Subject: [patch 0/9] mount ownership and unprivileged mount syscall (v6)
Posted by Miklos Szeredi on Tue, 08 Jan 2008 11:35:02 GMT
View Forum Message <> Reply to Message

After much sitting in -mm, Andrew dropped this patchset due to
conflicts with other stuff. It would be nice, if it could be reviewed
in time for 2.6.26 (2.6.25 is closed as far as I understand).

v5 -> v6:

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

Thanks,
Miklos

--

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

Subject: [patch 2/9] unprivileged mounts: allow unprivileged umount
Posted by Miklos Szeredi on Tue, 08 Jan 2008 11:35:04 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 2008-01-03 20:52:38.000000000 +0100
+++ linux/fs/namespace.c 2008-01-03 21:14:16.000000000 +0100
@@ -894,6 +894,27 @@ 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.
@@ -917,7 +938,7 @@ asmlinkage long sys_umount(char __user *
   goto dput_and_out;

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

  retval = do_umount(nd.path.mnt, flags);

--
```

Subject: [patch 4/9] unprivileged mounts: propagate error values from clone_mnt
Posted by Miklos Szeredi on Tue, 08 Jan 2008 11:35:06 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 2008-01-04 13:47:09.000000000 +0100
+++ linux/fs/namespace.c 2008-01-04 13:47:49.000000000 +0100
@@ -512,41 +512,42 @@ 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 */
+ 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);
-  }
+ 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;
 }
@@ -1021,11 +1022,11 @@ 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;
@@ -1046,8 +1047,8 @@ 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);
@@ -1055,7 +1056,7 @@ struct vfsmount *copy_tree(struct vfsmou
   }
  }
  return res;
-Enomem:
+ error:
  if (res) {
   LIST_HEAD(umount_list);
   spin_lock(&vfsmount_lock);
@@ -1063,7 +1064,7 @@ Enomem:
   spin_unlock(&vfsmount_lock);
   release_mounts(&umount_list);
  }
- return NULL;
+ return q;
 }

 struct vfsmount *collect_mounts(struct vfsmount *mnt, struct dentry *dentry)
@@ -1262,13 +1263,13 @@ 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);
@@ -1840,7 +1841,7 @@ 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 ERR_PTR(-ENOMEM);;
Index: linux/fs/pnode.c
===================================================================
--- linux.orig/fs/pnode.c 2008-01-04 13:45:46.000000000 +0100
+++ linux/fs/pnode.c 2008-01-04 13:47:49.000000000 +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;
   }

--
```

Subject: [patch 6/9] unprivileged mounts: allow unprivileged mounts
Posted by Miklos Szeredi on Tue, 08 Jan 2008 11:35:08 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.

Move subtype handling from do_kern_mount() into do_new_mount(). All
other callers are kernel-internal and do not need subtype support.

