Subject: [patch 00/10] mount ownership and unprivileged mount syscall (v7)
Posted by Miklos Szeredi on Wed, 16 Jan 2008 12:31:47 GMT
View Forum Message <> Reply to Message

Thanks to everyone for the comments on the previous submission.

Christoph, could you please look through the patches if they are
acceptable from the VFS point of view?

Thanks,
Miklos

v6 -> v7:

- add '/proc/sys/fs/types/<type>/usermount_safe' tunable (new patch)
- do not make FUSE safe by default, describe possible problems
  associated with unprivileged FUSE mounts in patch header
- return EMFILE instead of EPERM, if maximum user mount count is exceeded
- rename option 'nomnt' -> 'nosubmnt'
- clean up error propagation in dup_mnt_ns
- update util-linux-ng patch

v5 -> v6:

- update to latest -mm
- preliminary util-linux-ng support (will post right after this series)

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

v3 -> v4:

- simplify interface as much as possible, now only a single option
  ("user=UID") is used to control everything
- no longer allow/deny mounting based on file/directory permissions,
  that approach does not always make sense

v1 -> v3:

- add mount flags to set/clear mnt_flags individually
- add "usermnt" mount flag. If it is set, then allow unprivileged
  submounts under this mount
- make max number of user mounts default to 1024, since now the
  usermnt flag will prevent user mounts by default

--

_____

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 Wed, 16 Jan 2008 12:31:48 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/

After this series I'll be posting a preliminary patch for util-linux-ng,
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>
Acked-by: Serge Hallyn <serue@us.ibm.com>

Index: linux/fs/namespace.c

```
===================================================================
--- linux.orig/fs/namespace.c 2008-01-16 13:24:53.000000000 +0100
+++ linux/fs/namespace.c 2008-01-16 13:25:05.000000000 +0100
@@ -477,6 +477,13 @@ static struct vfsmount *skip_mnt_tree(st
  return p;
 }

+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 struct vfsmount *clone_mnt(struct vfsmount *old, struct dentry *root,
       int flag)
 {
@@ -491,6 +498,11 @@ 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;
@@ -644,6 +656,8 @@ 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");
@@ -1181,8 +1195,9 @@ 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_fl;
  struct nameidata old_nd;
  struct vfsmount *mnt = NULL;
```

```
  int err = mount_is_safe(nd);
@@ -1202,11 +1217,12 @@ static int do_loopback(struct nameidata
  if (!check_mnt(nd->path.mnt) || !check_mnt(old_nd.path.mnt))
   goto out;

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

  if (!mnt)
   goto out;
@@ -1268,8 +1284,11 @@ static int do_remount(struct nameidata *
   err = change_mount_flags(nd->path.mnt, flags);
  else
   err = do_remount_sb(sb, flags, data, 0);
- if (!err)
+ if (!err) {
   nd->path.mnt->mnt_flags = mnt_flags;
+  if (flags & MS_SETUSER)
+   set_mnt_user(nd->path.mnt);
+ }
  up_write(&sb->s_umount);
  if (!err)
   security_sb_post_remount(nd->path.mnt, flags, data);
@@ -1378,10 +1397,13 @@ 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);
 }

@@ -1413,7 +1435,8 @@ 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;

@@ -1735,7 +1758,7 @@ 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 2008-01-16 13:24:53.000000000 +0100
+++ linux/fs/pnode.h 2008-01-16 13:25:05.000000000 +0100
@@ -23,6 +23,7 @@
 #define CL_MAKE_SHARED   0x08
 #define CL_PROPAGATION   0x10
 #define CL_PRIVATE   0x20
+#define CL_SETUSER  0x40

 static inline void set_mnt_shared(struct vfsmount *mnt)
 {
Index: linux/include/linux/fs.h
===================================================================
--- linux.orig/include/linux/fs.h 2008-01-16 13:24:53.000000000 +0100
+++ linux/include/linux/fs.h 2008-01-16 13:25:05.000000000 +0100
@@ -125,6 +125,7 @@ extern int dir_notify_enable;
 #define MS_RELATIME (1<<21) /* Update atime relative to mtime/ctime. */
 #define MS_KERNMOUNT (1<<22) /* this is a kern_mount call */
 #define MS_I_VERSION (1<<23) /* Update inode I_version field */
+#define MS_SETUSER (1<<24) /* 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 2008-01-16 13:24:53.000000000 +0100
+++ linux/include/linux/mount.h 2008-01-16 13:25:05.000000000 +0100
@@ -33,6 +33,7 @@ struct mnt_namespace;

 #define MNT_SHRINKABLE 0x100
 #define MNT_IMBALANCED_WRITE_COUNT 0x200 /* just for debugging */
```

```
+#define MNT_USER 0x400

 #define MNT_SHARED 0x1000 /* if the vfsmount is a shared mount */
 #define MNT_UNBINDABLE 0x2000 /* if the vfsmount is a unbindable mount */
@@ -69,6 +70,8 @@ struct vfsmount {
   * are held, and all mnt_writer[]s on this mount have 0 as their ->count
   */
  atomic_t __mnt_writers;
+
+ 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 03/10] unprivileged mounts: propagate error values from clone_mnt
Posted by Miklos Szeredi on Wed, 16 Jan 2008 12:31:50 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.

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

