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
```

Subject: [PATCH 8/9] signal: Drop signals before sending them to init.
Posted by ebiederm on Wed, 12 Dec 2007 12:57:13 GMT
View Forum Message <> Reply to Message

Currently init drops all signals including SIGKILL that are sent to it
that it does not ignore and does have a handler for. Extending this to
pid namespaces where we want to maintain this semantic for signals sent
from inside the pid namespace (descendents of init) but we want the
namespace init to appear as a normal process is problematic, as a
naive approach requires always tracking the sender of a signal.

By making the rule (for init dropping signals):
When sending a signal to init, the presence of a signal handler that
is not SIG_DFL allows the signal to be sent to init. If the signal
is not sent it is silently dropped without becoming pending.

The only noticeable user space difference from todays init is that it
no longer needs to worry about signals becoming pending when it has
them marked as SIG_DFL and blocked.

This change by making the presence of a signal handler effectively
a permission check allows us to do all of the work before we enqueue
the signal, and there is no need for any fancy tracking of the
signal sender.

Which means that we can now allow force_sig_info to send signals to
init, that panic the kernel instead of going into an infinite busy
loop taking an exception sending a signal and then retaking the same
exception, eating all of the cpu time but accomplishing nothing.

This change also makes it possible to easily implement the desired
semantics of /sbin/init for pid namespaces where outer processes can
kill init but processes inside the pid namespace can not.

While it is now easy to remove the dropping of signals from individual
code paths such as force_sig_info this patch does not implement that,
to retain as much of the current behavior as possible. The only
behavioral difference besides not queuing blocked SIG_DFL signals
are signals directly added with sigaddset. In practice a threaded init
now receives SIGKILL sent from a core dump, a thread group exit, or an
exec shutting down extraneous threads.

This patch was inspired by a patch from Oleg and initial refined
in conversation with Suka and others on the containers list.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

```
---
 kernel/signal.c |   47 ++++++++++++++++++++++++++++++++++++++---------
 1 files changed, 38 insertions(+), 9 deletions(-)
```

```
diff --git a/kernel/signal.c b/kernel/signal.c
index c01e3cd..029a45d 100644
--- a/kernel/signal.c
+++ b/kernel/signal.c
@@ -64,6 +64,25 @@ static int sig_ignored(struct task_struct *t, int sig)
   (handler == SIG_DFL && sig_kernel_ignore(sig));
 }

+static int is_sig_init(struct task_struct *tsk)
+{
+ if (likely(!is_global_init(tsk->group_leader)))
+  return 0;
+
+ return 1;
+}
+
+static int sig_init_drop(struct task_struct *tsk, int sig)
+{
+ /* All signals for which init has a SIG_DFL handler are
+  * silently dropped without being sent.
+  */
+ if (!is_sig_init(tsk))
+  return 0;
+
+ return (tsk->sighand->action[sig-1].sa.sa_handler == SIG_DFL);
+}
+
 /*
  * Re-calculate pending state from the set of locally pending
  * signals, globally pending signals, and blocked signals.
@@ -834,6 +853,9 @@ force_sig_info(int sig, struct siginfo *info, struct task_struct *t)
  struct k_sigaction *action;

  spin_lock_irqsave(&t->sighand->siglock, flags);
+ ret = 0;
+ if (sig_init_drop(t, sig))
+  goto out;
  action = &t->sighand->action[sig-1];
  ignored = action->sa.sa_handler == SIG_IGN;
  blocked = sigismember(&t->blocked, sig);
@@ -845,6 +867,7 @@ force_sig_info(int sig, struct siginfo *info, struct task_struct *t)
   }
  }
  ret = specific_send_sig_info(sig, info, t);
+out:
  spin_unlock_irqrestore(&t->sighand->siglock, flags);
  return ret;
@@ -962,6 +985,9 @@ __group_send_sig_info(int sig, struct siginfo *info, struct task_struct *p,
  int ret = 0;

  assert_spin_locked(&p->sighand->siglock);
+ if (sig_init_drop(p, sig))
+  return ret;
+
  handle_stop_signal(sig, p);

  /* Short-circuit ignored signals.  */
@@ -1224,7 +1250,9 @@ send_sig_info(int sig, struct siginfo *info, struct task_struct *p)
   */
  read_lock(&tasklist_lock);
  spin_lock_irqsave(&p->sighand->siglock, flags);
- ret = specific_send_sig_info(sig, info, p);
+ ret = 0;
+ if (!sig_init_drop(p, sig))
+  ret = specific_send_sig_info(sig, info, p);
  spin_unlock_irqrestore(&p->sighand->siglock, flags);
  read_unlock(&tasklist_lock);
  return ret;
@@ -1392,6 +1420,11 @@ send_group_sigqueue(int sig, struct sigqueue *q, struct task_struct *p)
  read_lock(&tasklist_lock);
  /* Since it_lock is held, p->sighand cannot be NULL. */
  spin_lock_irqsave(&p->sighand->siglock, flags);
+ if (sig_init_drop(p, sig)) {
+  ret = 1;
+  goto out;
+ }
+
  handle_stop_signal(sig, p);

  /* Short-circuit ignored signals.  */
@@ -1844,12 +1877,6 @@ relock:
   if (sig_kernel_ignore(signr)) /* Default is nothing. */
    continue;

-  /*
-   * Global init gets no signals it doesn't want.
-   */
-  if (is_global_init(current))
-   continue;
-
   if (sig_kernel_stop(signr)) {
    /*
     * The default action is to stop all threads in
@@ -2272,8 +2299,10 @@ static int do_tkill(int tgid, int pid, int sig)
   */
  if (!error && sig && p->sighand) {
   spin_lock_irq(&p->sighand->siglock);
-  handle_stop_signal(sig, p);
-  error = specific_send_sig_info(sig, &info, p);
+  if (!sig_init_drop(p, sig)) {
+   handle_stop_signal(sig, p);
+   error = specific_send_sig_info(sig, &info, p);
+  }
   spin_unlock_irq(&p->sighand->siglock);
  }
 }
--
1.5.3.rc6.17.g1911
```

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

Subject: [PATCH 9/9] signal: Ignore signals sent to the pid namespace init
Posted by ebiederm on Wed, 12 Dec 2007 12:58:27 GMT

View Forum Message <> Reply to Message

This patch extends sit_init_drop to take a sender pid so
it can make decisions about dropping signals based on
the sender of the signal.

With knowledge of the signal sender sig_init_drop is extended
to drop signals meant for the pid namespace init processes
as well as the global init process. However only signals
with senders contained in the pid namespace of the init
process are dropped. Ensuring that outside of the pid
namespace the pid namespace init process continues to look
like an ordinary process.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

