Subject: [patch 00/10] mount ownership and unprivileged mount syscall (v5)
Posted by Miklos Szeredi on Fri, 27 Apr 2007 12:04:17 GMT
View Forum Message <> Reply to Message

v4 -> v5:

- fold back Andrew's changes
- fold back my update patch:
  - o use fsuid instead of ruid
  - o allow forced unpriv. unmounts for "safe" filesystems
  - o allow mounting over special files, but not over symlinks
  - o set nosuid and nodev based on lack of specific capability
- patch header updates
- new patch: on propagation inherit owner from parent
- new patch: add "no submounts" mount flag

The last two patches are up for discussion.

The rest I think is in pretty good shape for merging. If somebody feels otherwise, please complain now.

--

_____

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

Subject: [patch 01/10] unprivileged mounts: add user mounts to the kernel
Posted by Miklos Szeredi on Fri, 27 Apr 2007 12:04:18 GMT
View Forum Message <> Reply to Message

From: Miklos Szeredi <mszeredi@suse.cz>

This patchset adds support for keeping mount ownership information in the kernel, and allow unprivileged mount(2) and umount(2) in certain cases.

The mount owner has the following privileges:

- unmount the owned mount
- create a submount under the owned mount

The sysadmin can set the owner explicitly on mount and remount. When an unprivileged user creates a mount, then the owner is automatically set to the user.

The following use cases are envisioned:

1) Private namespace, with selected mounts owned by user. E.g. /home/$USER is a good candidate for allowing unpriv mounts and unmounts within.

2) Private namespace, with all mounts owned by user and having the "nosuid" flag. User can mount and umount anywhere within the namespace, but suid programs will not work.

3) Global namespace, with a designated directory, which is a mount owned by the user. E.g. /mnt/users/$USER is set up so that it is bind mounted onto itself, and set to be owned by $USER. The user can add/remove mounts only under this directory.

The following extra security measures are taken for unprivileged mounts:

- usermounts are limited by a sysctl tunable
- force "nosuid,nodev" mount options on the created mount

For testing unprivileged mounts (and for other purposes) simple mount/umount utilities are available from:

http://www.kernel.org/pub/linux/kernel/people/mszeredi/mmount/

I'll also submit a patch to util-linux, to add the same functionality to mount(8) and umount(8).

This patch:

A new mount flag, MS_SETUSER is used to make a mount owned by a user. If this flag is specified, then the owner will be set to the current fsuid and the mount will be marked with the MNT_USER flag. On remount don't preserve previous owner, and treat MS_SETUSER as for a new mount. The MS_SETUSER flag is ignored on mount move.

The MNT_USER flag is not copied on any kind of mount cloning: namespace creation, binding or propagation. For bind mounts the cloned mount(s) are set to MNT_USER depending on the MS_SETUSER mount flag. In all the other cases MNT_USER is always cleared.

For MNT_USER mounts a "user=UID" option is added to /proc/PID/mounts. This is compatible with how mount ownership is stored in /etc/mtab.

The rationale for using MS_SETUSER and MNT_USER, to distinguish "user" mounts from "non-user" or "legacy" mounts are follows:

