Subject: [patch 5/9] unprivileged mounts: allow unprivileged bind mounts
Posted by Miklos Szeredi on Tue, 08 Jan 2008 11:35:07 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
- parent mount is owned by the user
- the number of user mounts is below the maximum

Unprivileged mounts imply MS_SETUSER, and will also have the "nosuid" and "nodev" mount flags set.

In particular, if mounting process doesn't have CAP_SETUID capability, then the "nosuid" flag will be added, and if it doesn't have CAP_MKNOD capability, then the "nodev" flag will be added.

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

Index: linux/fs/namespace.c
```
===================================================================
--- linux.orig/fs/namespace.c 2008-01-04 13:47:49.000000000 +0100
+++ linux/fs/namespace.c 2008-01-04 13:48:01.000000000 +0100
@@ -487,11 +487,34 @@ 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->fsuid;
```

```
  mnt->mnt_flags |= MNT_USER;
+
+ if (!capable(CAP_SETUID))
+   mnt->mnt_flags |= MNT_NOSUID;
+ if (!capable(CAP_MKNOD))
+   mnt->mnt_flags |= MNT_NODEV;
+}
+
+static void set_mnt_user(struct vfsmount *mnt)
+{
+ __set_mnt_user(mnt);
  spin_lock(&vfsmount_lock);
  nr_user_mounts++;
  spin_unlock(&vfsmount_lock);
@@ -510,10 +533,16 @@ 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);
@@ -525,7 +554,7 @@ 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);
@@ -550,6 +579,11 @@ 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)
@@ -986,22 +1020,26 @@ asmlinkage long sys_oldumount(char __use

 #endif

-static int mount_is_safe(struct nameidata *nd)
+/*
+ * Conditions for unprivileged mounts are:
+ * - mountpoint is not a symlink
+ * - mountpoint is in a mount owned by the user
+ */
+static bool permit_mount(struct nameidata *nd, int *flags)
 {
+ struct inode *inode = nd->path.dentry->d_inode;
+
  if (capable(CAP_SYS_ADMIN))
-  return 0;
- return -EPERM;
-#ifdef notyet
- if (S_ISLNK(nd->path.dentry->d_inode->i_mode))
-  return -EPERM;
- if (nd->path.dentry->d_inode->i_mode & S_ISVTX) {
-  if (current->uid != nd->path.dentry->d_inode->i_uid)
-   return -EPERM;
- }
- if (vfs_permission(nd, MAY_WRITE))
-  return -EPERM;
- return 0;
-#endif
+  return true;
+
+ if (S_ISLNK(inode->i_mode))
+  return false;
+
+ if (!is_mount_owner(nd->path.mnt, current->fsuid))
+  return false;
+
+ *flags |= MS_SETUSER;
+ return true;
 }

 static int lives_below_in_same_fs(struct dentry *d, struct dentry *dentry)
@@ -1245,9 +1283,10 @@ static int do_loopback(struct nameidata
```

```
  int clone_fl;
  struct nameidata old_nd;
  struct vfsmount *mnt = NULL;
- int err = mount_is_safe(nd);
- if (err)
-  return err;
+ int err;
+
+ if (!permit_mount(nd, &flags))
+  return -EPERM;
  if (!old_name || !*old_name)
   return -EINVAL;
  err = path_lookup(old_name, LOOKUP_FOLLOW, &old_nd);

--
```

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

---

Subject: Re: [patch 5/9] unprivileged mounts: allow unprivileged bind mounts
Posted by Dave Hansen on Tue, 08 Jan 2008 18:12:38 GMT
View Forum Message <> Reply to Message

On Tue, 2008-01-08 at 12:35 +0100, Miklos Szeredi wrote:

```
> +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;
> +}
```

Would -ENOSPC or -ENOMEM be a more descriptive error here?

-- Dave

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

Subject: Re: [patch 5/9] unprivileged mounts: allow unprivileged bind mounts
Posted by Dave Hansen on Tue, 08 Jan 2008 18:26:26 GMT
View Forum Message <> Reply to Message

On Tue, 2008-01-08 at 12:35 +0100, Miklos Szeredi wrote:
> @@ -510,10 +533,16 @@ 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);

I think there's a little race here. We could have several users racing to get to this point when nr_user_mounts==max_user_mounts-1. One user wins the race and gets their mount reserved. The others get the error out of reserve_user_mount(), and return.