```
---
 kernel/signal.c |   24 +++++++++++++++---------
 1 files changed, 15 insertions(+), 9 deletions(-)

diff --git a/kernel/signal.c b/kernel/signal.c
index 029a45d..074905f 100644
--- a/kernel/signal.c
+++ b/kernel/signal.c
@@ -64,20 +64,26 @@ static int sig_ignored(struct task_struct *t, int sig)
```

```
   (handler == SIG_DFL && sig_kernel_ignore(sig));
 }

-static int is_sig_init(struct task_struct *tsk)
+static int is_sig_init(struct task_struct *init, struct pid *sender)
 {
- if (likely(!is_global_init(tsk->group_leader)))
+ if (!is_container_init(init))
+  return 0;
+
+ if (!sender)
+  sender = task_tgid(current);
+
+ if (!pid_in_pid_ns(sender, task_active_pid_ns(init)))
   return 0;

  return 1;
 }

-static int sig_init_drop(struct task_struct *tsk, int sig)
+static int sig_init_drop(struct task_struct *tsk, int sig, struct pid *sender)
 {
  /* All signals for which init has a SIG_DFL handler are
   * silently dropped without being sent.
   */
- if (!is_sig_init(tsk))
+ if (!is_sig_init(tsk, sender))
   return 0;

  return (tsk->sighand->action[sig-1].sa.sa_handler == SIG_DFL);
@@ -854,7 +860,7 @@ force_sig_info(int sig, struct siginfo *info, struct task_struct *t)

  spin_lock_irqsave(&t->sighand->siglock, flags);
  ret = 0;
- if (sig_init_drop(t, sig))
+ if (sig_init_drop(t, sig, NULL))
   goto out;
  action = &t->sighand->action[sig-1];
  ignored = action->sa.sa_handler == SIG_IGN;
@@ -985,7 +991,7 @@ __group_send_sig_info(int sig, struct siginfo *info, struct task_struct *p,
  int ret = 0;

  assert_spin_locked(&p->sighand->siglock);
- if (sig_init_drop(p, sig))
+ if (sig_init_drop(p, sig, sender))
   return ret;

  handle_stop_signal(sig, p);
```

```
@@ -1251,7 +1257,7 @@ send_sig_info(int sig, struct siginfo *info, struct task_struct *p)
  read_lock(&tasklist_lock);
  spin_lock_irqsave(&p->sighand->siglock, flags);
  ret = 0;
- if (!sig_init_drop(p, sig))
+ if (!sig_init_drop(p, sig, NULL))
   ret = specific_send_sig_info(sig, info, p);
  spin_unlock_irqrestore(&p->sighand->siglock, flags);
  read_unlock(&tasklist_lock);
@@ -1420,7 +1426,7 @@ send_group_sigqueue(int sig, struct sigqueue *q, struct task_struct *p)
  read_lock(&tasklist_lock);
  /* Since it_lock is held, p->sighand cannot be NULL. */
  spin_lock_irqsave(&p->sighand->siglock, flags);
- if (sig_init_drop(p, sig)) {
+ if (sig_init_drop(p, sig, NULL)) {
   ret = 1;
   goto out;
  }
@@ -2299,7 +2305,7 @@ static int do_tkill(int tgid, int pid, int sig)
    */
   if (!error && sig && p->sighand) {
    spin_lock_irq(&p->sighand->siglock);
-   if (!sig_init_drop(p, sig)) {
+   if (!sig_init_drop(p, sig, NULL)) {
     handle_stop_signal(sig, p);
     error = specific_send_sig_info(sig, &info, p);
    }
--
1.5.3.rc6.17.g1911
```

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

Subject: [PATCH 0/4] pid namespace infrastructure cleanups
Posted by ebiederm on Wed, 12 Dec 2007 13:09:42 GMT

View Forum Message <> Reply to Message

These next 4 patches are just generally nice cleanups to the pid
namespace infrastructure. There is a slight context dependence on the
previous patchset but nothing fundamental.

These patches remove yet another special case from de_thread.
Remove special cases from proc_get_sb
Kill proc_mnt
Move the pid_namespace logic out of the fork fast path.

Just generally making those code paths much nicer to live with.

Eric

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

Subject: [PATCH 1/4] pidns: Remove the child_reaper special case from de_thread.
Posted by ebiederm on Wed, 12 Dec 2007 13:27:00 GMT

View Forum Message <> Reply to Message

This patch inspired by Oleg's leader_pid change for the timer code now
tracks the child_reaper by pid instead of by task_struct, allowing us
to not need to update anything when de_thread is run.

The logic in de_thread that is supposed to let a threaded init call
exec has bit rotted. Since the signal drop code does not look at
child_reaper changing child_reaper early is of no help. Since
zap_other_threads uses sigaddset my previous patch to drop signals
sent to init before queueing them actually fixes this case.

Since the child_reaper never exits we don't have to worry about
reference counting the pid.

By tracking the child_reaper of a pid namespace by it's task group id,
we don't need to do anything when the leading task in the task group
exits.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

```
---
 fs/exec.c                     |    8 --------
 include/linux/pid_namespace.h |    9 ++-------
 init/main.c                   |    2 +-
 kernel/fork.c                 |    2 +-
 kernel/pid.c                  |   10 +++++++++-
 5 files changed, 13 insertions(+), 18 deletions(-)

diff --git a/fs/exec.c b/fs/exec.c
index f73edfc..96b4822 100644
--- a/fs/exec.c
+++ b/fs/exec.c
@@ -770,14 +770,6 @@ static int de_thread(struct task_struct *tsk)
   return -EAGAIN;
  }
```