a) Mount(2) and umount(2) on legacy mounts always need CAP_SYS_ADMIN

```
    capability.  As opposed to user mounts, which will only require,
    that the mount owner matches the current fsuid.  So a process
    with fsuid=0 should not be able to mount/umount legacy mounts
    without the CAP_SYS_ADMIN capability.

  b) Legacy userspace programs may set fsuid to nonzero before calling
     mount(2).  In such an unlikely case, this patchset would cause
     an unintended side effect of making the mount owned by the fsuid.

  c) For legacy mounts, no "user=UID" option should be shown in
     /proc/mounts for backwards compatibility.

Signed-off-by: Miklos Szeredi <mszeredi@suse.cz>
---

Index: linux/fs/namespace.c
===================================================================
--- linux.orig/fs/namespace.c 2007-04-26 13:08:35.000000000 +0200
+++ linux/fs/namespace.c 2007-04-26 13:10:48.000000000 +0200
@@ -227,6 +227,13 @@ static struct vfsmount *skip_mnt_tree(st
  return p;
 }

+static void set_mnt_user(struct vfsmount *mnt)
+{
+ BUG_ON(mnt->mnt_flags & MNT_USER);
+ mnt->mnt_uid = current->fsuid;
+ mnt->mnt_flags |= MNT_USER;
+}
+
 static struct vfsmount *clone_mnt(struct vfsmount *old, struct dentry *root,
       int flag)
 {
@@ -241,6 +248,11 @@ static struct vfsmount *clone_mnt(struct
   mnt->mnt_mountpoint = mnt->mnt_root;
   mnt->mnt_parent = mnt;

+  /* don't copy the MNT_USER flag */
+  mnt->mnt_flags &= ~MNT_USER;
+  if (flag & CL_SETUSER)
+   set_mnt_user(mnt);
+
   if (flag & CL_SLAVE) {
    list_add(&mnt->mnt_slave, &old->mnt_slave_list);
    mnt->mnt_master = old;
@@ -403,6 +415,8 @@ static int show_vfsmnt(struct seq_file *
   if (mnt->mnt_flags & fs_infop->flag)
    seq_puts(m, fs_infop->str);
```

```
  }
+ if (mnt->mnt_flags & MNT_USER)
+  seq_printf(m, ",user=%i", mnt->mnt_uid);
  if (mnt->mnt_sb->s_op->show_options)
   err = mnt->mnt_sb->s_op->show_options(m, mnt);
  seq_puts(m, " 0 0\n");
@@ -923,8 +937,9 @@ 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 flags)
 {
+ int clone_flags;
  struct nameidata old_nd;
  struct vfsmount *mnt = NULL;
  int err = mount_is_safe(nd);
@@ -944,11 +959,12 @@ static int do_loopback(struct nameidata
  if (!check_mnt(nd->mnt) || !check_mnt(old_nd.mnt))
   goto out;

+ clone_flags = (flags & MS_SETUSER) ? CL_SETUSER : 0;
  err = -ENOMEM;
- if (recurse)
-  mnt = copy_tree(old_nd.mnt, old_nd.dentry, 0);
+ if (flags & MS_REC)
+  mnt = copy_tree(old_nd.mnt, old_nd.dentry, clone_flags);
  else
-  mnt = clone_mnt(old_nd.mnt, old_nd.dentry, 0);
+  mnt = clone_mnt(old_nd.mnt, old_nd.dentry, clone_flags);

  if (!mnt)
   goto out;
@@ -990,8 +1006,11 @@ static int do_remount(struct nameidata *

  down_write(&sb->s_umount);
  err = do_remount_sb(sb, flags, data, 0);
- if (!err)
+ if (!err) {
   nd->mnt->mnt_flags = mnt_flags;
+  if (flags & MS_SETUSER)
+   set_mnt_user(nd->mnt);
+ }
  up_write(&sb->s_umount);
  if (!err)
   security_sb_post_remount(nd->mnt, flags, data);
@@ -1096,10 +1115,13 @@ static int do_new_mount(struct nameidata
  if (!capable(CAP_SYS_ADMIN))
```

```
   return -EPERM;

- mnt = do_kern_mount(type, flags, name, data);
+ mnt = do_kern_mount(type, flags & ~MS_SETUSER, name, data);
  if (IS_ERR(mnt))
   return PTR_ERR(mnt);

+ if (flags & MS_SETUSER)
+  set_mnt_user(mnt);
+
  return do_add_mount(mnt, nd, mnt_flags, NULL);
 }

@@ -1130,7 +1152,8 @@ int do_add_mount(struct vfsmount *newmnt
  if (S_ISLNK(newmnt->mnt_root->d_inode->i_mode))
   goto unlock;

- newmnt->mnt_flags = mnt_flags;
+ /* MNT_USER was set earlier */
+ newmnt->mnt_flags |= mnt_flags;
  if ((err = graft_tree(newmnt, nd)))
   goto unlock;

@@ -1450,7 +1473,7 @@ 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);
  else if (flags & (MS_SHARED | MS_PRIVATE | MS_SLAVE | MS_UNBINDABLE))
   retval = do_change_type(&nd, flags);
  else if (flags & MS_MOVE)
Index: linux/fs/pnode.h
===================================================================
--- linux.orig/fs/pnode.h 2007-04-26 13:08:35.000000000 +0200
+++ linux/fs/pnode.h 2007-04-26 13:08:36.000000000 +0200
@@ -22,6 +22,7 @@
 #define CL_COPY_ALL   0x04
 #define CL_MAKE_SHARED   0x08
 #define CL_PROPAGATION   0x10
+#define CL_SETUSER  0x20

 static inline void set_mnt_shared(struct vfsmount *mnt)
 {
Index: linux/include/linux/fs.h
===================================================================
--- linux.orig/include/linux/fs.h 2007-04-26 13:08:35.000000000 +0200
+++ linux/include/linux/fs.h 2007-04-26 13:08:36.000000000 +0200
```

```
@@ -123,6 +123,7 @@ extern int dir_notify_enable;
 #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_SETUSER (1<<22) /* set mnt_uid to current user */
 #define MS_ACTIVE (1<<30)
 #define MS_NOUSER (1<<31)

Index: linux/include/linux/mount.h
===================================================================
--- linux.orig/include/linux/mount.h 2007-04-26 13:08:35.000000000 +0200
+++ linux/include/linux/mount.h 2007-04-26 13:08:36.000000000 +0200
@@ -30,6 +30,7 @@ struct mnt_namespace;
 #define MNT_RELATIME 0x20

 #define MNT_SHRINKABLE 0x100
+#define MNT_USER 0x200

 #define MNT_SHARED 0x1000 /* if the vfsmount is a shared mount */
 #define MNT_UNBINDABLE 0x2000 /* if the vfsmount is a unbindable mount */
@@ -61,6 +62,8 @@ struct vfsmount {
   atomic_t mnt_count;
   int mnt_expiry_mark;  /* true if marked for expiry */
   int mnt_pinned;
+
+ uid_t mnt_uid;   /* owner of the mount */
 };

 static inline struct vfsmount *mntget(struct vfsmount *mnt)

--

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

Subject: [patch 02/10] unprivileged mounts: allow unprivileged umount
Posted by Miklos Szeredi on Fri, 27 Apr 2007 12:04:19 GMT
View Forum Message <> Reply to Message

From: Miklos Szeredi <mszeredi@suse.cz>

The owner doesn't need sysadmin capabilities to call umount().

Similar behavior as umount(8) on mounts having "user=UID" option in /etc/mtab.
The difference is that umount also checks /etc/fstab, presumably to exclude
another mount on the same mountpoint.

Signed-off-by: Miklos Szeredi <mszeredi@suse.cz>

Index: linux/fs/namespace.c
===
--- linux.orig/fs/namespace.c 2007-04-26 13:10:48.000000000 +0200
+++ linux/fs/namespace.c 2007-04-26 13:16:21.000000000 +0200
@@ -658,6 +658,27 @@ static int do_umount(struct vfsmount *mn
  return retval;
 }

+static bool is_mount_owner(struct vfsmount *mnt, uid_t uid)
+{
+ return (mnt->mnt_flags & MNT_USER) && mnt->mnt_uid == uid;
+}
+
+/*
+ * umount is permitted for
+ * - sysadmin
+ * - mount owner, if not forced umount
+ */
+static bool permit_umount(struct vfsmount *mnt, int flags)
+{
+ if (capable(CAP_SYS_ADMIN))
+ return true;
+
+ if (flags & MNT_FORCE)
+ return false;
+
+ return is_mount_owner(mnt, current->fsuid);
+}
+
 /*
  * Now umount can handle mount points as well as block devices.
  * This is important for filesystems which use unnamed block devices.
@@ -681,7 +702,7 @@ asmlinkage long sys_umount(char __user *
   goto dput_and_out;

  retval = -EPERM;
- if (!capable(CAP_SYS_ADMIN))
+ if (!permit_umount(nd.mnt, flags))
   goto dput_and_out;

  retval = do_umount(nd.mnt, flags);

--
_____

Subject: [patch 03/10] unprivileged mounts: account user mounts
Posted by Miklos Szeredi on Fri, 27 Apr 2007 12:04:20 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>

Index: linux/Documentation/filesystems/proc.txt
===
--- linux.orig/Documentation/filesystems/proc.txt 2007-04-26 13:08:35.000000000 +0200
+++ linux/Documentation/filesystems/proc.txt 2007-04-26 13:17:13.000000000 +0200
@@ -923,6 +923,15 @@ 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 2007-04-26 13:16:21.000000000 +0200
+++ linux/fs/namespace.c 2007-04-26 13:17:13.000000000 +0200