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

Index: linux/fs/namespace.c
```
===================================================================
--- linux.orig/fs/namespace.c 2008-01-03 21:20:11.000000000 +0100
+++ linux/fs/namespace.c 2008-01-03 21:21:06.000000000 +0100
@@ -960,14 +960,16 @@ 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);
@@ -1025,13 +1027,17 @@ 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->path.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;

@@ -1285,7 +1291,7 @@ 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;
@@ -1466,30 +1472,76 @@ out:
  return err;
 }

+static struct vfsmount *fs_set_subtype(struct vfsmount *mnt, const char *fstype)
+{
+ int err;
+ const char *subtype = strchr(fstype, '.');
+ if (subtype) {
+  subtype++;
+  err = -EINVAL;
+  if (!subtype[0])
+   goto err;
+ } else
+  subtype = "";
+
+ mnt->mnt_sb->s_subtype = kstrdup(subtype, GFP_KERNEL);
+ err = -ENOMEM;
+ if (!mnt->mnt_sb->s_subtype)
+  goto err;
+ return mnt;
+
+ err:
+ mntput(mnt);
+ return ERR_PTR(err);
+}
+
 /*
  * 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);
+ if (!IS_ERR(mnt) && (type->fs_flags & FS_HAS_SUBTYPE) &&
+     !mnt->mnt_sb->s_subtype)
+  mnt = fs_set_subtype(mnt, fstype);
+ 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;
 }

 /*
@@ -1520,7 +1572,7 @@ 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 2008-01-03 21:15:35.000000000 +0100
+++ linux/include/linux/fs.h 2008-01-03 21:21:06.000000000 +0100
@@ -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.
Index: linux/fs/super.c
===================================================================
--- linux.orig/fs/super.c 2008-01-02 21:42:10.000000000 +0100
+++ linux/fs/super.c 2008-01-03 21:21:06.000000000 +0100
@@ -906,29 +906,6 @@ out:

 EXPORT_SYMBOL_GPL(vfs_kern_mount);

-static struct vfsmount *fs_set_subtype(struct vfsmount *mnt, const char *fstype)
-{
- int err;
- const char *subtype = strchr(fstype, '.');
- if (subtype) {
-  subtype++;
-  err = -EINVAL;
-  if (!subtype[0])
-   goto err;
- } else
-  subtype = "";
-
- mnt->mnt_sb->s_subtype = kstrdup(subtype, GFP_KERNEL);
- err = -ENOMEM;
- if (!mnt->mnt_sb->s_subtype)
```

```
-  goto err;
- return mnt;
-
- err:
- mntput(mnt);
- return ERR_PTR(err);
-}
-
 struct vfsmount *
 do_kern_mount(const char *fstype, int flags, const char *name, void *data)
 {
@@ -937,9 +914,6 @@ do_kern_mount(const char *fstype, int fl
  if (!type)
   return ERR_PTR(-ENODEV);
  mnt = vfs_kern_mount(type, flags, name, data);
- if (!IS_ERR(mnt) && (type->fs_flags & FS_HAS_SUBTYPE) &&
-     !mnt->mnt_sb->s_subtype)
-  mnt = fs_set_subtype(mnt, fstype);
  put_filesystem(type);
  return mnt;
 }

--
```


Subject: [patch 8/9] unprivileged mounts: propagation: inherit owner from parent

Posted by Miklos Szeredi on Tue, 08 Jan 2008 11:35:10 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 2008-01-04 13:48:14.000000000 +0100
+++ linux/fs/namespace.c 2008-01-04 13:49:52.000000000 +0100
@@ -500,10 +500,10 @@ 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))
@@ -514,7 +514,7 @@ 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);
@@ -530,7 +530,7 @@ 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;
@@ -554,7 +554,10 @@ 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);
@@ -1060,7 +1063,7 @@ 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;
@@ -1068,7 +1071,7 @@ 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;
@@ -1090,7 +1093,7 @@ 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);
@@ -1115,7 +1118,7 @@ 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;
 }
@@ -1286,7 +1289,8 @@ 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;
@@ -1307,11 +1311,17 @@ 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))
@@ -1535,7 +1545,7 @@ 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);

@@ -1931,7 +1941,7 @@ 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 2008-01-04 13:47:49.000000000 +0100
+++ linux/fs/pnode.c 2008-01-04 13:49:12.000000000 +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 2008-01-04 13:45:45.000000000 +0100
+++ linux/fs/pnode.h 2008-01-04 13:49:12.000000000 +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 2008-01-04 13:48:14.000000000 +0100
+++ linux/include/linux/fs.h 2008-01-04 13:49:12.000000000 +0100
@@ -1492,7 +1492,6 @@ 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 *);

--
```

Subject: [patch 9/9] unprivileged mounts: add "no submounts" flag
Posted by Miklos Szeredi on Tue, 08 Jan 2008 11:35:11 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 2008-01-04 13:49:52.000000000 +0100
+++ linux/fs/namespace.c 2008-01-04 13:50:28.000000000 +0100
@@ -694,6 +694,7 @@ 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;
@@ -1044,6 +1045,9 @@ static bool permit_mount(struct nameidat
  if (S_ISLNK(inode->i_mode))
   return false;

+ if (nd->path.mnt->mnt_flags & MNT_NOMNT)
```