```
- /*
-  * child_reaper ignores SIGKILL, change it now.
-  * Reparenting needs write_lock on tasklist_lock,
-  * so it is safe to do it under read_lock.
-  */
- if (unlikely(tsk->group_leader == task_child_reaper(tsk)))
-  task_active_pid_ns(tsk)->child_reaper = tsk;
-
  sig->group_exit_task = tsk;
  zap_other_threads(tsk);
  read_unlock(&tasklist_lock);
diff --git a/include/linux/pid_namespace.h b/include/linux/pid_namespace.h
index ab13faa..2e67033 100644
--- a/include/linux/pid_namespace.h
+++ b/include/linux/pid_namespace.h
@@ -18,7 +18,7 @@ struct pid_namespace {
  struct kref kref;
  struct pidmap pidmap[PIDMAP_ENTRIES];
  int last_pid;
- struct task_struct *child_reaper;
+ struct pid *child_reaper;
  struct kmem_cache *pid_cachep;
  int level;
  struct pid_namespace *parent;
@@ -75,11 +75,6 @@ static inline void zap_pid_ns_processes(struct pid_namespace *ns)
 #endif /* CONFIG_PID_NS */

 extern struct pid_namespace *task_active_pid_ns(struct task_struct *tsk);
-
-static inline struct task_struct *task_child_reaper(struct task_struct *tsk)
-{
- BUG_ON(tsk != current);
- return tsk->nsproxy->pid_ns->child_reaper;
-}
+extern struct task_struct *task_child_reaper(struct task_struct *tsk);

 #endif /* _LINUX_PID_NS_H */
diff --git a/init/main.c b/init/main.c
index 5c6df6b..57404b6 100644
--- a/init/main.c
+++ b/init/main.c
@@ -828,7 +828,7 @@ static int __init kernel_init(void * unused)
   * assumptions about where in the task array this
   * can be found.
   */
- init_pid_ns.child_reaper = current;
+ init_pid_ns.child_reaper = task_pid(current);
```

```
  cad_pid = task_pid(current);

diff --git a/kernel/fork.c b/kernel/fork.c
index a210860..20d26a5 100644
--- a/kernel/fork.c
+++ b/kernel/fork.c
@@ -1305,7 +1305,7 @@ static struct task_struct *copy_process(unsigned long clone_flags,

   if (thread_group_leader(p)) {
    if (clone_flags & CLONE_NEWPID)
-    p->nsproxy->pid_ns->child_reaper = p;
+    p->nsproxy->pid_ns->child_reaper = pid;

    p->signal->leader_pid = pid;
    p->signal->tty = current->signal->tty;
diff --git a/kernel/pid.c b/kernel/pid.c
index 873c00f..294fc28 100644
--- a/kernel/pid.c
+++ b/kernel/pid.c
@@ -76,7 +76,7 @@ struct pid_namespace init_pid_ns = {
  },
  .last_pid = 0,
  .level = 0,
- .child_reaper = &init_task,
+ .child_reaper = &init_struct_pid,
 };
 EXPORT_SYMBOL_GPL(init_pid_ns);

@@ -484,6 +484,14 @@ struct pid_namespace *task_active_pid_ns(struct task_struct *tsk)
 }
 EXPORT_SYMBOL_GPL(task_active_pid_ns);

+struct task_struct *task_child_reaper(struct task_struct *tsk)
+{
+ struct pid_namespace *ns;
+ BUG_ON(tsk != current);
+ ns = task_active_pid_ns(tsk);
+ return pid_task(ns->child_reaper, PIDTYPE_PID);
+}
+
 /*
  * Used by proc to find the first pid that is greater then or equal to nr.
  *
--
1.5.3.rc6.17.g1911
```

____
Containers mailing list

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

Subject: [PATCH 2/4] proc: Simplify proc_get_sb.
Posted by ebiederm on Wed, 12 Dec 2007 13:30:15 GMT

View Forum Message <> Reply to Message

The idle_thread now has a struct pid, so we can always find out know
the pid of the child_reaper before we mount proc.

Therefore we can remove the special cases for getting the pid of the
child_reaper from proc_get_sb.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

```
---
 fs/proc/root.c |   17 +----------------
 1 files changed, 1 insertions(+), 16 deletions(-)

diff --git a/fs/proc/root.c b/fs/proc/root.c
index 81f99e6..f442967 100644
--- a/fs/proc/root.c
+++ b/fs/proc/root.c
@@ -46,17 +46,6 @@ static int proc_get_sb(struct file_system_type *fs_type,
  struct pid_namespace *ns;
  struct proc_inode *ei;
 
- if (proc_mnt) {
-  /* Seed the root directory with a pid so it doesn't need
-   * to be special in base.c.  I would do this earlier but
-   * the only task alive when /proc is mounted the first time
-   * is the init_task and it doesn't have any pids.
-   */
-  ei = PROC_I(proc_mnt->mnt_sb->s_root->d_inode);
-  if (!ei->pid)
-   ei->pid = find_get_pid(1);
- }
-
  if (flags & MS_KERNMOUNT)
   ns = (struct pid_namespace *)data;
  else
@@ -76,11 +65,7 @@ static int proc_get_sb(struct file_system_type *fs_type,
   }
 
   ei = PROC_I(sb->s_root->d_inode);
-  if (!ei->pid) {
-   rcu_read_lock();
-   ei->pid = get_pid(find_pid_ns(1, ns));
-   rcu_read_unlock();
-  }
+  ei->pid = get_pid(ns->child_reaper);

   sb->s_flags |= MS_ACTIVE;
   ns->proc_mnt = mnt;
--
1.5.3.rc6.17.g1911
```

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

Subject: [PATCH 3/4] proc: Remove the unnecessary global proc_mnt
Posted by ebiederm on Wed, 12 Dec 2007 13:31:22 GMT

View Forum Message <> Reply to Message

There are no longer any users of the proc_mnt global
and having it suggests that we have a single instance
of proc, so just kill it.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