```
@@ -39,6 +39,9 @@ static int hash_mask __read_mostly, hash
 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 */
 decl_subsys(fs, NULL, NULL);
 EXPORT_SYMBOL_GPL(fs_subsys);
@@ -227,11 +230,30 @@ static struct vfsmount *skip_mnt_tree(st
  return p;
 }

+static void dec_nr_user_mounts(void)
+{
+ spin_lock(&vfsmount_lock);
+ nr_user_mounts--;
+ spin_unlock(&vfsmount_lock);
+}
+
 static void set_mnt_user(struct vfsmount *mnt)
 {
  BUG_ON(mnt->mnt_flags & MNT_USER);
  mnt->mnt_uid = current->fsuid;
  mnt->mnt_flags |= MNT_USER;
+ 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,
@@ -283,6 +305,7 @@ static inline void __mntput(struct vfsmo
 {
  struct super_block *sb = mnt->mnt_sb;
  dput(mnt->mnt_root);
+ clear_mnt_user(mnt);
  free_vfsmnt(mnt);
  deactivate_super(sb);
```

```
 }
@@ -1028,6 +1051,7 @@ static int do_remount(struct nameidata *
  down_write(&sb->s_umount);
  err = do_remount_sb(sb, flags, data, 0);
  if (!err) {
+  clear_mnt_user(nd->mnt);
   nd->mnt->mnt_flags = mnt_flags;
   if (flags & MS_SETUSER)
    set_mnt_user(nd->mnt);
Index: linux/include/linux/fs.h
===================================================================
--- linux.orig/include/linux/fs.h 2007-04-26 13:08:36.000000000 +0200
+++ linux/include/linux/fs.h 2007-04-26 13:17:13.000000000 +0200
@@ -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 2007-04-26 13:08:35.000000000 +0200
+++ linux/kernel/sysctl.c 2007-04-26 13:17:13.000000000 +0200
@@ -1064,6 +1064,22 @@ static 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,

--
```

_____

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

---

Subject: [patch 04/10] unprivileged mounts: propagate error values from clone_mnt
Posted by Miklos Szeredi on Fri, 27 Apr 2007 12:04:21 GMT
View Forum Message <> Reply to Message

From: Miklos Szeredi <mszeredi@suse.cz>

Allow clone_mnt() to return errors other than ENOMEM. This will be used for returning a different error value when the number of user mounts goes over the limit.

Fix copy_tree() to return EPERM for unbindable mounts.

Don't propagate further from dup_mnt_ns() as that copy_tree() can only fail with -ENOMEM.

Signed-off-by: Miklos Szeredi <mszeredi@suse.cz>

Index: linux/fs/namespace.c
```
===================================================================
--- linux.orig/fs/namespace.c 2007-04-26 13:17:13.000000000 +0200
+++ linux/fs/namespace.c 2007-04-26 13:18:46.000000000 +0200
@@ -262,41 +262,42 @@ static struct vfsmount *clone_mnt(struct
  struct super_block *sb = old->mnt_sb;
  struct vfsmount *mnt = alloc_vfsmnt(old->mnt_devname);

- if (mnt) {
-  mnt->mnt_flags = old->mnt_flags;
-  atomic_inc(&sb->s_active);
-  mnt->mnt_sb = sb;
-  mnt->mnt_root = dget(root);
-  mnt->mnt_mountpoint = mnt->mnt_root;
-  mnt->mnt_parent = mnt;
-
-  /* don't copy the MNT_USER flag */
-  mnt->mnt_flags &= ~MNT_USER;
-  if (flag & CL_SETUSER)
-   set_mnt_user(mnt);
```

```
-
-  if (flag & CL_SLAVE) {
-   list_add(&mnt->mnt_slave, &old->mnt_slave_list);
-   mnt->mnt_master = old;
-   CLEAR_MNT_SHARED(mnt);
-  } else {
-   if ((flag & CL_PROPAGATION) || IS_MNT_SHARED(old))
-    list_add(&mnt->mnt_share, &old->mnt_share);
-   if (IS_MNT_SLAVE(old))
-    list_add(&mnt->mnt_slave, &old->mnt_slave);
-   mnt->mnt_master = old->mnt_master;
-  }
-  if (flag & CL_MAKE_SHARED)
-   set_mnt_shared(mnt);
+ if (!mnt)
+  return ERR_PTR(-ENOMEM);

-  /* stick the duplicate mount on the same expiry list
-   * as the original if that was on one */
-  if (flag & CL_EXPIRE) {
-   spin_lock(&vfsmount_lock);
-   if (!list_empty(&old->mnt_expire))
-    list_add(&mnt->mnt_expire, &old->mnt_expire);
-   spin_unlock(&vfsmount_lock);
-  }
+ mnt->mnt_flags = old->mnt_flags;
+ atomic_inc(&sb->s_active);
+ mnt->mnt_sb = sb;
+ mnt->mnt_root = dget(root);
+ mnt->mnt_mountpoint = mnt->mnt_root;
+ mnt->mnt_parent = mnt;
+
+ /* don't copy the MNT_USER flag */
+ mnt->mnt_flags &= ~MNT_USER;
+ if (flag & CL_SETUSER)
+  set_mnt_user(mnt);
+
+ if (flag & CL_SLAVE) {
+  list_add(&mnt->mnt_slave, &old->mnt_slave_list);
+  mnt->mnt_master = old;
+  CLEAR_MNT_SHARED(mnt);
+ } else {
+  if ((flag & CL_PROPAGATION) || IS_MNT_SHARED(old))
+   list_add(&mnt->mnt_share, &old->mnt_share);
+  if (IS_MNT_SLAVE(old))
+   list_add(&mnt->mnt_slave, &old->mnt_slave);
+  mnt->mnt_master = old->mnt_master;
+ }
```

