user namespace as the file owner.
>

> @@ -455,6 +460,9 @@ static const long band_table[NSIGPOLL] =

>  static inline int sigio_perm(struct task_struct *p,
>                               struct fown_struct *fown, int sig)
>  {
> + if (fown->user_ns != init_task.nsproxy->user_ns &&
> +     fown->user_ns != p->nsproxy->user_ns)
> +  return 0;

Why is the initial user namespace being treated specially here?
Especially when you start considering nested containers special treatment
like this is semantically a real problem, to maintain.

If we need to I can see doing something special if the process setting
fown has CAP_KILL and bypassing the security checks that way, but
hard coding rules like that when it doesn't appear we have any
experience to indicate we need the extra functionality looks
premature.

>   return (((fown->euid == 0) ||
>     (fown->euid == p->suid) || (fown->euid == p->uid) ||
>     (fown->uid == p->suid) || (fown->uid == p->uid)) &&

Eric

_____

Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 7/8] user ns: handle file sigio
Posted by serue on Wed, 24 Jan 2007 18:58:45 GMT
View Forum Message <> Reply to Message

Quoting Eric W. Biederman (ebiederm@xmission.com):
> "Serge E. Hallyn" <serue@us.ibm.com> writes:
>
> > From: Serge E. Hallyn <serue@us.ibm.com>
> > Subject: [PATCH 7/8] user ns: handle file sigio
> >
> > A process in one user namespace could set a fowner and sigio on a file in a
> > shared vfsmount, ending up killing a task in another user namespace.
> >
> > Prevent this by adding a user namespace pointer to the fown_struct, and
> > enforcing that a process causing a signal to be sent be in the same
> > user namespace as the file owner.
> >
>
>
> > @@ -455,6 +460,9 @@ static const long band_table[NSIGPOLL] =

> >  static inline int sigio_perm(struct task_struct *p,
> >                               struct fown_struct *fown, int sig)
> >  {
> > + if (fown->user_ns != init_task.nsproxy->user_ns &&
> > +    fown->user_ns != p->nsproxy->user_ns)
> > +  return 0;
>
> Why is the initial user namespace being treated specially here?

Because we haven't yet agreed upon any other security model.  For now, although I know you really dislike it, the fact is that "the initial namespace has special privileges" is our basic security model.

If you want to have a discussion about an appropriate security model, or an infrastructure to support multiple models, I think this would be a good time, given that several namespaces are out there.  And networkns, making its way up, also has concerns.

Three basic approaches I could see being simple to both implement and understand/use are

1. add a set of capabilities concerning cross-ns operations, not reassignable once they are removed.  Simple to understand, very limited.

2. maintain that any cross-ns operation is allowed if and only if the target ns is a child of the subject ns.

3. cross-ns operations are not permitted.  The only way to achieve them is using a (as-yet unimplemented, but i'm working on it) namespace enter feature to execute code in a child namespace.

> Especially when you start considering nested containers special treatment
> like this is semantically a real problem, to maintain.

Yup.

> If we need to I can see doing something special if the process setting
> fown has CAP_KILL

Obviously CAP_KILL is insufficient :)  I assume you mean a new CAP_XNS_CAP_KILL?

> and bypassing the security checks that way, but
> hard coding rules like that when it doesn't appear we have any
> experience to indicate we need the extra functionality looks
> premature.

Ok, in this case actually I suspect you're right and we can just ditch
the exception.  But in general the security discussion is one we should
still have.

> >   return (((fown->euid == 0) ||
> >     (fown->euid == p->suid) || (fown->euid == p->uid) ||
> >     (fown->uid == p->suid) || (fown->uid == p->uid)) &&
>
> Eric

_____

Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 4/8] user ns: hook permission
Posted by serue on Wed, 24 Jan 2007 19:06:33 GMT
View Forum Message <> Reply to Message

Quoting Eric W. Biederman (ebiederm@xmission.com):
> "Serge E. Hallyn" <serue@us.ibm.com> writes:
>
> > From: Serge E. Hallyn <serue@us.ibm.com>
> > Subject: [PATCH 4/8] user ns: hook permission
> >
> > Hook permission to check vfsmnt->user_ns against current.
>
> This looks wrong on several levels.
> - This should ultimately be inside generic_permission instead of
>   permission as there are some distributed filesystems that know how to cope with
>   multiple mount namespaces simultaneous.
>
> - As implemented the test is not what I would expect.  I would
>   expect comparisons of uid X == uid Y and gid X == gid Y to
>   be replaced by comparing the tuples of uid namesspace and uid.
>   Which would allow access to world readable/writeable files,
>   and it would allow users with CAP_DAC_OVERRIDE to be able to access
>   everything.

