Subject: [patch 04/10] unprivileged mounts: account user mounts
Posted by Miklos Szeredi on Wed, 16 Jan 2008 12:31:51 GMT
View Forum Message <> Reply to Message

From: Miklos Szeredi <mszeredi@suse.cz>

Add sysctl variables for accounting and limiting the number of user
mounts.

The maximum number of user mounts is set to 1024 by default. This
won't in itself enable user mounts, setting a mount to be owned by a
user is first needed.

[akpm]
- don't use enumerated sysctls

Signed-off-by: Miklos Szeredi <mszeredi@suse.cz>
Acked-by: Serge Hallyn <serue@us.ibm.com>

Index: linux/Documentation/filesystems/proc.txt
===
--- linux.orig/Documentation/filesystems/proc.txt 2008-01-16 13:24:53.000000000 +0100
+++ linux/Documentation/filesystems/proc.txt 2008-01-16 13:25:07.000000000 +0100
@@ -1012,6 +1012,15 @@ reaches aio-max-nr then io_setup will fa
raising aio-max-nr does not result in the pre-allocation or re-sizing
of any kernel data structures.

+nr_user_mounts and max_user_mounts
+---------------------------------
+
+These represent the number of "user" mounts and the maximum number of
+"user" mounts respectively. User mounts may be created by
+unprivileged users. User mounts may also be created with sysadmin
+privileges on behalf of a user, in which case nr_user_mounts may
+exceed max_user_mounts.
+
2.2 /proc/sys/fs/binfmt_misc - Miscellaneous binary formats