```
+ if (flag & CL_MAKE_SHARED)
+  set_mnt_shared(mnt);
+
+ /* stick the duplicate mount on the same expiry list
+  * as the original if that was on one */
+ if (flag & CL_EXPIRE) {
+  spin_lock(&vfsmount_lock);
+  if (!list_empty(&old->mnt_expire))
+   list_add(&mnt->mnt_expire, &old->mnt_expire);
+  spin_unlock(&vfsmount_lock);
  }
  return mnt;
 }
@@ -783,11 +784,11 @@ struct vfsmount *copy_tree(struct vfsmou
  struct nameidata nd;

  if (!(flag & CL_COPY_ALL) && IS_MNT_UNBINDABLE(mnt))
-  return NULL;
+  return ERR_PTR(-EPERM);

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

  p = mnt;
@@ -808,8 +809,8 @@ 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 (IS_ERR(q))
+    goto error;
    spin_lock(&vfsmount_lock);
    list_add_tail(&q->mnt_list, &res->mnt_list);
    attach_mnt(q, &nd);
@@ -817,7 +818,7 @@ struct vfsmount *copy_tree(struct vfsmou
   }
  }
  return res;
-Enomem:
+ error:
  if (res) {
   LIST_HEAD(umount_list);
   spin_lock(&vfsmount_lock);
```

```
@@ -825,7 +826,7 @@ Enomem:
   spin_unlock(&vfsmount_lock);
   release_mounts(&umount_list);
  }
- return NULL;
+ return q;
 }

 /*
@@ -1004,13 +1005,13 @@ static int do_loopback(struct nameidata
   goto out;

  clone_flags = (flags & MS_SETUSER) ? CL_SETUSER : 0;
- err = -ENOMEM;
  if (flags & MS_REC)
   mnt = copy_tree(old_nd.mnt, old_nd.dentry, clone_flags);
  else
   mnt = clone_mnt(old_nd.mnt, old_nd.dentry, clone_flags);

- if (!mnt)
+ err = PTR_ERR(mnt);
+ if (IS_ERR(mnt))
   goto out;

  err = graft_tree(mnt, nd);
@@ -1555,7 +1556,7 @@ static struct mnt_namespace *dup_mnt_ns(
  /* 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 (IS_ERR(new_ns->root)) {
   up_write(&namespace_sem);
   kfree(new_ns);
   return NULL;
Index: linux/fs/pnode.c
===================================================================
--- linux.orig/fs/pnode.c 2007-04-26 13:08:35.000000000 +0200
+++ linux/fs/pnode.c 2007-04-26 13:18:46.000000000 +0200
@@ -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 (IS_ERR(child)) {
+   ret = PTR_ERR(child);
    list_splice(tree_list, tmp_list.prev);
```

```
  goto out;
 }
```

--

_____

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

---

Subject: [patch 05/10] unprivileged mounts: allow unprivileged bind mounts
Posted by Miklos Szeredi on Fri, 27 Apr 2007 12:04:22 GMT
View Forum Message <> Reply to Message

From: Miklos Szeredi <mszeredi@suse.cz>

Allow bind mounts to unprivileged users if the following conditions are met:

- mountpoint is not a symlink
- parent mount is owned by the user
- the number of user mounts is below the maximum

Unprivileged mounts imply MS_SETUSER, and will also have the "nosuid" and
"nodev" mount flags set.

In particular, if mounting process doesn't have CAP_SETUID capability,
then the "nosuid" flag will be added, and if it doesn't have CAP_MKNOD
capability, then the "nodev" flag will be added.

Signed-off-by: Miklos Szeredi <mszeredi@suse.cz>

Index: linux/fs/namespace.c
```
===================================================================
--- linux.orig/fs/namespace.c 2007-04-26 13:18:46.000000000 +0200
+++ linux/fs/namespace.c 2007-04-26 13:30:04.000000000 +0200
@@ -237,11 +237,34 @@ static void dec_nr_user_mounts(void)
  spin_unlock(&vfsmount_lock);
 }

-static void set_mnt_user(struct vfsmount *mnt)
+static int reserve_user_mount(void)
+{
+ int err = 0;
+
+ spin_lock(&vfsmount_lock);
+ if (nr_user_mounts >= max_user_mounts && !capable(CAP_SYS_ADMIN))
+  err = -EPERM;
```

```
+ else
+  nr_user_mounts++;
+ spin_unlock(&vfsmount_lock);
+ return err;
+}
+
+static void __set_mnt_user(struct vfsmount *mnt)
 {
  BUG_ON(mnt->mnt_flags & MNT_USER);
  mnt->mnt_uid = current->fsuid;
  mnt->mnt_flags |= MNT_USER;
+
+ if (!capable(CAP_SETUID))
+  mnt->mnt_flags |= MNT_NOSUID;
+ if (!capable(CAP_MKNOD))
+  mnt->mnt_flags |= MNT_NODEV;
+}
+
+static void set_mnt_user(struct vfsmount *mnt)
+{
+ __set_mnt_user(mnt);
  spin_lock(&vfsmount_lock);
  nr_user_mounts++;
  spin_unlock(&vfsmount_lock);
@@ -260,10 +283,16 @@ 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 (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);
@@ -275,7 +304,7 @@ 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);
@@ -300,6 +329,11 @@ 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)
@@ -748,22 +782,26 @@ asmlinkage long sys_oldumount(char __use

