Subject: [PATCH 7/9] sig: Handle pid namespace crossing when sending signals.
Posted by ebiederm on Wed, 12 Dec 2007 12:52:41 GMT
View Forum Message <> Reply to Message

Setting si_pid correctly in the context of pid namespaces is tricky.
Currently with the special cases in do_notify_parent and
do_notify_parent_cldstop we handle all of the day to day cases
properly except sending a signal to a task in a child pid namespace.
For that case we need to pretend to be the kernel and set si_pid to 0.

There are also a few theoretical cases where we can trigger sending a
signal from a task in one pid namespace to a task in another pid
namespace. With no necessary correlation between one or the other.
In those cases when the source pid namespace is a child of the
destination pid namespace we actually have a valid pid value
we can and should report to user space.

This patch modifies the code to handle the full general case for
setting si_pid. The code is a little longer but occurs only once and
making it some easier to understand and verify it is correct.

I add a struct pid sender parameter to __group_send_sig_info, as that is
the only function called with si_pid != task_tgid_vnr(current). So we can
correctly handle the sending of a signal to the parent of an arbitrary
task.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
drivers/char/tty_io.c | 4 +-
include/linux/signal.h | 3 +-
ipc/mqueue.c | 2 +-
kernel/posix-cpu-timers.c | 8 ++--
kernel/signal.c | 88 ++++++++++++++++++++++++++++++++----------
5 files changed, 63 insertions(+), 42 deletions(-)

```
diff --git a/drivers/char/tty_io.c b/drivers/char/tty_io.c
index 613ec81..c121cdb 100644
--- a/drivers/char/tty_io.c
+++ b/drivers/char/tty_io.c
@@ -1435,8 +1435,8 @@ static void do_tty_hangup(struct work_struct *work)
     spin_unlock_irq(&p->sighand->siglock);
     continue;
    }
-   __group_send_sig_info(SIGHUP, SEND_SIG_PRIV, p);
-   __group_send_sig_info(SIGCONT, SEND_SIG_PRIV, p);
+   __group_send_sig_info(SIGHUP, SEND_SIG_PRIV, p, NULL);
+   __group_send_sig_info(SIGCONT, SEND_SIG_PRIV, p, NULL);
    put_pid(p->signal->tty_old_pgrp);  /* A noop */
```

```
    if (tty->pgrp)
     p->signal->tty_old_pgrp = get_pid(tty->pgrp);
diff --git a/include/linux/signal.h b/include/linux/signal.h
index 42d2e0a..0a13489 100644
--- a/include/linux/signal.h
+++ b/include/linux/signal.h
@@ -234,7 +234,8 @@ static inline int valid_signal(unsigned long sig)

 extern int next_signal(struct sigpending *pending, sigset_t *mask);
 extern int group_send_sig_info(int sig, struct siginfo *info, struct task_struct *p);
-extern int __group_send_sig_info(int, struct siginfo *, struct task_struct *);
+extern int __group_send_sig_info(int, struct siginfo *, struct task_struct *,
+     struct pid *sender);
 extern long do_sigpending(void __user *, unsigned long);
 extern int sigprocmask(int, sigset_t *, sigset_t *);
 extern int show_unhandled_signals;
diff --git a/ipc/mqueue.c b/ipc/mqueue.c
index d3feadf..b0bf0b0 100644
--- a/ipc/mqueue.c
+++ b/ipc/mqueue.c
@@ -510,7 +510,7 @@ static void __do_notify(struct mqueue_inode_info *info)
    sig_i.si_errno = 0;
    sig_i.si_code = SI_MESGQ;
    sig_i.si_value = info->notify.sigev_value;
-   sig_i.si_pid = task_tgid_vnr(current);
+   sig_i.si_pid = 0; /* Uses default current tgid */
    sig_i.si_uid = current->uid;

    kill_pid_info(info->notify.sigev_signo,
diff --git a/kernel/posix-cpu-timers.c b/kernel/posix-cpu-timers.c
index 68c9637..91f80b9 100644
--- a/kernel/posix-cpu-timers.c
+++ b/kernel/posix-cpu-timers.c
@@ -1109,7 +1109,7 @@ static void check_process_timers(struct task_struct *tsk,
     sig->it_prof_expires = cputime_add(
      sig->it_prof_expires, ptime);
    }
-   __group_send_sig_info(SIGPROF, SEND_SIG_PRIV, tsk);
+   __group_send_sig_info(SIGPROF, SEND_SIG_PRIV, tsk, NULL);
   }
   if (!cputime_eq(sig->it_prof_expires, cputime_zero) &&
       (cputime_eq(prof_expires, cputime_zero) ||
@@ -1125,7 +1125,7 @@ static void check_process_timers(struct task_struct *tsk,
     sig->it_virt_expires = cputime_add(
      sig->it_virt_expires, utime);
    }
-   __group_send_sig_info(SIGVTALRM, SEND_SIG_PRIV, tsk);
+   __group_send_sig_info(SIGVTALRM, SEND_SIG_PRIV, tsk, NULL);
```

```
   }
   if (!cputime_eq(sig->it_virt_expires, cputime_zero) &&
       (cputime_eq(virt_expires, cputime_zero) ||
@@ -1141,14 +1141,14 @@ static void check_process_timers(struct task_struct *tsk,
     * At the hard limit, we just die.
