

---

Subject: [patch 1/9] unprivileged mounts: add user mounts to the kernel  
Posted by [Miklos Szeredi](#) on Tue, 08 Jan 2008 11:35:03 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

From: Miklos Szeredi <[mszeredi@suse.cz](mailto: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>

---

Index: linux/fs/namespace.c

```
=====
--- linux.orig/fs/namespace.c 2008-01-03 22:10:10.000000000 +0100
+++ linux/fs/namespace.c 2008-01-04 13:46:33.000000000 +0100
@@ -477,6 +477,13 @@ 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)
{
@@ -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 2008-01-03 22:10:10.000000000 +0100
```

```
+++ linux/fs/pnode.h 2008-01-04 13:45:45.000000000 +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 2008-01-03 22:10:10.000000000 +0100
```

```
+++ linux/include/linux/fs.h 2008-01-04 13:45:46.000000000 +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 2008-01-03 22:10:10.000000000 +0100
```

```
+++ linux/include/linux/mount.h 2008-01-04 13:45:45.000000000 +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: Re: [patch 1/9] unprivileged mounts: add user mounts to the kernel  
Posted by [Pavel Machek](#) on Tue, 08 Jan 2008 21:34:21 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Tue 2008-01-08 12:35:03, Miklos Szeredi wrote:

> From: Miklos Szeredi <[mszeredi@suse.cz](mailto: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

- create unkillable processes

- block suspend/hibernation

?

Pavel

--

(english) <http://www.livejournal.com/~pavelmachek>

(cesky, pictures) <http://atrey.karlin.mff.cuni.cz/~pavel/picture/horses/blog.html>

---

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

---

---

Subject: Re: [patch 1/9] unprivileged mounts: add user mounts to the kernel  
Posted by [Pavel Machek](#) on Tue, 08 Jan 2008 21:47:22 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Tue 2008-01-08 12:35:03, Miklos Szeredi wrote:

> From: Miklos Szeredi <[mszeredi@suse.cz](mailto: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

- create traps for updatedb, etc?

Is there Doc\* file somewhere describing dangers of allowing this?

Pavel

--

(english) <http://www.livejournal.com/~pavelmachek>

(cesky, pictures) <http://atrey.karlin.mff.cuni.cz/~pavel/picture/horses/blog.html>

---

Containers mailing list

[Containers@lists.linux-foundation.org](mailto:Containers@lists.linux-foundation.org)

<https://lists.linux-foundation.org/mailman/listinfo/containers>

---

---

Subject: Re: [patch 1/9] unprivileged mounts: add user mounts to the kernel

Posted by [serue](#) on Mon, 14 Jan 2008 21:46:24 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

Quoting Miklos Szeredi ([miklos@szeredi.hu](mailto:miklos@szeredi.hu)):

> From: Miklos Szeredi <[mszeredi@suse.cz](mailto: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>

This looks good to me.

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

thanks,  
-serge

```
> ---
>
> Index: linux/fs/namespace.c
> =====
> --- linux.orig/fs/namespace.c 2008-01-03 22:10:10.000000000 +0100
> +++ linux/fs/namespace.c 2008-01-04 13:46:33.000000000 +0100
> @@ -477,6 +477,13 @@ 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)
> {
> @@ -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_infolp->flag)
>    seq_puts(m, fs_infolp->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 2008-01-03 22:10:10.000000000 +0100
> +++ linux/fs/pnode.h 2008-01-04 13:45:45.000000000 +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 2008-01-03 22:10:10.000000000 +0100
> +++ linux/include/linux/fs.h 2008-01-04 13:45:46.000000000 +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 2008-01-03 22:10:10.000000000 +0100
> +++ linux/include/linux/mount.h 2008-01-04 13:45:45.000000000 +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>

---