 #endif

-static int mount_is_safe(struct nameidata *nd)
+/*
+ * Conditions for unprivileged mounts are:
+ * - mountpoint is not a symlink
+ * - mountpoint is in a mount owned by the user
+ */
+static bool permit_mount(struct nameidata *nd, int *flags)
 {
+ struct inode *inode = nd->dentry->d_inode;
+
  if (capable(CAP_SYS_ADMIN))
-  return 0;
- return -EPERM;
-#ifdef notyet
- if (S_ISLNK(nd->dentry->d_inode->i_mode))
-  return -EPERM;
- if (nd->dentry->d_inode->i_mode & S_ISVTX) {
-  if (current->uid != nd->dentry->d_inode->i_uid)
-   return -EPERM;
- }
- if (vfs_permission(nd, MAY_WRITE))
-  return -EPERM;
- return 0;
-#endif
+  return true;
+
+ if (S_ISLNK(inode->i_mode))
+  return false;
```

```
+
+ if (!is_mount_owner(nd->mnt, current->fsuid))
+  return false;
+
+ *flags |= MS_SETUSER;
+ return true;
 }

 static int lives_below_in_same_fs(struct dentry *d, struct dentry *dentry)
@@ -987,9 +1025,10 @@ static int do_loopback(struct nameidata
  int clone_flags;
  struct nameidata old_nd;
  struct vfsmount *mnt = NULL;
- int err = mount_is_safe(nd);
- if (err)
-  return err;
+ int err;
+
+ if (!permit_mount(nd, &flags))
+  return -EPERM;
  if (!old_name || !*old_name)
   return -EINVAL;
  err = path_lookup(old_name, LOOKUP_FOLLOW, &old_nd);
```

--

Subject: [patch 06/10] unprivileged mounts: put declaration of put_filesystem() in fs.h

Posted by Miklos Szeredi on Fri, 27 Apr 2007 12:04:23 GMT

View Forum Message <> Reply to Message

From: Miklos Szeredi <mszeredi@suse.cz>

Declarations go into headers.

Signed-off-by: Miklos Szeredi <mszeredi@suse.cz>

Index: linux/fs/super.c
```
===================================================================
--- linux.orig/fs/super.c 2007-04-26 13:08:34.000000000 +0200
+++ linux/fs/super.c 2007-04-26 13:46:26.000000000 +0200
@@ -40,10 +40,6 @@
```

```
 #include <asm/uaccess.h>


-void get_filesystem(struct file_system_type *fs);
-void put_filesystem(struct file_system_type *fs);
-struct file_system_type *get_fs_type(const char *name);
-
 LIST_HEAD(super_blocks);
 DEFINE_SPINLOCK(sb_lock);

Index: linux/include/linux/fs.h
===================================================================
--- linux.orig/include/linux/fs.h 2007-04-26 13:17:13.000000000 +0200
+++ linux/include/linux/fs.h 2007-04-26 13:46:26.000000000 +0200
@@ -1919,6 +1919,8 @@ extern int vfs_fstat(unsigned int, struc

 extern int vfs_ioctl(struct file *, unsigned int, unsigned int, unsigned long);

+extern void get_filesystem(struct file_system_type *fs);
+extern void put_filesystem(struct file_system_type *fs);
 extern struct file_system_type *get_fs_type(const char *name);
 extern struct super_block *get_super(struct block_device *);
 extern struct super_block *user_get_super(dev_t);

--

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

Subject: [patch 07/10] unprivileged mounts: allow unprivileged mounts
Posted by Miklos Szeredi on Fri, 27 Apr 2007 12:04:24 GMT
View Forum Message <> Reply to Message

From: Miklos Szeredi <mszeredi@suse.cz>

Define a new fs flag FS_SAFE, which denotes, that unprivileged mounting of
this filesystem may not constitute a security problem.

Since most filesystems haven't been designed with unprivileged mounting in
mind, a thorough audit is needed before setting this flag.

For "safe" filesystems also allow unprivileged forced unmounting.