Whoa - why on earth would we want that?

> All we are really saying as I understand a user namespace is that
> instead of uid's uniquely identifying a user the pair the pair uidns,
> uid is uniquely identifies a user.

Ok that would be one way to interpret it, but it is insufficient for
preventing root in one vserver from messing with users in another
vserver.

> Because you didn't pick what I would consider the obvious choice
> you now need an extra mount flag to disable the uid namespace all
> together, so you can transition through the intermediate uid namespace
> state.  That really feels wrong.

Some bit of required bootstrapping seems both acceptable and expected
to me.

> All mounts should have an associated uid namespace and the only

check

> way you should be able to ignore that is to access filesystems
> that can cope with multiple uid namespaces simultaneously.

But it's my fs on my box, why shouldn't i be able to say all uid
namespaces can acces this subtree read-only, just bc you feel the
fs is inadequate?  :)

Note that the tiniest of trees, with just a statically compiled bash,
mount, pivot_mount, and initrc, should suffice, mounted readonly for
all uid namespaces to use to bootstrap.

-serge

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 7/8] user ns: handle file sigio
Posted by Andrew Morton on Thu, 25 Jan 2007 08:12:53 GMT
View Forum Message <> Reply to Message

On Wed, 24 Jan 2007 12:58:45 -0600
"Serge E. Hallyn" <serue@us.ibm.com> wrote:

> > If we need to I can see doing something special if the process setting
> > fown has CAP_KILL
>
> Obviously CAP_KILL is insufficient :)  I assume you mean a new
> CAP_XNS_CAP_KILL?
>
> > and bypassing the security checks that way, but
> > hard coding rules like that when it doesn't appear we have any
> > experience to indicate we need the extra functionality looks
> > premature.

>
> Ok, in this case actually I suspect you're right and we can just ditch
> the exception.  But in general the security discussion is one we should
> still have.

People like security.

Where do we now stand with this patch, and with "[PATCH 4/8] user ns: hook permission"?

_____

Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 7/8] user ns: handle file sigio
Posted by serue on Thu, 25 Jan 2007 15:32:39 GMT
View Forum Message <> Reply to Message

Quoting Andrew Morton (akpm@osdl.org):
> On Wed, 24 Jan 2007 12:58:45 -0600
> "Serge E. Hallyn" <serue@us.ibm.com> wrote:
>
> > > If we need to I can see doing something special if the process setting
> > > fown has CAP_KILL
> >
> > Obviously CAP_KILL is insufficient :)  I assume you mean a new
> > CAP_XNS_CAP_KILL?
> >
> > > and bypassing the security checks that way, but
> > > hard coding rules like that when it doesn't appear we have any
> > > experience to indicate we need the extra functionality looks
> > > premature.
> >
> > Ok, in this case actually I suspect you're right and we can just ditch
> > the exception.  But in general the security discussion is one we should
> > still have.
>
> People like security.
>
> Where do we now stand with this patch, and with "[PATCH 4/8] user ns: hook permission"?

Later today I can send a patch against this set which removes the
the init_task exceptions (out of patch 3 and patch 7), but I'd prefer
to leave the MS_SHARED_NS option (patch 6) in.

thanks,
-serge

_____

Subject: Re: [PATCH 7/8] user ns: handle file sigio
Posted by serue on Fri, 26 Jan 2007 05:38:08 GMT
View Forum Message <> Reply to Message

Quoting Serge E. Hallyn (serue@us.ibm.com):
> Quoting Andrew Morton (akpm@osdl.org):
> > On Wed, 24 Jan 2007 12:58:45 -0600
> > "Serge E. Hallyn" <serue@us.ibm.com> wrote:
> >
> > > > > If we need to I can see doing something special if the process setting
> > > > > fown has CAP_KILL
> > >
> > > > Obviously CAP_KILL is insufficient :)  I assume you mean a new
> > > > CAP_XNS_CAP_KILL?
> > >
> > > > > and bypassing the security checks that way, but
> > > > > hard coding rules like that when it doesn't appear we have any
> > > > > experience to indicate we need the extra functionality looks
> > > > > premature.
> > >
> > > > Ok, in this case actually I suspect you're right and we can just ditch
> > > > the exception.  But in general the security discussion is one we should
> > > > still have.
> > >
> > People like security.
> >
> > Where do we now stand with this patch, and with "[PATCH 4/8] user ns: hook permission"?
>
> Later today I can send a patch against this set which removes the
> the init_task exceptions (out of patch 3 and patch 7), but I'd prefer
> to leave the MS_SHARED_NS option (patch 6) in.
>
> thanks,
> -serge

Boots with USER_NS=n (given Cedric's patch to fix that original problem)
and passes my testcases with USER_NS=y.