But, the winner goes on to error out on some condition further down in clone_mnt() and never actually instantiates the mount.

Do you think this is a problem?

I think just about the one solution is to block new mounters on a sleepable lock until the race winner actually finishes their mount operation.

-- Dave

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

Subject: Re: [patch 5/9] unprivileged mounts: allow unprivileged bind mounts

Posted by Miklos Szeredi on Tue, 08 Jan 2008 19:08:34 GMT
View Forum Message <> Reply to Message

> On Tue, 2008-01-08 at 12:35 +0100, Miklos Szeredi wrote:
> > +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;
> > +}
>
> Would -ENOSPC or -ENOMEM be a more descriptive error here?

The logic behind EPERM, is that this failure is only for unprivileged
callers. ENOMEM is too specifically about OOM. It could be changed
to ENOSPC, ENFILE, EMFILE, or it could remain EPERM. What do others
think?

Miklos

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

Subject: Re: [patch 5/9] unprivileged mounts: allow unprivileged bind mounts
Posted by Dave Hansen on Tue, 08 Jan 2008 19:15:10 GMT
View Forum Message <> Reply to Message

On Tue, 2008-01-08 at 20:08 +0100, Miklos Szeredi wrote:
>
> The logic behind EPERM, is that this failure is only for unprivileged
> callers. ENOMEM is too specifically about OOM. It could be changed
> to ENOSPC, ENFILE, EMFILE, or it could remain EPERM. What do others
> think?

Since you're patching mount anyway, maybe you could add a little pointer
for people to go check the sysctl if they get a certain error code back
from here.

-- Dave

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

Subject: Re: [patch 5/9] unprivileged mounts: allow unprivileged bind mounts
Posted by Miklos Szeredi on Tue, 08 Jan 2008 19:21:35 GMT
View Forum Message <> Reply to Message

```
> > @@ -510,10 +533,16 @@ 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);
>
> I think there's a little race here.  We could have several users racing
> to get to this point when nr_user_mounts==max_user_mounts-1.  One user
> wins the race and gets their mount reserved.  The others get the error
> out of reserve_user_mount(), and return.
>
> But, the winner goes on to error out on some condition further down in
> clone_mnt() and never actually instantiates the mount.
>
> Do you think this is a problem?
```

For similar reasons as stated in the previous mail, I don't think this matters. If nr_user_mounts is getting remotely close to max_user_mounts, then something is wrong (or the max needs to be raised anyway).

Thanks for the review, Dave!

Miklos

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

Subject: Re: [patch 5/9] unprivileged mounts: allow unprivileged bind mounts
Posted by Szabolcs Szakacsits on Tue, 08 Jan 2008 20:44:20 GMT
View Forum Message <> Reply to Message

On Tue, 8 Jan 2008, Miklos Szeredi wrote:
> > On Tue, 2008-01-08 at 12:35 +0100, Miklos Szeredi wrote:
> > > +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;
> > > +}
> >
> > Would -ENOSPC or -ENOMEM be a more descriptive error here?
>
> The logic behind EPERM, is that this failure is only for unprivileged
> callers. ENOMEM is too specifically about OOM. It could be changed
> to ENOSPC, ENFILE, EMFILE, or it could remain EPERM. What do others
> think?

I think it would be important to log the non-trivial errors. Several
mount(8) hints to check for the reason by dmesg since it's already too
challanging to figure out what's exactly the problem by the errno value.
This could also prevent to mislead troubleshooters with the mount/sysctl
race.