```
Index: linux/fs/namespace.c
===================================================================
--- linux.orig/fs/namespace.c 2008-01-16 13:25:06.000000000 +0100
+++ linux/fs/namespace.c 2008-01-16 13:25:07.000000000 +0100
@@ -490,41 +490,42 @@ 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_PRIVATE)) {
-   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_PRIVATE)) {
+  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;
 }
@@ -998,11 +999,11 @@ 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;
@@ -1023,8 +1024,8 @@ struct vfsmount *copy_tree(struct vfsmou
    nd.path.mnt = q;
    nd.path.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);
```

```
@@ -1032,7 +1033,7 @@ struct vfsmount *copy_tree(struct vfsmou
   }
  }
  return res;
-Enomem:
+ error:
  if (res) {
   LIST_HEAD(umount_list);
   spin_lock(&vfsmount_lock);
@@ -1040,7 +1041,7 @@ Enomem:
   spin_unlock(&vfsmount_lock);
   release_mounts(&umount_list);
  }
- return NULL;
+ return q;
 }

 struct vfsmount *collect_mounts(struct vfsmount *mnt, struct dentry *dentry)
@@ -1239,13 +1240,13 @@ static int do_loopback(struct nameidata
   goto out;

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

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

  err = graft_tree(mnt, nd);
@@ -1816,10 +1817,10 @@ 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 ERR_PTR(-ENOMEM);;
+  return ERR_CAST(new_ns->root);
  }
  spin_lock(&vfsmount_lock);
  list_add_tail(&new_ns->list, &new_ns->root->mnt_list);
Index: linux/fs/pnode.c
```

```
===================================================================
--- linux.orig/fs/pnode.c 2008-01-16 13:24:53.000000000 +0100
+++ linux/fs/pnode.c 2008-01-16 13:25:07.000000000 +0100
@@ -189,8 +189,9 @@ 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 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,

--
```


Subject: [patch 05/10] unprivileged mounts: allow unprivileged bind mounts
Posted by Miklos Szeredi on Wed, 16 Jan 2008 12:31:52 GMT

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>
Acked-by: Serge Hallyn <serue@us.ibm.com>

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 13:25:08.000000000 +0100
@@ -511,6 +511,11 @@ static void __set_mnt_user(struct vfsmou
  WARN_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)
@@ -1021,22 +1026,26 @@ 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->path.dentry->d_inode;
+
  if (capable(CAP_SYS_ADMIN))
-  return 0;
- return -EPERM;
-#ifdef notyet
- if (S_ISLNK(nd->path.dentry->d_inode->i_mode))
-  return -EPERM;
- if (nd->path.dentry->d_inode->i_mode & S_ISVTX) {
-  if (current->uid != nd->path.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->path.mnt, current->fsuid))
+  return false;
+
+ *flags |= MS_SETUSER;
+ return true;
 }

 static int lives_below_in_same_fs(struct dentry *d, struct dentry *dentry)
@@ -1280,9 +1289,10 @@ static int do_loopback(struct nameidata
  int clone_fl;
  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);
```

--

_____

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

Subject: [patch 09/10] unprivileged mounts: propagation: inherit owner from parent
Posted by Miklos Szeredi on Wed, 16 Jan 2008 12:31:56 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.

