
Subject: Re: [PATCH] Make access to task's nsproxy liter

Posted by [serue](#) on Fri, 10 Aug 2007 13:40:03 GMT

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

Quoting Pavel Emelyanov (xemul@openvz.org):

> When someone wants to deal with some other task's namespaces
> it has to lock the task and then to get the desired namespace
> if the one exists. This is slow on read-only paths and may be
> impossible in some cases.
>
> E.g. Oleg recently noticed a race between unshare() and the
> (sent for review in containers) pid namespaces - when the task notifies the
> parent it has to know the parent's namespace, but taking the task_lock() is
> impossible there - the code is under write locked tasklist lock.
>
> On the other hand switching the namespace on task (daemonize) and releasing
> the namespace (after the last task exit) is rather
> rare operation and we can sacrifice its speed to solve the
> issues above.
>
> The access to other task namespaces is proposed to be performed
> like this:
>
> rcu_read_lock();
> nsproxy = task_nsproxy(tsk);
> if (nsproxy != NULL) {
> /*
> * work with the namespaces here
> * e.g. get the reference on one of them
> * /
> } / *
> * NULL task_nsproxy() means that this task is
> * almost dead (zombie)
> * /
> rcu_read_unlock();
>
> This patch has passed the review by Eric and Oleg :) and,
> of course, tested.
>
> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
> Cc: Eric W. Biederman <ebiederm@xmission.com>
> Cc: Oleg Nesterov <oleg@tv-sign.ru>
> Cc: Paul E. McKenney <paulmck@linux.vnet.ibm.com>
> Cc: Serge Hallyn <serue@us.ibm.com>
>
> ---
>
> fs/proc/base.c | 27 ++++++++-----

```

> include/linux/nsproxy.h | 44 ++++++-----
> kernel/exit.c          | 4 +---
> kernel/fork.c          | 11 +++++-----
> kernel/nsproxy.c       | 39 ++++++-----
> 5 files changed, 86 insertions(+), 39 deletions(-)
>
> diff --git a/fs/proc/base.c b/fs/proc/base.c
> index ed2b224..614851a 100644
> --- a/fs/proc/base.c
> +++ b/fs/proc/base.c
> @@ -371,18 +371,21 @@ struct proc_mounts {
> static int mounts_open(struct inode *inode, struct file *file)
> {
>     struct task_struct *task = get_proc_task(inode);
>     + struct nsproxy *nsp;
>     struct mnt_namespace *ns = NULL;
>     struct proc_mounts *p;
>     int ret = -EINVAL;
>
>     if (task) {
>         - task_lock(task);
>         - if (task->nsproxy) {
>             ns = task->nsproxy->mnt_ns;
>         + rcu_read_lock();
>         + nsp = task_nsproxy(task);
>         + if (nsp) {
>             ns = nsp->mnt_ns;
>             if (ns)
>                 get_mnt_ns(ns);
>         }
>         - task_unlock(task);
>         + rcu_read_unlock();
>         +
>         put_task_struct(task);
>     }
>
> @@ -445,16 +448,20 @@ static int mountstats_open(struct inode
> if (!ret) {
>     struct seq_file *m = file->private_data;
>     + struct nsproxy *nsp;
>     struct mnt_namespace *mnt_ns = NULL;
>     struct task_struct *task = get_proc_task(inode);
>
>     if (task) {
>         - task_lock(task);
>         - if (task->nsproxy)
>             mnt_ns = task->nsproxy->mnt_ns;
>         - if (mnt_ns)

```

```

> - get_mnt_ns(mnt_ns);
> - task_unlock(task);
> + rcu_read_lock();
> + nsp = task_nsproxy(task);
> + if (nsp) {
> +     mnt_ns = nsp->mnt_ns;
> +     if (mnt_ns)
> +         get_mnt_ns(mnt_ns);
> + }
> + rcu_read_unlock();
> +
> put_task_struct(task);
> }
>
> diff --git a/include/linux/nsproxy.h b/include/linux/nsproxy.h
> index 525d8fc..ef4e863 100644
> --- a/include/linux/nsproxy.h
> +++ b/include/linux/nsproxy.h
> @@ -32,8 +32,39 @@ struct nsproxy {
> };
> extern struct nsproxy init_nsproxy;
>
> +/*
> + * the namespaces access rules are:
> + *
> + * 1. only current task is allowed to change tsk->nsproxy pointer or
> + *    any pointer on the nsproxy itself
> + *
> + * 2. when accessing (i.e. reading) current task's namespaces - no
> + *    precautions should be taken - just dereference the pointers
> + *
> + * 3. the access to other task namespaces is performed like this
> + *    rcu_read_lock();
> + *    nsproxy = task_nsproxy(tsk);
> + *    if (nsproxy != NULL) {
> + *        /*
> + *         * work with the namespaces here
> + *         * e.g. get the reference on one of them
> + *         */
> + *    } /*
> + *    * NULL task_nsproxy() means that this task is
> + *    * almost dead (zombie)
> + *    */
> + *    rcu_read_unlock();

```

And lastly, I guess that the caller to `switch_task_namespaces()` has to ensure that `new_nsproxy` either (1) is the init namespace, (2) is a brand-new namespace to which noone else has a reference, or (3) the

caller has to hold a reference to the new_nsproxy across the call to switch_task_namespaces().