Index: linux/fs/namespace.c
===
--- linux.orig/fs/namespace.c 2008-01-16 13:25:07.000000000 +0100
+++ linux/fs/namespace.c 2008-01-16 13:25:07.000000000 +0100
@@ -44,6 +44,9 @@ static struct list_head *mount_hashtable
static struct kmem_cache *mnt_cache __read_mostly;
static struct rw_semaphore namespace_sem;

```
+int nr_user_mounts;
+int max_user_mounts = 1024;
+
 /* /sys/fs */
 struct kobject *fs_kobj;
 EXPORT_SYMBOL_GPL(fs_kobj);
@@ -477,21 +480,70 @@ static struct vfsmount *skip_mnt_tree(st
  return p;
 }

-static void set_mnt_user(struct vfsmount *mnt)
+static void dec_nr_user_mounts(void)
+{
+ spin_lock(&vfsmount_lock);
+ nr_user_mounts--;
+ spin_unlock(&vfsmount_lock);
+}
+
+static int reserve_user_mount(void)
+{
+ int err = 0;
+
+ spin_lock(&vfsmount_lock);
+ /*
+  * EMFILE was error returned by mount(2) in the old days, when
+  * the mount count was limited.  Reuse this error value to
+  * mean, that the maximum number of user mounts has been
+  * exceeded.
+  */
+ if (nr_user_mounts >= max_user_mounts && !capable(CAP_SYS_ADMIN))
+  err = -EMFILE;
+ else
+  nr_user_mounts++;
+ spin_unlock(&vfsmount_lock);
+ return err;
+}
+
+static void __set_mnt_user(struct vfsmount *mnt)
 {
  WARN_ON(mnt->mnt_flags & MNT_USER);
  mnt->mnt_uid = current->fsuid;
  mnt->mnt_flags |= MNT_USER;
 }

+static void set_mnt_user(struct vfsmount *mnt)
+{
+ __set_mnt_user(mnt);
+ spin_lock(&vfsmount_lock);
```

```
+ nr_user_mounts++;
+ spin_unlock(&vfsmount_lock);
+}
+
+static void clear_mnt_user(struct vfsmount *mnt)
+{
+ if (mnt->mnt_flags & MNT_USER) {
+  mnt->mnt_uid = 0;
+  mnt->mnt_flags &= ~MNT_USER;
+  dec_nr_user_mounts();
+ }
+}
+
 static struct vfsmount *clone_mnt(struct vfsmount *old, struct dentry *root,
       int flag)
 {
  struct super_block *sb = old->mnt_sb;
- struct vfsmount *mnt = alloc_vfsmnt(old->mnt_devname);
+ struct vfsmount *mnt;

+ if (flag & CL_SETUSER) {
+  int err = reserve_user_mount();
+  if (err)
+   return ERR_PTR(err);
+ }
+ mnt = alloc_vfsmnt(old->mnt_devname);
  if (!mnt)
-  return ERR_PTR(-ENOMEM);
+  goto alloc_failed;

  mnt->mnt_flags = old->mnt_flags;
  atomic_inc(&sb->s_active);
@@ -503,7 +555,7 @@ static struct vfsmount *clone_mnt(struct
  /* don't copy the MNT_USER flag */
  mnt->mnt_flags &= ~MNT_USER;
  if (flag & CL_SETUSER)
-  set_mnt_user(mnt);
+  __set_mnt_user(mnt);

  if (flag & CL_SLAVE) {
   list_add(&mnt->mnt_slave, &old->mnt_slave_list);
@@ -528,6 +580,11 @@ static struct vfsmount *clone_mnt(struct
   spin_unlock(&vfsmount_lock);
  }
  return mnt;
+
+ alloc_failed:
+ if (flag & CL_SETUSER)
```

```
+  dec_nr_user_mounts();
+ return ERR_PTR(-ENOMEM);
 }

 static inline void __mntput(struct vfsmount *mnt)
@@ -543,6 +600,7 @@ static inline void __mntput(struct vfsmo
   */
  WARN_ON(atomic_read(&mnt->__mnt_writers));
  dput(mnt->mnt_root);
+ clear_mnt_user(mnt);
  free_vfsmnt(mnt);
  deactivate_super(sb);
 }
@@ -1307,6 +1365,7 @@ static int do_remount(struct nameidata *
  else
   err = do_remount_sb(sb, flags, data, 0);
  if (!err) {
+  clear_mnt_user(nd->path.mnt);
   nd->path.mnt->mnt_flags = mnt_flags;
   if (flags & MS_SETUSER)
    set_mnt_user(nd->path.mnt);
Index: linux/include/linux/fs.h
===================================================================
--- linux.orig/include/linux/fs.h 2008-01-16 13:25:05.000000000 +0100
+++ linux/include/linux/fs.h 2008-01-16 13:25:07.000000000 +0100
@@ -50,6 +50,9 @@ extern struct inodes_stat_t inodes_stat;

 extern int leases_enable, lease_break_time;

+extern int nr_user_mounts;
+extern int max_user_mounts;
+
 #ifdef CONFIG_DNOTIFY
 extern int dir_notify_enable;
 #endif
Index: linux/kernel/sysctl.c
===================================================================
--- linux.orig/kernel/sysctl.c 2008-01-16 13:24:53.000000000 +0100
+++ linux/kernel/sysctl.c 2008-01-16 13:25:07.000000000 +0100
@@ -1288,6 +1288,22 @@ static struct ctl_table fs_table[] = {
 #endif
 #endif
  {
+  .ctl_name = CTL_UNNUMBERED,
+  .procname = "nr_user_mounts",
+  .data  = &nr_user_mounts,
+  .maxlen  = sizeof(int),
+  .mode  = 0444,
```

```
+  .proc_handler = &proc_dointvec,
+ },
+ {
+  .ctl_name = CTL_UNNUMBERED,
+  .procname = "max_user_mounts",
+  .data  = &max_user_mounts,
+  .maxlen  = sizeof(int),
+  .mode  = 0644,
+  .proc_handler = &proc_dointvec,
+ },
+ {
   .ctl_name = KERN_SETUID_DUMPABLE,
   .procname = "suid_dumpable",
   .data  = &suid_dumpable,

--
```