Signed-off-by: Miklos Szeredi <mszeredi@suse.cz>

```
Index: linux/fs/namespace.c
===================================================================
--- linux.orig/fs/namespace.c 2007-04-26 13:30:04.000000000 +0200
+++ linux/fs/namespace.c 2007-04-26 13:51:29.000000000 +0200
@@ -724,14 +724,16 @@ static bool is_mount_owner(struct vfsmou
 /*
  * umount is permitted for
  *  - sysadmin
- *  - mount owner, if not forced umount
+ *  - mount owner
+ *    o if not forced umount,
+ *    o if forced umount, and filesystem is "safe"
  */
 static bool permit_umount(struct vfsmount *mnt, int flags)
 {
  if (capable(CAP_SYS_ADMIN))
   return true;

- if (flags & MNT_FORCE)
+ if ((flags & MNT_FORCE) && !(mnt->mnt_sb->s_type->fs_flags & FS_SAFE))
   return false;

  return is_mount_owner(mnt, current->fsuid);
@@ -787,13 +789,17 @@ asmlinkage long sys_oldumount(char __use
  * - mountpoint is not a symlink
  * - mountpoint is in a mount owned by the user
  */
-static bool permit_mount(struct nameidata *nd, int *flags)
+static bool permit_mount(struct nameidata *nd, struct file_system_type *type,
+    int *flags)
 {
  struct inode *inode = nd->dentry->d_inode;

  if (capable(CAP_SYS_ADMIN))
   return true;

+ if (type && !(type->fs_flags & FS_SAFE))
+  return false;
+
  if (S_ISLNK(inode->i_mode))
   return false;

@@ -1027,7 +1033,7 @@ static int do_loopback(struct nameidata
  struct vfsmount *mnt = NULL;
  int err;

- if (!permit_mount(nd, &flags))
+ if (!permit_mount(nd, NULL, &flags))
```

```
   return -EPERM;
  if (!old_name || !*old_name)
   return -EINVAL;
@@ -1188,26 +1194,46 @@ out:
  * create a new mount for userspace and request it to be added into the
  * namespace's tree
  */
-static int do_new_mount(struct nameidata *nd, char *type, int flags,
+static int do_new_mount(struct nameidata *nd, char *fstype, int flags,
     int mnt_flags, char *name, void *data)
 {
+ int err;
  struct vfsmount *mnt;
+ struct file_system_type *type;

- if (!type || !memchr(type, 0, PAGE_SIZE))
+ if (!fstype || !memchr(fstype, 0, PAGE_SIZE))
   return -EINVAL;

- /* we need capabilities... */
- if (!capable(CAP_SYS_ADMIN))
-  return -EPERM;
-
- mnt = do_kern_mount(type, flags & ~MS_SETUSER, name, data);
- if (IS_ERR(mnt))
+ type = get_fs_type(fstype);
+ if (!type)
+  return -ENODEV;
+
+ err = -EPERM;
+ if (!permit_mount(nd, type, &flags))
+  goto out_put_filesystem;
+
+ if (flags & MS_SETUSER) {
+  err = reserve_user_mount();
+  if (err)
+   goto out_put_filesystem;
+ }
+
+ mnt = vfs_kern_mount(type, flags & ~MS_SETUSER, name, data);
+ put_filesystem(type);
+ if (IS_ERR(mnt)) {
+  if (flags & MS_SETUSER)
+   dec_nr_user_mounts();
   return PTR_ERR(mnt);
+ }

  if (flags & MS_SETUSER)
```

```
-  set_mnt_user(mnt);
+   __set_mnt_user(mnt);

  return do_add_mount(mnt, nd, mnt_flags, NULL);
+
+ out_put_filesystem:
+ put_filesystem(type);
+ return err;
 }

 /*
@@ -1237,7 +1263,7 @@ int do_add_mount(struct vfsmount *newmnt
  if (S_ISLNK(newmnt->mnt_root->d_inode->i_mode))
   goto unlock;

- /* MNT_USER was set earlier */
+ /* some flags may have been set earlier */
  newmnt->mnt_flags |= mnt_flags;
  if ((err = graft_tree(newmnt, nd)))
   goto unlock;
Index: linux/include/linux/fs.h
===================================================================
--- linux.orig/include/linux/fs.h 2007-04-26 13:46:26.000000000 +0200
+++ linux/include/linux/fs.h 2007-04-26 13:48:14.000000000 +0200
@@ -96,6 +96,7 @@ extern int dir_notify_enable;
 #define FS_REQUIRES_DEV 1
 #define FS_BINARY_MOUNTDATA 2
 #define FS_HAS_SUBTYPE 4
+#define FS_SAFE 8  /* Safe to mount by unprivileged users */
 #define FS_REVAL_DOT 16384 /* Check the paths ".", ".." for staleness */
 #define FS_RENAME_DOES_D_MOVE 32768 /* FS will handle d_move()
       * during rename() internally.

--

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

Subject: [patch 08/10] unprivileged mounts: allow unprivileged fuse mounts
Posted by Miklos Szeredi on Fri, 27 Apr 2007 12:04:25 GMT
View Forum Message <> Reply to Message

From: Miklos Szeredi <mszeredi@suse.cz>

Use FS_SAFE for "fuse" fs type, but not for "fuseblk".

FUSE was designed from the beginning to be safe for unprivileged users. This
has also been verified in practice over many years. In addition unprivileged
mounts require the parent mount to be owned by the user, which is more strict
than the current userspace policy.

This will enable future installations to remove the suid-root fusermount
utility.

Don't require the "user_id=" and "group_id=" options for unprivileged mounts,
but if they are present, verify them for sanity.

Disallow the "allow_other" option for unprivileged mounts.

Signed-off-by: Miklos Szeredi <mszeredi@suse.cz>

Index: linux/fs/fuse/inode.c
===
--- linux.orig/fs/fuse/inode.c 2007-04-26 13:07:11.000000000 +0200
+++ linux/fs/fuse/inode.c 2007-04-26 13:07:33.000000000 +0200
@@ -311,6 +311,19 @@ static int parse_fuse_opt(char *opt, str
  d->max_read = ~0;
  d->blksize = 512;

+ /*
+ * For unprivileged mounts use current uid/gid. Still allow
+ * "user_id" and "group_id" options for compatibility, but
+ * only if they match these values.
+ */
+ if (!capable(CAP_SYS_ADMIN)) {
+ d->user_id = current->uid;
+ d->user_id_present = 1;
+ d->group_id = current->gid;
+ d->group_id_present = 1;
+
+ }
+
  while ((p = strsep(&opt, ",")) != NULL) {
   int token;
   int value;
@@ -339,6 +352,8 @@ static int parse_fuse_opt(char *opt, str
   case OPT_USER_ID:
    if (match_int(&args[0], &value))
     return 0;
+   if (d->user_id_present && d->user_id != value)
+   return 0;
    d->user_id = value;
    d->user_id_present = 1;

```
    break;
@@ -346,6 +361,8 @@ static int parse_fuse_opt(char *opt, str
   case OPT_GROUP_ID:
    if (match_int(&args[0], &value))
     return 0;
+   if (d->group_id_present && d->group_id != value)
+    return 0;
    d->group_id = value;
    d->group_id_present = 1;
    break;
@@ -536,6 +553,10 @@ static int fuse_fill_super(struct super_
  if (!parse_fuse_opt((char *) data, &d, is_bdev))
   return -EINVAL;

+ /* This is a privileged option */
+ if ((d.flags & FUSE_ALLOW_OTHER) && !capable(CAP_SYS_ADMIN))
+  return -EPERM;
+
  if (is_bdev) {
 #ifdef CONFIG_BLOCK
   if (!sb_set_blocksize(sb, d.blksize))
@@ -639,6 +660,7 @@ static struct file_system_type fuse_fs_t
  .fs_flags = FS_HAS_SUBTYPE,
  .get_sb  = fuse_get_sb,
  .kill_sb = kill_anon_super,
+ .fs_flags = FS_SAFE,
 };