 Szaka

--
NTFS-3G: http://ntfs-3g.org

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

Subject: Re: [patch 5/9] unprivileged mounts: allow unprivileged bind mounts
Posted by Jan Engelhardt on Wed, 09 Jan 2008 12:44:31 GMT
View Forum Message <> Reply to Message

On Jan 8 2008 20:08, Miklos Szeredi wrote:
>> On Tue, 2008-01-08 at 12:35 +0100, Miklos Szeredi wrote:
>> > +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;
>> > +}
>>
>> Would -ENOSPC or -ENOMEM be a more descriptive error here?
>
>The logic behind EPERM, is that this failure is only for unprivileged
>callers. ENOMEM is too specifically about OOM. It could be changed
>to ENOSPC, ENFILE, EMFILE, or it could remain EPERM. What do others
>think?

ENOSPC: No space remaining on device => 'wth'.
ENOMEM: I usually think of a userspace OOM (e.g. malloc'ed out all of your
32-bit address space on 32-bit processes)
EMFILE: "Too many open files"
ENFILE: "Too many open files in system".

ENFILE seems like a temporary winner among these four.

Back in the old days, when the number of mounts was limited in Linux,
what error value did it return? That one could be used.

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

Subject: Re: [patch 5/9] unprivileged mounts: allow unprivileged bind mounts
Posted by Karel Zak on Wed, 09 Jan 2008 13:25:45 GMT
View Forum Message <> Reply to Message

On Wed, Jan 09, 2008 at 01:45:09PM +0100, Jan Engelhardt wrote:
>

> On Jan 8 2008 20:08, Miklos Szeredi wrote:
> >> On Tue, 2008-01-08 at 12:35 +0100, Miklos Szeredi wrote:
> >> > +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;
> >> > +}
> >>
> >> Would -ENOSPC or -ENOMEM be a more descriptive error here?
> >
> >The logic behind EPERM, is that this failure is only for unprivileged
> >callers. ENOMEM is too specifically about OOM. It could be changed
> >to ENOSPC, ENFILE, EMFILE, or it could remain EPERM. What do others
> >think?
>
> ENOSPC: No space remaining on device => 'wth'.
> ENOMEM: I usually think of a userspace OOM (e.g. malloc'ed out all of your
> 32-bit address space on 32-bit processes)
> EMFILE: "Too many open files"
> ENFILE: "Too many open files in system".
>
> ENFILE seems like a temporary winner among these four.

I see "EMFILE", it's still supported by the latest mount(8).

> Back in the old days, when the number of mounts was limited in Linux,
> what error value did it return? That one could be used.

Copy & past from mount-0.99.2:

```
 /* Mount failed, complain, but don't die.  */
 switch (mnt_err)
   {
   case EPERM:
     if (geteuid() == 0)
error ("mount: mount point %s is not a directory", node);
     else
error ("mount: must be superuser to use mount");
     break;
   case EBUSY:
     error ("mount: wrong fs type, %s already mounted, %s busy, "
```

```
 "or other error", spec, node);
      break;
    case ENOENT:
      error ("mount: mount point %s does not exist", node); break;
    case ENOTDIR:
      error ("mount: mount point %s is not a directory", node); break;
    case EINVAL:
      error ("mount: %s not a mount point", spec); break;
    case EMFILE:
      error ("mount table full"); break;
    case EIO:
      error ("mount: %s: can't read superblock", spec); break;
    case ENODEV:
      error ("mount: fs type %s not supported by kernel", type); break;
    case ENOTBLK:
      error ("mount: %s is not a block device", spec); break;
    case ENXIO:
      error ("mount: %s is not a valid block device", spec); break;
    case EACCES:
      error ("mount: block device %s is not permitted on its filesystem", spec);
      break;
    default:
      error ("mount: %s", strerror (mnt_err)); break;
    }
```

Karel

--
Karel Zak <kzak@redhat.com>

_____

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

---

Subject: Re: [patch 5/9] unprivileged mounts: allow unprivileged bind mounts
Posted by Miklos Szeredi on Wed, 09 Jan 2008 13:31:59 GMT
View Forum Message <> Reply to Message

> case EMFILE:
> error ("mount table full"); break;

OK, we could go with EMFILE, but the message should be changed to
something like "maximum unprivileged mount count exceeded".

