Subject: [PATCH 4/6] user ns: hook generic_permission()
Posted by serue on Mon, 04 Jun 2007 19:41:44 GMT
View Forum Message <> Reply to Message

>From nobody Mon Sep 17 00:00:00 2001
From: Serge E. Hallyn <serue@us.ibm.com>
Date: Thu, 5 Apr 2007 17:17:23 -0400
Subject: [PATCH 4/6] user ns: hook generic_permission()

Hook generic_permission() to check for user namespaces.
Also define task_ino_capable() which denies a capability
if the subject task is not in the target inode's user
namespace.

Once user namespace keys exist, it will be possible to
grant a user a capability for a specific user namespace.

We will also need to do a user namespace equiv check for
capabilities with a target task.

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

---

 fs/namei.c | 33 +++++++++++++++----------
 include/linux/user_namespace.h | 51 ++++++++++++++++++++++++++++++++++++++--
 kernel/user_namespace.c | 8 ------
 3 files changed, 70 insertions(+), 22 deletions(-)

ef97a7bb9846d40108cab755ceec37b012b4bcf6
diff --git a/fs/namei.c b/fs/namei.c
index 6326faf..f4fa166 100644
--- a/fs/namei.c
+++ b/fs/namei.c
@@ -32,6 +32,7 @@ #include <linux/capability.h>
 #include <linux/file.h>
 #include <linux/fcntl.h>
 #include <linux/namei.h>
+#include <linux/user_namespace.h>
 #include <asm/namei.h>
 #include <asm/uaccess.h>