 #ifdef CONFIG_BLOCK

--
```

Subject: [patch 09/10] unprivileged mounts: propagation: inherit owner from parent

Posted by Miklos Szeredi on Fri, 27 Apr 2007 12:04:26 GMT

View Forum Message <> Reply to Message

From: Miklos Szeredi <mszeredi@suse.cz>

On mount propagation, let the owner of the clone be inherited from the parent into which it has been propagated. Also if the parent has the "nosuid" flag, set this flag for the child as well.

This makes sense for example, when propagation is set up from the

initial namespace into a per-user namespace, where some or all of the
mounts may be owned by the user.

Signed-off-by: Miklos Szeredi <mszeredi@suse.cz>

Index: linux/fs/namespace.c
```
===================================================================
--- linux.orig/fs/namespace.c 2007-04-27 12:57:01.000000000 +0200
+++ linux/fs/namespace.c 2007-04-27 12:57:11.000000000 +0200
@@ -250,10 +250,10 @@ static int reserve_user_mount(void)
  return err;
 }

-static void __set_mnt_user(struct vfsmount *mnt)
+static void __set_mnt_user(struct vfsmount *mnt, uid_t owner)
 {
  BUG_ON(mnt->mnt_flags & MNT_USER);
- mnt->mnt_uid = current->fsuid;
+ mnt->mnt_uid = owner;
  mnt->mnt_flags |= MNT_USER;

  if (!capable(CAP_SETUID))
@@ -264,7 +264,7 @@ static void __set_mnt_user(struct vfsmou

 static void set_mnt_user(struct vfsmount *mnt)
 {
- __set_mnt_user(mnt);
+ __set_mnt_user(mnt, current->fsuid);
  spin_lock(&vfsmount_lock);
  nr_user_mounts++;
  spin_unlock(&vfsmount_lock);
@@ -280,7 +280,7 @@ static void clear_mnt_user(struct vfsmou
 }

 static struct vfsmount *clone_mnt(struct vfsmount *old, struct dentry *root,
-     int flag)
+     int flag, uid_t owner)
 {
  struct super_block *sb = old->mnt_sb;
  struct vfsmount *mnt;
@@ -304,7 +304,10 @@ 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, owner);
+
```

```
+ if (flag & CL_NOSUID)
+  mnt->mnt_flags |= MNT_NOSUID;

  if (flag & CL_SLAVE) {
   list_add(&mnt->mnt_slave, &old->mnt_slave_list);
@@ -822,7 +825,7 @@ static int lives_below_in_same_fs(struct
 }

 struct vfsmount *copy_tree(struct vfsmount *mnt, struct dentry *dentry,
-     int flag)
+     int flag, uid_t owner)
 {
  struct vfsmount *res, *p, *q, *r, *s;
  struct nameidata nd;
@@ -830,7 +833,7 @@ struct vfsmount *copy_tree(struct vfsmou
  if (!(flag & CL_COPY_ALL) && IS_MNT_UNBINDABLE(mnt))
   return ERR_PTR(-EPERM);

- res = q = clone_mnt(mnt, dentry, flag);
+ res = q = clone_mnt(mnt, dentry, flag, owner);
  if (IS_ERR(q))
   goto error;
  q->mnt_mountpoint = mnt->mnt_mountpoint;
@@ -852,7 +855,7 @@ struct vfsmount *copy_tree(struct vfsmou
    p = s;
    nd.mnt = q;
    nd.dentry = p->mnt_mountpoint;
-   q = clone_mnt(p, p->mnt_root, flag);
+   q = clone_mnt(p, p->mnt_root, flag, owner);
    if (IS_ERR(q))
     goto error;
    spin_lock(&vfsmount_lock);
@@ -1028,7 +1031,8 @@ static int do_change_type(struct nameida
  */
 static int do_loopback(struct nameidata *nd, char *old_name, int flags)
 {
- int clone_flags;
+ int clone_flags = 0;
+ uid_t owner = 0;
  struct nameidata old_nd;
  struct vfsmount *mnt = NULL;
  int err;
@@ -1049,11 +1053,15 @@ static int do_loopback(struct nameidata
  if (!check_mnt(nd->mnt) || !check_mnt(old_nd.mnt))
   goto out;

- clone_flags = (flags & MS_SETUSER) ? CL_SETUSER : 0;
+ if (flags & MS_SETUSER) {
```

```
+  clone_flags |= CL_SETUSER;
+  owner = current->fsuid;
+ }
+
   if (flags & MS_REC)
-  mnt = copy_tree(old_nd.mnt, old_nd.dentry, clone_flags);
+  mnt = copy_tree(old_nd.mnt, old_nd.dentry, clone_flags, owner);
   else
-  mnt = clone_mnt(old_nd.mnt, old_nd.dentry, clone_flags);
+  mnt = clone_mnt(old_nd.mnt, old_nd.dentry, clone_flags, owner);

   err = PTR_ERR(mnt);
   if (IS_ERR(mnt))
@@ -1227,7 +1235,7 @@ static int do_new_mount(struct nameidata
   }

   if (flags & MS_SETUSER)
-  __set_mnt_user(mnt);
+  __set_mnt_user(mnt, current->fsuid);

   return do_add_mount(mnt, nd, mnt_flags, NULL);

@@ -1620,7 +1628,7 @@ static struct mnt_namespace *dup_mnt_ns(
   down_write(&namespace_sem);
   /* First pass: copy the tree topology */
   new_ns->root = copy_tree(mnt_ns->root, mnt_ns->root->mnt_root,
-      CL_COPY_ALL | CL_EXPIRE);
+      CL_COPY_ALL | CL_EXPIRE, 0);
   if (IS_ERR(new_ns->root)) {
    up_write(&namespace_sem);
    kfree(new_ns);
Index: linux/fs/pnode.c
===================================================================
--- linux.orig/fs/pnode.c 2007-04-27 12:57:00.000000000 +0200
+++ linux/fs/pnode.c 2007-04-27 12:57:11.000000000 +0200
@@ -179,15 +179,28 @@ int propagate_mnt(struct vfsmount *dest_

   for (m = propagation_next(dest_mnt, dest_mnt); m;
     m = propagation_next(m, dest_mnt)) {
-  int type;
+  int clflags;
+  uid_t owner = 0;
    struct vfsmount *source;

    if (IS_MNT_NEW(m))
     continue;