If the parent has the "nosuid" flag, set this flag for the child as well. This is needed for the suid-less namespace (use case #2 in the first patch header), where all mounts are owned by the user and have the nosuid flag set. In this case the propagated mount needs to have nosuid, otherwise a suid executable may be misused by the user.

Similar treatment is not needed for "nodev", because devices can't be abused this way: the user is not able to gain privileges to devices by rearranging the mount namespace.

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

Index: linux/fs/namespace.c
```
===================================================================
--- linux.orig/fs/namespace.c 2008-01-16 13:25:09.000000000 +0100
+++ linux/fs/namespace.c 2008-01-16 13:25:11.000000000 +0100
@@ -506,10 +506,10 @@ 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)
 {
  WARN_ON(mnt->mnt_flags & MNT_USER);
- mnt->mnt_uid = current->fsuid;
+ mnt->mnt_uid = owner;
  mnt->mnt_flags |= MNT_USER;

  if (!capable(CAP_SETUID))
@@ -520,7 +520,7 @@ 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);
@@ -536,7 +536,7 @@ 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;
@@ -560,7 +560,10 @@ 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);
@@ -1066,7 +1069,7 @@ 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;
@@ -1074,7 +1077,7 @@ 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;
@@ -1096,7 +1099,7 @@ struct vfsmount *copy_tree(struct vfsmou
    p = s;
    nd.path.mnt = q;
    nd.path.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);
@@ -1121,7 +1124,7 @@ struct vfsmount *collect_mounts(struct v
 {
  struct vfsmount *tree;
  down_read(&namespace_sem);
- tree = copy_tree(mnt, dentry, CL_COPY_ALL | CL_PRIVATE);
+ tree = copy_tree(mnt, dentry, CL_COPY_ALL | CL_PRIVATE, 0);
  up_read(&namespace_sem);
  return tree;
 }
@@ -1292,7 +1295,8 @@ static int do_change_type(struct nameida
  */
 static int do_loopback(struct nameidata *nd, char *old_name, int flags)
 {
- int clone_fl;
+ int clone_fl = 0;
+ uid_t owner = 0;
  struct nameidata old_nd;
  struct vfsmount *mnt = NULL;
  int err;
@@ -1313,11 +1317,17 @@ static int do_loopback(struct nameidata
  if (!check_mnt(nd->path.mnt) || !check_mnt(old_nd.path.mnt))
   goto out;

- clone_fl = (flags & MS_SETUSER) ? CL_SETUSER : 0;
+ if (flags & MS_SETUSER) {
+  clone_fl |= CL_SETUSER;
+  owner = current->fsuid;
+ }
+
  if (flags & MS_REC)
-  mnt = copy_tree(old_nd.path.mnt, old_nd.path.dentry, clone_fl);
+  mnt = copy_tree(old_nd.path.mnt, old_nd.path.dentry, clone_fl,
+    owner);
  else
-  mnt = clone_mnt(old_nd.path.mnt, old_nd.path.dentry, clone_fl);
+  mnt = clone_mnt(old_nd.path.mnt, old_nd.path.dentry, clone_fl,
+    owner);

  err = PTR_ERR(mnt);
  if (IS_ERR(mnt))
@@ -1541,7 +1551,7 @@ 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);

@@ -1937,7 +1947,7 @@ 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 2008-01-16 13:25:07.000000000 +0100
+++ linux/fs/pnode.c 2008-01-16 13:25:11.000000000 +0100
@@ -181,15 +181,28 @@ 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 2008-01-16 13:25:05.000000000 +0100
+++ linux/fs/pnode.h 2008-01-16 13:25:11.000000000 +0100
@@ -24,6 +24,7 @@
 #define CL_PROPAGATION   0x10
 #define CL_PRIVATE   0x20
 #define CL_SETUSER  0x40
+#define CL_NOSUID  0x80

 static inline void set_mnt_shared(struct vfsmount *mnt)
 {
@@ -36,4 +37,6 @@ 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 2008-01-16 13:25:09.000000000 +0100
+++ linux/include/linux/fs.h 2008-01-16 13:25:11.000000000 +0100
@@ -1493,7 +1493,6 @@ 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 *);
 extern struct vfsmount *collect_mounts(struct vfsmount *, struct dentry *);

--

_____
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 Wed, 16 Jan 2008 12:31:57 GMT
View Forum Message <> Reply to Message

From: Miklos Szeredi <mszeredi@suse.cz>

Add a new mount flag "nosubmnt", 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>
Acked-by: Serge Hallyn <serue@us.ibm.com>

Index: linux/fs/namespace.c
```
===================================================================
--- linux.orig/fs/namespace.c 2008-01-16 13:25:11.000000000 +0100
+++ linux/fs/namespace.c 2008-01-16 13:25:12.000000000 +0100
@@ -700,6 +700,7 @@ static int show_vfsmnt(struct seq_file *
   { MNT_NOATIME, ",noatime" },
   { MNT_NODIRATIME, ",nodiratime" },
   { MNT_RELATIME, ",relatime" },
+  { MNT_NOSUBMNT, ",nosubmnt" },
   { 0, NULL }
  };
  struct proc_fs_info *fs_infop;
@@ -1050,6 +1051,9 @@ static bool permit_mount(struct nameidat
  if (S_ISLNK(inode->i_mode))
   return false;