```
---
 fs/proc/inode.c         |    2 --
 fs/proc/root.c          |    5 ++---
 include/linux/proc_fs.h |    1 -
 3 files changed, 2 insertions(+), 6 deletions(-)

diff --git a/fs/proc/inode.c b/fs/proc/inode.c
index 82b3a1b..96446ac 100644
--- a/fs/proc/inode.c
+++ b/fs/proc/inode.c
@@ -71,8 +71,6 @@ static void proc_delete_inode(struct inode *inode)
  clear_inode(inode);
 }

-struct vfsmount *proc_mnt;
-
 static struct kmem_cache * proc_inode_cachep;

 static struct inode *proc_alloc_inode(struct super_block *sb)
diff --git a/fs/proc/root.c b/fs/proc/root.c
index f442967..e4b7c5a 100644
--- a/fs/proc/root.c
+++ b/fs/proc/root.c
@@ -97,9 +97,8 @@ void __init proc_root_init(void)
```

```
  err = register_filesystem(&proc_fs_type);
  if (err)
   return;
- proc_mnt = kern_mount_data(&proc_fs_type, &init_pid_ns);
- err = PTR_ERR(proc_mnt);
- if (IS_ERR(proc_mnt)) {
+ err = pid_ns_prepare_proc(&init_pid_ns);
+ if (err) {
   unregister_filesystem(&proc_fs_type);
   return;
  }
diff --git a/include/linux/proc_fs.h b/include/linux/proc_fs.h
index a5ab8ad..40da820 100644
--- a/include/linux/proc_fs.h
+++ b/include/linux/proc_fs.h
@@ -128,7 +128,6 @@ extern struct proc_dir_entry *create_proc_entry(const char *name,
mode_t mode,
       struct proc_dir_entry *parent);
 extern void remove_proc_entry(const char *name, struct proc_dir_entry *parent);

-extern struct vfsmount *proc_mnt;
 struct pid_namespace;
 extern int proc_fill_super(struct super_block *);
 extern struct inode *proc_get_inode(struct super_block *, unsigned int, struct proc_dir_entry *);
--
1.5.3.rc6.17.g1911
```

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

Subject: [PATCH 4/4] pid: Move all of the pid_namespace logic into copy_pid_ns.
Posted by ebiederm on Wed, 12 Dec 2007 13:33:27 GMT

View Forum Message <> Reply to Message

Currently we have a bunch of pid namespace logic unnecessarily
scattered through fork, this patch moves it into copy_pid_ns
where it is easy to find and review.

To do this the prototype of copy_pid_ns is changed to take
a task parameter and the pid passed into copy_process
is passed in the task_pid field of that task.

After this cleanup things are actually clean enough we can contemplate
what an unshare of the pid namespace would mean.

```
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 include/linux/pid_namespace.h |    5 +++--
 kernel/fork.c                 |   16 +++++-----------
 kernel/nsproxy.c              |    2 +-
 kernel/pid_namespace.c        |   31 ++++++++++++++++++++++++++++---
 4 files changed, 37 insertions(+), 17 deletions(-)

diff --git a/include/linux/pid_namespace.h b/include/linux/pid_namespace.h
index 2e67033..8e2e346 100644
--- a/include/linux/pid_namespace.h
+++ b/include/linux/pid_namespace.h
@@ -37,7 +37,7 @@ 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 task_struct *tsk);
 extern void free_pid_ns(struct kref *kref);
 extern void zap_pid_ns_processes(struct pid_namespace *pid_ns);

@@ -56,8 +56,9 @@ static inline struct pid_namespace *get_pid_ns(struct pid_namespace *ns)
 }

 static inline struct pid_namespace *
-copy_pid_ns(unsigned long flags, struct pid_namespace *ns)
+copy_pid_ns(unsigned long flags, struct task_struct *task)
 {
+ struct pid_namespace *ns = task->nsproxy->pid_ns;
  if (flags & CLONE_NEWPID)
   ns = ERR_PTR(-EINVAL);
  return ns;
diff --git a/kernel/fork.c b/kernel/fork.c
index 20d26a5..bbd8bcd 100644
--- a/kernel/fork.c
+++ b/kernel/fork.c
@@ -50,7 +50,6 @@
 #include <linux/taskstats_kern.h>
 #include <linux/random.h>
 #include <linux/tty.h>
-#include <linux/proc_fs.h>
 #include <linux/memcontrol.h>

 #include <asm/pgtable.h>
@@ -1013,6 +1012,9 @@ static struct task_struct *copy_process(unsigned long clone_flags,
  if (!p)
   goto fork_out;
```

```
+ /* Remember the pid */
+ p->pids[PIDTYPE_PID].pid = pid;
+
  rt_mutex_init_task(p);

 #ifdef CONFIG_TRACE_IRQFLAGS
@@ -1162,17 +1164,12 @@ static struct task_struct *copy_process(unsigned long clone_flags,
  if (retval)
   goto bad_fork_cleanup_namespaces;

- if (pid != &init_struct_pid) {
+ pid = task_pid(p);
+ if (!pid) {
   retval = -ENOMEM;
   pid = alloc_pid(p->nsproxy->pid_ns);
   if (!pid)
    goto bad_fork_cleanup_namespaces;
-
-  if (clone_flags & CLONE_NEWPID) {
-   retval = pid_ns_prepare_proc(p->nsproxy->pid_ns);
-   if (retval < 0)
-    goto bad_fork_free_pid;
-  }
  }

  p->pid = pid_nr(pid);
@@ -1304,9 +1301,6 @@ static struct task_struct *copy_process(unsigned long clone_flags,
    __ptrace_link(p, current->parent);

   if (thread_group_leader(p)) {
-   if (clone_flags & CLONE_NEWPID)
-    p->nsproxy->pid_ns->child_reaper = pid;
-
    p->signal->leader_pid = pid;
    p->signal->tty = current->signal->tty;
    set_task_pgrp(p, task_pgrp_nr(current));
diff --git a/kernel/nsproxy.c b/kernel/nsproxy.c
index f5d332c..7b6d2dc 100644
--- a/kernel/nsproxy.c
+++ b/kernel/nsproxy.c
@@ -75,7 +75,7 @@ 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);
  if (IS_ERR(new_nsp->pid_ns)) {
   err = PTR_ERR(new_nsp->pid_ns);
   goto out_pid;
diff --git a/kernel/pid_namespace.c b/kernel/pid_namespace.c
index 6d792b6..739a72b 100644
--- a/kernel/pid_namespace.c
+++ b/kernel/pid_namespace.c
@@ -12,6 +12,7 @@
 #include <linux/pid_namespace.h>
 #include <linux/syscalls.h>
 #include <linux/err.h>
+#include <linux/proc_fs.h>