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

@@ -1888,9 +1892,11 @@ long do_mount(char *dev_name, char *dir_
   mnt_flags |= MNT_RELATIME;
  if (flags & MS_RDONLY)
   mnt_flags |= MNT_READONLY;
+ if (flags & MS_NOMNT)
+  mnt_flags |= MNT_NOMNT;

- 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_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 2008-01-04 13:49:12.000000000 +0100
+++ linux/include/linux/fs.h 2008-01-04 13:49:58.000000000 +0100
@@ -130,6 +130,7 @@ 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_NOMNT (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 2008-01-04 13:45:45.000000000 +0100
+++ linux/include/linux/mount.h 2008-01-04 13:49:58.000000000 +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_NOMNT 0x80

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

--
```

Subject: Re: [patch 6/9] unprivileged mounts: allow unprivileged mounts
Posted by Karel Zak on Wed, 09 Jan 2008 11:11:20 GMT
View Forum Message <> Reply to Message

On Tue, Jan 08, 2008 at 12:35:08PM +0100, Miklos Szeredi wrote:
> 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.

What about to list "safe" filesystems anywhere in /proc/fs/ ? I think it's very important information for admins.

Note, your patch for mount(8) is always trying to use unprivileged mount(2) for non-root users. It's overkill when unprivileged mounts are supported for bind mounts and fuse only. It would be nice to check if FS is "safe" before switch to unprivileged mode.

The "safe" definition is also very subjective and it depends on your level of paranoia. There should be a way (e.g. /proc) how control and modify the list of "safe" filesystems. For example I have no problem to mark cifs as "safe" for my home server.

Karel

--
Karel Zak <kzak@redhat.com>

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

Subject: Re: [patch 6/9] unprivileged mounts: allow unprivileged mounts
Posted by Miklos Szeredi on Wed, 09 Jan 2008 12:41:40 GMT
View Forum Message <> Reply to Message

> On Tue, Jan 08, 2008 at 12:35:08PM +0100, Miklos Szeredi wrote:
> > 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.
>
> What about to list "safe" filesystems anywhere in /proc/fs/ ? I think
> it's very important information for admins.

Makes sense. I'll cook up something.

> Note, your patch for mount(8) is always trying to use unprivileged
> mount(2) for non-root users. It's overkill when unprivileged mounts are
> supported for bind mounts and fuse only. It would be nice to check
> if FS is "safe" before switch to unprivileged mode.

I think the little gain in performance is not worth the added
complexity. Especially if the added complexity is in the privileged
part, and itself can be a source of security holes.

> The "safe" definition is also very subjective and it depends on your
> level of paranoia. There should be a way (e.g. /proc) how control and
> modify the list of "safe" filesystems. For example I have no problem
> to mark cifs as "safe" for my home server.

OK, also makes some sense. Pavel's examples do point out that fuse
isn't as safe as I'd like it to be, so perhaps it would make sense to
default to just bind mounts being allowed, and having to explicity
enable unprivileged fuse mounts with a sysctl or whatever.

Miklos

Subject: Re: [patch 2/9] unprivileged mounts: allow unprivileged umount
Posted by serue on Mon, 14 Jan 2008 21:48:41 GMT
View Forum Message <> Reply to Message

Quoting Miklos Szeredi (miklos@szeredi.hu):
> 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>

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

> ---
>
> Index: linux/fs/namespace.c
> ===
> --- linux.orig/fs/namespace.c 2008-01-03 20:52:38.000000000 +0100
> +++ linux/fs/namespace.c 2008-01-03 21:14:16.000000000 +0100
> @@ -894,6 +894,27 @@ 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.
> @@ -917,7 +938,7 @@ asmlinkage long sys_umount(char __user *
> goto dput_and_out;
>
> retval = -EPERM;
> - if (!capable(CAP_SYS_ADMIN))
> + if (!permit_umount(nd.path.mnt, flags))
> goto dput_and_out;
>
> retval = do_umount(nd.path.mnt, flags);

>
> --

_____

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

---

Subject: Re: [patch 4/9] unprivileged mounts: propagate error values from clone_mnt
Posted by serue on Mon, 14 Jan 2008 22:23:05 GMT
View Forum Message <> Reply to Message

Quoting Miklos Szeredi (miklos@szeredi.hu):
> 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.

I see what you're saying, but it just seems like it's bound to be more
confusing this way.

What's the reason to insist on doing this? To force people to think
about it as a form of documentation?

Still,

> 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 2008-01-04 13:47:09.000000000 +0100
> +++ linux/fs/namespace.c 2008-01-04 13:47:49.000000000 +0100
> @@ -512,41 +512,42 @@ 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;
>  }
> @@ -1021,11 +1022,11 @@ 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;
> @@ -1046,8 +1047,8 @@ 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);
> @@ -1055,7 +1056,7 @@ struct vfsmount *copy_tree(struct vfsmou
>     }
>    }
>    return res;
> -Enomem:
> + error:
>    if (res) {
>     LIST_HEAD(umount_list);
>     spin_lock(&vfsmount_lock);
> @@ -1063,7 +1064,7 @@ Enomem:
>     spin_unlock(&vfsmount_lock);
>     release_mounts(&umount_list);
>    }
> - return NULL;
> + return q;
>  }
>
>  struct vfsmount *collect_mounts(struct vfsmount *mnt, struct dentry *dentry)
> @@ -1262,13 +1263,13 @@ 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);
> @@ -1840,7 +1841,7 @@ 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 ERR_PTR(-ENOMEM);;
> Index: linux/fs/pnode.c
> ===================================================================
> --- linux.orig/fs/pnode.c 2008-01-04 13:45:46.000000000 +0100
```