+ if (nd->path.mnt->mnt_flags & MNT_NOSUBMNT)
+  return false;
+
  if (!is_mount_owner(nd->path.mnt, current->fsuid))
   return false;

@@ -1894,9 +1898,11 @@ long do_mount(char *dev_name, char *dir_
   mnt_flags |= MNT_RELATIME;
  if (flags & MS_RDONLY)
   mnt_flags |= MNT_READONLY;
+ if (flags & MS_NOSUBMNT)
+  mnt_flags |= MNT_NOSUBMNT;

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

  /* ... and get the mountpoint */
  retval = path_lookup(dir_name, LOOKUP_FOLLOW, &nd);
Index: linux/include/linux/fs.h
```

```
===================================================================
--- linux.orig/include/linux/fs.h 2008-01-16 13:25:11.000000000 +0100
+++ linux/include/linux/fs.h 2008-01-16 13:25:12.000000000 +0100
@@ -129,6 +129,7 @@ extern int dir_notify_enable;
 #define MS_KERNMOUNT (1<<22) /* this is a kern_mount call */
 #define MS_I_VERSION (1<<23) /* Update inode I_version field */
 #define MS_SETUSER (1<<24) /* set mnt_uid to current user */
+#define MS_NOSUBMNT (1<<25) /* 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 2008-01-16 13:25:05.000000000 +0100
+++ linux/include/linux/mount.h 2008-01-16 13:25:12.000000000 +0100
@@ -30,6 +30,7 @@ struct mnt_namespace;
 #define MNT_NODIRATIME 0x10
 #define MNT_RELATIME 0x20
 #define MNT_READONLY 0x40 /* does the user want this to be r/o? */
+#define MNT_NOSUBMNT 0x80

 #define MNT_SHRINKABLE 0x100
 #define MNT_IMBALANCED_WRITE_COUNT 0x200 /* just for debugging */

--
```


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

Posted by serue on Mon, 21 Jan 2008 20:23:06 GMT

View Forum Message <> Reply to Message

Quoting Miklos Szeredi (miklos@szeredi.hu):
> 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.
>
> If the parent has the "nosuid" flag, set this flag for the child as
> well. This is needed for the suid-less namespace (use case #2 in the
> first patch header), where all mounts are owned by the user and have
> the nosuid flag set. In this case the propagated mount needs to have
> nosuid, otherwise a suid executable may be misused by the user.
>

> Similar treatment is not needed for "nodev", because devices can't be
> abused this way: the user is not able to gain privileges to devices by
> rearranging the mount namespace.
>
> Signed-off-by: Miklos Szeredi <mszeredi@suse.cz>

As discussed many months ago this does seem like the most appropriate
behavior for propagation.

Acked-by: Serge Hallyn <serue@us.ibm.com>

> ---
>
> Index: linux/fs/namespace.c
> ===
> --- linux.orig/fs/namespace.c 2008-01-16 13:25:09.000000000 +0100
> +++ linux/fs/namespace.c 2008-01-16 13:25:11.000000000 +0100
> @@ -506,10 +506,10 @@ 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)
> {
> WARN_ON(mnt->mnt_flags & MNT_USER);
> - mnt->mnt_uid = current->fsuid;
> + mnt->mnt_uid = owner;
> mnt->mnt_flags |= MNT_USER;
>
> if (!capable(CAP_SETUID))
> @@ -520,7 +520,7 @@ 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);
> @@ -536,7 +536,7 @@ 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;