@@ -183,8 +184,9 @@ int generic_permission(struct inode *ino
   int (*check_acl)(struct inode *inode, int mask))
 {
  umode_t   mode = inode->i_mode;
+ int   sameuserns = task_inode_same_userns(current, inode);

```
- if (current->fsuid == inode->i_uid)
+ if (sameuserns && current->fsuid == inode->i_uid)
   mode >>= 6;
  else {
   if (IS_POSIXACL(inode) && (mode & S_IRWXG) && check_acl) {
@@ -195,7 +197,7 @@ int generic_permission(struct inode *ino
     return error;
   }

-  if (in_group_p(inode->i_gid))
+  if (sameuserns && in_group_p(inode->i_gid))
    mode >>= 3;
  }

@@ -212,14 +214,14 @@ int generic_permission(struct inode *ino
   */
  if (!(mask & MAY_EXEC) ||
      (inode->i_mode & S_IXUGO) || S_ISDIR(inode->i_mode))
-  if (capable(CAP_DAC_OVERRIDE))
+  if (task_ino_capable(inode, CAP_DAC_OVERRIDE))
    return 0;

  /*
   * Searching includes executable on directories, else just read.
   */
  if (mask == MAY_READ || (S_ISDIR(inode->i_mode) && !(mask & MAY_WRITE)))
-  if (capable(CAP_DAC_READ_SEARCH))
+  if (task_ino_capable(inode, CAP_DAC_READ_SEARCH))
    return 0;

  return -EACCES;
@@ -436,21 +438,25 @@ static int exec_permission_lite(struct i
  if (inode->i_op && inode->i_op->permission)
   return -EAGAIN;

+ if (!task_inode_same_userns(current, inode))
+  goto different_userns;
+
  if (current->fsuid == inode->i_uid)
   mode >>= 6;
  else if (in_group_p(inode->i_gid))
   mode >>= 3;

+different_userns:
  if (mode & MAY_EXEC)
   goto ok;

- if ((inode->i_mode & S_IXUGO) && capable(CAP_DAC_OVERRIDE))
```

```
+ if ((inode->i_mode & S_IXUGO) && task_ino_capable(inode, CAP_DAC_OVERRIDE))
   goto ok;

- if (S_ISDIR(inode->i_mode) && capable(CAP_DAC_OVERRIDE))
+ if (S_ISDIR(inode->i_mode) && task_ino_capable(inode, CAP_DAC_OVERRIDE))
   goto ok;

- if (S_ISDIR(inode->i_mode) && capable(CAP_DAC_READ_SEARCH))
+ if (S_ISDIR(inode->i_mode) && task_ino_capable(inode, CAP_DAC_READ_SEARCH))
   goto ok;

  return -EACCES;
@@ -1339,16 +1345,19 @@ int fastcall __user_walk(const char __us
 /*
  * It's inline, so penalty for filesystems that don't use sticky bit is
  * minimal.
+ *
+ * Alas, the task_inode_same_fsuid() calls are starting to change that.
  */
 static inline int check_sticky(struct inode *dir, struct inode *inode)
 {
  if (!(dir->i_mode & S_ISVTX))
   return 0;
- if (inode->i_uid == current->fsuid)
+ if (task_inode_same_fsuid(current, inode))
   return 0;
- if (dir->i_uid == current->fsuid)
+ if (task_inode_same_fsuid(current, dir))
   return 0;
- return !capable(CAP_FOWNER);
+
+ return !task_ino_capable(inode, CAP_FOWNER);
 }

 /*
@@ -1547,7 +1556,7 @@ int may_open(struct nameidata *nd, int a

  /* O_NOATIME can only be set by the owner or superuser */
  if (flag & O_NOATIME)
-  if (current->fsuid != inode->i_uid && !capable(CAP_FOWNER))
+  if (!task_inode_same_fsuid(current, inode) && !task_ino_capable(inode, CAP_FOWNER))
    return -EPERM;

  /*
@@ -1832,7 +1841,7 @@ int vfs_mknod(struct inode *dir, struct
  if (error)
   return error;
```

```
- if ((S_ISCHR(mode) || S_ISBLK(mode)) && !capable(CAP_MKNOD))
+ if ((S_ISCHR(mode) || S_ISBLK(mode)) && !task_ino_capable(dir, CAP_MKNOD))
   return -EPERM;

  if (!dir->i_op || !dir->i_op->mknod)
diff --git a/include/linux/user_namespace.h b/include/linux/user_namespace.h
index 3b5e040..073f3e0 100644
--- a/include/linux/user_namespace.h
+++ b/include/linux/user_namespace.h
@@ -5,6 +5,7 @@ #include <linux/kref.h>
 #include <linux/nsproxy.h>
 #include <linux/sched.h>
 #include <linux/err.h>
+#include <linux/fs.h>

 #define UIDHASH_BITS (CONFIG_BASE_SMALL ? 3 : 8)
 #define UIDHASH_SZ (1 << UIDHASH_BITS)
@@ -37,7 +38,6 @@ static inline void put_user_ns(struct us
 }

 struct user_namespace *task_user_ns(struct task_struct *tsk);
-struct user_namespace *inode_user_ns(struct inode *inode);
 /*
  * task_inode_same_userns:
  * If inode->i_userns is NULL, the fs does not support user namespaces, so any
@@ -51,10 +51,43 @@ struct user_namespace *inode_user_ns(str
 static inline int
 task_inode_same_userns(struct task_struct *tsk, struct inode *inode)
 {
- struct user_namespace *ino_userns = inode_user_ns(inode);
+ struct user_namespace *ino_userns = inode->i_userns;

  return (!ino_userns || task_user_ns(tsk)== ino_userns);
 }
+
+/*
+ * task_inode_same_fsuid:
+ * Does the inode's user equal the task's fsuid.
+ * Same behavior applies as for task_inode_same_userns() above.
+ * So once userns keys are implemented in the keyring, this will
+ * be modified check whether (inode->i_userns, inode->i_uid) is in
+ * tsk->user's keychain.
+ */
+static inline int
+task_inode_same_fsuid(struct task_struct *tsk, struct inode *inode)
+{
+ struct user_namespace *ino_userns = inode->i_userns;
+
```

```
+ if (ino_userns && task_user_ns(tsk) != ino_userns)
+  return 0;
+ if (tsk->fsuid != inode->i_uid)
+  return 0;
+ return 1;
+}
+
+/*
+ * task_ino_capable:
+ * Again, when userns keys exist, we will need to check for a
+ * (inode->i_userns, cap) key if !task_inode_same_userns(current, inode)
+ */
+static inline int
+task_ino_capable(struct inode *inode, int cap)
+{
+ if (!task_inode_same_userns(current, inode))
+  return 0;
+ return capable(cap);
+}
 #else

 static inline struct user_namespace *get_user_ns(struct user_namespace *ns)
@@ -80,6 +113,20 @@ task_inode_same_userns(struct task_struc
 {
  return 1;
 }
+
+static inline int
+task_inode_same_fsuid(struct task_struct *tsk, struct inode *inode)
+{
+ if (tsk->fsuid != inode->i_uid)
+  return 0;
+ return 1;
+}
+
+static inline int
+task_ino_capable(struct inode *inode, int cap)
+{
+ return capable(cap);
+}
 #endif

 #endif /* _LINUX_USER_H */
diff --git a/kernel/user_namespace.c b/kernel/user_namespace.c
index f487361..ffabd04 100644
--- a/kernel/user_namespace.c
+++ b/kernel/user_namespace.c
@@ -94,12 +94,4 @@ struct user_namespace *get_task_user_ns(
```

```
  return get_user_ns(task_user_ns(tsk));
 }

-/* should this just go into fs.h? */
-#include <linux/fs.h>
-struct user_namespace *inode_user_ns(struct inode *inode)
-{
- return inode->i_userns;
-}
-
-
 #endif /* CONFIG_USER_NS */
--
1.3.2
```