 #define BITS_PER_PAGE  (PAGE_SIZE*8)

@@ -115,9 +116,12 @@ 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 task_struct *tsk)
 {
+ struct pid_namespace *old_ns = tsk->nsproxy->pid_ns;
  struct pid_namespace *new_ns;
+ struct pid *pid = NULL;
+ int err;

  BUG_ON(!old_ns);
  new_ns = get_pid_ns(old_ns);
@@ -129,13 +133,34 @@ struct pid_namespace *copy_pid_ns(unsigned long flags, struct
pid_namespace *old
   goto out_put;

  new_ns = create_pid_namespace(old_ns->level + 1);
- if (!IS_ERR(new_ns))
-  new_ns->parent = get_pid_ns(old_ns);
+ if (IS_ERR(new_ns))
+  goto out_put;
+
+ new_ns->parent = get_pid_ns(old_ns);
+ pid = task_pid(tsk);
+ if (!pid) {
+  err = -ENOMEM;
+  tsk->pids[PIDTYPE_PID].pid = pid = alloc_pid(new_ns);
+  if (!pid)
+   goto out_put_new;
+ }
+
+ new_ns->child_reaper = pid;
+ err = pid_ns_prepare_proc(new_ns);
+ if (err)
+  goto out_put_new;

 out_put:
  put_pid_ns(old_ns);
 out:
  return new_ns;
+
+out_put_new:
+ if (pid)
+  free_pid(pid);
+ put_pid_ns(new_ns);
+ new_ns = ERR_PTR(err);
+ goto out_put;
 }

 void free_pid_ns(struct kref *kref)
--
1.5.3.rc6.17.g1911
```

Subject: Re: [PATCH 2/4] proc: Simplify proc_get_sb.
Posted by Pavel Emelianov on Wed, 12 Dec 2007 13:42:21 GMT

View Forum Message <> Reply to Message

Eric W. Biederman wrote:

```
> The idle_thread now has a struct pid, so we can always find out know
> the pid of the child_reaper before we mount proc.
>
> Therefore we can remove the special cases for getting the pid of the
> child_reaper from proc_get_sb.
>
> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
> ---
>  fs/proc/root.c |   17 +----------------
>  1 files changed, 1 insertions(+), 16 deletions(-)
>
> diff --git a/fs/proc/root.c b/fs/proc/root.c
> index 81f99e6..f442967 100644
> --- a/fs/proc/root.c
> +++ b/fs/proc/root.c
> @@ -46,17 +46,6 @@ static int proc_get_sb(struct file_system_type *fs_type,
>   struct pid_namespace *ns;
>   struct proc_inode *ei;
>
> - if (proc_mnt) {
> -  /* Seed the root directory with a pid so it doesn't need
> -   * to be special in base.c.  I would do this earlier but
> -   * the only task alive when /proc is mounted the first time
> -   * is the init_task and it doesn't have any pids.
> -   */
> -  ei = PROC_I(proc_mnt->mnt_sb->s_root->d_inode);
> -  if (!ei->pid)
> -   ei->pid = find_get_pid(1);
> - }
> -
>   if (flags & MS_KERNMOUNT)
>    ns = (struct pid_namespace *)data;
>   else
> @@ -76,11 +65,7 @@ static int proc_get_sb(struct file_system_type *fs_type,
>    }
>
>    ei = PROC_I(sb->s_root->d_inode);
> -  if (!ei->pid) {
> -   rcu_read_lock();
> -   ei->pid = get_pid(find_pid_ns(1, ns));
> -   rcu_read_unlock();
> -  }
> +  ei->pid = get_pid(ns->child_reaper);
```

That's not git-bisect safe - you move the child_reaper initialization
before the call to prepare_proc only in the 4th patch.

```
>    sb->s_flags |= MS_ACTIVE;
>    ns->proc_mnt = mnt;
```

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

Subject: Re: [PATCH 2/4] proc: Simplify proc_get_sb.
Posted by ebiederm on Wed, 12 Dec 2007 14:06:18 GMT

View Forum Message <> Reply to Message

Pavel Emelyanov <xemul@openvz.org> writes:

> Eric W. Biederman wrote:
>> The idle_thread now has a struct pid, so we can always find out know
>> the pid of the child_reaper before we mount proc.