Miklos

_____

Subject: Re: [patch 5/9] unprivileged mounts: allow unprivileged bind mounts
Posted by serue on Thu, 10 Jan 2008 04:47:24 GMT
View Forum Message <> Reply to Message

Quoting Miklos Szeredi (miklos@szeredi.hu):
> From: Miklos Szeredi <mszeredi@suse.cz>
>
> Allow bind mounts to unprivileged users if the following conditions are met:
>
> - mountpoint is not a symlink
> - parent mount is owned by the user
> - the number of user mounts is below the maximum
>
> Unprivileged mounts imply MS_SETUSER, and will also have the "nosuid" and
> "nodev" mount flags set.
>
> In particular, if mounting process doesn't have CAP_SETUID capability,
> then the "nosuid" flag will be added, and if it doesn't have CAP_MKNOD
> capability, then the "nodev" flag will be added.

That little part by itself is really needed in order to make the ability
to remove CAP_MKNOD from a process tree's bounding set meaningful.
Else instead of creating /dev/hda1, the user can just mount a filesystem
with hda1 existing on it. (Which is why I was surprised when one day
I found this code missing :)

But of course I'm a fan of the patchset altogether. I plan to review in
more detail early next week, but since I liked the previous submission I
don't see myself having any complaints, so I'm glad to see the reviews
by others.

thanks,
-serge

> Signed-off-by: Miklos Szeredi <mszeredi@suse.cz>
> ---
>
> Index: linux/fs/namespace.c
> ===
> --- linux.orig/fs/namespace.c 2008-01-04 13:47:49.000000000 +0100
> +++ linux/fs/namespace.c 2008-01-04 13:48:01.000000000 +0100
> @@ -487,11 +487,34 @@ 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->fsuid;
>   mnt->mnt_flags |= MNT_USER;
> +
> + if (!capable(CAP_SETUID))
> +  mnt->mnt_flags |= MNT_NOSUID;
> + if (!capable(CAP_MKNOD))
> +  mnt->mnt_flags |= MNT_NODEV;
> +}
> +
> +static void set_mnt_user(struct vfsmount *mnt)
> +{
> + __set_mnt_user(mnt);
>   spin_lock(&vfsmount_lock);
>   nr_user_mounts++;
>   spin_unlock(&vfsmount_lock);
> @@ -510,10 +533,16 @@ 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);
> @@ -525,7 +554,7 @@ 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);
> @@ -550,6 +579,11 @@ 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)
> @@ -986,22 +1020,26 @@ asmlinkage long sys_oldumount(char __use
>
>  #endif
>
> -static int mount_is_safe(struct nameidata *nd)
> +/*
> + * Conditions for unprivileged mounts are:
> + * - mountpoint is not a symlink
> + * - mountpoint is in a mount owned by the user
> + */
> +static bool permit_mount(struct nameidata *nd, int *flags)
>  {
> + struct inode *inode = nd->path.dentry->d_inode;
> +
>   if (capable(CAP_SYS_ADMIN))
> -  return 0;
> - return -EPERM;
> -#ifdef notyet
> - if (S_ISLNK(nd->path.dentry->d_inode->i_mode))
> -  return -EPERM;
> - if (nd->path.dentry->d_inode->i_mode & S_ISVTX) {
> -  if (current->uid != nd->path.dentry->d_inode->i_uid)
```

> -   return -EPERM;
> - }
> - if (vfs_permission(nd, MAY_WRITE))
> - return -EPERM;
> - return 0;
> -#endif
> + return true;
> +
> + if (S_ISLNK(inode->i_mode))
> + return false;
> +
> + if (!is_mount_owner(nd->path.mnt, current->fsuid))
> + return false;
> +
> + *flags |= MS_SETUSER;
> + return true;
> }
>
> static int lives_below_in_same_fs(struct dentry *d, struct dentry *dentry)
> @@ -1245,9 +1283,10 @@ static int do_loopback(struct nameidata
>   int clone_fl;
>   struct nameidata old_nd;
>   struct vfsmount *mnt = NULL;
> - int err = mount_is_safe(nd);
> - if (err)
> - return err;
> + int err;
> +
> + if (!permit_mount(nd, &flags))
> + return -EPERM;
>   if (!old_name || !*old_name)
>   return -EINVAL;
>   err = path_lookup(old_name, LOOKUP_FOLLOW, &old_nd);
>
> --

_____

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

---

Subject: Re: [patch 5/9] unprivileged mounts: allow unprivileged bind mounts
Posted by serue on Mon, 14 Jan 2008 22:42:40 GMT
View Forum Message <> Reply to Message

Quoting Miklos Szeredi (miklos@szeredi.hu):
> From: Miklos Szeredi <mszeredi@suse.cz>
>

> Allow bind mounts to unprivileged users if the following conditions are met:
>
> - mountpoint is not a symlink
> - parent mount is owned by the user
> - the number of user mounts is below the maximum
>
> Unprivileged mounts imply MS_SETUSER, and will also have the "nosuid" and
> "nodev" mount flags set.
>
> In particular, if mounting process doesn't have CAP_SETUID capability,
> then the "nosuid" flag will be added, and if it doesn't have CAP_MKNOD
> capability, then the "nodev" flag will be added.
>
> Signed-off-by: Miklos Szeredi <mszeredi@suse.cz>

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

> ---
>
> Index: linux/fs/namespace.c
> ===
> --- linux.orig/fs/namespace.c 2008-01-04 13:47:49.000000000 +0100
> +++ linux/fs/namespace.c 2008-01-04 13:48:01.000000000 +0100
> @@ -487,11 +487,34 @@ 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->fsuid;
> mnt->mnt_flags |= MNT_USER;
> +
> + if (!capable(CAP_SETUID))