     * No need to calculate anything else now.
     */
-   __group_send_sig_info(SIGKILL, SEND_SIG_PRIV, tsk);
+   __group_send_sig_info(SIGKILL, SEND_SIG_PRIV, tsk, NULL);
    return;
   }
   if (psecs >= sig->rlim[RLIMIT_CPU].rlim_cur) {
    /*
     * At the soft limit, send a SIGXCPU every second.
     */
-   __group_send_sig_info(SIGXCPU, SEND_SIG_PRIV, tsk);
+   __group_send_sig_info(SIGXCPU, SEND_SIG_PRIV, tsk, NULL);
    if (sig->rlim[RLIMIT_CPU].rlim_cur
        < sig->rlim[RLIMIT_CPU].rlim_max) {
     sig->rlim[RLIMIT_CPU].rlim_cur++;
diff --git a/kernel/signal.c b/kernel/signal.c
index 694a643..c01e3cd 100644
--- a/kernel/signal.c
+++ b/kernel/signal.c
@@ -657,8 +657,40 @@ static void handle_stop_signal(int sig, struct task_struct *p)
  }
 }

+static void set_sigqueue_pid(struct sigqueue *q, struct task_struct *t,
+        struct pid *sender)
+{
+ struct pid_namespace *ns;
+
+ /* Set si_pid to the pid number of sender in the pid namespace of
+  * our destination task for all siginfo types that support it.
+  */
+ switch(q->info.si_code & __SI_MASK) {
+  /* siginfo without si_pid */
+ case __SI_TIMER:
+ case __SI_POLL:
+ case __SI_FAULT:
+  break;
+  /* siginfo with si_pid */
+ case __SI_KILL:
+ case __SI_CHLD:
+ case __SI_RT:
+ case __SI_MESGQ:
+ default:
```

```
+  /* si_pid for SI_KERNEL is always 0 */
+  if (q->info.si_code == SI_KERNEL)
+   break;
+  /* Is current not the sending task? */
+  if (!sender)
+   sender = task_tgid(current);
+  ns = task_active_pid_ns(t);
+  q->info.si_pid = pid_nr_ns(sender, ns);
+  break;
+ }
+}
+
 static int send_signal(int sig, struct siginfo *info, struct task_struct *t,
-   struct sigpending *signals)
+   struct sigpending *signals, struct pid *sender)
 {
  struct sigqueue * q = NULL;
  int ret = 0;
@@ -694,8 +726,9 @@ static int send_signal(int sig, struct siginfo *info, struct task_struct *t,
    q->info.si_signo = sig;
    q->info.si_errno = 0;
    q->info.si_code = SI_USER;
-   q->info.si_pid = task_tgid_vnr(current);
+   q->info.si_pid = 0; /* Uses current tgid */
    q->info.si_uid = current->uid;
+   sender = task_tgid(current);
    break;
   case (unsigned long) SEND_SIG_PRIV:
    q->info.si_signo = sig;
@@ -708,6 +741,7 @@ static int send_signal(int sig, struct siginfo *info, struct task_struct *t,
    copy_siginfo(&q->info, info);
    break;
   }
+  set_sigqueue_pid(q, t, sender);
  } else if (!is_si_special(info)) {
   if (sig >= SIGRTMIN && info->si_code != SI_USER)
   /*
@@ -775,7 +809,7 @@ specific_send_sig_info(int sig, struct siginfo *info, struct task_struct *t)
  if (LEGACY_QUEUE(&t->pending, sig))
   goto out;

- ret = send_signal(sig, info, t, &t->pending);
+ ret = send_signal(sig, info, t, &t->pending, NULL);
  if (!ret && !sigismember(&t->blocked, sig))
   signal_wake_up(t, sig == SIGKILL);
 out:
@@ -922,7 +956,8 @@ __group_complete_signal(int sig, struct task_struct *p)
 }
```

```
 int
-__group_send_sig_info(int sig, struct siginfo *info, struct task_struct *p)
+__group_send_sig_info(int sig, struct siginfo *info, struct task_struct *p,
+        struct pid *sender)
 {
  int ret = 0;

@@ -942,7 +977,7 @@ __group_send_sig_info(int sig, struct siginfo *info, struct task_struct *p)
   * We always use the shared queue for process-wide signals,
   * to avoid several races.