```
>>
>> Therefore we can remove the special cases for getting the pid of the
>> child_reaper from proc_get_sb.
>>
>> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
>> ---
>>  fs/proc/root.c |   17 +----------------
>>  1 files changed, 1 insertions(+), 16 deletions(-)
>>
>> diff --git a/fs/proc/root.c b/fs/proc/root.c
>> index 81f99e6..f442967 100644
>> --- a/fs/proc/root.c
>> +++ b/fs/proc/root.c
>> @@ -46,17 +46,6 @@ static int proc_get_sb(struct file_system_type *fs_type,
>>   struct pid_namespace *ns;
>>   struct proc_inode *ei;
>>
>> - if (proc_mnt) {
>> -  /* Seed the root directory with a pid so it doesn't need
>> -   * to be special in base.c.  I would do this earlier but
>> -   * the only task alive when /proc is mounted the first time
>> -   * is the init_task and it doesn't have any pids.
>> -   */
>> -  ei = PROC_I(proc_mnt->mnt_sb->s_root->d_inode);
>> -  if (!ei->pid)
>> -   ei->pid = find_get_pid(1);
>> - }
>> -
>>   if (flags & MS_KERNMOUNT)
>>    ns = (struct pid_namespace *)data;
>>   else
>> @@ -76,11 +65,7 @@ static int proc_get_sb(struct file_system_type *fs_type,
>>    }
>>
>>    ei = PROC_I(sb->s_root->d_inode);
>> -  if (!ei->pid) {
>> -   rcu_read_lock();
>> -   ei->pid = get_pid(find_pid_ns(1, ns));
>> -   rcu_read_unlock();
>> -  }
>> +  ei->pid = get_pid(ns->child_reaper);
>
> That's not git-bisect safe - you move the child_reaper initialization
> before the call to prepare_proc only in the 4th patch.
```

Bugger. I had overlooked that dependency. So patch 4 really should have been patch 2.

Eric

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

Subject: Re: [PATCH 8/9] signal: Drop signals before sending them to init.
Posted by serue on Wed, 12 Dec 2007 19:00:42 GMT

View Forum Message <> Reply to Message

Quoting Eric W. Biederman (ebiederm@xmission.com):
>
> Currently init drops all signals including SIGKILL that are sent to it
> that it does not ignore and does have a handler for. Extending this to

Description is crucial in this patchset, so should the above not be:

 "does not have a handler for"

?

Otherwise, this whole patchset looks good to me (deferring to Pavel for
his NAK on patch 5 of course).

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

thanks,
-serge

> pid namespaces where we want to maintain this semantic for signals sent
> from inside the pid namespace (descendents of init) but we want the
> namespace init to appear as a normal process is problematic, as a
> naive approach requires always tracking the sender of a signal.
>
> By making the rule (for init dropping signals):
> When sending a signal to init, the presence of a signal handler that
> is not SIG_DFL allows the signal to be sent to init. If the signal
> is not sent it is silently dropped without becoming pending.
>
> The only noticeable user space difference from todays init is that it
> no longer needs to worry about signals becoming pending when it has
> them marked as SIG_DFL and blocked.
>
> This change by making the presence of a signal handler effectively
> a permission check allows us to do all of the work before we enqueue
> the signal, and there is no need for any fancy tracking of the
> signal sender.

>
> Which means that we can now allow force_sig_info to send signals to
> init, that panic the kernel instead of going into an infinite busy
> loop taking an exception sending a signal and then retaking the same
> exception, eating all of the cpu time but accomplishing nothing.
>
> This change also makes it possible to easily implement the desired
> semantics of /sbin/init for pid namespaces where outer processes can
> kill init but processes inside the pid namespace can not.
>
> While it is now easy to remove the dropping of signals from individual
> code paths such as force_sig_info this patch does not implement that,
> to retain as much of the current behavior as possible. The only
> behavioral difference besides not queuing blocked SIG_DFL signals
> are signals directly added with sigaddset. In practice a threaded init
> now receives SIGKILL sent from a core dump, a thread group exit, or an
> exec shutting down extraneous threads.
>
> This patch was inspired by a patch from Oleg and initial refined
> in conversation with Suka and others on the containers list.
>
> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

```
> ---
>  kernel/signal.c |   47 +++++++++++++++++++++++++++++++++++++++++---------
>  1 files changed, 38 insertions(+), 9 deletions(-)
>
> diff --git a/kernel/signal.c b/kernel/signal.c
> index c01e3cd..029a45d 100644
> --- a/kernel/signal.c
> +++ b/kernel/signal.c
> @@ -64,6 +64,25 @@ static int sig_ignored(struct task_struct *t, int sig)
>    (handler == SIG_DFL && sig_kernel_ignore(sig));
>  }
>
> +static int is_sig_init(struct task_struct *tsk)
> +{
> + if (likely(!is_global_init(tsk->group_leader)))
> +  return 0;
> +
> + return 1;
> +}
> +
> +static int sig_init_drop(struct task_struct *tsk, int sig)
> +{
> + /* All signals for which init has a SIG_DFL handler are
> +  * silently dropped without being sent.
> +  */
> + if (!is_sig_init(tsk))
> +  return 0;
> +
> + return (tsk->sighand->action[sig-1].sa.sa_handler == SIG_DFL);
> +}
> +
>  /*
>   * Re-calculate pending state from the set of locally pending
>   * signals, globally pending signals, and blocked signals.
> @@ -834,6 +853,9 @@ force_sig_info(int sig, struct siginfo *info, struct task_struct *t)
>   struct k_sigaction *action;
>
>   spin_lock_irqsave(&t->sighand->siglock, flags);
> + ret = 0;
> + if (sig_init_drop(t, sig))
> +  goto out;
>   action = &t->sighand->action[sig-1];
>   ignored = action->sa.sa_handler == SIG_IGN;
>   blocked = sigismember(&t->blocked, sig);
> @@ -845,6 +867,7 @@ force_sig_info(int sig, struct siginfo *info, struct task_struct *t)
>    }
>   }
>   ret = specific_send_sig_info(sig, info, t);
> +out:
>   spin_unlock_irqrestore(&t->sighand->siglock, flags);
>
>   return ret;
> @@ -962,6 +985,9 @@ __group_send_sig_info(int sig, struct siginfo *info, struct task_struct *p,
>   int ret = 0;
>
>   assert_spin_locked(&p->sighand->siglock);
> + if (sig_init_drop(p, sig))
> +  return ret;
> +
>   handle_stop_signal(sig, p);
>
>   /* Short-circuit ignored signals.  */
> @@ -1224,7 +1250,9 @@ send_sig_info(int sig, struct siginfo *info, struct task_struct *p)
>    */
>   read_lock(&tasklist_lock);
>   spin_lock_irqsave(&p->sighand->siglock, flags);
> - ret = specific_send_sig_info(sig, info, p);
> + ret = 0;
> + if (!sig_init_drop(p, sig))
> +  ret = specific_send_sig_info(sig, info, p);
>   spin_unlock_irqrestore(&p->sighand->siglock, flags);
>   read_unlock(&tasklist_lock);
>   return ret;
> @@ -1392,6 +1420,11 @@ send_group_sigqueue(int sig, struct sigqueue *q, struct task_struct
*p)
>   read_lock(&tasklist_lock);
>   /* Since it_lock is held, p->sighand cannot be NULL. */
>   spin_lock_irqsave(&p->sighand->siglock, flags);
> + if (sig_init_drop(p, sig)) {
> +  ret = 1;
> +  goto out;
> + }
> +
>   handle_stop_signal(sig, p);
>
>   /* Short-circuit ignored signals.  */
> @@ -1844,12 +1877,6 @@ relock:
>    if (sig_kernel_ignore(signr)) /* Default is nothing. */
>     continue;
>
> -  /*
> -   * Global init gets no signals it doesn't want.
> -   */
> -  if (is_global_init(current))
> -   continue;
> -
>    if (sig_kernel_stop(signr)) {
>     /*
>      * The default action is to stop all threads in
> @@ -2272,8 +2299,10 @@ static int do_tkill(int tgid, int pid, int sig)
>     */
>    if (!error && sig && p->sighand) {
>     spin_lock_irq(&p->sighand->siglock);
> -   handle_stop_signal(sig, p);
> -   error = specific_send_sig_info(sig, &info, p);
> +   if (!sig_init_drop(p, sig)) {
> +    handle_stop_signal(sig, p);
> +    error = specific_send_sig_info(sig, &info, p);
> +   }
>     spin_unlock_irq(&p->sighand->siglock);
>    }
>   }
> --
> 1.5.3.rc6.17.g1911
```

Subject: Re: [PATCH 8/9] signal: Drop signals before sending them to init.

Posted by ebiederm on Wed, 12 Dec 2007 19:33:08 GMT
View Forum Message <> Reply to Message

"Serge E. Hallyn" <serue@us.ibm.com> writes:

> Quoting Eric W. Biederman (ebiederm@xmission.com):
>>
>> Currently init drops all signals including SIGKILL that are sent to it
>> that it does not ignore and does have a handler for. Extending this to
>
> Description is crucial in this patchset, so should the above not be:
>
> "does not have a handler for"

Yes. It should say "does not have a handler for." Instead of
"does have a handler for." Oops. I thought that was what I wrote!

>
> ?

Eric

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

Subject: Re: [PATCH 8/9] signal: Drop signals before sending them to init.

Posted by Oleg Nesterov on Thu, 13 Dec 2007 16:25:02 GMT
View Forum Message <> Reply to Message

On 12/12, Eric W. Biederman wrote:
>
> By making the rule (for init dropping signals):
> When sending a signal to init, the presence of a signal handler that
> is not SIG_DFL allows the signal to be sent to init. If the signal
> is not sent it is silently dropped without becoming pending.

But isn't it better to modify sig_ignore() and handle_stop_signal()
instead? This way we seem to need less changes,

 http://marc.info/?l=linux-kernel&m=118753610515859

(the patch above itself is not complete and a bit obsolete)

> The only noticeable user space difference from todays init is that it
> no longer needs to worry about signals becoming pending when it has
> them marked as SIG_DFL and blocked.

Ugh. I have to apologize again. I got a fever, and it turns out I just
can't read English.

So, do you mean we can ignore the problems with the signals which are
currently blocked by /sbin/init?

I personally agree, but I'm not sure I understand this right.

