Subject: [patch 0/8] mount ownership and unprivileged mount syscall (v4)
Posted by Miklos Szeredi on Fri, 20 Apr 2007 10:25:32 GMT

View Forum Message <> Reply to Message

This patchset has now been bared to the "lowest common denominator"
that everybody can agree on. Or at least there weren't any objections
to this proposal.

Andrew, please consider it for -mm.

Thanks,
Miklos
----

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

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

--

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

Subject: [patch 1/8] add user mounts to the kernel
Posted by Miklos Szeredi on Fri, 20 Apr 2007 10:25:33 GMT

View Forum Message <> Reply to Message

From: Miklos Szeredi <mszeredi@suse.cz>

Add ownership information to mounts.

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
real user id 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.

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

```
Index: linux/fs/namespace.c
===================================================================
--- linux.orig/fs/namespace.c 2007-04-20 11:55:02.000000000 +0200
+++ linux/fs/namespace.c 2007-04-20 11:55:05.000000000 +0200
@@ -227,6 +227,13 @@ static struct vfsmount *skip_mnt_tree(st
  return p;
 }
```

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

+  /* don't copy the MNT_USER flag */
+  mnt->mnt_flags &= ~MNT_USER;
+  if (flag & CL_SETUSER)
+   set_mnt_user(mnt);
+
   if (flag & CL_SLAVE) {
    list_add(&mnt->mnt_slave, &old->mnt_slave_list);
    mnt->mnt_master = old;
@@ -403,6 +415,8 @@ static int show_vfsmnt(struct seq_file *
   if (mnt->mnt_flags & fs_infop->flag)
    seq_puts(m, fs_infop->str);
  }
+ if (mnt->mnt_flags & MNT_USER)
+  seq_printf(m, ",user=%i", mnt->mnt_uid);
  if (mnt->mnt_sb->s_op->show_options)
   err = mnt->mnt_sb->s_op->show_options(m, mnt);
  seq_puts(m, " 0 0\n");
@@ -920,8 +934,9 @@ static int do_change_type(struct nameida
 /*
  * do loopback mount.
  */
-static int do_loopback(struct nameidata *nd, char *old_name, int recurse)
+static int do_loopback(struct nameidata *nd, char *old_name, int flags)
 {
+ int clone_flags;
  struct nameidata old_nd;
  struct vfsmount *mnt = NULL;
  int err = mount_is_safe(nd);
@@ -941,11 +956,12 @@ static int do_loopback(struct nameidata
  if (!check_mnt(nd->mnt) || !check_mnt(old_nd.mnt))
   goto out;

+ clone_flags = (flags & MS_SETUSER) ? CL_SETUSER : 0;
  err = -ENOMEM;
```

```
- if (recurse)
-  mnt = copy_tree(old_nd.mnt, old_nd.dentry, 0);
+ if (flags & MS_REC)
+  mnt = copy_tree(old_nd.mnt, old_nd.dentry, clone_flags);
  else
-  mnt = clone_mnt(old_nd.mnt, old_nd.dentry, 0);
+  mnt = clone_mnt(old_nd.mnt, old_nd.dentry, clone_flags);

  if (!mnt)
   goto out;
@@ -987,8 +1003,11 @@ static int do_remount(struct nameidata *

  down_write(&sb->s_umount);
  err = do_remount_sb(sb, flags, data, 0);
- if (!err)
+ if (!err) {
   nd->mnt->mnt_flags = mnt_flags;
+  if (flags & MS_SETUSER)
+   set_mnt_user(nd->mnt);
+ }
  up_write(&sb->s_umount);
  if (!err)
   security_sb_post_remount(nd->mnt, flags, data);
@@ -1093,10 +1112,13 @@ 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);
 }

@@ -1127,7 +1149,8 @@ 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;
```

```
@@ -1447,7 +1470,7 @@ 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/include/linux/fs.h
===================================================================
--- linux.orig/include/linux/fs.h 2007-04-20 11:55:02.000000000 +0200
+++ linux/include/linux/fs.h 2007-04-20 11:55:05.000000000 +0200
@@ -123,6 +123,7 @@ extern int dir_notify_enable;
 #define MS_SLAVE (1<<19) /* change to slave */
 #define MS_SHARED (1<<20) /* change to shared */
 #define MS_RELATIME (1<<21) /* Update atime relative to mtime/ctime. */
+#define MS_SETUSER (1<<22) /* set mnt_uid to current user */
 #define MS_ACTIVE (1<<30)
 #define MS_NOUSER (1<<31)

Index: linux/include/linux/mount.h
===================================================================
--- linux.orig/include/linux/mount.h 2007-04-20 11:55:02.000000000 +0200
+++ linux/include/linux/mount.h 2007-04-20 11:55:05.000000000 +0200
@@ -30,6 +30,7 @@ struct mnt_namespace;
 #define MNT_RELATIME 0x20

 #define MNT_SHRINKABLE 0x100
+#define MNT_USER 0x200

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

 static inline struct vfsmount *mntget(struct vfsmount *mnt)
Index: linux/fs/pnode.h
===================================================================
--- linux.orig/fs/pnode.h 2007-04-20 11:55:02.000000000 +0200
+++ linux/fs/pnode.h 2007-04-20 11:55:05.000000000 +0200
@@ -22,6 +22,7 @@
 #define CL_COPY_ALL   0x04
 #define CL_MAKE_SHARED   0x08
 #define CL_PROPAGATION   0x10
+#define CL_SETUSER  0x20

 static inline void set_mnt_shared(struct vfsmount *mnt)
 {
```

--

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

Subject: [patch 2/8] allow unprivileged umount
Posted by Miklos Szeredi on Fri, 20 Apr 2007 10:25:34 GMT

View Forum Message <> Reply to Message

From: Miklos Szeredi <mszeredi@suse.cz>

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

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

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

```
Index: linux/fs/namespace.c
===================================================================
--- linux.orig/fs/namespace.c 2007-04-20 11:55:05.000000000 +0200
+++ linux/fs/namespace.c 2007-04-20 11:55:06.000000000 +0200
@@ -659,6 +659,25 @@ static int do_umount(struct vfsmount *mn
 }

 /*
+ * 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 (!(mnt->mnt_flags & MNT_USER))
+  return false;
```

```
+
+ if (flags & MNT_FORCE)
+  return false;
+
+ return mnt->mnt_uid == current->uid;
+}
+
+/*
  * Now umount can handle mount points as well as block devices.
  * This is important for filesystems which use unnamed block devices.
  *
@@ -681,7 +700,7 @@ asmlinkage long sys_umount(char __user *
   goto dput_and_out;

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

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

--

____

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

---

Subject: [patch 3/8] account user mounts

Posted by Miklos Szeredi on Fri, 20 Apr 2007 10:25:35 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

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

```
Index: linux/include/linux/sysctl.h
===================================================================
--- linux.orig/include/linux/sysctl.h 2007-04-20 11:55:02.000000000 +0200
+++ linux/include/linux/sysctl.h 2007-04-20 11:55:07.000000000 +0200
```

```
@@ -818,6 +818,8 @@ enum
  FS_AIO_NR=18, /* current system-wide number of aio requests */
  FS_AIO_MAX_NR=19, /* system-wide maximum number of aio requests */
  FS_INOTIFY=20, /* inotify submenu */
+ FS_NR_USER_MOUNTS=21, /* int:current number of user mounts */
+ FS_MAX_USER_MOUNTS=22, /* int:maximum number of user mounts */
  FS_OCFS2=988, /* ocfs2 */
 };

Index: linux/kernel/sysctl.c
===================================================================
--- linux.orig/kernel/sysctl.c 2007-04-20 11:55:02.000000000 +0200
+++ linux/kernel/sysctl.c 2007-04-20 11:55:07.000000000 +0200
@@ -1063,6 +1063,22 @@ static ctl_table fs_table[] = {
 #endif
 #endif
  {
+  .ctl_name = FS_NR_USER_MOUNTS,
+  .procname = "nr_user_mounts",
+  .data  = &nr_user_mounts,
+  .maxlen  = sizeof(int),
+  .mode  = 0444,
+  .proc_handler = &proc_dointvec,
+ },
+ {
+  .ctl_name = FS_MAX_USER_MOUNTS,
+  .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,
Index: linux/Documentation/filesystems/proc.txt
===================================================================
--- linux.orig/Documentation/filesystems/proc.txt 2007-04-20 11:55:02.000000000 +0200
+++ linux/Documentation/filesystems/proc.txt 2007-04-20 11:55:07.000000000 +0200
@@ -923,6 +923,15 @@ reaches aio-max-nr then io_setup will fa
 raising aio-max-nr does not result in the pre-allocation or re-sizing
 of any kernel data structures.

+nr_user_mounts and max_user_mounts
+----------------------------------
+
+These represent the number of "user" mounts and the maximum number of
```

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

Index: linux/fs/namespace.c
===================================================================
--- linux.orig/fs/namespace.c 2007-04-20 11:55:06.000000000 +0200
+++ linux/fs/namespace.c 2007-04-20 11:55:07.000000000 +0200
@@ -39,6 +39,9 @@ static int hash_mask __read_mostly, hash
 static struct kmem_cache *mnt_cache __read_mostly;
 static struct rw_semaphore namespace_sem;

+int nr_user_mounts;
+int max_user_mounts = 1024;
+
 /* /sys/fs */
 decl_subsys(fs, NULL, NULL);
 EXPORT_SYMBOL_GPL(fs_subsys);
@@ -227,11 +230,30 @@ static struct vfsmount *skip_mnt_tree(st
  return p;
 }

+static void dec_nr_user_mounts(void)
+{
+ spin_lock(&vfsmount_lock);
+ nr_user_mounts--;
+ spin_unlock(&vfsmount_lock);
+}
+
 static void set_mnt_user(struct vfsmount *mnt)
 {
  BUG_ON(mnt->mnt_flags & MNT_USER);
  mnt->mnt_uid = current->uid;
  mnt->mnt_flags |= MNT_USER;
+ spin_lock(&vfsmount_lock);
+ nr_user_mounts++;
+ spin_unlock(&vfsmount_lock);
+}
+
+static void clear_mnt_user(struct vfsmount *mnt)
+{
+ if (mnt->mnt_flags & MNT_USER) {
+  mnt->mnt_uid = 0;
+  mnt->mnt_flags &= ~MNT_USER;
+  dec_nr_user_mounts();
+ }
 }

 static struct vfsmount *clone_mnt(struct vfsmount *old, struct dentry *root,
@@ -283,6 +305,7 @@ static inline void __mntput(struct vfsmo
 {
  struct super_block *sb = mnt->mnt_sb;
  dput(mnt->mnt_root);
+ clear_mnt_user(mnt);
  free_vfsmnt(mnt);
  deactivate_super(sb);
 }
@@ -1023,6 +1046,7 @@ static int do_remount(struct nameidata *
  down_write(&sb->s_umount);
  err = do_remount_sb(sb, flags, data, 0);
  if (!err) {
+  clear_mnt_user(nd->mnt);
   nd->mnt->mnt_flags = mnt_flags;
   if (flags & MS_SETUSER)
    set_mnt_user(nd->mnt);
Index: linux/include/linux/fs.h
===================================================================
--- linux.orig/include/linux/fs.h 2007-04-20 11:55:05.000000000 +0200
+++ linux/include/linux/fs.h 2007-04-20 11:55:07.000000000 +0200
@@ -50,6 +50,9 @@ extern struct inodes_stat_t inodes_stat;

 extern int leases_enable, lease_break_time;

+extern int nr_user_mounts;
+extern int max_user_mounts;
+
 #ifdef CONFIG_DNOTIFY
 extern int dir_notify_enable;
 #endif
```

--

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

Subject: [patch 4/8] propagate error values from clone_mnt
Posted by Miklos Szeredi on Fri, 20 Apr 2007 10:25:36 GMT

View Forum Message <> Reply to Message

From: Miklos Szeredi <mszeredi@suse.cz>

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

Fix copy_tree() to return EPERM for unbindable mounts.

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

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

```
Index: linux/fs/namespace.c
===================================================================
--- linux.orig/fs/namespace.c 2007-04-20 11:55:07.000000000 +0200
+++ linux/fs/namespace.c 2007-04-20 11:55:09.000000000 +0200
@@ -261,42 +261,42 @@ static struct vfsmount *clone_mnt(struct
 {
  struct super_block *sb = old->mnt_sb;
  struct vfsmount *mnt = alloc_vfsmnt(old->mnt_devname);
+ if (!mnt)
+  return ERR_PTR(-ENOMEM);