```
> +  mnt->mnt_flags |= MNT_NOSUID;
> + if (!capable(CAP_MKNOD))
> +  mnt->mnt_flags |= MNT_NODEV;
> +}
> +
> +static void set_mnt_user(struct vfsmount *mnt)
> +{
> + __set_mnt_user(mnt);
>   spin_lock(&vfsmount_lock);
>   nr_user_mounts++;
>   spin_unlock(&vfsmount_lock);
> @@ -510,10 +533,16 @@ 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);
> @@ -525,7 +554,7 @@ 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);
> @@ -550,6 +579,11 @@ 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)
> @@ -986,22 +1020,26 @@ asmlinkage long sys_oldumount(char __use
>
>  #endif
>
> -static int mount_is_safe(struct nameidata *nd)
> +/*
> + * Conditions for unprivileged mounts are:
> + * - mountpoint is not a symlink
> + * - mountpoint is in a mount owned by the user
> + */
> +static bool permit_mount(struct nameidata *nd, int *flags)
>  {
> + struct inode *inode = nd->path.dentry->d_inode;
> +
>   if (capable(CAP_SYS_ADMIN))
> -  return 0;
> - return -EPERM;
> -#ifdef notyet
> - if (S_ISLNK(nd->path.dentry->d_inode->i_mode))
> -  return -EPERM;
> - if (nd->path.dentry->d_inode->i_mode & S_ISVTX) {
> -  if (current->uid != nd->path.dentry->d_inode->i_uid)
> -   return -EPERM;
> - }
> - if (vfs_permission(nd, MAY_WRITE))
> -  return -EPERM;
> - return 0;
> -#endif
> +  return true;
> +
> + if (S_ISLNK(inode->i_mode))
> +  return false;
> +
> + if (!is_mount_owner(nd->path.mnt, current->fsuid))
> +  return false;
> +
> + *flags |= MS_SETUSER;
> + return true;
>  }
>
>  static int lives_below_in_same_fs(struct dentry *d, struct dentry *dentry)
> @@ -1245,9 +1283,10 @@ static int do_loopback(struct nameidata
>   int clone_fl;
>   struct nameidata old_nd;
>   struct vfsmount *mnt = NULL;
```

> - int err = mount_is_safe(nd);
> - if (err)
> - return err;
> + int err;
> +
> + if (!permit_mount(nd, &flags))
> + return -EPERM;
>   if (!old_name || !*old_name)
>   return -EINVAL;
>   err = path_lookup(old_name, LOOKUP_FOLLOW, &old_nd);
>
> --

_____

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