> +++ linux/fs/pnode.c 2008-01-04 13:47:49.000000000 +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;
> }
>
> --
> -
> 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

---

Subject: Re: [patch 6/9] unprivileged mounts: allow unprivileged mounts
Posted by serue on Mon, 14 Jan 2008 22:58:26 GMT
View Forum Message <> Reply to Message

Quoting Miklos Szeredi (miklos@szeredi.hu):
> 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.
>
> Move subtype handling from do_kern_mount() into do_new_mount(). All
> other callers are kernel-internal and do not need subtype support.
>
> Signed-off-by: Miklos Szeredi <mszeredi@suse.cz>

This patch itself doesn't assign FS_SAFE to any filesystems, so
presuming that there is such a thing as an fs safe for users to
mount, and/or users sign their systems away through a sysctl,

this patch in itself appears right.

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

```
> ---
>
> Index: linux/fs/namespace.c
> ===================================================================
> --- linux.orig/fs/namespace.c 2008-01-03 21:20:11.000000000 +0100
> +++ linux/fs/namespace.c 2008-01-03 21:21:06.000000000 +0100
> @@ -960,14 +960,16 @@ 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);
> @@ -1025,13 +1027,17 @@ 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->path.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;
>
```

```
> @@ -1285,7 +1291,7 @@ 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;
> @@ -1466,30 +1472,76 @@ out:
>   return err;
>  }
>
> +static struct vfsmount *fs_set_subtype(struct vfsmount *mnt, const char *fstype)
> +{
> + int err;
> + const char *subtype = strchr(fstype, '.');
> + if (subtype) {
> +  subtype++;
> +  err = -EINVAL;
> +  if (!subtype[0])
> +   goto err;
> + } else
> +  subtype = "";
> +
> + mnt->mnt_sb->s_subtype = kstrdup(subtype, GFP_KERNEL);
> + err = -ENOMEM;
> + if (!mnt->mnt_sb->s_subtype)
> +  goto err;
> + return mnt;
> +
> + err:
> + mntput(mnt);
> + return ERR_PTR(err);
> +}
> +
>  /*
>   * 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);
> + if (!IS_ERR(mnt) && (type->fs_flags & FS_HAS_SUBTYPE) &&
> +   !mnt->mnt_sb->s_subtype)
> + mnt = fs_set_subtype(mnt, fstype);
> + 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;
> }
>
> /*
> @@ -1520,7 +1572,7 @@ 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 2008-01-03 21:15:35.000000000 +0100
> +++ linux/include/linux/fs.h 2008-01-03 21:21:06.000000000 +0100
> @@ -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.
> Index: linux/fs/super.c
> ===
> --- linux.orig/fs/super.c 2008-01-02 21:42:10.000000000 +0100
> +++ linux/fs/super.c 2008-01-03 21:21:06.000000000 +0100
> @@ -906,29 +906,6 @@ out:
>
> EXPORT_SYMBOL_GPL(vfs_kern_mount);
>
> -static struct vfsmount *fs_set_subtype(struct vfsmount *mnt, const char *fstype)
> -{
> - int err;
> - const char *subtype = strchr(fstype, '.');
> - if (subtype) {
> - subtype++;
> - err = -EINVAL;
> - if (!subtype[0])
> -   goto err;
> - } else
> - subtype = "";
> -
> - mnt->mnt_sb->s_subtype = kstrdup(subtype, GFP_KERNEL);
> - err = -ENOMEM;
> - if (!mnt->mnt_sb->s_subtype)
> - goto err;
> - return mnt;
> -
> - err:
> - mntput(mnt);

> - return ERR_PTR(err);
> -}
> -
> struct vfsmount *
> do_kern_mount(const char *fstype, int flags, const char *name, void *data)
> {
> @@ -937,9 +914,6 @@ do_kern_mount(const char *fstype, int fl
>   if (!type)
>   return ERR_PTR(-ENODEV);
>   mnt = vfs_kern_mount(type, flags, name, data);
> - if (!IS_ERR(mnt) && (type->fs_flags & FS_HAS_SUBTYPE) &&