```
> @@ -560,7 +560,10 @@ 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);
> @@ -1066,7 +1069,7 @@ 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;
> @@ -1074,7 +1077,7 @@ 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;
> @@ -1096,7 +1099,7 @@ struct vfsmount *copy_tree(struct vfsmou
>     p = s;
>     nd.path.mnt = q;
>     nd.path.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);
> @@ -1121,7 +1124,7 @@ struct vfsmount *collect_mounts(struct v
>  {
>   struct vfsmount *tree;
>   down_read(&namespace_sem);
> - tree = copy_tree(mnt, dentry, CL_COPY_ALL | CL_PRIVATE);
> + tree = copy_tree(mnt, dentry, CL_COPY_ALL | CL_PRIVATE, 0);
>   up_read(&namespace_sem);
>   return tree;
>  }
```

```
> @@ -1292,7 +1295,8 @@ static int do_change_type(struct nameida
>   */
>  static int do_loopback(struct nameidata *nd, char *old_name, int flags)
>  {
> - int clone_fl;
> + int clone_fl = 0;
> + uid_t owner = 0;
>   struct nameidata old_nd;
>   struct vfsmount *mnt = NULL;
>   int err;
> @@ -1313,11 +1317,17 @@ static int do_loopback(struct nameidata
>   if (!check_mnt(nd->path.mnt) || !check_mnt(old_nd.path.mnt))
>    goto out;
>
> - clone_fl = (flags & MS_SETUSER) ? CL_SETUSER : 0;
> + if (flags & MS_SETUSER) {
> +  clone_fl |= CL_SETUSER;
> +  owner = current->fsuid;
> + }
> +
>   if (flags & MS_REC)
> -  mnt = copy_tree(old_nd.path.mnt, old_nd.path.dentry, clone_fl);
> +  mnt = copy_tree(old_nd.path.mnt, old_nd.path.dentry, clone_fl,
> +    owner);
>   else
> -  mnt = clone_mnt(old_nd.path.mnt, old_nd.path.dentry, clone_fl);
> +  mnt = clone_mnt(old_nd.path.mnt, old_nd.path.dentry, clone_fl,
> +    owner);
>
>   err = PTR_ERR(mnt);
>   if (IS_ERR(mnt))
> @@ -1541,7 +1551,7 @@ 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);
>
> @@ -1937,7 +1947,7 @@ 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 2008-01-16 13:25:07.000000000 +0100
> +++ linux/fs/pnode.c 2008-01-16 13:25:11.000000000 +0100
> @@ -181,15 +181,28 @@ 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 2008-01-16 13:25:05.000000000 +0100
> +++ linux/fs/pnode.h 2008-01-16 13:25:11.000000000 +0100
> @@ -24,6 +24,7 @@
> #define CL_PROPAGATION   0x10
> #define CL_PRIVATE   0x20
> #define CL_SETUSER 0x40
> +#define CL_NOSUID 0x80
>
> static inline void set_mnt_shared(struct vfsmount *mnt)

> {
> @@ -36,4 +37,6 @@ 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 2008-01-16 13:25:09.000000000 +0100
> +++ linux/include/linux/fs.h 2008-01-16 13:25:11.000000000 +0100
> @@ -1493,7 +1493,6 @@ 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 *);
> extern struct vfsmount *collect_mounts(struct vfsmount *, struct dentry *);
>
> --
> -
> To unsubscribe from this list: send the line "unsubscribe linux-fsdevel" in
> the body of a message to majordomo@vger.kernel.org
> More majordomo info at http://vger.kernel.org/majordomo-info.html

_____

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