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

```
+  source =  get_source(m, prev_dest_mnt, prev_src_mnt, &clflags);

-  child = copy_tree(source, source->mnt_root, type);
+  if (m->mnt_flags & MNT_USER) {
+   clflags |= CL_SETUSER;
+   owner = m->mnt_uid;
+
+   /*
+    * If propagating into a user mount which doesn't
+    * allow suid, then make sure, the child(ren) won't
+    * allow suid either
+    */
+   if (m->mnt_flags & MNT_NOSUID)
+    clflags |= CL_NOSUID;
+  }
+  child = copy_tree(source, source->mnt_root, clflags, owner);
   if (IS_ERR(child)) {
    ret = PTR_ERR(child);
    list_splice(tree_list, tmp_list.prev);
Index: linux/fs/pnode.h
===================================================================
--- linux.orig/fs/pnode.h 2007-04-27 12:57:00.000000000 +0200
+++ linux/fs/pnode.h 2007-04-27 12:57:11.000000000 +0200
@@ -23,6 +23,7 @@
 #define CL_MAKE_SHARED   0x08
 #define CL_PROPAGATION   0x10
 #define CL_SETUSER  0x20
+#define CL_NOSUID  0x40

 static inline void set_mnt_shared(struct vfsmount *mnt)
 {
@@ -35,4 +36,6 @@ int propagate_mnt(struct vfsmount *, str
   struct list_head *);
 int propagate_umount(struct list_head *);
 int propagate_mount_busy(struct vfsmount *, int);
+struct vfsmount *copy_tree(struct vfsmount *, struct dentry *, int, uid_t);
+
 #endif /* _LINUX_PNODE_H */
Index: linux/include/linux/fs.h
===================================================================
--- linux.orig/include/linux/fs.h 2007-04-27 12:57:01.000000000 +0200
+++ linux/include/linux/fs.h 2007-04-27 12:57:11.000000000 +0200
@@ -1467,7 +1467,6 @@ extern int may_umount(struct vfsmount *)
 extern void umount_tree(struct vfsmount *, int, struct list_head *);
 extern void release_mounts(struct list_head *);
 extern long do_mount(char *, char *, char *, unsigned long, void *);
-extern struct vfsmount *copy_tree(struct vfsmount *, struct dentry *, int);
 extern void mnt_set_mountpoint(struct vfsmount *, struct dentry *,
```

struct vfsmount *);

--

_____

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

---

Subject: [patch 10/10] unprivileged mounts: add "no submounts" flag
Posted by Miklos Szeredi on Fri, 27 Apr 2007 12:04:27 GMT
View Forum Message <> Reply to Message

From: Miklos Szeredi <mszeredi@suse.cz>

Add a new mount flag "nomnt", which denies submounts for the owner.
This would be useful, if we want to support traditional /etc/fstab
based user mounts.

In this case mount(8) would still have to be suid-root, to check the
mountpoint against the user/users flag in /etc/fstab, but /etc/mtab
would no longer be mandatory for storing the actual owner of the
mount.

Signed-off-by: Miklos Szeredi <mszeredi@suse.cz>

Index: linux/fs/namespace.c
```
===================================================================
--- linux.orig/fs/namespace.c 2007-04-27 12:57:11.000000000 +0200
+++ linux/fs/namespace.c 2007-04-27 12:57:14.000000000 +0200
@@ -449,6 +449,7 @@ static int show_vfsmnt(struct seq_file *
   { MNT_NOATIME, ",noatime" },
   { MNT_NODIRATIME, ",nodiratime" },
   { MNT_RELATIME, ",relatime" },
+  { MNT_NOMNT, ",nomnt" },
   { 0, NULL }
  };
  struct proc_fs_info *fs_infop;
@@ -806,6 +807,9 @@ static bool permit_mount(struct nameidat
  if (S_ISLNK(inode->i_mode))
   return false;

+ if (nd->mnt->mnt_flags & MNT_NOMNT)
+  return false;
+
  if (!is_mount_owner(nd->mnt, current->fsuid))
```

```
   return false;

@@ -1575,9 +1579,11 @@ long do_mount(char *dev_name, char *dir_
   mnt_flags |= MNT_NODIRATIME;
  if (flags & MS_RELATIME)
   mnt_flags |= MNT_RELATIME;
+ if (flags & MS_NOMNT)
+  mnt_flags |= MNT_NOMNT;

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

  /* ... and get the mountpoint */
  retval = path_lookup(dir_name, LOOKUP_FOLLOW, &nd);
Index: linux/include/linux/fs.h
===================================================================
--- linux.orig/include/linux/fs.h 2007-04-27 12:57:11.000000000 +0200
+++ linux/include/linux/fs.h 2007-04-27 12:57:14.000000000 +0200
@@ -128,6 +128,7 @@ extern int dir_notify_enable;
 #define MS_SHARED (1<<20) /* change to shared */
 #define MS_RELATIME (1<<21) /* Update atime relative to mtime/ctime. */
 #define MS_SETUSER (1<<22) /* set mnt_uid to current user */
+#define MS_NOMNT (1<<23) /* don't allow unprivileged submounts */
 #define MS_ACTIVE (1<<30)
 #define MS_NOUSER (1<<31)

Index: linux/include/linux/mount.h
===================================================================
--- linux.orig/include/linux/mount.h 2007-04-27 12:57:01.000000000 +0200
+++ linux/include/linux/mount.h 2007-04-27 12:57:14.000000000 +0200
@@ -28,6 +28,7 @@ struct mnt_namespace;
 #define MNT_NOATIME 0x08
 #define MNT_NODIRATIME 0x10
 #define MNT_RELATIME 0x20
+#define MNT_NOMNT 0x40

 #define MNT_SHRINKABLE 0x100
 #define MNT_USER 0x200

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