> -   !mnt->mnt_sb->s_subtype)
> - mnt = fs_set_subtype(mnt, fstype);
>   put_filesystem(type);
>   return mnt;
> }
>
> --

_____

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

---

Subject: Re: [patch 8/9] unprivileged mounts: propagation: inherit owner from parent
Posted by serue on Mon, 14 Jan 2008 23:13:40 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. Also if the parent has the
> "nosuid" flag, set this flag for the child as well.

What about nodev?

thanks,
-serge

>
> 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 2008-01-04 13:48:14.000000000 +0100
> +++ linux/fs/namespace.c 2008-01-04 13:49:52.000000000 +0100
> @@ -500,10 +500,10 @@ 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))
> @@ -514,7 +514,7 @@ 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);
> @@ -530,7 +530,7 @@ 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;
> @@ -554,7 +554,10 @@ 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);
> @@ -1060,7 +1063,7 @@ 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;
> @@ -1068,7 +1071,7 @@ 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;
> @@ -1090,7 +1093,7 @@ 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);
> @@ -1115,7 +1118,7 @@ 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;
>  }
> @@ -1286,7 +1289,8 @@ 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;
> @@ -1307,11 +1311,17 @@ 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))
> @@ -1535,7 +1545,7 @@ 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);
>
> @@ -1931,7 +1941,7 @@ 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 2008-01-04 13:47:49.000000000 +0100
> +++ linux/fs/pnode.c 2008-01-04 13:49:12.000000000 +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 2008-01-04 13:45:45.000000000 +0100
> +++ linux/fs/pnode.h 2008-01-04 13:49:12.000000000 +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 2008-01-04 13:48:14.000000000 +0100
> +++ linux/include/linux/fs.h 2008-01-04 13:49:12.000000000 +0100
> @@ -1492,7 +1492,6 @@ 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: Re: [patch 9/9] unprivileged mounts: add "no submounts" flag
Posted by serue on Mon, 14 Jan 2008 23:39:10 GMT
View Forum Message <> Reply to Message

Quoting Miklos Szeredi (miklos@szeredi.hu):
> 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.

Ah, I see, so the floppy drive could be mounted as a MNT_NOMNT but
MNT_USER mount with mnt_owner set. Makes sense. I'd ask for a better
name than 'nomnt', but I can't think of one myself.

> 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 2008-01-04 13:49:52.000000000 +0100
> +++ linux/fs/namespace.c 2008-01-04 13:50:28.000000000 +0100
> @@ -694,6 +694,7 @@ 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;
> @@ -1044,6 +1045,9 @@ static bool permit_mount(struct nameidat
>   if (S_ISLNK(inode->i_mode))
>    return false;
>
> + if (nd->path.mnt->mnt_flags & MNT_NOMNT)
> +  return false;
> +
>   if (!is_mount_owner(nd->path.mnt, current->fsuid))
>    return false;
>
> @@ -1888,9 +1892,11 @@ long do_mount(char *dev_name, char *dir_
>    mnt_flags |= MNT_RELATIME;
>   if (flags & MS_RDONLY)
>    mnt_flags |= MNT_READONLY;
> + if (flags & MS_NOMNT)
> +  mnt_flags |= MNT_NOMNT;
>
> - 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_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 2008-01-04 13:49:12.000000000 +0100
> +++ linux/include/linux/fs.h 2008-01-04 13:49:58.000000000 +0100
> @@ -130,6 +130,7 @@ 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_NOMNT (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 2008-01-04 13:45:45.000000000 +0100