- if (mnt) {
-  mnt->mnt_flags = old->mnt_flags;
-  atomic_inc(&sb->s_active);
-  mnt->mnt_sb = sb;
-  mnt->mnt_root = dget(root);
-  mnt->mnt_mountpoint = mnt->mnt_root;
-  mnt->mnt_parent = mnt;
-
-  /* don't copy the MNT_USER flag */
-  mnt->mnt_flags &= ~MNT_USER;
-  if (flag & CL_SETUSER)
-   set_mnt_user(mnt);
-
-  if (flag & CL_SLAVE) {
-   list_add(&mnt->mnt_slave, &old->mnt_slave_list);
-   mnt->mnt_master = old;
-   CLEAR_MNT_SHARED(mnt);
-  } else {
-   if ((flag & CL_PROPAGATION) || IS_MNT_SHARED(old))
-    list_add(&mnt->mnt_share, &old->mnt_share);
-   if (IS_MNT_SLAVE(old))
-    list_add(&mnt->mnt_slave, &old->mnt_slave);
-   mnt->mnt_master = old->mnt_master;
-  }
-  if (flag & CL_MAKE_SHARED)
-   set_mnt_shared(mnt);
+ 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);

-  /* 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);
-  }
+ if (flag & CL_SLAVE) {
+  list_add(&mnt->mnt_slave, &old->mnt_slave_list);
+  mnt->mnt_master = old;
+  CLEAR_MNT_SHARED(mnt);
+ } else {
+  if ((flag & CL_PROPAGATION) || IS_MNT_SHARED(old))
+   list_add(&mnt->mnt_share, &old->mnt_share);
+  if (IS_MNT_SLAVE(old))
+   list_add(&mnt->mnt_slave, &old->mnt_slave);
+  mnt->mnt_master = old->mnt_master;
+ }
+ if (flag & CL_MAKE_SHARED)
+  set_mnt_shared(mnt);
+
+ /* stick the duplicate mount on the same expiry list
+  * as the original if that was on one */
+ if (flag & CL_EXPIRE) {
+  spin_lock(&vfsmount_lock);
+  if (!list_empty(&old->mnt_expire))
+   list_add(&mnt->mnt_expire, &old->mnt_expire);
+  spin_unlock(&vfsmount_lock);
  }
  return mnt;
 }
@@ -781,11 +781,11 @@ 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;
@@ -806,8 +806,8 @@ struct vfsmount *copy_tree(struct vfsmou
    nd.mnt = q;
    nd.dentry = p->mnt_mountpoint;
    q = clone_mnt(p, p->mnt_root, flag);
-   if (!q)
-    goto Enomem;
+   if (IS_ERR(q))
+    goto error;
    spin_lock(&vfsmount_lock);
    list_add_tail(&q->mnt_list, &res->mnt_list);
    attach_mnt(q, &nd);
@@ -815,7 +815,7 @@ struct vfsmount *copy_tree(struct vfsmou
   }
  }
  return res;
-Enomem:
+ error:
  if (res) {
   LIST_HEAD(umount_list);
   spin_lock(&vfsmount_lock);
@@ -823,7 +823,7 @@ Enomem:
   spin_unlock(&vfsmount_lock);
   release_mounts(&umount_list);
  }
- return NULL;
+ return q;
 }

 /*
@@ -999,13 +999,13 @@ static int do_loopback(struct nameidata
   goto out;

  clone_flags = (flags & MS_SETUSER) ? CL_SETUSER : 0;
- err = -ENOMEM;
  if (flags & MS_REC)
```

```
   mnt = copy_tree(old_nd.mnt, old_nd.dentry, clone_flags);
  else
   mnt = clone_mnt(old_nd.mnt, old_nd.dentry, clone_flags);

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

  err = graft_tree(mnt, nd);
@@ -1550,7 +1550,7 @@ static struct mnt_namespace *dup_mnt_ns(
  /* First pass: copy the tree topology */
  new_ns->root = copy_tree(mnt_ns->root, mnt_ns->root->mnt_root,
      CL_COPY_ALL | CL_EXPIRE);
- if (!new_ns->root) {
+ if (IS_ERR(new_ns->root)) {
   up_write(&namespace_sem);
   kfree(new_ns);
   return NULL;
Index: linux/fs/pnode.c
===================================================================
--- linux.orig/fs/pnode.c 2007-04-20 11:55:02.000000000 +0200
+++ linux/fs/pnode.c 2007-04-20 11:55:09.000000000 +0200
@@ -187,8 +187,9 @@ int propagate_mnt(struct vfsmount *dest_

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

-  if (!(child = copy_tree(source, source->mnt_root, type))) {
-   ret = -ENOMEM;
+  child = copy_tree(source, source->mnt_root, type);
+  if (IS_ERR(child)) {
+   ret = PTR_ERR(child);
    list_splice(tree_list, tmp_list.prev);
    goto out;
   }
```

--

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

Subject: [patch 5/8] allow unprivileged bind mounts
Posted by Miklos Szeredi on Fri, 20 Apr 2007 10:25:37 GMT

View Forum Message <> Reply to Message

From: Miklos Szeredi <mszeredi@suse.cz>

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

  - mountpoint is not a symlink or special file
  - 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.

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

```
Index: linux/fs/namespace.c
===================================================================
--- linux.orig/fs/namespace.c 2007-04-20 11:55:09.000000000 +0200
+++ linux/fs/namespace.c 2007-04-20 11:55:10.000000000 +0200
@@ -237,11 +237,30 @@ static void dec_nr_user_mounts(void)
  spin_unlock(&vfsmount_lock);
 }

-static void set_mnt_user(struct vfsmount *mnt)
+static int reserve_user_mount(void)
+{
+ int err = 0;
+ spin_lock(&vfsmount_lock);
+ if (nr_user_mounts >= max_user_mounts && !capable(CAP_SYS_ADMIN))
+  err = -EPERM;
+ else
+  nr_user_mounts++;
+ spin_unlock(&vfsmount_lock);
+ return err;
+}
+
+static void __set_mnt_user(struct vfsmount *mnt)
 {
  BUG_ON(mnt->mnt_flags & MNT_USER);
  mnt->mnt_uid = current->uid;
  mnt->mnt_flags |= MNT_USER;
+ if (!capable(CAP_SYS_ADMIN))
+  mnt->mnt_flags |= MNT_NOSUID | MNT_NODEV;
+}
+
+static void set_mnt_user(struct vfsmount *mnt)
+{
+ __set_mnt_user(mnt);
  spin_lock(&vfsmount_lock);
  nr_user_mounts++;
  spin_unlock(&vfsmount_lock);
@@ -260,9 +279,16 @@ static struct vfsmount *clone_mnt(struct
      int flag)
 {
  struct super_block *sb = old->mnt_sb;
- struct vfsmount *mnt = alloc_vfsmnt(old->mnt_devname);
+ struct vfsmount *mnt;
+
+ if (flag & CL_SETUSER) {
+  int err = reserve_user_mount();
+  if (err)
+   return ERR_PTR(err);
+ }
+ mnt = alloc_vfsmnt(old->mnt_devname);
  if (!mnt)
-  return ERR_PTR(-ENOMEM);
+  goto alloc_failed;

  mnt->mnt_flags = old->mnt_flags;
  atomic_inc(&sb->s_active);
@@ -274,7 +300,7 @@ 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);
@@ -299,6 +325,11 @@ 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)
@@ -745,22 +776,29 @@ 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 or special file
+ * - mountpoint is in a mount owned by the user
+ */
+static bool permit_mount(struct nameidata *nd, int *flags)
 {
+ struct inode *inode = nd->dentry->d_inode;
+
  if (capable(CAP_SYS_ADMIN))
-  return 0;
- return -EPERM;
-#ifdef notyet
- if (S_ISLNK(nd->dentry->d_inode->i_mode))
-  return -EPERM;
- if (nd->dentry->d_inode->i_mode & S_ISVTX) {
-  if (current->uid != nd->dentry->d_inode->i_uid)
-   return -EPERM;
- }
- if (vfs_permission(nd, MAY_WRITE))
-  return -EPERM;
- return 0;
-#endif
+  return true;
+
+ if (!S_ISDIR(inode->i_mode) && !S_ISREG(inode->i_mode))
+  return false;
+
+ if (!(nd->mnt->mnt_flags & MNT_USER))
+  return false;
+
+ if (nd->mnt->mnt_uid != current->uid)
+  return false;
+
+ *flags |= MS_SETUSER;
+ return true;
 }

 static int lives_below_in_same_fs(struct dentry *d, struct dentry *dentry)
@@ -981,9 +1019,10 @@ static int do_loopback(struct nameidata
  int clone_flags;
  struct nameidata old_nd;
  struct vfsmount *mnt = NULL;
- int err = mount_is_safe(nd);
- if (err)
-  return err;
+ int err;
+
+ if (!permit_mount(nd, &flags))
+  return -EPERM;
  if (!old_name || !*old_name)
   return -EINVAL;
  err = path_lookup(old_name, LOOKUP_FOLLOW, &old_nd);
```

--

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

Subject: [patch 6/8] put declaration of put_filesystem() in fs.h
Posted by Miklos Szeredi on Fri, 20 Apr 2007 10:25:38 GMT
View Forum Message <> Reply to Message

From: Miklos Szeredi <mszeredi@suse.cz>

Declarations go into headers.

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

```
Index: linux/fs/super.c
===================================================================
--- linux.orig/fs/super.c 2007-04-20 11:55:02.000000000 +0200
+++ linux/fs/super.c 2007-04-20 11:55:11.000000000 +0200
@@ -40,10 +40,6 @@
 #include <asm/uaccess.h>


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

Index: linux/include/linux/fs.h
===================================================================
--- linux.orig/include/linux/fs.h 2007-04-20 11:55:07.000000000 +0200
+++ linux/include/linux/fs.h 2007-04-20 11:55:11.000000000 +0200
@@ -1918,6 +1918,8 @@ extern int vfs_fstat(unsigned int, struc

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

+extern void get_filesystem(struct file_system_type *fs);
+extern void put_filesystem(struct file_system_type *fs);
```

```
 extern struct file_system_type *get_fs_type(const char *name);
 extern struct super_block *get_super(struct block_device *);
 extern struct super_block *user_get_super(dev_t);
```

--

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

Subject: [patch 7/8] allow unprivileged mounts
Posted by Miklos Szeredi on Fri, 20 Apr 2007 10:25:39 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.

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

```
Index: linux/fs/namespace.c
===================================================================
--- linux.orig/fs/namespace.c 2007-04-20 11:55:10.000000000 +0200
+++ linux/fs/namespace.c 2007-04-20 11:55:13.000000000 +0200
@@ -781,13 +781,17 @@ asmlinkage long sys_oldumount(char __use
  * - mountpoint is not a symlink or special file
  * - mountpoint is in a mount owned by the user
  */
-static bool permit_mount(struct nameidata *nd, int *flags)
+static bool permit_mount(struct nameidata *nd, struct file_system_type *type,
+    int *flags)
 {
  struct inode *inode = nd->dentry->d_inode;

  if (capable(CAP_SYS_ADMIN))
   return true;

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

```
@@ -1021,7 +1025,7 @@ 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;
@@ -1182,26 +1186,46 @@ out:
  * create a new mount for userspace and request it to be added into the
  * namespace's tree
  */
-static int do_new_mount(struct nameidata *nd, char *type, int flags,
+static int do_new_mount(struct nameidata *nd, char *fstype, int flags,
    int mnt_flags, char *name, void *data)
 {
+ int err;
  struct vfsmount *mnt;
+ struct file_system_type *type;

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

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

  if (flags & MS_SETUSER)
-  set_mnt_user(mnt);
+  __set_mnt_user(mnt);

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

 /*
@@ -1231,7 +1255,7 @@ int do_add_mount(struct vfsmount *newmnt
  if (S_ISLNK(newmnt->mnt_root->d_inode->i_mode))
   goto unlock;

- /* MNT_USER was set earlier */
+ /* some flags may have been set earlier */
  newmnt->mnt_flags |= mnt_flags;
  if ((err = graft_tree(newmnt, nd)))
   goto unlock;
Index: linux/include/linux/fs.h
===================================================================
--- linux.orig/include/linux/fs.h 2007-04-20 11:55:11.000000000 +0200
+++ linux/include/linux/fs.h 2007-04-20 11:55:13.000000000 +0200
@@ -96,6 +96,7 @@ extern int dir_notify_enable;
 #define FS_REQUIRES_DEV 1
 #define FS_BINARY_MOUNTDATA 2
 #define FS_HAS_SUBTYPE 4
+#define FS_SAFE 8  /* Safe to mount by unprivileged users */
 #define FS_REVAL_DOT 16384 /* Check the paths ".", ".." for staleness */
 #define FS_RENAME_DOES_D_MOVE 32768 /* FS will handle d_move()
       * during rename() internally.

--
```

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

Subject: [patch 8/8] allow unprivileged fuse mounts
Posted by Miklos Szeredi on Fri, 20 Apr 2007 10:25:40 GMT
View Forum Message <> Reply to Message

From: Miklos Szeredi <mszeredi@suse.cz>

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

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

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

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

Disallow the "allow_other" option for unprivileged mounts.

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

```
---

Index: linux/fs/fuse/inode.c
===================================================================
--- linux.orig/fs/fuse/inode.c 2007-04-20 11:55:01.000000000 +0200
+++ linux/fs/fuse/inode.c 2007-04-20 11:55:14.000000000 +0200
@@ -311,6 +311,19 @@ static int parse_fuse_opt(char *opt, str
  d->max_read = ~0;
  d->blksize = 512;

+ /*
+  * For unprivileged mounts use current uid/gid.  Still allow
+  * "user_id" and "group_id" options for compatibility, but
+  * only if they match these values.
+  */
+ if (!capable(CAP_SYS_ADMIN)) {
+  d->user_id = current->uid;
+  d->user_id_present = 1;
+  d->group_id = current->gid;
+  d->group_id_present = 1;
+
+ }
+
  while ((p = strsep(&opt, ",")) != NULL) {
   int token;
   int value;
@@ -339,6 +352,8 @@ static int parse_fuse_opt(char *opt, str
   case OPT_USER_ID:
    if (match_int(&args[0], &value))
     return 0;
+   if (d->user_id_present && d->user_id != value)
+    return 0;
    d->user_id = value;
    d->user_id_present = 1;
    break;
@@ -346,6 +361,8 @@ static int parse_fuse_opt(char *opt, str
   case OPT_GROUP_ID:
    if (match_int(&args[0], &value))
     return 0;
+   if (d->group_id_present && d->group_id != value)
+    return 0;
    d->group_id = value;
    d->group_id_present = 1;
    break;
@@ -536,6 +553,10 @@ static int fuse_fill_super(struct super_
  if (!parse_fuse_opt((char *) data, &d, is_bdev))
   return -EINVAL;

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

 #ifdef CONFIG_BLOCK

--
```

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

Subject: Re: [patch 1/8] add user mounts to the kernel
Posted by akpm on Sat, 21 Apr 2007 07:55:03 GMT
View Forum Message <> Reply to Message

On Fri, 20 Apr 2007 12:25:33 +0200 Miklos Szeredi <miklos@szeredi.hu> wrote:

> From: Miklos Szeredi <mszeredi@suse.cz>
>
> Add ownership information to mounts.
>
> 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
> real user id 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.

So is a modified mount(8) needed? If so, is there some convenient way
in which testers can get hold of it?

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

```
> Index: linux/fs/namespace.c
> ===================================================================
> --- linux.orig/fs/namespace.c 2007-04-20 11:55:02.000000000 +0200
> +++ linux/fs/namespace.c 2007-04-20 11:55:05.000000000 +0200
> @@ -227,6 +227,13 @@ static struct vfsmount *skip_mnt_tree(st
>   return p;
>  }
>
> +static void set_mnt_user(struct vfsmount *mnt)
> +{
> + BUG_ON(mnt->mnt_flags & MNT_USER);
> + mnt->mnt_uid = current->uid;
> + mnt->mnt_flags |= MNT_USER;
> +}
```

I'm a bit surprised to see this. Using uids in-kernel is all rather
old-fashioned and restricted. I'd have expected

```
 mnt->user = get_uid(current->user);
```

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

Subject: Re: [patch 2/8] allow unprivileged umount
Posted by akpm on Sat, 21 Apr 2007 07:55:06 GMT
View Forum Message <> Reply to Message

On Fri, 20 Apr 2007 12:25:34 +0200 Miklos Szeredi <miklos@szeredi.hu> wrote:

```
> +static bool permit_umount(struct vfsmount *mnt, int flags)
> +{
>
> ...
>
> + return mnt->mnt_uid == current->uid;
> +}
```

Yes, this seems very wrong. I'd have thought that comparing user_struct*'s would get us a heck of a lot closer to being able to support aliasing of UIDs between different namespaces.

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

Subject: Re: [patch 3/8] account user mounts
Posted by akpm on Sat, 21 Apr 2007 07:55:09 GMT
View Forum Message <> Reply to Message

On Fri, 20 Apr 2007 12:25:35 +0200 Miklos Szeredi <miklos@szeredi.hu> wrote:

> 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
>
> Signed-off-by: Miklos Szeredi <mszeredi@suse.cz>
> ---
>
> Index: linux/include/linux/sysctl.h

```
> ===================================================================
> --- linux.orig/include/linux/sysctl.h 2007-04-20 11:55:02.000000000 +0200
> +++ linux/include/linux/sysctl.h 2007-04-20 11:55:07.000000000 +0200
> @@ -818,6 +818,8 @@ enum
>   FS_AIO_NR=18, /* current system-wide number of aio requests */
>   FS_AIO_MAX_NR=19, /* system-wide maximum number of aio requests */
>   FS_INOTIFY=20, /* inotify submenu */
> + FS_NR_USER_MOUNTS=21, /* int:current number of user mounts */
> + FS_MAX_USER_MOUNTS=22, /* int:maximum number of user mounts */
>   FS_OCFS2=988, /* ocfs2 */
```

Is there a special reason why the enumerated sysctls are needed? We're
trying to get away from using them.

```
diff -puN include/linux/sysctl.h~unprivileged-mounts-account-user-mounts-fix include/linux/sysctl.h
--- a/include/linux/sysctl.h~unprivileged-mounts-account-user-mounts-fix
+++ a/include/linux/sysctl.h
@@ -819,8 +819,6 @@ enum
  FS_AIO_NR=18, /* current system-wide number of aio requests */
  FS_AIO_MAX_NR=19, /* system-wide maximum number of aio requests */
  FS_INOTIFY=20, /* inotify submenu */
- FS_NR_USER_MOUNTS=21, /* int:current number of user mounts */
- FS_MAX_USER_MOUNTS=22, /* int:maximum number of user mounts */
  FS_OCFS2=988, /* ocfs2 */
 };

diff -puN kernel/sysctl.c~unprivileged-mounts-account-user-mounts-fix kernel/sysctl.c
--- a/kernel/sysctl.c~unprivileged-mounts-account-user-mounts-fix
+++ a/kernel/sysctl.c
@@ -1028,7 +1028,7 @@ static ctl_table fs_table[] = {
 #endif
 #endif
  {
-  .ctl_name = FS_NR_USER_MOUNTS,
+  .ctl_name = CTL_UNNUMBERED,
   .procname = "nr_user_mounts",
   .data  = &nr_user_mounts,
   .maxlen  = sizeof(int),
@@ -1036,7 +1036,7 @@ static ctl_table fs_table[] = {
   .proc_handler = &proc_dointvec,
  },
  {
-  .ctl_name = FS_MAX_USER_MOUNTS,
+  .ctl_name = CTL_UNNUMBERED,
   .procname = "max_user_mounts",
   .data  = &max_user_mounts,
   .maxlen  = sizeof(int),
```

–

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

Subject: Re: [patch 7/8] allow unprivileged mounts
Posted by akpm on Sat, 21 Apr 2007 07:55:13 GMT

View Forum Message <> Reply to Message

On Fri, 20 Apr 2007 12:25:39 +0200 Miklos Szeredi <miklos@szeredi.hu> 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.

Practically speaking, is there any realistic likelihood that any filesystem
apart from FUSE will ever use this?

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

Subject: Re: [patch 8/8] allow unprivileged fuse mounts
Posted by akpm on Sat, 21 Apr 2007 07:55:16 GMT

View Forum Message <> Reply to Message

On Fri, 20 Apr 2007 12:25:40 +0200 Miklos Szeredi <miklos@szeredi.hu> wrote:

> Use FS_SAFE for "fuse" fs type, but not for "fuseblk".
>
> FUSE was designed from the beginning to be safe for unprivileged
> users. This has also been verified in practice over many years.

How does FUSE do this?

There are obvious cases like crafting a filesystem which has setuid executables
or world-writeable device nodes or whatever. I'm sure there are lots of other
cases.

Where is FUSE's implementation of all this protection described?

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

Subject: Re: [patch 2/8] allow unprivileged umount
Posted by hpa on Sat, 21 Apr 2007 08:01:58 GMT

View Forum Message <> Reply to Message

Andrew Morton wrote:
> On Fri, 20 Apr 2007 12:25:34 +0200 Miklos Szeredi <miklos@szeredi.hu> wrote:
>
>> +static bool permit_umount(struct vfsmount *mnt, int flags)
>> +{
>>
>> ...
>>
>> + return mnt->mnt_uid == current->uid;
>> +}
>
> Yes, this seems very wrong. I'd have thought that comparing user_struct*'s
> would get us a heck of a lot closer to being able to support aliasing of
> UIDs between different namespaces.
>

Not to mention it should be fsuid, not uid.

-hpa

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

Subject: Re: [patch 1/8] add user mounts to the kernel
Posted by Miklos Szeredi on Sat, 21 Apr 2007 08:06:28 GMT

View Forum Message <> Reply to Message

> > From: Miklos Szeredi <mszeredi@suse.cz>
> >
> > Add ownership information to mounts.
> >
> > 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
> > real user id 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.
>
> So is a modified mount(8) needed? If so, is there some convenient way
> in which testers can get hold of it?

I haven't yet added this to mount(8). I'll do that and post patches.

There's an ultra sophisticated mount tester program I use. It's
slightly user unfriendly...

Miklos

```
----
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mount.h>

int main(int argc, char *argv[])
{
 int res;
 int un = (strrchr(argv[0], '/') ?: argv[0] - 1)[1] == 'u';
 if ((un && argc != 3) || (!un && argc != 6)) {
  fprintf(stderr, "usage: %s %s\n", argv[0],
   un ? "dir flags" : "dev dir type flags opts");
  return 1;
 }
 if (un)
  res = umount2(argv[1], atoi(argv[2]));
 else
  res = mount(argv[1], argv[2], argv[3], atoi(argv[4]), argv[5]);
 if (res == -1) {
  perror(argv[0]);
  return 1;
 }
 return 0;
}
```

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

Subject: Re: [patch 2/8] allow unprivileged umount
Posted by Miklos Szeredi on Sat, 21 Apr 2007 08:09:42 GMT

View Forum Message <> Reply to Message

> > +static bool permit_umount(struct vfsmount *mnt, int flags)
> > +{
> >
> > ...
> >
> > + return mnt->mnt_uid == current->uid;
> > +}
>
> Yes, this seems very wrong. I'd have thought that comparing user_struct*'s
> would get us a heck of a lot closer to being able to support aliasing of
> UIDs between different namespaces.
>

OK, I'll fix this up.

Actually an earlier version of this patch did use user_struct's but
I'd changed it to uids, because it's simpler. I didn't think about
this being contrary to the id namespaces thing.

Miklos

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

Subject: Re: [patch 7/8] allow unprivileged mounts
Posted by Miklos Szeredi on Sat, 21 Apr 2007 08:13:22 GMT

View Forum Message <> Reply to Message

> On Fri, 20 Apr 2007 12:25:39 +0200 Miklos Szeredi <miklos@szeredi.hu> 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.
>
> Practically speaking, is there any realistic likelihood that any filesystem
> apart from FUSE will ever use this?

V9FS people did express an interest in this. Yeah, I should've CC-ed
them, but forgot. Sorry.

Miklos

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

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

Subject: Re: [patch 8/8] allow unprivileged fuse mounts
Posted by Miklos Szeredi on Sat, 21 Apr 2007 08:16:02 GMT
View Forum Message <> Reply to Message

> > Use FS_SAFE for "fuse" fs type, but not for "fuseblk".
> >
> > FUSE was designed from the beginning to be safe for unprivileged
> > users. This has also been verified in practice over many years.
>
> How does FUSE do this?
>
> There are obvious cases like crafting a filesystem which has setuid executables
> or world-writeable device nodes or whatever. I'm sure there are lots of other
> cases.
>
> Where is FUSE's implementation of all this protection described?

Most of it is in Documentation/filesystems/fuse.txt, some of it is
code comments.

Miklos

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

Subject: Re: [patch 7/8] allow unprivileged mounts
Posted by Miklos Szeredi on Sat, 21 Apr 2007 08:23:27 GMT
View Forum Message <> Reply to Message

> >
> > > 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.
> >
> > Practically speaking, is there any realistic likelihood that any filesystem
> > apart from FUSE will ever use this?
>
> V9FS people did express an interest in this. Yeah, I should've CC-ed
> them, but forgot. Sorry.

And CIFS maybe. They also have an unprivileged mounting suid hack.
But I'm not very optimistic about CIFS, seeing some of the code,
that's in there.

Miklos

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

---

Subject: Re: [patch 2/8] allow unprivileged umount
Posted by akpm on Sat, 21 Apr 2007 08:36:22 GMT

View Forum Message <> Reply to Message

On Sat, 21 Apr 2007 10:09:42 +0200 Miklos Szeredi <miklos@szeredi.hu> wrote:

```
> > > +static bool permit_umount(struct vfsmount *mnt, int flags)
> > > +{
> > >
> > > ...
> > >
> > > + return mnt->mnt_uid == current->uid;
> > > +}
> >
> > Yes, this seems very wrong.  I'd have thought that comparing user_struct*'s
> > would get us a heck of a lot closer to being able to support aliasing of
> > UIDs between different namespaces.
> >
>
> OK, I'll fix this up.
>
> Actually an earlier version of this patch did use user_struct's but
> I'd changed it to uids, because it's simpler.
```

OK..

```
>  I didn't think about
> this being contrary to the id namespaces thing.
```

Well I was madly assuming that when serarate UID namespaces are in use, UID
42 in container A will have a different user_struct from UID 42 in
container B. I'd suggest that we provoke an opinion from Eric & co before
you do work on this.

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

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

Subject: Re: [patch 2/8] allow unprivileged umount
Posted by ebiederm on Sat, 21 Apr 2007 12:53:58 GMT
View Forum Message <> Reply to Message

Andrew Morton <akpm@linux-foundation.org> writes:

```
> On Sat, 21 Apr 2007 10:09:42 +0200 Miklos Szeredi <miklos@szeredi.hu> wrote:
>
>> > > +static bool permit_umount(struct vfsmount *mnt, int flags)
>> > > +{
>> > >
>> > > ...
>> > >
>> > > + return mnt->mnt_uid == current->uid;
>> > > +}
>> >
>> > Yes, this seems very wrong.  I'd have thought that comparing user_struct*'s
>> > would get us a heck of a lot closer to being able to support aliasing of
>> > UIDs between different namespaces.
>> >
>>
>> OK, I'll fix this up.
>>
>> Actually an earlier version of this patch did use user_struct's but
>> I'd changed it to uids, because it's simpler.
>
> OK..
>
>>  I didn't think about
>> this being contrary to the id namespaces thing.
>
> Well I was madly assuming that when serarate UID namespaces are in use, UID
> 42 in container A will have a different user_struct from UID 42 in
> container B.  I'd suggest that we provoke an opinion from Eric & co before
> you do work on this.
```

That is what I what I have been thinking as well, storing a user
struct on each mount point seems sane, plus it allows per user mount
rlimits which is major plus. Especially since we seem to be doing
accounting only for user mounts a per user rlimit seems good.

To get the user we should be user fs_uid as HPA suggested.

Finally I'm pretty certain the capability we should care about in
this context is CAP_SETUID. Instead of CAP_SYS_ADMIN.

If we have CAP_SETUID we can become which ever user owns the mount,
and the root user in a container needs this, so he can run login
programs. So changing the appropriate super user checks from
CAP_SYS_ADMIN to CAP_SETUID I think is the right thing todo.

With the CAP_SETUID thing handled I'm not currently seeing any adverse
implications to using this in containers.

Ok. Now that I have a reasonable approximation of the 10,000 foot
view now to see how the patches match up.

Eric

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

---

Subject: Re: [patch 1/8] add user mounts to the kernel
Posted by ebiederm on Sat, 21 Apr 2007 13:14:23 GMT

View Forum Message <> Reply to Message

Miklos Szeredi <miklos@szeredi.hu> writes:

> From: Miklos Szeredi <mszeredi@suse.cz>
>
> Add ownership information to mounts.
>
> 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
> real user id 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.

I half agree, and as an initial approximation this works.
Ultimately we should be at the point that for mount propagation
that we copy the owner of the from the owner of our parent mount
at the propagation destination.

Mount propagation semantics are a major pain.

> 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.

Ok. While I generally agree with the concept this can be simplified some
more.

Eric

```
> Signed-off-by: Miklos Szeredi <mszeredi@suse.cz>
> ---
>
> Index: linux/fs/namespace.c
> ===================================================================
> --- linux.orig/fs/namespace.c 2007-04-20 11:55:02.000000000 +0200
> +++ linux/fs/namespace.c 2007-04-20 11:55:05.000000000 +0200
> @@ -227,6 +227,13 @@ static struct vfsmount *skip_mnt_tree(st
>   return p;
>  }
>
> +static void set_mnt_user(struct vfsmount *mnt)
> +{
> + BUG_ON(mnt->mnt_flags & MNT_USER);
> + mnt->mnt_uid = current->uid;
```

This should be based on fsuid. Unless I'm completely confused.

I think getting the mount user from current when we do mount
propagation could be a problem. In particular I'm pretty
certain in the mount propagation case the mount should be owned
by the user who owns the destination mount that is above us.

```
> + mnt->mnt_flags |= MNT_USER;
> +}
> +
>  static struct vfsmount *clone_mnt(struct vfsmount *old, struct dentry *root,
>       int flag)
>  {
> @@ -241,6 +248,11 @@ static struct vfsmount *clone_mnt(struct
>    mnt->mnt_mountpoint = mnt->mnt_root;
>    mnt->mnt_parent = mnt;
>
> +  /* don't copy the MNT_USER flag */
> +  mnt->mnt_flags &= ~MNT_USER;
> +  if (flag & CL_SETUSER)
> +   set_mnt_user(mnt);
> +
>    if (flag & CL_SLAVE) {
>     list_add(&mnt->mnt_slave, &old->mnt_slave_list);
>     mnt->mnt_master = old;
> @@ -403,6 +415,8 @@ static int show_vfsmnt(struct seq_file *
>    if (mnt->mnt_flags & fs_infop->flag)
>     seq_puts(m, fs_infop->str);
>   }
> + if (mnt->mnt_flags & MNT_USER)
> +  seq_printf(m, ",user=%i", mnt->mnt_uid);
```

How about making the test "if (mnt->mnt_user != &root_user)"

```
>   if (mnt->mnt_sb->s_op->show_options)
>    err = mnt->mnt_sb->s_op->show_options(m, mnt);
>   seq_puts(m, " 0 0\n");
> @@ -920,8 +934,9 @@ static int do_change_type(struct nameida
>  /*
>   * do loopback mount.
>   */
> -static int do_loopback(struct nameidata *nd, char *old_name, int recurse)
> +static int do_loopback(struct nameidata *nd, char *old_name, int flags)
>  {
> + int clone_flags;
>   struct nameidata old_nd;
>   struct vfsmount *mnt = NULL;
>   int err = mount_is_safe(nd);
> @@ -941,11 +956,12 @@ static int do_loopback(struct nameidata
>   if (!check_mnt(nd->mnt) || !check_mnt(old_nd.mnt))
>    goto out;
>
> + clone_flags = (flags & MS_SETUSER) ? CL_SETUSER : 0;
>   err = -ENOMEM;
> - if (recurse)
> -  mnt = copy_tree(old_nd.mnt, old_nd.dentry, 0);
> + if (flags & MS_REC)
> +  mnt = copy_tree(old_nd.mnt, old_nd.dentry, clone_flags);
>   else
> -  mnt = clone_mnt(old_nd.mnt, old_nd.dentry, 0);
> +  mnt = clone_mnt(old_nd.mnt, old_nd.dentry, clone_flags);
>
>   if (!mnt)
>    goto out;
> @@ -987,8 +1003,11 @@ static int do_remount(struct nameidata *
>
>   down_write(&sb->s_umount);
>   err = do_remount_sb(sb, flags, data, 0);
> - if (!err)
> + if (!err) {
>    nd->mnt->mnt_flags = mnt_flags;
> +  if (flags & MS_SETUSER)
> +   set_mnt_user(nd->mnt);
> + }
>   up_write(&sb->s_umount);
>   if (!err)
>    security_sb_post_remount(nd->mnt, flags, data);
> @@ -1093,10 +1112,13 @@ 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);
>  }
>
> @@ -1127,7 +1149,8 @@ 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;
>
> @@ -1447,7 +1470,7 @@ 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/include/linux/fs.h
> ===================================================================
> --- linux.orig/include/linux/fs.h 2007-04-20 11:55:02.000000000 +0200
> +++ linux/include/linux/fs.h 2007-04-20 11:55:05.000000000 +0200
> @@ -123,6 +123,7 @@ extern int dir_notify_enable;
>  #define MS_SLAVE (1<<19) /* change to slave */
>  #define MS_SHARED (1<<20) /* change to shared */
>  #define MS_RELATIME (1<<21) /* Update atime relative to mtime/ctime. */
> +#define MS_SETUSER (1<<22) /* set mnt_uid to current user */
```

If we unconditionally use the fsuid I think we can get away without
this flag.

```
>  #define MS_ACTIVE (1<<30)
>  #define MS_NOUSER (1<<31)
>
> Index: linux/include/linux/mount.h
> ===================================================================
> --- linux.orig/include/linux/mount.h 2007-04-20 11:55:02.000000000 +0200
> +++ linux/include/linux/mount.h 2007-04-20 11:55:05.000000000 +0200
> @@ -30,6 +30,7 @@ struct mnt_namespace;
>  #define MNT_RELATIME 0x20
>
>  #define MNT_SHRINKABLE 0x100
> +#define MNT_USER 0x200
```

If we assign a user to all mount points and root gets to own the
initial set of mounts then we don't need the internal MNT_USER
flag.

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

Can we please make this a user struct.   That requires a bit of
reference counting but it has uid namespace benefits as well
as making it easy to implement per user mount rlimits.

```
>  };
>
>  static inline struct vfsmount *mntget(struct vfsmount *mnt)
> Index: linux/fs/pnode.h
> ===================================================================
> --- linux.orig/fs/pnode.h 2007-04-20 11:55:02.000000000 +0200
> +++ linux/fs/pnode.h 2007-04-20 11:55:05.000000000 +0200
> @@ -22,6 +22,7 @@
>  #define CL_COPY_ALL   0x04
>  #define CL_MAKE_SHARED   0x08
>  #define CL_PROPAGATION   0x10
> +#define CL_SETUSER  0x20


>  static inline void set_mnt_shared(struct vfsmount *mnt)
>  {
>
> --
```

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

Subject: Re: [patch 2/8] allow unprivileged umount
Posted by ebiederm on Sat, 21 Apr 2007 13:29:51 GMT
View Forum Message <> Reply to Message

Miklos Szeredi <miklos@szeredi.hu> writes:

```
> From: Miklos Szeredi <mszeredi@suse.cz>
>
> The owner doesn't need sysadmin capabilities to call umount().
>
> Similar behavior as umount(8) on mounts having "user=UID" option in
> /etc/mtab.  The difference is that umount also checks /etc/fstab,
> presumably to exclude another mount on the same mountpoint.
>
> Signed-off-by: Miklos Szeredi <mszeredi@suse.cz>
> ---
>
> Index: linux/fs/namespace.c
> ===================================================================
> --- linux.orig/fs/namespace.c 2007-04-20 11:55:05.000000000 +0200
> +++ linux/fs/namespace.c 2007-04-20 11:55:06.000000000 +0200
> @@ -659,6 +659,25 @@ static int do_umount(struct vfsmount *mn
>  }
>
>  /*
> + * 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 (!(mnt->mnt_flags & MNT_USER))
> +  return false;
> +
> + if (flags & MNT_FORCE)
> +  return false;
> +
> + return mnt->mnt_uid == current->uid;
> +}
```

I think this should be:

```
static bool permit_umount(struct vfsmount *mnt, int flags)
{
 if ((mnt->mnt_uid != current->fsuid) && !capable(CAP_SETUID))
        return false;

 if ((flags & MNT_FORCE) && !capable(CAP_SYS_ADMIN))
  return false;

 return true;
}
```

I.e.
  MNT_USER gone.
  compare against fsuid.
  Only require setuid for unmounts unless force is specified.

  I suspect we can allow MNT_FORCE for non-privileged users
  as well if we can trust the filesystem.

```
> +/*
>   * Now umount can handle mount points as well as block devices.
>   * This is important for filesystems which use unnamed block devices.
>   *
> @@ -681,7 +700,7 @@ asmlinkage long sys_umount(char __user *
>    goto dput_and_out;
>
>   retval = -EPERM;
> - if (!capable(CAP_SYS_ADMIN))
> + if (!permit_umount(nd.mnt, flags))
>    goto dput_and_out;
>
>   retval = do_umount(nd.mnt, flags);
>
> --
```

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

Subject: Re: [patch 3/8] account user mounts
Posted by ebiederm on Sat, 21 Apr 2007 13:37:21 GMT
View Forum Message <> Reply to Message

Miklos Szeredi <miklos@szeredi.hu> writes:

```
> 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
```

Since each mount has a user can we just make this a per user rlimit?

If we are going to implement a sysctl at this point I think it should
be a global limit that doesn't care if who you are. Even root can
have recursive mounts that attempt to get out of control.

Also currently you are not checking the max_users. It looks like
you do this in a later patch but still it is a little strange to
allow user own mounts and have accounting but to not check the
limit at this state.

Eric

```
>
> Signed-off-by: Miklos Szeredi <mszeredi@suse.cz>
> ---
>
> Index: linux/include/linux/sysctl.h
> ===================================================================
> --- linux.orig/include/linux/sysctl.h 2007-04-20 11:55:02.000000000 +0200
> +++ linux/include/linux/sysctl.h 2007-04-20 11:55:07.000000000 +0200
> @@ -818,6 +818,8 @@ enum
>   FS_AIO_NR=18, /* current system-wide number of aio requests */
>   FS_AIO_MAX_NR=19, /* system-wide maximum number of aio requests */
>   FS_INOTIFY=20, /* inotify submenu */
> + FS_NR_USER_MOUNTS=21, /* int:current number of user mounts */
> + FS_MAX_USER_MOUNTS=22, /* int:maximum number of user mounts */
>   FS_OCFS2=988, /* ocfs2 */
>  };
>
> Index: linux/kernel/sysctl.c
> ===================================================================
> --- linux.orig/kernel/sysctl.c 2007-04-20 11:55:02.000000000 +0200
> +++ linux/kernel/sysctl.c 2007-04-20 11:55:07.000000000 +0200
> @@ -1063,6 +1063,22 @@ static ctl_table fs_table[] = {
>  #endif
>  #endif
>   {
> +  .ctl_name = FS_NR_USER_MOUNTS,
> +  .procname = "nr_user_mounts",
> +  .data  = &nr_user_mounts,
> +  .maxlen  = sizeof(int),
> +  .mode  = 0444,
> +  .proc_handler = &proc_dointvec,
> + },
> + {
> +  .ctl_name = FS_MAX_USER_MOUNTS,
> +  .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,
> Index: linux/Documentation/filesystems/proc.txt
> ===================================================================
> --- linux.orig/Documentation/filesystems/proc.txt 2007-04-20 11:55:02.000000000
> +0200
> +++ linux/Documentation/filesystems/proc.txt 2007-04-20 11:55:07.000000000 +0200
> @@ -923,6 +923,15 @@ reaches aio-max-nr then io_setup will fa
>  raising aio-max-nr does not result in the pre-allocation or re-sizing
>  of any kernel data structures.
>
> +nr_user_mounts and max_user_mounts
> +----------------------------------
> +
> +These represent the number of "user" mounts and the maximum number of
> +"user" mounts respectively.  User mounts may be created by
> +unprivileged users.  User mounts may also be created with sysadmin
> +privileges on behalf of a user, in which case nr_user_mounts may
> +exceed max_user_mounts.
> +
>  2.2 /proc/sys/fs/binfmt_misc - Miscellaneous binary formats
>  -----------------------------------------------------------
>
> Index: linux/fs/namespace.c
> ===================================================================
> --- linux.orig/fs/namespace.c 2007-04-20 11:55:06.000000000 +0200
> +++ linux/fs/namespace.c 2007-04-20 11:55:07.000000000 +0200
> @@ -39,6 +39,9 @@ static int hash_mask __read_mostly, hash
>  static struct kmem_cache *mnt_cache __read_mostly;
>  static struct rw_semaphore namespace_sem;
>
> +int nr_user_mounts;
> +int max_user_mounts = 1024;
> +
>  /* /sys/fs */
>  decl_subsys(fs, NULL, NULL);
>  EXPORT_SYMBOL_GPL(fs_subsys);
> @@ -227,11 +230,30 @@ static struct vfsmount *skip_mnt_tree(st
>   return p;
>  }
>
> +static void dec_nr_user_mounts(void)
> +{
> + spin_lock(&vfsmount_lock);
> + nr_user_mounts--;
> + spin_unlock(&vfsmount_lock);
> +}
> +
>  static void set_mnt_user(struct vfsmount *mnt)
>  {
>   BUG_ON(mnt->mnt_flags & MNT_USER);
>   mnt->mnt_uid = current->uid;
>   mnt->mnt_flags |= MNT_USER;
> + spin_lock(&vfsmount_lock);
> + nr_user_mounts++;
> + spin_unlock(&vfsmount_lock);
> +}
> +
> +static void clear_mnt_user(struct vfsmount *mnt)
> +{
> + if (mnt->mnt_flags & MNT_USER) {
> +  mnt->mnt_uid = 0;
> +  mnt->mnt_flags &= ~MNT_USER;
> +  dec_nr_user_mounts();
> + }
>  }
>
>  static struct vfsmount *clone_mnt(struct vfsmount *old, struct dentry *root,
> @@ -283,6 +305,7 @@ static inline void __mntput(struct vfsmo
>  {
>   struct super_block *sb = mnt->mnt_sb;
>   dput(mnt->mnt_root);
> + clear_mnt_user(mnt);
>   free_vfsmnt(mnt);
>   deactivate_super(sb);
>  }
> @@ -1023,6 +1046,7 @@ static int do_remount(struct nameidata *
>   down_write(&sb->s_umount);
>   err = do_remount_sb(sb, flags, data, 0);
>   if (!err) {
> +  clear_mnt_user(nd->mnt);
>    nd->mnt->mnt_flags = mnt_flags;
>    if (flags & MS_SETUSER)
>     set_mnt_user(nd->mnt);
> Index: linux/include/linux/fs.h
> ===================================================================
> --- linux.orig/include/linux/fs.h 2007-04-20 11:55:05.000000000 +0200
> +++ linux/include/linux/fs.h 2007-04-20 11:55:07.000000000 +0200
> @@ -50,6 +50,9 @@ extern struct inodes_stat_t inodes_stat;
>
>  extern int leases_enable, lease_break_time;
>
> +extern int nr_user_mounts;
> +extern int max_user_mounts;
> +
>  #ifdef CONFIG_DNOTIFY
>  extern int dir_notify_enable;
>  #endif
>
> --
```

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

Subject: Re: [patch 4/8] propagate error values from clone_mnt
Posted by ebiederm on Sat, 21 Apr 2007 13:40:20 GMT
View Forum Message <> Reply to Message

Miklos Szeredi <miklos@szeredi.hu> writes:

> 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.

At a quick skim this looks ok.

Eric

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

Subject: Re: [patch 5/8] allow unprivileged bind mounts
Posted by ebiederm on Sat, 21 Apr 2007 14:00:00 GMT

View Forum Message <> Reply to Message

Miklos Szeredi <miklos@szeredi.hu> writes:

> From: Miklos Szeredi <mszeredi@suse.cz>
>
> Allow bind mounts to unprivileged users if the following conditions
> are met:
>
>   - mountpoint is not a symlink or special file

Why? This sounds like a left over from when we were checking permissions.

>   - 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.

So in principle I agree, but in detail I disagree.

capable(CAP_SETUID) should be required to leave MNT_NOSUID clear.
capable(CAP_MKNOD) should be required to leave MNT_NODEV clear.

I.e. We should not special case this as a user mount but rather
simply check to see if the user performing the mount has the appropriate
capabilities to allow the flags.

How we properly propagate MNT_NOSUID and MNT_NODEV in the context of a
user id namespace is still a puzzle to me. Because to the user capability
should theoretically at least be namespace local. However until we
get to the user id namespace we don't have that problem.

Eric

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

```
>
> Index: linux/fs/namespace.c
> ===================================================================
> --- linux.orig/fs/namespace.c 2007-04-20 11:55:09.000000000 +0200
> +++ linux/fs/namespace.c 2007-04-20 11:55:10.000000000 +0200
> @@ -237,11 +237,30 @@ static void dec_nr_user_mounts(void)
>   spin_unlock(&vfsmount_lock);
>  }
>
> -static void set_mnt_user(struct vfsmount *mnt)
> +static int reserve_user_mount(void)
> +{
> + int err = 0;
> + spin_lock(&vfsmount_lock);
> + if (nr_user_mounts >= max_user_mounts && !capable(CAP_SYS_ADMIN))
> +  err = -EPERM;
> + else
> +  nr_user_mounts++;
> + spin_unlock(&vfsmount_lock);
> + return err;
> +}
> +
> +static void __set_mnt_user(struct vfsmount *mnt)
>  {
>   BUG_ON(mnt->mnt_flags & MNT_USER);
>   mnt->mnt_uid = current->uid;
>   mnt->mnt_flags |= MNT_USER;
> + if (!capable(CAP_SYS_ADMIN))
> +  mnt->mnt_flags |= MNT_NOSUID | MNT_NODEV;
> +}
> +
> +static void set_mnt_user(struct vfsmount *mnt)
> +{
> + __set_mnt_user(mnt);
>   spin_lock(&vfsmount_lock);
>   nr_user_mounts++;
>   spin_unlock(&vfsmount_lock);
> @@ -260,9 +279,16 @@ static struct vfsmount *clone_mnt(struct
>      int flag)
>  {
>   struct super_block *sb = old->mnt_sb;
> - struct vfsmount *mnt = alloc_vfsmnt(old->mnt_devname);
> + struct vfsmount *mnt;
> +
> + if (flag & CL_SETUSER) {
> +  int err = reserve_user_mount();
> +  if (err)
> +   return ERR_PTR(err);
> + }
> + mnt = alloc_vfsmnt(old->mnt_devname);
>   if (!mnt)
> -  return ERR_PTR(-ENOMEM);
> +  goto alloc_failed;
>
>   mnt->mnt_flags = old->mnt_flags;
>   atomic_inc(&sb->s_active);
> @@ -274,7 +300,7 @@ 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);
> @@ -299,6 +325,11 @@ 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)
> @@ -745,22 +776,29 @@ 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 or special file
> + * - mountpoint is in a mount owned by the user
> + */
> +static bool permit_mount(struct nameidata *nd, int *flags)
>  {
> + struct inode *inode = nd->dentry->d_inode;
> +
>   if (capable(CAP_SYS_ADMIN))
> -  return 0;
> - return -EPERM;
> -#ifdef notyet
> - if (S_ISLNK(nd->dentry->d_inode->i_mode))
> -  return -EPERM;
> - if (nd->dentry->d_inode->i_mode & S_ISVTX) {
> -  if (current->uid != nd->dentry->d_inode->i_uid)
> -   return -EPERM;
> - }
> - if (vfs_permission(nd, MAY_WRITE))
> -  return -EPERM;
> - return 0;
> -#endif
> +  return true;
> +
> + if (!S_ISDIR(inode->i_mode) && !S_ISREG(inode->i_mode))
> +  return false;
> +
> + if (!(nd->mnt->mnt_flags & MNT_USER))
> +  return false;
> +
> + if (nd->mnt->mnt_uid != current->uid)
> +  return false;
> +
> + *flags |= MS_SETUSER;
> + return true;
>  }
```

Can't this just be:

```
static bool permit_mount(struct nameidata *nd, uid_t *mnt_uid)
{
 *mnt_uid = current->fsuid;

 if ((nd->mnt->mnt_uid != current->fsuid) && !capable(CAP_SETUID))
         return false;

 return true;
}
```

```
>
>  static int lives_below_in_same_fs(struct dentry *d, struct dentry *dentry)
> @@ -981,9 +1019,10 @@ static int do_loopback(struct nameidata
>   int clone_flags;
>   struct nameidata old_nd;
>   struct vfsmount *mnt = NULL;
> - int err = mount_is_safe(nd);
> - if (err)
> -  return err;
> + int err;
> +
> + if (!permit_mount(nd, &flags))
> +  return -EPERM;
```

>   if (!old_name || !*old_name)
>   return -EINVAL;
>   err = path_lookup(old_name, LOOKUP_FOLLOW, &old_nd);
>
> --

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

Subject: Re: [patch 7/8] allow unprivileged mounts
Posted by ebiederm on Sat, 21 Apr 2007 14:10:18 GMT
View Forum Message <> Reply to Message

Andrew Morton <akpm@linux-foundation.org> writes:

> On Fri, 20 Apr 2007 12:25:39 +0200 Miklos Szeredi <miklos@szeredi.hu> 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.
>
> Practically speaking, is there any realistic likelihood that any filesystem
> apart from FUSE will ever use this?

Also potentially some of the kernel virtual filesystems. /proc should be safe already. If you don't have any kind of backing store this problem gets easier.

With unprivileged users allowed to create mounts the utility of kernel functionality exported as filesystems goes up quite a bit. We are not plan9 but this is the last bottleneck in allowing the everything is a filesystem paradigm from being really usable in linux.

Eric

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

Subject: Re: [patch 8/8] allow unprivileged fuse mounts
Posted by ebiederm on Sat, 21 Apr 2007 14:18:47 GMT
View Forum Message <> Reply to Message

Miklos Szeredi <miklos@szeredi.hu> writes:

```
> From: Miklos Szeredi <mszeredi@suse.cz>
>
> Use FS_SAFE for "fuse" fs type, but not for "fuseblk".
>
> FUSE was designed from the beginning to be safe for unprivileged
> users.  This has also been verified in practice over many years.  In
> addition unprivileged mounts require the parent mount to be owned by
> the user, which is more strict than the current userspace policy.
>
> This will enable future installations to remove the suid-root
> fusermount utility.
>
> Don't require the "user_id=" and "group_id=" options for unprivileged
> mounts, but if they are present, verify them for sanity.
>
> Disallow the "allow_other" option for unprivileged mounts.
>
> Signed-off-by: Miklos Szeredi <mszeredi@suse.cz>
> ---
>
> Index: linux/fs/fuse/inode.c
> ===================================================================
> --- linux.orig/fs/fuse/inode.c 2007-04-20 11:55:01.000000000 +0200
> +++ linux/fs/fuse/inode.c 2007-04-20 11:55:14.000000000 +0200
> @@ -311,6 +311,19 @@ static int parse_fuse_opt(char *opt, str
>   d->max_read = ~0;
>   d->blksize = 512;
>
> + /*
> +  * For unprivileged mounts use current uid/gid.  Still allow
> +  * "user_id" and "group_id" options for compatibility, but
> +  * only if they match these values.
> +  */
> + if (!capable(CAP_SYS_ADMIN)) {
> +  d->user_id = current->uid;
> +  d->user_id_present = 1;
> +  d->group_id = current->gid;
> +  d->group_id_present = 1;
> +
> + }
```

CAP_SETUID is the appropriate capability...

This is not a dimension we have not fully explored.
What is the problem with a user controlled mount having different
uid and gid values.

Yes they map into different users but how is this a problem.
The only problem that I can recall is the historic chown problem
where you could give files to other users and mess up their quotas.

Or is the problem other users writing to this user controlled
filesystem?

Eric

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

Subject: Re: [patch 7/8] allow unprivileged mounts
Posted by Jan Engelhardt on Sat, 21 Apr 2007 15:43:28 GMT

View Forum Message <> Reply to Message

On Apr 21 2007 08:10, Eric W. Biederman 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.
>>
>> Practically speaking, is there any realistic likelihood that any filesystem
>> apart from FUSE will ever use this?
>
\>Also potentially some of the kernel virtual filesystems. /proc should
\>be safe already. If you don't have any kind of backing store this problem
\>gets easier.

tmpfs!

Jan
--

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

Subject: Re: [patch 7/8] allow unprivileged mounts
Posted by ebiederm on Sat, 21 Apr 2007 16:57:10 GMT

View Forum Message <> Reply to Message

Jan Engelhardt <jengelh@linux01.gwdg.de> writes:

> On Apr 21 2007 08:10, Eric W. Biederman 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.
>>>
>>> Practically speaking, is there any realistic likelihood that any filesystem
>>> apart from FUSE will ever use this?
>>
>>Also potentially some of the kernel virtual filesystems. /proc should
>>be safe already. If you don't have any kind of backing store this problem
>>gets easier.
>
> tmpfs!

tmpfs is a possible problem because it can consume lots of ram/swap. Which
is why it has limits on the amount of space it can consume. Those are set as
mount options as I recall. Which means that we would need to do something
different with respect to limits before tmpfs could become safe for
an untrusted user to mount.

Still it's close.

Eric

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

Subject: Re: [patch 7/8] allow unprivileged mounts
Posted by Jan Engelhardt on Sat, 21 Apr 2007 17:10:55 GMT

View Forum Message <> Reply to Message

On Apr 21 2007 10:57, Eric W. Biederman wrote:
>
>> tmpfs!
>
>tmpfs is a possible problem because it can consume lots of ram/swap.
>Which is why it has limits on the amount of space it can consume.

Users can gobble up all RAM and swap already today. (Unless they are
confined into an rlimit, which, in most systems, is not the case.)
And in case /dev/shm exists, they can already fill it without running
into an rlimit early.

>Those are set as mount options as I recall. Which means that we
>would need to do something different with respect to limits before
>tmpfs could become safe for an untrusted user to mount.
>
>Still it's close.

Jan
--

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

Subject: Re: [patch 7/8] allow unprivileged mounts
Posted by ebiederm on Sat, 21 Apr 2007 21:00:16 GMT
View Forum Message <> Reply to Message

Jan Engelhardt <jengelh@linux01.gwdg.de> writes:

> On Apr 21 2007 10:57, Eric W. Biederman wrote:
>>
>>> tmpfs!
>>
>>tmpfs is a possible problem because it can consume lots of ram/swap.
>>Which is why it has limits on the amount of space it can consume.
>
> Users can gobble up all RAM and swap already today. (Unless they are
> confined into an rlimit, which, in most systems, is not the case.)
> And in case /dev/shm exists, they can already fill it without running
> into an rlimit early.

There are systems that care about rlimits and there is strong intersection
between caring about rlimits and user mounts. Although I do agree that
it looks like we have gotten lazy with the default mount options for
/dev/shm.

Going a little farther any filesystem that is safe to put on a usb
stick and mount automatically should ultimately be safe for unprivileged
mounts as well.

So it looks to me like ultimately most of the common filesystems will actually be safe for non-privileged mounting.

Regardless this looks like an important discussion as soon as we have the
glitches out of the non-privileged mount code.

Eric

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

Subject: Re: [patch 7/8] allow unprivileged mounts
Posted by ebiederm on Sat, 21 Apr 2007 21:33:25 GMT

View Forum Message <> Reply to Message

Andi Kleen <andi@firstfloor.org> writes:

> Andrew Morton <akpm@linux-foundation.org> writes:
>
>> On Fri, 20 Apr 2007 12:25:39 +0200 Miklos Szeredi <miklos@szeredi.hu> 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.
>>
>> Practically speaking, is there any realistic likelihood that any filesystem
>> apart from FUSE will ever use this?
>
> If it worked for mount --bind for any fs I could see uses of this. I haven't
> thought
> through the security implications though, so it might not work.

Binding a directory that you have access to in other was is essentially
the same thing as a symlink. So there are no real security implications
there. The only problem case I can think of is removal media that you
want to remove but someone has made a bind mount to. But that is
essentially the same case as opening a file so there are no new
real issues. Although our diagnostic tools will likely fall behind
for a bit.

We handle the security implications by assigning an owner to all mounts
and only allowing you to add additional mounts on top of a mount you
already own.

If you have the right capabilities you can create a mount owned by another user.

For a new mount if you don't have the appropriate capabilities nodev
and nosuid will be forced.

Initial super block creation is a lot more delicate so we need the
FS_SAFE flag, to know that the kernel is prepared to deal with the
crazy things that a hostile user space is prepared to do.

Eric

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

Subject: Re: [patch 7/8] allow unprivileged mounts
Posted by Andi Kleen on Sat, 21 Apr 2007 22:06:51 GMT

View Forum Message <> Reply to Message

Andrew Morton <akpm@linux-foundation.org> writes:

> On Fri, 20 Apr 2007 12:25:39 +0200 Miklos Szeredi <miklos@szeredi.hu> 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.
>
> Practically speaking, is there any realistic likelihood that any filesystem
> apart from FUSE will ever use this?

If it worked for mount --bind for any fs I could see uses of this. I haven't thought
through the security implications though, so it might not work.

-Andi

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

Subject: Re: [patch 7/8] allow unprivileged mounts
Posted by Shaya Potter on Sun, 22 Apr 2007 00:46:25 GMT

View Forum Message <> Reply to Message

Andrew Morton wrote:
> On Fri, 20 Apr 2007 12:25:39 +0200 Miklos Szeredi <miklos@szeredi.hu> 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.
>
> Practically speaking, is there any realistic likelihood that any filesystem
> apart from FUSE will ever use this?

Would it be interesting to support mounting of external file systems (be
it USB, NFS or whatever) in a way that automatically forces it to ignore
suid and devices (which are already mount time options)?   The question
I guess is, how much do you gain over a setuid program (hack?) that can
handle this?

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

Subject: Re: [patch 2/8] allow unprivileged umount
Posted by Miklos Szeredi on Sun, 22 Apr 2007 06:47:31 GMT

View Forum Message <> Reply to Message

> > On Sat, 21 Apr 2007 10:09:42 +0200 Miklos Szeredi <miklos@szeredi.hu> wrote:
> >
> >> > > +static bool permit_umount(struct vfsmount *mnt, int flags)
> >> > > +{
> >> > >
> >> > > ...
> >> > >
> >> > > + return mnt->mnt_uid == current->uid;
> >> > > +}
> >> >
> >> > Yes, this seems very wrong. I'd have thought that comparing user_struct*'s
> >> > would get us a heck of a lot closer to being able to support aliasing of
> >> > UIDs between different namespaces.
> >> >
> >>
> >> OK, I'll fix this up.
> >>
> >> Actually an earlier version of this patch did use user_struct's but
> >> I'd changed it to uids, because it's simpler.
> >
> > OK..

> >
> >> I didn't think about
> >> this being contrary to the id namespaces thing.
> >
> > Well I was madly assuming that when serarate UID namespaces are in use, UID
> > 42 in container A will have a different user_struct from UID 42 in
> > container B. I'd suggest that we provoke an opinion from Eric & co before
> > you do work on this.
>
> That is what I what I have been thinking as well,

Does this mean, that containers will need this? Or that you don't
know yet?

> storing a user struct on each mount point seems sane, plus it allows
> per user mount rlimits which is major plus. Especially since we
> seem to be doing accounting only for user mounts a per user rlimit
> seems good.

I'm not against per-user rlimits for mounts, but I'd rather do this
later...

> To get the user we should be user fs_uid as HPA suggested.

fsuid is exclusively used for checking file permissions, which we
don't do here anymore. So while it could be argued, that mount() _is_
a filesystem operation, it is really a different sort of filesystem
operation than the rest.

OTOH it wouldn't hurt to use fsuid instead of ruid...

> Finally I'm pretty certain the capability we should care about in
> this context is CAP_SETUID. Instead of CAP_SYS_ADMIN.
>
> If we have CAP_SETUID we can become which ever user owns the mount,
> and the root user in a container needs this, so he can run login
> programs. So changing the appropriate super user checks from
> CAP_SYS_ADMIN to CAP_SETUID I think is the right thing todo.

That's a flawed logic. If you want to mount as a specific user, and
you have CAP_SETUID, then just use set*uid() and then mount().

Changing the capability check for mount() would break the userspace
ABI.

Miklos

____
Containers mailing list

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

Subject: Re: [patch 1/8] add user mounts to the kernel
Posted by Miklos Szeredi on Sun, 22 Apr 2007 07:02:08 GMT
View Forum Message <> Reply to Message

> > The MNT_USER flag is not copied on any kind of mount cloning:
> > namespace creation, binding or propagation.
>
> I half agree, and as an initial approximation this works.
> Ultimately we should be at the point that for mount propagation
> that we copy the owner of the from the owner of our parent mount
> at the propagation destination.

Yes, that sounds the most sane.

Ram, what do you think?

> > + if (mnt->mnt_flags & MNT_USER)
> > + seq_printf(m, ",user=%i", mnt->mnt_uid);
> How about making the test "if (mnt->mnt_user != &root_user)"

We don't want to treat root_user special. That's what capabilities
were invented for.

> > Index: linux/include/linux/fs.h
> > ===
> > --- linux.orig/include/linux/fs.h 2007-04-20 11:55:02.000000000 +0200
> > +++ linux/include/linux/fs.h 2007-04-20 11:55:05.000000000 +0200
> > @@ -123,6 +123,7 @@ extern int dir_notify_enable;
> > #define MS_SLAVE (1<<19) /* change to slave */
> > #define MS_SHARED (1<<20) /* change to shared */
> > #define MS_RELATIME (1<<21) /* Update atime relative to mtime/ctime. */
> > +#define MS_SETUSER (1<<22) /* set mnt_uid to current user */
>
> If we unconditionally use the fsuid I think we can get away without
> this flag.

That coudl work if we wouldn't have to worry about breaking the user
interface. As it is, we cannot be sure, that existing callers of
mount(2) don't have fsuid set to some random value.

> > #define MNT_SHRINKABLE 0x100
> > +#define MNT_USER 0x200
>
> If we assign a user to all mount points and root gets to own the

> initial set of mounts then we don't need the internal MNT_USER
> flag.

I think we do want to treat "owned" mounts special, rather than
treating user=0 mounts special.

> > +
> > + uid_t mnt_uid;   /* owner of the mount */
>
> Can we please make this a user struct.   That requires a bit of
> reference counting but it has uid namespace benefits as well
> as making it easy to implement per user mount rlimits.

OK, can you ellaborate, what the uid namespace benifits are?

Miklos

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

---

Subject: Re: [patch 2/8] allow unprivileged umount
Posted by Miklos Szeredi on Sun, 22 Apr 2007 07:05:58 GMT
View Forum Message <> Reply to Message

>   I suspect we can allow MNT_FORCE for non-privileged users
>   as well if we can trust the filesystem.

I don't think so. MNT_FORCE has side effects on the superblock. So a
user shouldn't be able to force an unmount on a bind mount s/he did,
but there's no problem with allowing plain/lazy unmounts.

We could possibly allow MNT_FORCE, for FS_SAFE filesystems.

Miklos

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

---

Subject: Re: [patch 2/8] allow unprivileged umount
Posted by ebiederm on Sun, 22 Apr 2007 07:09:54 GMT
View Forum Message <> Reply to Message

Miklos Szeredi <miklos@szeredi.hu> writes:

> Does this mean, that containers will need this? Or that you don't
> know yet?

The uid namespace is something we have to handle carefully and we
have not decided on the final design.

What is clear is that all permission checks will need to become
either (uid namspace, uid) tuple comparisons. Or struct user
pointer comparisons. To see if we are talking about the same
uid.

So the eventual uid namespace combined with the possibility
for rlimits if we use struct user *. See to make using a struct
user a clear win.

>> storing a user struct on each mount point seems sane, plus it allows
>> per user mount rlimits which is major plus. Especially since we
>> seem to be doing accounting only for user mounts a per user rlimit
>> seems good.
>
> I'm not against per-user rlimits for mounts, but I'd rather do this
> later...

Then let's add a non-discriminate limit. Instead of a limit that
applies only to root.

>> To get the user we should be user fs_uid as HPA suggested.
>
> fsuid is exclusively used for checking file permissions, which we
> don't do here anymore. So while it could be argued, that mount() _is_
> a filesystem operation, it is really a different sort of filesystem
> operation than the rest.
>
> OTOH it wouldn't hurt to use fsuid instead of ruid...

Yes. I may be confused but I'm pretty certain we want either
the fsuid or the euid to be the mount owner. ruid just looks wrong.
The fsuid is a special case of the effective uid. Which is who
we should perform operations as. Unless I'm just confused.

>> Finally I'm pretty certain the capability we should care about in
>> this context is CAP_SETUID. Instead of CAP_SYS_ADMIN.
>>
>> If we have CAP_SETUID we can become which ever user owns the mount,
>> and the root user in a container needs this, so he can run login
>> programs. So changing the appropriate super user checks from
>> CAP_SYS_ADMIN to CAP_SETUID I think is the right thing todo.

>
> That's a flawed logic. If you want to mount as a specific user, and
> you have CAP_SETUID, then just use set*uid() and then mount().

Totally agreed for mount.

> Changing the capability check for mount() would break the userspace
> ABI.

Sorry I apparently wasn't clear. CAP_SETUID should be the capability
check for umount.

Hopefully my other more detail replies helped with this.

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

---

Subject: Re: [patch 3/8] account user mounts
Posted by Miklos Szeredi on Sun, 22 Apr 2007 07:10:49 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
>
> Since each mount has a user can we just make this a per user rlimit?
>
> If we are going to implement a sysctl at this point I think it should
> be a global limit that doesn't care if who you are. Even root can
> have recursive mounts that attempt to get out of control.

Recursive bind mounts are done carefully enough, so they don't get out
of control.

Recursive mount propagations can get out of control. But root can
shoot itself in the foot any number of ways, and it's not for the
kernel to police that.

> Also currently you are not checking the max_users. It looks like

> you do this in a later patch but still it is a little strange to
> allow user own mounts and have accounting but to not check the
> limit at this state.

Yeah, but at this stage user mounts are not yet allowed, so this is
safe.

Miklos

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

Subject: Re: [patch 5/8] allow unprivileged bind mounts
Posted by Miklos Szeredi on Sun, 22 Apr 2007 07:19:45 GMT

View Forum Message <> Reply to Message

> > From: Miklos Szeredi <mszeredi@suse.cz>
> >
> > Allow bind mounts to unprivileged users if the following conditions
> > are met:
> >
> >   - mountpoint is not a symlink or special file
>
> Why? This sounds like a left over from when we were checking permissions.

Hmm, yes. Don't know. Maybe only the symlink check.

Bind mounts of directory over non-directy, and vica versa are already
excluded, even for root.

> >   - 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.
>
> So in principle I agree, but in detail I disagree.
>
> capable(CAP_SETUID) should be required to leave MNT_NOSUID clear.
> capable(CAP_MKNOD) should be required to leave MNT_NODEV clear.
>
> I.e. We should not special case this as a user mount but rather
> simply check to see if the user performing the mount has the appropriate
> capabilities to allow the flags.

Sounds sane. Will fix.

Miklos

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

---

Subject: Re: [patch 8/8] allow unprivileged fuse mounts
Posted by Miklos Szeredi on Sun, 22 Apr 2007 07:22:44 GMT
View Forum Message <> Reply to Message

> > + /*
> > + * For unprivileged mounts use current uid/gid. Still allow
> > + * "user_id" and "group_id" options for compatibility, but
> > + * only if they match these values.
> > + */
> > + if (!capable(CAP_SYS_ADMIN)) {
> > + d->user_id = current->uid;
> > + d->user_id_present = 1;
> > + d->group_id = current->gid;
> > + d->group_id_present = 1;
> > +
> > + }
>
> CAP_SETUID is the appropriate capability...
>
> This is not a dimension we have not fully explored.
> What is the problem with a user controlled mount having different
> uid and gid values.
>
> Yes they map into different users but how is this a problem.
> The only problem that I can recall is the historic chown problem
> where you could give files to other users and mess up their quotas.
>
> Or is the problem other users writing to this user controlled
> filesystem?

Yes. Or even just a suid process trying to access the user controlled filesystem. See Documentation/filesystems/fuse.txt for the gory details.

Eric, thanks for the detailed review :)

Miklos

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

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

Subject: Re: [patch 2/8] allow unprivileged umount
Posted by Miklos Szeredi on Sun, 22 Apr 2007 07:32:34 GMT
View Forum Message <> Reply to Message

> > Does this mean, that containers will need this? Or that you don't
> > know yet?
>
> The uid namespace is something we have to handle carefully and we
> have not decided on the final design.
>
> What is clear is that all permission checks will need to become
> either (uid namspace, uid) tuple comparisons. Or struct user
> pointer comparisons. To see if we are talking about the same
> uid.
>
> So the eventual uid namespace combined with the possibility
> for rlimits if we use struct user *. See to make using a struct
> user a clear win.

OK, if we don't yet know, I'd rather leave this for later. It will be
trivial to change to user_struct if we want per-user rlimits.

> >> storing a user struct on each mount point seems sane, plus it allows
> >> per user mount rlimits which is major plus. Especially since we
> >> seem to be doing accounting only for user mounts a per user rlimit
> >> seems good.
> >
> > I'm not against per-user rlimits for mounts, but I'd rather do this
> > later...
>
> Then let's add a non-discriminate limit. Instead of a limit that
> applies only to root.

See reply to relevant patch.

> >> To get the user we should be user fs_uid as HPA suggested.
> >
> > fsuid is exclusively used for checking file permissions, which we
> > don't do here anymore. So while it could be argued, that mount() _is_
> > a filesystem operation, it is really a different sort of filesystem
> > operation than the rest.
> >
> > OTOH it wouldn't hurt to use fsuid instead of ruid...
>
> Yes. I may be confused but I'm pretty certain we want either

> the fsuid or the euid to be the mount owner. ruid just looks wrong.
> The fsuid is a special case of the effective uid. Which is who
> we should perform operations as. Unless I'm just confused.

Definitely not euid. Euid is the one which is effective, i.e. it will
basically always be zero for a privileged mount().

Ruid is the one which is returned by getuid(). If a user execs a
suid-root program, then ruid will be the id of the user, while euid
will be zero.

> >> Finally I'm pretty certain the capability we should care about in
> >> this context is CAP_SETUID. Instead of CAP_SYS_ADMIN.
> >>
> >> If we have CAP_SETUID we can become which ever user owns the mount,
> >> and the root user in a container needs this, so he can run login
> >> programs. So changing the appropriate super user checks from
> >> CAP_SYS_ADMIN to CAP_SETUID I think is the right thing todo.
> >
> > That's a flawed logic. If you want to mount as a specific user, and
> > you have CAP_SETUID, then just use set*uid() and then mount().
>
> Totally agreed for mount.
>
> > Changing the capability check for mount() would break the userspace
> > ABI.
>
> Sorry I apparently wasn't clear. CAP_SETUID should be the capability
> check for umount.

The argument applies to umount as well. For compatibility, we _need_
the CAP_SYS_ADMIN check. And if program has CAP_SETUID but not
CAP_SYS_ADMIN, it can just set the id to the mount owner before
calling umount.

Miklos

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

Subject: Re: [patch 1/8] add user mounts to the kernel
Posted by ebiederm on Sun, 22 Apr 2007 07:43:44 GMT
View Forum Message <> Reply to Message

Miklos Szeredi <miklos@szeredi.hu> writes:

>> > The MNT_USER flag is not copied on any kind of mount cloning:
>> > namespace creation, binding or propagation.
>>
>> I half agree, and as an initial approximation this works.
>> Ultimately we should be at the point that for mount propagation
>> that we copy the owner of the from the owner of our parent mount
>> at the propagation destination.
>
> Yes, that sounds the most sane.
>
> Ram, what do you think?
>
>> > + if (mnt->mnt_flags & MNT_USER)
>> > + seq_printf(m, ",user=%i", mnt->mnt_uid);
>> How about making the test "if (mnt->mnt_user != &root_user)"
>
> We don't want to treat root_user special. That's what capabilities
> were invented for.

For the print statement? What ever it is minor.

>> > Index: linux/include/linux/fs.h
>> > ===
>> > --- linux.orig/include/linux/fs.h 2007-04-20 11:55:02.000000000 +0200
>> > +++ linux/include/linux/fs.h 2007-04-20 11:55:05.000000000 +0200
>> > @@ -123,6 +123,7 @@ extern int dir_notify_enable;
>> > #define MS_SLAVE (1<<19) /* change to slave */
>> > #define MS_SHARED (1<<20) /* change to shared */
>> > #define MS_RELATIME (1<<21) /* Update atime relative to mtime/ctime. */
>> > +#define MS_SETUSER (1<<22) /* set mnt_uid to current user */
>>
>> If we unconditionally use the fsuid I think we can get away without
>> this flag.
>
> That coudl work if we wouldn't have to worry about breaking the user
> interface. As it is, we cannot be sure, that existing callers of
> mount(2) don't have fsuid set to some random value.

If we can get away without an extra flag it would really be
preferable.

In the container case we have an interesting and very common
scenario struct user *our_user != &root_user. our_user->uid == 0.

I.e. The root in the what is the container but not the root
of the entire system.

So I want to minimize the changes needed to existing programs.
Now if all we have to do is specify MS_SETUSER when root a
user with CAP_SETUID is setting up a mount as a user other
then himself then I don't much care. If we have to call MS_SETUSER
as unprivileged users I will have to modify mount binaries to work
differently inside and outside of containers.

Further there is only one or two versions of mount in widespread
use on linux, and unless you do something special fsuid == euid.
So the chance of fsuid set to some random value is pretty low.
So yes I think we can be 99.9% certain that existing callers of
mount(2) don't have fsuid set to some random value just by
inspecting the code of mount(1).

>> > #define MNT_SHRINKABLE 0x100
>> > +#define MNT_USER 0x200
>>
>> If we assign a user to all mount points and root gets to own the
>> initial set of mounts then we don't need the internal MNT_USER
>> flag.
>
> I think we do want to treat "owned" mounts special, rather than
> treating user=0 mounts special.

I don't think we should treat any mount special and all mounts
should be owned.

>> > +
>> > + uid_t mnt_uid;   /* owner of the mount */
>>
>> Can we please make this a user struct.   That requires a bit of
>> reference counting but it has uid namespace benefits as well
>> as making it easy to implement per user mount rlimits.
>
> OK, can you ellaborate, what the uid namespace benifits are?

In the uid namespace the comparison is simpler as are the propagations
rules. Basically if you use a struct user you will never need to
care about a uid namespace. If you don't we will have to tear through
this code another time.

Plus like I was mentioning earlier. If we do have a struct user there
implementing per user mount rlimits becomes trivial.

Eric
____
Containers mailing list

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

Subject: Re: [patch 3/8] account user mounts
Posted by ebiederm on Sun, 22 Apr 2007 07:49:22 GMT
View Forum Message <> Reply to Message

Miklos Szeredi <miklos@szeredi.hu> writes:

>> > 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
>>
>> Since each mount has a user can we just make this a per user rlimit?
>>
>> If we are going to implement a sysctl at this point I think it should
>> be a global limit that doesn't care if who you are. Even root can
>> have recursive mounts that attempt to get out of control.
>
> Recursive bind mounts are done carefully enough, so they don't get out
> of control.
>
> Recursive mount propagations can get out of control. But root can
> shoot itself in the foot any number of ways, and it's not for the
> kernel to police that.

Yes. It is.

This is mostly about removing special cases.

We routinely have limits on resources that are global and apply
to root along with every one else. Root can change them but
they still apply to root. Things like the number of inodes
in the system or the total number of files.

Since it is perfectly possible to do a per user rlimit at this stage
in the design. I contend that either:
- We implement a per user rlimit of mounts.
- We implement a global limit on mounts.

No other case makes sense. The previous objections were at least in
part because the limit only applied to user mounts but the name of

the limit did not apply to user mounts.

>> Also currently you are not checking the max_users. It looks like
>> you do this in a later patch but still it is a little strange to
>> allow user own mounts and have accounting but to not check the
>> limit at this state.
>
> Yeah, but at this stage user mounts are not yet allowed, so this is
> safe.

Eric

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

---

Subject: Re: [patch 1/8] add user mounts to the kernel
Posted by Miklos Szeredi on Sun, 22 Apr 2007 08:05:05 GMT

View Forum Message <> Reply to Message

> >> > + if (mnt->mnt_flags & MNT_USER)
> >> > + seq_printf(m, ",user=%i", mnt->mnt_uid);
> >> How about making the test "if (mnt->mnt_user != &root_user)"
> >
> > We don't want to treat root_user special. That's what capabilities
> > were invented for.
>
> For the print statement? What ever it is minor.

It is a user interface, not a print statement. Your suggested change
would be vetoed by any number of people.

So either we have all mounts having owners, AND have /proc/mounts add
"user=0" to all mounts. While I don't _think_ this would actually
break userspace, it would definitely make people complain.

The other choice is what the current patchset does: is to have
"legacy" mounts without owners, and "new generation" mounts with
owners having "user=UID" in /proc/mounts, regardless of the value of
UID.

> So I want to minimize the changes needed to existing programs.
> Now if all we have to do is specify MS_SETUSER when root a
> user with CAP_SETUID is setting up a mount as a user other
> then himself then I don't much care. If we have to call MS_SETUSER
> as unprivileged users

You don't. Unprivileged mounts _imply_ MS_SETUSER.

> >> > +
> >> > + uid_t mnt_uid;   /* owner of the mount */
> >>
> >> Can we please make this a user struct.   That requires a bit of
> >> reference counting but it has uid namespace benefits as well
> >> as making it easy to implement per user mount rlimits.
> >
> > OK, can you ellaborate, what the uid namespace benifits are?
>
> In the uid namespace the comparison is simpler as are the propagations
> rules. Basically if you use a struct user you will never need to
> care about a uid namespace. If you don't we will have to tear through
> this code another time.

Well, OK. I'll do the user_struct thing then.

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

---

Subject: Re: [patch 3/8] account user mounts
Posted by Miklos Szeredi on Sun, 22 Apr 2007 08:08:49 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
> >>
> >> Since each mount has a user can we just make this a per user rlimit?
> >>
> >> If we are going to implement a sysctl at this point I think it should
> >> be a global limit that doesn't care if who you are. Even root can
> >> have recursive mounts that attempt to get out of control.
> >
> > Recursive bind mounts are done carefully enough, so they don't get out
> > of control.
> >

> > Recursive mount propagations can get out of control. But root can
> > shoot itself in the foot any number of ways, and it's not for the
> > kernel to police that.
>
> Yes. It is.
>
> This is mostly about removing special cases.
>
> We routinely have limits on resources that are global and apply
> to root along with every one else. Root can change them but
> they still apply to root. Things like the number of inodes
> in the system or the total number of files.

There's no max_inodes any more. As for max_files:

get_empty_filp():

```
 /*
  * Privileged users can go above max_files
  */
 if (get_nr_files() >= files_stat.max_files && !capable(CAP_SYS_ADMIN)) {
```

Miklos

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

Subject: Re: [patch 7/8] allow unprivileged mounts
Posted by Miklos Szeredi on Sun, 22 Apr 2007 08:19:54 GMT

View Forum Message <> Reply to Message

> > On Apr 21 2007 10:57, Eric W. Biederman wrote:
> >>
> >>> tmpfs!
> >>
> >>tmpfs is a possible problem because it can consume lots of ram/swap.
> >>Which is why it has limits on the amount of space it can consume.
> >
> > Users can gobble up all RAM and swap already today. (Unless they are
> > confined into an rlimit, which, in most systems, is not the case.)
> > And in case /dev/shm exists, they can already fill it without running
> > into an rlimit early.
>
> There are systems that care about rlimits and there is strong intersection
> between caring about rlimits and user mounts. Although I do agree that
> it looks like we have gotten lazy with the default mount options for

> /dev/shm.
>
> Going a little farther any filesystem that is safe to put on a usb
> stick and mount automatically should ultimately be safe for unprivileged
> mounts as well.

Actually, that's not as simple.

For the usb stick or cdrom you need physical access to the machine.
And once you have that you basically have full control over the system
anyway.

But with block filesystems, the user would still need access to the
device (currently kernel doesn't even check this I think).

So it may make sense to mark all block based filesystems safe, and
defer permission checking to user access on the block device.

But the safe flag is still needed for filesystems, which don't have
such an additional access checking, such as network filesystems.

Miklos

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

Subject: Re: [patch 1/8] add user mounts to the kernel
Posted by Miklos Szeredi on Sun, 22 Apr 2007 16:22:40 GMT

View Forum Message <> Reply to Message

> >> > +
> >> > + uid_t mnt_uid;   /* owner of the mount */
> >>
> >> Can we please make this a user struct.   That requires a bit of
> >> reference counting but it has uid namespace benefits as well
> >> as making it easy to implement per user mount rlimits.
> >
> > OK, can you ellaborate, what the uid namespace benifits are?
>
> In the uid namespace the comparison is simpler as are the propagations
> rules. Basically if you use a struct user you will never need to
> care about a uid namespace.

I tried to implement it but got stuck on this: fsuid doesn't have a
user_struct in task_struct (yet), so we'd now have to convert
current->fsuid to a user_struct. This can be done with alloc_uid(),

but this can fail, bringing in extra error handling complexity.

Also we'd have to compare current->fsuid with a user_struct, which we
don't yet know how will actually be done in the future.

So it seems, we still have to care about the uid namespace, at least
if fsuid is preferred to ruid.

Anyway, here's a patch fixing the other things you brought up, and
which I agree with. Does this look OK?

Thanks,
Miklos

```
Index: linux/fs/namespace.c
===================================================================
--- linux.orig/fs/namespace.c 2007-04-22 17:48:18.000000000 +0200
+++ linux/fs/namespace.c 2007-04-22 18:19:51.000000000 +0200
@@ -252,10 +252,12 @@ static int reserve_user_mount(void)
 static void __set_mnt_user(struct vfsmount *mnt)
 {
  BUG_ON(mnt->mnt_flags & MNT_USER);
- mnt->mnt_uid = current->uid;
+ mnt->mnt_uid = current->fsuid;
  mnt->mnt_flags |= MNT_USER;
- if (!capable(CAP_SYS_ADMIN))
-  mnt->mnt_flags |= MNT_NOSUID | MNT_NODEV;
+ 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)
@@ -725,10 +727,10 @@ static bool permit_umount(struct vfsmoun
  if (!(mnt->mnt_flags & MNT_USER))
   return false;

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

- return mnt->mnt_uid == current->uid;
+ return mnt->mnt_uid == current->fsuid;
 }

 /*
```

```
@@ -792,13 +794,13 @@ static bool permit_mount(struct nameidat
  if (type && !(type->fs_flags & FS_SAFE))
   return false;

- if (!S_ISDIR(inode->i_mode) && !S_ISREG(inode->i_mode))
+ if (S_ISLNK(inode->i_mode))
   return false;

  if (!(nd->mnt->mnt_flags & MNT_USER))
   return false;

- if (nd->mnt->mnt_uid != current->uid)
+ if (nd->mnt->mnt_uid != current->fsuid)
   return false;

  *flags |= MS_SETUSER;
```

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

Subject: Re: [patch 0/8] mount ownership and unprivileged mount syscall (v4)
Posted by Karel Zak on Wed, 25 Apr 2007 00:04:14 GMT

View Forum Message <> Reply to Message

On Fri, Apr 20, 2007 at 12:25:32PM +0200, Miklos Szeredi wrote:

> 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

The original userspace "user=" solution also implies the "noexec" option by default (you can override the default by "exec" option).

It means the kernel based solution is not fully compatible ;-(

Karel

--
Karel Zak <kzak@redhat.com>

Red Hat Czech s.r.o.
Purkynova 99/71, 612 45 Brno, Czech Republic
Reg.id: CZ27690016

____
Containers mailing list

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

Subject: Re: [patch 0/8] mount ownership and unprivileged mount syscall (v4)
Posted by ebiederm on Wed, 25 Apr 2007 01:04:36 GMT
View Forum Message <> Reply to Message

Karel Zak <kzak@redhat.com> writes:

> On Fri, Apr 20, 2007 at 12:25:32PM +0200, Miklos Szeredi wrote:
>> 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
>
> The original userspace "user=" solution also implies the "noexec"
> option by default (you can override the default by "exec" option).
>
> It means the kernel based solution is not fully compatible ;-(

Why noexec? Either it was a silly or arbitrary decision, or
our kernel design may be incomplete.

Now I can see not wanting to support executables if you are locking
down a system. The classic don't execute a program from a CD just because
the CD was stuck in the drive problem.

So I can see how executing code from an untrusted source could prevent
exploitation of other problems, and we certainly don't want to do it
automatically.

Eric

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

Subject: Re: [patch 0/8] mount ownership and unprivileged mount syscall (v4)
Posted by Miklos Szeredi on Wed, 25 Apr 2007 07:18:28 GMT
View Forum Message <> Reply to Message

> > 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
>
> The original userspace "user=" solution also implies the "noexec"
> option by default (you can override the default by "exec" option).

Unlike "nosuid" and "nodev", I don't think "noexec" has real security benefits.

> It means the kernel based solution is not fully compatible ;-(

Oh, I don't think that matters. For traditional /etc/fstab based user mounts, mount(8) will have to remain suid-root, the kernel can't replace the fstab check.

In fact the latest patches don't even support these "legacy" user mounts too well: setting the owner of a mount gives not only umount privilege, but the ability to submount. This is not necessarily a good thing for these kinds of user mounts.

We could add a new "nosubmount" or similar flag, to prevent submounting, but that again would go against the simplicity of the current approach, so I'm not sure it's worth it.

Miklos

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

---

Subject: Re: [patch 0/8] mount ownership and unprivileged mount syscall (v4)
Posted by Karel Zak on Wed, 25 Apr 2007 09:23:56 GMT

View Forum Message <> Reply to Message

On Wed, Apr 25, 2007 at 09:18:28AM +0200, Miklos Szeredi wrote:
> > > 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
> >
> > The original userspace "user=" solution also implies the "noexec"
> > option by default (you can override the default by "exec" option).
>
> Unlike "nosuid" and "nodev", I don't think "noexec" has real security
> benefits.

Yes. I agree.

> > It means the kernel based solution is not fully compatible ;-(
>
> Oh, I don't think that matters. For traditional /etc/fstab based user
> mounts, mount(8) will have to remain suid-root, the kernel can't
> replace the fstab check.

Ok, it makes sense. You're right that for the mount(8) is more
important the fstab check.

Please, prepare a mount(8) patch -- with the patch it will be more
clear.

> We could add a new "nosubmount" or similar flag, to prevent
> submounting, but that again would go against the simplicity of the
> current approach, so I'm not sure it's worth it.

The "nosubmount" is probably good idea.

The patches seem much better in v4. I'm fun for the feature in the
kernel (and also for every change that makes mtab more and more
obsolete :-).

Karel

>
> Miklos
> -
> 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

--
Karel Zak <kzak@redhat.com>

Red Hat Czech s.r.o.
Purkynova 99/71, 612 45 Brno, Czech Republic
Reg.id: CZ27690016

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