Subject: [PATCH 2/2] pidns: Support unsharing the pid namespace.
Posted by Daniel Lezcano on Tue, 15 Feb 2011 16:53:44 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com>

- Allow CLONEW_NEWPID into unshare.
- Pass both nsproxy->pid_ns and task_active_pid_ns to copy_pid_ns
  As they can now be different.

Unsharing of the pid namespace unlike unsharing of other namespaces
does not take effect immediately. Instead it affects the children
created with fork and clone. The first of these children becomes the init
process of the new pid namespace, the rest become oddball children
of pid 0. From the point of view of the new pid namespace the process
that created it is pid 0, as it's pid does not map.

A couple of different semantics were considered but this one was
settled on because it is easy to implement and it is usable from
pam modules. The core reasons for the existence of unshare.

I took a survey of the callers of pam modules and the following
appears to be a representative sample of their logic.

```
{
 setup stuff include pam
 child = fork();
 if (!child) {
  setuid()
            exec /bin/bash
       }
       waitpid(child);

       pam and other cleanup
}
```

As you can see there is a fork to create the unprivileged user
space process. Which means that the unprivileged user space
process will appear as pid 1 in the new pid namespace. Further
most login processes do not cope with extraneous children which
means shifting the duty of reaping extraneous child process to
the creator of those extraneous children makes the system more
comprehensible.

The practical reason for this set of pid namespace semantics is
that it is simple to implement and verify they work correctly.
Whereas an implementation that requires changing the struct
pid on a process comes with a lot more races and pain. Not
the least of which is that glibc caches getpid().

These semantics are implemented by having two notions
of the pid namespace of a process. There is task_active_pid_ns
which is the pid namspace the process was created with
and the pid namespace that all pids are presented to
that process in. The task_active_pid_ns is stored
in the struct pid of the task.

There is the pid namespace that will be used for children
that pid namespace is stored in task->nsproxy->pid_ns.

There is one really nasty corner case in all of this. Which
pid namespace are you in if your parent unshared it's pid
namespace and then on clone you also unshare the pid namespace.
To me there are only two possible answers. Either the cases
is so bizarre and we deny it completely. or the new pid
namespace is a descendent of our parent's active pid namespace,
and we ignore the task->nsproxy->pid_ns.

To that end I have modified copy_pid_ns to take both of these
pid namespaces. The active pid namespace and the default
pid namespace of children. Allowing me to simply implement
unsharing a pid namespace in clone after already unsharing
a pid namespace with unshare.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
Signed-off-by: Daniel Lezcano <daniel.lezcano@free.fr>
---
 include/linux/pid_namespace.h | 14 +++++++++-----
 kernel/fork.c | 3 ++-
 kernel/nsproxy.c | 5 +++--
 kernel/pid_namespace.c | 8 +++++---
 4 files changed, 19 insertions(+), 11 deletions(-)

diff --git a/include/linux/pid_namespace.h b/include/linux/pid_namespace.h
index b447d37..4316347 100644
--- a/include/linux/pid_namespace.h
+++ b/include/linux/pid_namespace.h
@@ -43,7 +43,10 @@ static inline struct pid_namespace *get_pid_ns(struct pid_namespace *ns)
  return ns;
 }

-extern struct pid_namespace *copy_pid_ns(unsigned long flags, struct pid_namespace *ns);
+extern struct pid_namespace *copy_pid_ns(unsigned long flags,
+	struct pid_namespace *default_ns,
+	struct pid_namespace *active_ns);
+
 extern void free_pid_ns(struct kref *kref);

```
 extern void zap_pid_ns_processes(struct pid_namespace *pid_ns);

@@ -61,12 +64,13 @@ static inline struct pid_namespace *get_pid_ns(struct pid_namespace
*ns)
  return ns;
 }

-static inline struct pid_namespace *
-copy_pid_ns(unsigned long flags, struct pid_namespace *ns)
+static inline struct pid_namespace *copy_pid_ns(unsigned long flags,
+       struct pid_namespace *default_ns,
+       struct pid_namespace *active_ns)
 {
  if (flags & CLONE_NEWPID)
-  ns = ERR_PTR(-EINVAL);
- return ns;
+  return ERR_PTR(-EINVAL);
+ return default_ns;
 }

 static inline void put_pid_ns(struct pid_namespace *ns)
diff --git a/kernel/fork.c b/kernel/fork.c
index e7a5907..4b019f1 100644
--- a/kernel/fork.c
+++ b/kernel/fork.c
@@ -1633,7 +1633,8 @@ SYSCALL_DEFINE1(unshare, unsigned long, unshare_flags)
  err = -EINVAL;
  if (unshare_flags & ~(CLONE_THREAD|CLONE_FS|CLONE_NEWNS|CLONE_SIGHAND|
     CLONE_VM|CLONE_FILES|CLONE_SYSVSEM|
-    CLONE_NEWUTS|CLONE_NEWIPC|CLONE_NEWNET))
+           CLONE_NEWUTS|CLONE_NEWIPC|CLONE_NEWNET|
+           CLONE_NEWPID))
   goto bad_unshare_out;

  /*
diff --git a/kernel/nsproxy.c b/kernel/nsproxy.c
index f74e6c0..a9cf251 100644
--- a/kernel/nsproxy.c
+++ b/kernel/nsproxy.c
@@ -81,7 +81,8 @@ static struct nsproxy *create_new_namespaces(unsigned long flags,
   goto out_ipc;
  }

- new_nsp->pid_ns = copy_pid_ns(flags, task_active_pid_ns(tsk));
+ new_nsp->pid_ns = copy_pid_ns(flags, tsk->nsproxy->pid_ns,
+         task_active_pid_ns(tsk));
  if (IS_ERR(new_nsp->pid_ns)) {
   err = PTR_ERR(new_nsp->pid_ns);
```

```
   goto out_pid;
@@ -185,7 +186,7 @@ int unshare_nsproxy_namespaces(unsigned long unshare_flags,
  int err = 0;

  if (!(unshare_flags & (CLONE_NEWNS | CLONE_NEWUTS | CLONE_NEWIPC |
-          CLONE_NEWNET)))
+          CLONE_NEWNET | CLONE_NEWPID)))
   return 0;

  if (!capable(CAP_SYS_ADMIN))
diff --git a/kernel/pid_namespace.c b/kernel/pid_namespace.c
index e8ea25d..9e101c1 100644
--- a/kernel/pid_namespace.c
+++ b/kernel/pid_namespace.c
@@ -123,13 +123,15 @@ static void destroy_pid_namespace(struct pid_namespace *ns)
  kmem_cache_free(pid_ns_cachep, ns);
 }

-struct pid_namespace *copy_pid_ns(unsigned long flags, struct pid_namespace *old_ns)
+struct pid_namespace *copy_pid_ns(unsigned long flags,
+       struct pid_namespace *default_ns,
+       struct pid_namespace *active_ns)
 {
  if (!(flags & CLONE_NEWPID))
-  return get_pid_ns(old_ns);
+  return get_pid_ns(default_ns);
  if (flags & (CLONE_THREAD|CLONE_PARENT))
   return ERR_PTR(-EINVAL);
- return create_pid_namespace(old_ns);
+ return create_pid_namespace(active_ns);
 }

 void free_pid_ns(struct kref *kref)
--
1.7.1
```

_____

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