> +++ linux/include/linux/mount.h 2008-01-04 13:49:58.000000000 +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_NOMNT 0x80
>
> #define MNT_SHRINKABLE 0x100
> #define MNT_IMBALANCED_WRITE_COUNT 0x200 /* just for debugging */
>
> --

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

---

Subject: Re: [patch 4/9] unprivileged mounts: propagate error values from clone_mnt
Posted by Miklos Szeredi on Tue, 15 Jan 2008 10:15:55 GMT
View Forum Message <> Reply to Message

> Quoting Miklos Szeredi (miklos@szeredi.hu):
> > 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.
>
> I see what you're saying, but it just seems like it's bound to be more
> confusing this way.

Ah yes, this is indeed confusing. Last time dup_mnt_ns() returned a
namespace pointer or NULL. But now I see it returns an ERR_PTR(error)
instead, which means it's cleaner to just propagate the error value.
I'll fix this.

Thanks,
Miklos

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

---

Subject: Re: [patch 8/9] unprivileged mounts: propagation: inherit owner from parent

Posted by Miklos Szeredi on Tue, 15 Jan 2008 10:39:08 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. Also if the parent has the
> > "nosuid" flag, set this flag for the child as well.
>
> What about nodev?

Hmm, I think the nosuid thing is meant to prevent suid mounts being introduced into a "suidless" namespace. This doesn't apply to dev mounts, which are quite safe in a suidless environment, as long as the user is not able to create devices. But that should be taken care of by capability tests.

I'll update the description.

Thanks,
Miklos

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

Subject: Re: [patch 9/9] unprivileged mounts: add "no submounts" flag
Posted by Miklos Szeredi on Tue, 15 Jan 2008 10:41:10 GMT
View Forum Message <> Reply to Message

> Quoting Miklos Szeredi (miklos@szeredi.hu):
> > 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.
>
> Ah, I see, so the floppy drive could be mounted as a MNT_NOMNT but
> MNT_USER mount with mnt_owner set. Makes sense. I'd ask for a better
> name than 'nomnt', but I can't think of one myself.

Me neither.

Thanks for the review, Serge!

Miklos

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

---

Subject: Re: [patch 9/9] unprivileged mounts: add "no submounts" flag
Posted by accensi on Tue, 15 Jan 2008 10:53:42 GMT
View Forum Message <> Reply to Message

On Jan 15, 2008 8:41 AM, Miklos Szeredi <miklos@szeredi.hu> wrote:
> > Quoting Miklos Szeredi (miklos@szeredi.hu):
> > > 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.
> >
> > Ah, I see, so the floppy drive could be mounted as a MNT_NOMNT but
> > MNT_USER mount with mnt_owner set. Makes sense. I'd ask for a better
> > name than 'nomnt', but I can't think of one myself.
>
> Me neither.
>

Why not "nosubmnt"?

--
A. C. Censi
accensi [em] gmail [ponto] com
accensi [em] montreal [ponto] com [ponto] br

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

---

Subject: Re: [patch 9/9] unprivileged mounts: add "no submounts" flag
Posted by Miklos Szeredi on Tue, 15 Jan 2008 10:58:27 GMT
View Forum Message <> Reply to Message

> Why not "nosubmnt"?

Why not indeed. Maybe I should try to use my brain sometime.

Thanks,
Miklos

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

Subject: Re: [patch 9/9] unprivileged mounts: add "no submounts" flag
Posted by serue on Tue, 15 Jan 2008 13:47:19 GMT
View Forum Message <> Reply to Message

Quoting Miklos Szeredi (miklos@szeredi.hu):
> > Why not "nosubmnt"?
>
> Why not indeed. Maybe I should try to use my brain sometime.

Well it really should have 'user' or 'unpriv' in the name
somewhere. 'nosubmnt' is more confusing than 'nomnt' because
it no submounts really sounds like a reasonable thing in
itself...

But I never win naming arguments, so I accept that I have poor
naming judgement :)

-serge

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

Subject: Re: [patch 8/9] unprivileged mounts: propagation: inherit owner from parent
Posted by serue on Tue, 15 Jan 2008 14:21:15 GMT
View Forum Message <> Reply to Message

Quoting Miklos Szeredi (miklos@szeredi.hu):
> > 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. Also if the parent has the
> > > "nosuid" flag, set this flag for the child as well.
> >
> > What about nodev?
>
> Hmm, I think the nosuid thing is meant to prevent suid mounts being
> introduced into a "suidless" namespace. This doesn't apply to dev
> mounts, which are quite safe in a suidless environment, as long as the
> user is not able to create devices. But that should be taken care of
> by capability tests.
>
> I'll update the description.

Hmm,

Part of me wants to say the safest thing for now would be to refuse
mounts propagation from non-user mounts to user mounts.

I assume you're thinking about a fully user-mounted chroot, where
the user woudl still want to be able to stick in a cdrom and have
it automounted under /mnt/cdrom, propagated from the root mounts ns?

But then are there no devices which the user could create on a floppy
while inserted into his own laptop, owned by his own uid, then insert
into this machine, and use the device under the auto-mounted /dev/floppy
to gain inappropriate access?

-serge

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

---

Subject: Re: [patch 8/9] unprivileged mounts: propagation: inherit owner from parent
Posted by Miklos Szeredi on Tue, 15 Jan 2008 14:37:37 GMT
View Forum Message <> Reply to Message

> > > > 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.
> > >
> > > What about nodev?
> >
> > Hmm, I think the nosuid thing is meant to prevent suid mounts being
> > introduced into a "suidless" namespace. This doesn't apply to dev
> > mounts, which are quite safe in a suidless environment, as long as the

> > user is not able to create devices. But that should be taken care of
> > by capability tests.
> >
> > I'll update the description.
>
> Hmm,
>
> Part of me wants to say the safest thing for now would be to refuse
> mounts propagation from non-user mounts to user mounts.
>
> I assume you're thinking about a fully user-mounted chroot, where
> the user woudl still want to be able to stick in a cdrom and have
> it automounted under /mnt/cdrom, propagated from the root mounts ns?

Right.

> But then are there no devices which the user could create on a floppy
> while inserted into his own laptop, owned by his own uid, then insert
> into this machine, and use the device under the auto-mounted /dev/floppy
> to gain inappropriate access?

I assume, that the floppy and cdrom are already mounted with
nosuid,nodev.

The problem case is I think is if a sysadmin does some mounting in the
initial namespace, and this is propagated into the fully user-mounted
namespace (or chroot), so that a mount with suid binaries slips in.
Which is bad, because the user may be able rearange the namespace, to
trick the suid program to something it should not do.

OTOH, a mount with devices can't be abused this way, since it is not
possible to gain privileges to files/devices just by rearanging the
mounts.

Miklos

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

---

Subject: Re: [patch 8/9] unprivileged mounts: propagation: inherit owner from parent
Posted by serue on Tue, 15 Jan 2008 14:59:19 GMT
View Forum Message <> Reply to Message

Quoting Miklos Szeredi (miklos@szeredi.hu):
> > > > > 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.
> > > > >
> > > > > What about nodev?
> > > >
> > > > Hmm, I think the nosuid thing is meant to prevent suid mounts being
> > > > introduced into a "suidless" namespace. This doesn't apply to dev
> > > > mounts, which are quite safe in a suidless environment, as long as the
> > > > user is not able to create devices. But that should be taken care of
> > > > by capability tests.
> > > >
> > > > I'll update the description.
> > >
> > > Hmm,
> > >
> > > Part of me wants to say the safest thing for now would be to refuse
> > > mounts propagation from non-user mounts to user mounts.
> > >
> > > I assume you're thinking about a fully user-mounted chroot, where
> > > the user woudl still want to be able to stick in a cdrom and have
> > > it automounted under /mnt/cdrom, propagated from the root mounts ns?
> >
> > Right.
> >
> > > But then are there no devices which the user could create on a floppy
> > > while inserted into his own laptop, owned by his own uid, then insert
> > > into this machine, and use the device under the auto-mounted /dev/floppy
> > > to gain inappropriate access?
> >
> > I assume, that the floppy and cdrom are already mounted with
> > nosuid,nodev.

Yeah, of course, what I'm saying is no different whether the upper mount
is a user mount or not. You're right.

> The problem case is I think is if a sysadmin does some mounting in the
> initial namespace, and this is propagated into the fully user-mounted
> namespace (or chroot), so that a mount with suid binaries slips in.
> Which is bad, because the user may be able rearange the namespace, to
> trick the suid program to something it should not do.

And really this shouldn't be an issue at all - the usermount chroot
would be set up under something like /share/hallyn/root, so the admin
would have to purposely set up propagation into that tree, so this
won't be happening by accident.

> OTOH, a mount with devices can't be abused this way, since it is not
> possible to gain privileges to files/devices just by rearanging the
> mounts.

Thanks for humoring me,

-serge

_____

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

---

Subject: Re: [patch 9/9] unprivileged mounts: add "no submounts" flag
Posted by Miklos Szeredi on Wed, 16 Jan 2008 09:43:10 GMT
View Forum Message <> Reply to Message

> > > Why not "nosubmnt"?
> >
> > Why not indeed. Maybe I should try to use my brain sometime.
>
> Well it really should have 'user' or 'unpriv' in the name
> somewhere. 'nosubmnt' is more confusing than 'nomnt' because
> it no submounts really sounds like a reasonable thing in
> itself...

I slept on it, and I still think 'nosubmnt' might be the best
compromise. Obviously the superuser has privileges, that override
what is normally allowed, and we don't find it strange when a
read-only file is happily being written by root.

It may feel wrong in the context of mounts, because we are so used to
mounts being privileged-only.

Objections? Once this goes in, it will stay the same forever, so now
is the time to express any doubts...

Thanks,
Miklos

_____

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

---