From: Serge E. Hallyn <serue@us.ibm.com>
Subject: [PATCH] user namespace: remove exceptions for initial namespace

Both sigio and file access checks for user namespace equivalence
were being skipped for processes in the initial namespace.

Remove these exceptions, enforcing the same cross-namespace
checks for all processes in all user namespaces.

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

---

 fs/fcntl.c            |    3 +--
 include/linux/sched.h |    4 +---
 2 files changed, 2 insertions(+), 5 deletions(-)

939c4da5209a2c00aca70048915007d0eef8ad75
diff --git a/fs/fcntl.c b/fs/fcntl.c
index 6a774c1..d7113d5 100644
--- a/fs/fcntl.c
+++ b/fs/fcntl.c
@@ -460,8 +460,7 @@ static const long band_table[NSIGPOLL] =
 static inline int sigio_perm(struct task_struct *p,
                              struct fown_struct *fown, int sig)
 {
- if (fown->user_ns != init_task.nsproxy->user_ns &&
-     fown->user_ns != p->nsproxy->user_ns)
+ if (fown->user_ns != p->nsproxy->user_ns)
    return 0;
   return (((fown->euid == 0) ||
     (fown->euid == p->suid) || (fown->euid == p->uid) ||
diff --git a/include/linux/sched.h b/include/linux/sched.h
index edbdce2..5c3438b 100644
--- a/include/linux/sched.h
+++ b/include/linux/sched.h
@@ -1614,12 +1614,10 @@ extern int cond_resched_softirq(void);
 static inline int task_mnt_same_uidns(struct task_struct *tsk,
       struct vfsmount *mnt)
 {
- if (tsk->nsproxy == init_task.nsproxy)
+ if (mnt->mnt_user_ns == tsk->nsproxy->user_ns)
    return 1;
    if (mnt->mnt_flags & MNT_SHARE_NS)
     return 1;
- if (mnt->mnt_user_ns == tsk->nsproxy->user_ns)
-   return 1;
   return 0;
 }
 #else
--
1.1.6

_____
Containers mailing list

Subject: Re: [PATCH 7/8] user ns: handle file sigio
Posted by Andrew Morton on Fri, 26 Jan 2007 06:09:16 GMT
View Forum Message <> Reply to Message

On Thu, 25 Jan 2007 23:38:08 -0600
"Serge E. Hallyn" <serue@us.ibm.com> wrote:

> fs/fcntl.c | 3 +--
> include/linux/sched.h | 4 +---
> 2 files changed, 2 insertions(+), 5 deletions(-)

My confidence in this lot:

introduce-and-use-get_task_mnt_ns.patch
introduce-and-use-get_task_mnt_ns-tweaks.patch
nsproxy-externalizes-exit_task_namespaces.patch
user-namespace-add-the-framework.patch
user-namespace-add-the-framework-fix.patch
user-namespace-add-the-framework-fixes.patch
user-ns-add-user_namespace-ptr-to-vfsmount.patch
user-ns-add-user_namespace-ptr-to-vfsmount-fixes.patch
# user-ns-hook-permission.patch: Eric no like
user-ns-hook-permission.patch
user-ns-prepare-copy_tree-copy_mnt-and-their-callers-to-handle-errs.patch
user-ns-prepare-copy_tree-copy_mnt-and-their-callers-to-handle-errs-fix.patch
user-ns-implement-shared-mounts.patch
user-ns-implement-shared-mounts-fixes.patch
# user_ns-handle-file-sigio.patch: Eric no like
user_ns-handle-file-sigio.patch
user_ns-handle-file-sigio-fix.patch
user_ns-handle-file-sigio-fix-2.patch
user-ns-implement-user-ns-unshare.patch
user-ns-implement-user-ns-unshare-tidy.patch
#
rename-attach_pid-to-find_attach_pid.patch
attach_pid-with-struct-pid-parameter.patch
remove-find_attach_pid.patch
statically-initialize-struct-pid-for-swapper.patch
explicitly-set-pgid-sid-of-init.patch
#
uts-namespace-remove-config_uts_ns.patch
ipc-namespace-remove-config_ipc_ns.patch

is very low. If I can by some miracle get all this gunk to compile and

boot a bit, I'd ask that we all very carefully review the patches which
landed and make sure that we're all happy with it.

That's still a couple of days away, I expect.

_____

Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers