

---

Subject: Re: [patch 6/9] unprivileged mounts: allow unprivileged mounts

Posted by [serue](#) on Mon, 14 Jan 2008 22:58:26 GMT

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

---

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

> From: Miklos Szeredi <[mszeredi@suse.cz](mailto:mszeredi@suse.cz)>  
>  
> Define a new fs flag FS\_SAFE, which denotes, that unprivileged mounting of  
> this filesystem may not constitute a security problem.  
>  
> Since most filesystems haven't been designed with unprivileged mounting in  
> mind, a thorough audit is needed before setting this flag.  
>  
> For "safe" filesystems also allow unprivileged forced unmounting.  
>  
> Move subtype handling from do\_kern\_mount() into do\_new\_mount(). All  
> other callers are kernel-internal and do not need subtype support.  
>  
> Signed-off-by: Miklos Szeredi <[mszeredi@suse.cz](mailto:mszeredi@suse.cz)>

This patch itself doesn't assign FS\_SAFE to any filesystems, so presuming that there is such a thing as an fs safe for users to mount, and/or users sign their systems away through a sysctl, this patch in itself appears right.

Acked-by: Serge Hallyn <[serue@us.ibm.com](mailto:serue@us.ibm.com)>

```
> ---  
>  
> Index: linux/fs/namespace.c  
> ======  
> --- linux.orig/fs/namespace.c 2008-01-03 21:20:11.000000000 +0100  
> +++ linux/fs/namespace.c 2008-01-03 21:21:06.000000000 +0100  
> @@ -960,14 +960,16 @@ static bool is_mount_owner(struct vfsmou  
> /*  
> * umount is permitted for  
> * - sysadmin  
> - * - mount owner, if not forced umount  
> + * - mount owner  
> + * o if not forced umount,  
> + * o if forced umount, and filesystem is "safe"  
> */  
> static bool permit_umount(struct vfsmount *mnt, int flags)  
> {  
>     if (capable(CAP_SYS_ADMIN))  
>         return true;  
>  
> - if (flags & MNT_FORCE)
```

```

> + if ((flags & MNT_FORCE) && !(mnt->mnt_sb->s_type->fs_flags & FS_SAFE))
>   return false;
>
>   return is_mount_owner(mnt, current->fsuid);
> @@ -1025,13 +1027,17 @@ asmlinkage long sys_oldumount(char __use
>   * - mountpoint is not a symlink
>   * - mountpoint is in a mount owned by the user
>   */
> -static bool permit_mount(struct nameidata *nd, int *flags)
> +static bool permit_mount(struct nameidata *nd, struct file_system_type *type,
> +  int *flags)
> {
>   struct inode *inode = nd->path.dentry->d_inode;
>
>   if (capable(CAP_SYS_ADMIN))
>     return true;
>
> + if (type && !(type->fs_flags & FS_SAFE))
> +  return false;
> +
>   if (S_ISLNK(inode->i_mode))
>     return false;
>
> @@ -1285,7 +1291,7 @@ static int do_loopback(struct nameidata
>   struct vfsmount *mnt = NULL;
>   int err;
>
> - if (!permit_mount(nd, &flags))
> + if (!permit_mount(nd, NULL, &flags))
>   return -EPERM;
>   if (!old_name || !*old_name)
>     return -EINVAL;
> @@ -1466,30 +1472,76 @@ out:
>   return err;
> }
>
> +static struct vfsmount *fs_set_subtype(struct vfsmount *mnt, const char *fstype)
> +{
> + int err;
> + const char *subtype = strchr(fstype, '.');
> + if (subtype) {
> + subtype++;
> + err = -EINVAL;
> + if (!subtype[0])
> + goto err;
> + } else
> + subtype = "";
> +

```

```

> + mnt->mnt_sb->s_subtype = kstrdup(subtype, GFP_KERNEL);
> + err = -ENOMEM;
> + if (!mnt->mnt_sb->s_subtype)
> +     goto err;
> + return mnt;
> +
> +
> + err:
> + mntput(mnt);
> + return ERR_PTR(err);
> +}
> +
> /*
> * create a new mount for userspace and request it to be added into the
> * namespace's tree
> */
> -static int do_new_mount(struct nameidata *nd, char *type, int flags,
> +static int do_new_mount(struct nameidata *nd, char *fstype, int flags,
>     int mnt_flags, char *name, void *data)
> {
> + int err;
> struct vfsmount *mnt;
> + struct file_system_type *type;
>
> - if (!type || !memchr(type, 0, PAGE_SIZE))
> + if (!fstype || !memchr(fstype, 0, PAGE_SIZE))
>     return -EINVAL;
>
> - /* we need capabilities... */
> - if (!capable(CAP_SYS_ADMIN))
> -     return -EPERM;
> -
> - mnt = do_kern_mount(type, flags & ~MS_SETUSER, name, data);
> - if (IS_ERR(mnt))
> + type = get_fs_type(fstype);
> + if (!type)
> +     return -ENODEV;
> +
> + err = -EPERM;
> + if (!permit_mount(nd, type, &flags))
> +     goto out_put_filesystem;
> +
> + if (flags & MS_SETUSER) {
> +     err = reserve_user_mount();
> +     if (err)
> +         goto out_put_filesystem;
> + }
> +
> + mnt = vfs_kern_mount(type, flags & ~MS_SETUSER, name, data);