```
> +static int sig_init_drop(struct task_struct *tsk, int sig)
> +{
> + /* All signals for which init has a SIG_DFL handler are
> +  * silently dropped without being sent.
> +  */
> + if (!is_sig_init(tsk))
> +  return 0;
> +
> + return (tsk->sighand->action[sig-1].sa.sa_handler == SIG_DFL);
> +}
```

What if /sbin/init has a handler, but before this signal is delivered
/sbin/init does signal(SIG_DFL) ? We should modify so_sigaction() to
prevent this. Note again the patch above.

Oleg.

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

Subject: Re: [PATCH 9/9] signal: Ignore signals sent to the pid namespace init
Posted by Oleg Nesterov on Thu, 13 Dec 2007 16:28:11 GMT

View Forum Message <> Reply to Message

On 12/12, Eric W. Biederman wrote:

```
>
> -static int is_sig_init(struct task_struct *tsk)
> +static int is_sig_init(struct task_struct *init, struct pid *sender)
>  {
> - if (likely(!is_global_init(tsk->group_leader)))
> + if (!is_container_init(init))
> +  return 0;
> +
> + if (!sender)
> +  sender = task_tgid(current);
```

What if this signal is sent from the interrupt and sender == NULL?

```
> +
> + if (!pid_in_pid_ns(sender, task_active_pid_ns(init)))
>    return 0;
```

In that case the result of the above check can be wrong, no?

Oleg.

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

---

Subject: Re: [PATCH 8/9] signal: Drop signals before sending them to init.
Posted by ebiederm on Thu, 13 Dec 2007 17:50:59 GMT

View Forum Message <> Reply to Message

Oleg Nesterov <oleg@tv-sign.ru> writes:

> On 12/12, Eric W. Biederman wrote:
>>
>> By making the rule (for init dropping signals):
>> When sending a signal to init, the presence of a signal handler that
>> is not SIG_DFL allows the signal to be sent to init. If the signal
>> is not sent it is silently dropped without becoming pending.
>
> But isn't it better to modify sig_ignore() and handle_stop_signal()
> instead? This way we seem to need less changes,
>
> http://marc.info/?l=linux-kernel&m=118753610515859
>
> (the patch above itself is not complete and a bit obsolete)

No. That way leads to semantic disaster.

In particular if it is traced or blocked your patch did not
ignore the signal, the signal would be queued up. We would
not know who the sender is and thus not know the proper
disposition for the signal.

For a clean definition that we can maintain into the future
either we must either drop the signal before it is placed on the
queue or otherwise processed. Or we must track the sender.

Tracking the sender is expensive.

>> The only noticeable user space difference from todays init is that it
>> no longer needs to worry about signals becoming pending when it has
>> them marked as SIG_DFL and blocked.
>
> Ugh. I have to apologize again. I got a fever, and it turns out I just
> can't read English.
>
> So, do you mean we can ignore the problems with the signals which are
> currently blocked by /sbin/init?

Yes. Further I am saying those signals will never become pending if
we do not have a signal handler installed.

> I personally agree, but I'm not sure I understand this right.
>
>> +static int sig_init_drop(struct task_struct *tsk, int sig)
>> +{
>> + /* All signals for which init has a SIG_DFL handler are
>> + * silently dropped without being sent.
>> + */
>> + if (!is_sig_init(tsk))
>> + return 0;
>> +
>> + return (tsk->sighand->action[sig-1].sa.sa_handler == SIG_DFL);
>> +}
>
> What if /sbin/init has a handler, but before this signal is delivered
> /sbin/init does signal(SIG_DFL) ? We should modify so_sigaction() to
> prevent this. Note again the patch above.

No. We should treat signals that we process for /sbin/init completely
normally.

If we try and do the right thing with pending signals we have the question
which pid namespaces sent the signal. If the signal came from a decedent
of /sbin/init we should drop the signal. If the signal came from outside
the pid namespace of init we should keep the signal. There is no right
correct if you define what happens that way.

Therefore we make a small change to the definition of where we drop
signals to init. We always unconditionally drop them in the sender
if init has the signal handler set to SIG_DFL. Otherwise the signals
are processed normally.

This gives /sbin/init completely normal signal handling if the signal is
ever enqueued. Something trivial to implement and explain.

Theoretically the new definition can deliver a few more signals to
/sbin/init and see them processed. In practice I do not expect this
to matter. Especially since /sbin/init did ask for the signals at
one time.

Eric

Subject: Re: [PATCH 9/9] signal: Ignore signals sent to the pid namespace init
Posted by ebiederm on Thu, 13 Dec 2007 18:16:21 GMT

View Forum Message <> Reply to Message

Oleg Nesterov <oleg@tv-sign.ru> writes:

