Subject: Re: [PATCH 9/9] userns: check user namespace for task->file uid equivalence checks
Posted by ebiederm on Fri, 18 Feb 2011 01:29:32 GMT
View Forum Message <> Reply to Message

"Serge E. Hallyn" <serge@hallyn.com> writes:

> Cheat for now and say all files belong to init_user_ns. Next
> step will be to let superblocks belong to a user_ns, and derive
> inode_userns(inode) from inode->i_sb->s_user_ns. Finally we'll
> introduce more flexible arrangements.

This looks good. I am a little worried that a concept like
inode_user_ns will imply that there is only ever one.

However this looks like a good place to start and it will only
be strange filesystems that implement a notion of permissions
that is namespace aware so I don't expect the generic code
needs to handle that case other than allowing the permission checks
to be overridden.

Acked-by: "Eric W. Biederman" <ebiederm@xmission.com>

>
> Changelog:
> Feb 15: make is_owner_or_cap take const struct inode
>
> Signed-off-by: Serge E. Hallyn <serge.hallyn@canonical.com>
> ---
> fs/inode.c | 17 ++++++++++++++++
> fs/namei.c | 20 +++++++++++++++++++-----
> include/linux/fs.h | 9 +++++++--
> 3 files changed, 39 insertions(+), 7 deletions(-)
>
> diff --git a/fs/inode.c b/fs/inode.c
> index da85e56..1930b45 100644
> --- a/fs/inode.c
> +++ b/fs/inode.c
> @@ -25,6 +25,7 @@
> #include <linux/async.h>
> #include <linux/posix_acl.h>
> #include <linux/ima.h>
> +#include <linux/cred.h>
>
> /*
>   * This is needed for the following functions:
> @@ -1722,3 +1723,19 @@ void inode_init_owner(struct inode *inode, const struct inode *dir,
>   inode->i_mode = mode;

> }
> EXPORT_SYMBOL(inode_init_owner);
> +
> +/*
> + * return 1 if current either has CAP_FOWNER to the
> + * file, or owns the file.
> + */
> +int is_owner_or_cap(const struct inode *inode)
> +{
> + struct user_namespace *ns = inode_userns(inode);
> +
> + if (current_user_ns() == ns && current_fsuid() == inode->i_uid)
> + return 1;
> + if (ns_capable(ns, CAP_FOWNER))
> + return 1;
> + return 0;
> +}
> +EXPORT_SYMBOL(is_owner_or_cap);
> diff --git a/fs/namei.c b/fs/namei.c
> index 9e701e2..cfac5b4 100644
> --- a/fs/namei.c
> +++ b/fs/namei.c
> @@ -176,6 +176,9 @@ static int acl_permission_check(struct inode *inode, int mask, unsigned
int flag
>
>   mask &= MAY_READ | MAY_WRITE | MAY_EXEC;
>
> + if (current_user_ns() != inode_userns(inode))
> + goto other_perms;
> +
>   if (current_fsuid() == inode->i_uid)
>   mode >>= 6;
>   else {
> @@ -189,6 +192,7 @@ static int acl_permission_check(struct inode *inode, int mask, unsigned
int flag
>   mode >>= 3;
>   }
>
> +other_perms:
>   /*
>   * If the DACs are ok we don't need any capability check.
>   */
> @@ -230,7 +234,7 @@ int generic_permission(struct inode *inode, int mask, unsigned int
flags,
>   * Executable DACs are overridable if at least one exec bit is set.
>   */
>   if (!(mask & MAY_EXEC) || execute_ok(inode))
> - if (capable(CAP_DAC_OVERRIDE))

```
> +  if (ns_capable(inode_users(inode), CAP_DAC_OVERRIDE))
>     return 0;
>
>   /*
> @@ -238,7 +242,7 @@ int generic_permission(struct inode *inode, int mask, unsigned int
flags,
>    */
>   mask &= MAY_READ | MAY_WRITE | MAY_EXEC;
>   if (mask == MAY_READ || (S_ISDIR(inode->i_mode) && !(mask & MAY_WRITE)))
> -  if (capable(CAP_DAC_READ_SEARCH))
> +  if (ns_capable(inode_users(inode), CAP_DAC_READ_SEARCH))
>     return 0;
>
>   return -EACCES;
> @@ -675,6 +679,7 @@ force_reval_path(struct path *path, struct nameidata *nd)
>  static inline int exec_permission(struct inode *inode, unsigned int flags)
>  {
>   int ret;
> + struct user_namespace *ns = inode_users(inode);
>
>   if (inode->i_op->permission) {
>    ret = inode->i_op->permission(inode, MAY_EXEC, flags);
> @@ -687,7 +692,7 @@ static inline int exec_permission(struct inode *inode, unsigned int flags)
>   if (ret == -ECHILD)
>    return ret;
>
> - if (capable(CAP_DAC_OVERRIDE) || capable(CAP_DAC_READ_SEARCH))
> + if (ns_capable(ns, CAP_DAC_OVERRIDE) || ns_capable(ns, CAP_DAC_READ_SEARCH))
>    goto ok;
>
>   return ret;
> @@ -1940,11 +1945,15 @@ static inline int check_sticky(struct inode *dir, struct inode *inode)
>
>   if (!(dir->i_mode & S_ISVTX))
>    return 0;
> + if (current_user_ns() != inode_users(inode))
> +  goto other_userns;
>   if (inode->i_uid == fsuid)
>    return 0;
>   if (dir->i_uid == fsuid)
>    return 0;
> - return !capable(CAP_FOWNER);
> +
> +other_userns:
> + return !ns_capable(inode_users(inode), CAP_FOWNER);
>  }
>
>  /*
```

> @@ -2635,7 +2644,8 @@ int vfs_mknod(struct inode *dir, struct dentry *dentry, int mode, dev_t dev)
> if (error)
> return error;
>
> - if ((S_ISCHR(mode) || S_ISBLK(mode)) && !capable(CAP_MKNOD))
> + if ((S_ISCHR(mode) || S_ISBLK(mode)) &&
> + !ns_capable(inode_userns(dir), CAP_MKNOD))
> return -EPERM;
>
> if (!dir->i_op->mknod)
> diff --git a/include/linux/fs.h b/include/linux/fs.h
> index bd32159..c84417a 100644
> --- a/include/linux/fs.h
> +++ b/include/linux/fs.h
> @@ -1446,8 +1446,13 @@ enum {
> #define put_fs_excl() atomic_dec(&current->fs_excl)
> #define has_fs_excl() atomic_read(&current->fs_excl)
>
> -#define is_owner_or_cap(inode) \
> - ((current_fsuid() == (inode)->i_uid) || capable(CAP_FOWNER))
> +/*
> + * until VFS tracks user namespaces for inodes, just make all files
> + * belong to init_user_ns
> + */
> +extern struct user_namespace init_user_ns;
> +#define inode_userns(inode) (&init_user_ns)
> +extern int is_owner_or_cap(const struct inode *inode);
>
> /* not quite ready to be deprecated, but... */
> extern void lock_super(struct super_block *);

_____

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