```

```

> + if (!IS_ERR(mnt) && (type->fs_flags & FS_HAS_SUBTYPE) &&
> +     !mnt->mnt_sb->s_subtype)
> +     mnt = fs_set_subtype(mnt, fstype);
> + put_filesystem(type);
> + if (IS_ERR(mnt)) {
> +     if (flags & MS_SETUSER)
> +         dec_nr_user_mounts();
>     return PTR_ERR(mnt);
> +
>
>     if (flags & MS_SETUSER)
> -     set_mnt_user(mnt);
> +     __set_mnt_user(mnt);
>
>     return do_add_mount(mnt, nd, mnt_flags, NULL);
> +
> + out_put_filesystem:
> + put_filesystem(type);
> + return err;
> }
>
> /*
> @@ -1520,7 +1572,7 @@ int do_add_mount(struct vfsmount *newmnt
>     if (S_ISLNK(newmnt->mnt_root->d_inode->i_mode))
>         goto unlock;
>
> - /* MNT_USER was set earlier */
> + /* some flags may have been set earlier */
>     newmnt->mnt_flags |= mnt_flags;
>     if ((err = graft_tree(newmnt, nd)))
>         goto unlock;
> Index: linux/include/linux/fs.h
> =====
> --- linux.orig/include/linux/fs.h 2008-01-03 21:15:35.000000000 +0100
> +++ linux/include/linux/fs.h 2008-01-03 21:21:06.000000000 +0100
> @@ -96,6 +96,7 @@ extern int dir_notify_enable;
> #define FS_REQUIRES_DEV 1
> #define FS_BINARY_MOUNTDATA 2
> #define FS_HAS_SUBTYPE 4
> +#define FS_SAFE 8 /* Safe to mount by unprivileged users */
> #define FS_REVAL_DOT 16384 /* Check the paths ".", ".." for staleness */
> #define FS_RENAME_DOES_D_MOVE 32768 /* FS will handle d_move()
>     * during rename() internally.
> Index: linux/fs/super.c
> =====
> --- linux.orig/fs/super.c 2008-01-02 21:42:10.000000000 +0100
> +++ linux/fs/super.c 2008-01-03 21:21:06.000000000 +0100
> @@ -906,29 +906,6 @@ out:

```

```

>
> EXPORT_SYMBOL_GPL(vfs_kern_mount);
>
> -static struct vfsmount *fs_set_subtype(struct vfsmount *mnt, const char *fstype)
> -{
> - int err;
> - const char *subtype = strchr(fstype, '.');
> - if (subtype) {
> - subtype++;
> - err = -EINVAL;
> - if (!subtype[0])
> - goto err;
> - } else
> - subtype = "";
> -
> - mnt->mnt_sb->s_subtype = kstrdup(subtype, GFP_KERNEL);
> - err = -ENOMEM;
> - if (!mnt->mnt_sb->s_subtype)
> - goto err;
> - return mnt;
> -
> - err:
> - mntput(mnt);
> - return ERR_PTR(err);
> -}
> -
> struct vfsmount *
> do_kern_mount(const char *fstype, int flags, const char *name, void *data)
> {
> @@ -937,9 +914,6 @@ do_kern_mount(const char *fstype, int fl
> if (!type)
> return ERR_PTR(-ENODEV);
> mnt = vfs_kern_mount(type, flags, name, data);
> - if (!IS_ERR(mnt) && (type->fs_flags & FS_HAS_SUBTYPE) &&
> - !mnt->mnt_sb->s_subtype)
> - mnt = fs_set_subtype(mnt, fstype);
> put_filesystem(type);
> return mnt;
> }
>
> --

```

---

Containers mailing list  
[Containers@lists.linux-foundation.org](mailto:Containers@lists.linux-foundation.org)  
<https://lists.linux-foundation.org/mailman/listinfo/containers>

---