   */
- ret = send_signal(sig, info, p, &p->signal->shared_pending);
+ ret = send_signal(sig, info, p, &p->signal->shared_pending, sender);
  if (unlikely(ret))
   return ret;

@@ -1008,7 +1043,7 @@ int group_send_sig_info(int sig, struct siginfo *info, struct task_struct
*p)
  if (!ret && sig) {
   ret = -ESRCH;
   if (lock_task_sighand(p, &flags)) {
-   ret = __group_send_sig_info(sig, info, p);
+   ret = __group_send_sig_info(sig, info, p, NULL);
    unlock_task_sighand(p, &flags);
   }
  }
@@ -1114,7 +1149,7 @@ int kill_pid_info_as_uid(int sig, struct siginfo *info, struct pid *pid,
  if (sig && p->sighand) {
   unsigned long flags;
   spin_lock_irqsave(&p->sighand->siglock, flags);
-  ret = __group_send_sig_info(sig, info, p);
+  ret = __group_send_sig_info(sig, info, p, NULL);
   spin_unlock_irqrestore(&p->sighand->siglock, flags);
  }
 out_unlock:
@@ -1415,6 +1450,7 @@ void do_notify_parent(struct task_struct *tsk, int sig)
  struct siginfo info;
  unsigned long flags;
  struct sighand_struct *psig;
+ struct pid *sender;

  BUG_ON(sig == -1);

@@ -1424,24 +1460,11 @@ void do_notify_parent(struct task_struct *tsk, int sig)
  BUG_ON(!tsk->ptrace &&
        (tsk->group_leader != tsk || !thread_group_empty(tsk)));
```

```
+ /* We are under tasklist_lock so no need to call get_pid */
+ sender = task_pid(tsk);
  info.si_signo = sig;
  info.si_errno = 0;
- /*
-  * we are under tasklist_lock here so our parent is tied to
-  * us and cannot exit and release its namespace.
-  *
-  * the only it can is to switch its nsproxy with sys_unshare,
-  * bu uncharing pid namespaces is not allowed, so we'll always
-  * see relevant namespace
-  *
-  * write_lock() currently calls preempt_disable() which is the
-  * same as rcu_read_lock(), but according to Oleg, this is not
-  * correct to rely on this
-  */
- rcu_read_lock();
- info.si_pid = task_pid_nr_ns(tsk, tsk->parent->nsproxy->pid_ns);
- rcu_read_unlock();
-
+ info.si_pid = 0; /* Filled in later from sender */
  info.si_uid = tsk->uid;

  /* FIXME: find out whether or not this is supposed to be c*time. */
@@ -1485,7 +1508,7 @@ void do_notify_parent(struct task_struct *tsk, int sig)
    sig = 0;
  }
  if (valid_signal(sig) && sig > 0)
-  __group_send_sig_info(sig, &info, tsk->parent);
+  __group_send_sig_info(sig, &info, tsk->parent, sender);
  __wake_up_parent(tsk, tsk->parent);
  spin_unlock_irqrestore(&psig->siglock, flags);
 }
@@ -1496,6 +1519,7 @@ static void do_notify_parent_cldstop(struct task_struct *tsk, int why)
  unsigned long flags;
  struct task_struct *parent;
  struct sighand_struct *sighand;
+ struct pid *sender;

  if (tsk->ptrace & PT_PTRACED)
   parent = tsk->parent;
@@ -1504,15 +1528,11 @@ static void do_notify_parent_cldstop(struct task_struct *tsk, int why)
   parent = tsk->real_parent;
  }

- /* We are under tasklist_lock so no need to call get_pid */
- sender = task_pid(tsk);
  info.si_signo = SIGCHLD;
```

```
  info.si_errno = 0;
- /*
-  * see comment in do_notify_parent() abot the following 3 lines
-  */
- rcu_read_lock();
- info.si_pid = task_pid_nr_ns(tsk, tsk->parent->nsproxy->pid_ns);
- rcu_read_unlock();
-
+ info.si_pid = 0; /* Filled in later from sender */
  info.si_uid = tsk->uid;

  /* FIXME: find out whether or not this is supposed to be c*time. */
@@ -1538,7 +1558,7 @@ static void do_notify_parent_cldstop(struct task_struct *tsk, int why)
  spin_lock_irqsave(&sighand->siglock, flags);
  if (sighand->action[SIGCHLD-1].sa.sa_handler != SIG_IGN &&
      !(sighand->action[SIGCHLD-1].sa.sa_flags & SA_NOCLDSTOP))
-  __group_send_sig_info(SIGCHLD, &info, parent);
+  __group_send_sig_info(SIGCHLD, &info, parent, sender);
  /*
   * Even if SIGCHLD is not generated, we must wake up wait4 calls.
   */
@@ -2223,7 +2243,7 @@ sys_kill(int pid, int sig)
  info.si_signo = sig;
  info.si_errno = 0;
  info.si_code = SI_USER;
- info.si_pid = task_tgid_vnr(current);
+ info.si_pid = 0; /* Uses default current tgid */
  info.si_uid = current->uid;

  return kill_something_info(sig, &info, pid);
@@ -2239,7 +2259,7 @@ static int do_tkill(int tgid, int pid, int sig)
  info.si_signo = sig;
  info.si_errno = 0;
  info.si_code = SI_TKILL;
- info.si_pid = task_tgid_vnr(current);
+ info.si_pid = 0; /* Uses default current tgid */
  info.si_uid = current->uid;

  read_lock(&tasklist_lock);
--
1.5.3.rc6.17.g1911
```