As it happens the current calls fit (1) or (2). Again if we happen to jump into the game of switching a task into another task's nsproxy, we'll need to be mindful of (3) so that new_nsproxy can't be tossed into the bin between

```
if (new)
    get_nsproxy(new);
```

below.

```
> + *
> + */
> +
> +static inline struct nsproxy *task_nsproxy(struct task_struct *tsk)
> +{
> + return rcu_dereference(tsk->nsproxy);
> +}
> +
> int copy_namespaces(unsigned long flags, struct task_struct *tsk);
> -void get_task_namespaces(struct task_struct *tsk);
> +void exit_task_namespaces(struct task_struct *tsk);
> +void switch_task_namespaces(struct task_struct *tsk, struct nsproxy *new);
> void free_nsproxy(struct nsproxy *ns);
> int unshare_nsproxy_namespaces(unsigned long, struct nsproxy **,
> struct fs_struct *);
> @@ -45,17 +76,6 @@ static inline void put_nsproxy(struct ns
> }
> }
>
> -static inline void exit_task_namespaces(struct task_struct *p)
> -{
> - struct nsproxy *ns = p->nsproxy;
> - if (ns) {
> - task_lock(p);
> - p->nsproxy = NULL;
> - task_unlock(p);
> - put_nsproxy(ns);
> - }
> -}
> -
> #ifdef CONFIG_CONTAINER_NS
> int ns_container_clone(struct task_struct *tsk);
> #else
> diff --git a/kernel/exit.c b/kernel/exit.c
> index 2f4f35e..a2aef1f 100644
```

```

> --- a/kernel/exit.c
> +++ b/kernel/exit.c
> @@ -409,9 +409,7 @@ void daemonize(const char *name, ...)
> current->fs = fs;
> atomic_inc(&fs->count);
>
> - exit_task_namespaces(current);
> - current->nsproxy = init_task.nsproxy;
> - get_task_namespaces(current);
> + switch_task_namespaces(current, init_task.nsproxy);
>
> exit_files(current);
> current->files = init_task.files;
> diff --git a/kernel/fork.c b/kernel/fork.c
> index bc48e0f..0c6dd67 100644
> --- a/kernel/fork.c
> +++ b/kernel/fork.c
> @@ -1618,7 +1618,7 @@ asmlinkage long sys_unshare(unsigned lon
> struct mm_struct *mm, *new_mm = NULL, *active_mm = NULL;
> struct files_struct *fd, *new_fd = NULL;
> struct sem_undo_list *new_ulist = NULL;
> - struct nsproxy *new_nsproxy = NULL, *old_nsproxy = NULL;
> + struct nsproxy *new_nsproxy = NULL;
>
> check_unshare_flags(&unshare_flags);
>
> @@ -1647,14 +1647,13 @@ asmlinkage long sys_unshare(unsigned lon
>
> if (new_fs || new_mm || new_fd || new_ulist || new_nsproxy) {
>
> - task_lock(current);
> -
> if (new_nsproxy) {
> - old_nsproxy = current->nsproxy;
> - current->nsproxy = new_nsproxy;
> - new_nsproxy = old_nsproxy;
> + switch_task_namespaces(current, new_nsproxy);
> + new_nsproxy = NULL;
> }
>
> + task_lock(current);
> +
> if (new_fs) {
> fs = current->fs;
> current->fs = new_fs;
> diff --git a/kernel/nsproxy.c b/kernel/nsproxy.c
> index 58843ae..48bc070 100644
> --- a/kernel/nsproxy.c

```

```

> +++ b/kernel/nsproxy.c
> @@ -30,14 +30,6 @@ static inline void get_nsproxy(struct ns
> atomic_inc(&ns->count);
> }
>
> -void get_task_namespaces(struct task_struct *tsk)
> -{
> - struct nsproxy *ns = tsk->nsproxy;
> - if (ns) {
> - get_nsproxy(ns);
> - }
> -}
> -
> /*
>  * creates a copy of "orig" with refcount 1.
>  */
> @@ -205,6 +197,37 @@ out:
> return err;
> }
>
> +void switch_task_namespaces(struct task_struct *p, struct nsproxy *new)
> +{
> + struct nsproxy *ns;
> +
> + might_sleep();
> +
> + ns = p->nsproxy;
> + if (ns == new)
> + return;
> +
> + if (new)
> + get_nsproxy(new);
> + rcu_assign_pointer(p->nsproxy, new);
> +
> + if (ns && atomic_dec_and_test(&ns->count)) {
> + /*
> +  * wait for others to get what they want from this nsproxy.
> +  *
> +  * cannot release this nsproxy via the call_rcu() since
> +  * put_mnt_ns() will want to sleep
> +  */
> + synchronize_rcu();
> + free_nsproxy(ns);
> + }
> +}
> +
> +void exit_task_namespaces(struct task_struct *p)
> +{

```

```
> + switch_task_namespaces(p, NULL);  
> +}  
> +  
> static int __init nsproxy_cache_init(void)  
> {  
> nsproxy_cachep = kmem_cache_create("nsproxy", sizeof(struct nsproxy),
```

thanks,
-serge