```
> On 12/12, Eric W. Biederman wrote:
>>
>> -static int is_sig_init(struct task_struct *tsk)
>> +static int is_sig_init(struct task_struct *init, struct pid *sender)
>>  {
>> - if (likely(!is_global_init(tsk->group_leader)))
>> + if (!is_container_init(init))
>> +  return 0;
>> +
>> + if (!sender)
>> +  sender = task_tgid(current);
>
> What if this signal is sent from the interrupt and sender == NULL?
>
>> +
>> + if (!pid_in_pid_ns(sender, task_active_pid_ns(init)))
>>    return 0;
>
> In that case the result of the above check can be wrong, no?
```

Yes. If we are not in process context (in_interrupt) we do infer
the sender incorrectly. Duh. I saw something in earlier patches
people had posted didn't understand it, and didn't get an answer
when I asked about it. I guess I should have thought about that
corner case and looked a little harder.

In this case the sender would always be the kernel. Oleg do you
know which signals are sent from interrupt context and through
which signal sending entry points?

I'm inclined to say:
if (!sender && in_interrupt())
 sender = &init_struct_pid;

Which will cause the signal to be dropped if SIG_DFL for the real init
(same pid namespace), and otherwise cause the signal to be sent. Which
sight unseen feels like the right thing.

It worries me that we are likely going through check_kill_permission and
all of the rest.

I expect instead of testing for in_interrupt I should be doing something
like the siginfo test in check_kill_permssion. And just lumping all of
the kernel related signals into the same bin.

I place this one on the backburner of my thoughts and come back to it in
a bit. I am close enough that it should be a simple straight forward code
change whatever the outcome.

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

---

Subject: Re: [PATCH 8/9] signal: Drop signals before sending them to init.
Posted by Oleg Nesterov on Thu, 13 Dec 2007 18:18:02 GMT

View Forum Message <> Reply to Message

On 12/13, Eric W. Biederman wrote:
>
> Oleg Nesterov <oleg@tv-sign.ru> writes:
>
> > So, do you mean we can ignore the problems with the signals which are
> > currently blocked by /sbin/init?
>
> Yes. Further I am saying those signals will never become pending if
> we do not have a signal handler installed.

OK, if we change the semantics for /sbin/init signals we can avoid
a lot of problems,

> > I personally agree, but I'm not sure I understand this right.
> >
> >> +static int sig_init_drop(struct task_struct *tsk, int sig)
> >> +{
> >> + /* All signals for which init has a SIG_DFL handler are

> >> + * silently dropped without being sent.
> >> + */
> >> + if (!is_sig_init(tsk))
> >> + return 0;
> >> +
> >> + return (tsk->sighand->action[sig-1].sa.sa_handler == SIG_DFL);
> >> +}
> >
> > What if /sbin/init has a handler, but before this signal is delivered
> > /sbin/init does signal(SIG_DFL) ? We should modify so_sigaction() to
> > prevent this. Note again the patch above.
>
> No. We should treat signals that we process for /sbin/init completely
> normally.

... including this one. I am not arguing.

> This gives /sbin/init completely normal signal handling if the signal is
> ever enqueued. Something trivial to implement and explain.

Well, I am not sure about "explain" though. Unless I missed something
this makes the semantics a bit special.

Suppose that init does sigtimedwait() but the handler == SIG_DFL.

Oleg.

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

Subject: Re: [PATCH 9/9] signal: Ignore signals sent to the pid namespace init
Posted by ebiederm on Thu, 13 Dec 2007 18:33:10 GMT

View Forum Message <> Reply to Message

ebiederm@xmission.com (Eric W. Biederman) writes:
>
> Yes. If we are not in process context (in_interrupt) we do infer
> the sender incorrectly. Duh. I saw something in earlier patches
> people had posted didn't understand it, and didn't get an answer
> when I asked about it.

Grr. Rather I didn't see a reply when I asked about it. I just
spotted Suka's old reply telling me it was to attempt to prevent
init from getting sigio.   Clearly I have a big fat blind spot
here you could drive a truck through.

Kernel generated signals are a pain because they are neither fish nor
fowl. They are sent privileged, but we don't necessarily want to give
them all curtsies and deliver them. Grr.

Eric

Subject: Re: [PATCH 8/9] signal: Drop signals before sending them to init.
Posted by ebiederm on Thu, 13 Dec 2007 18:50:09 GMT

View Forum Message <> Reply to Message

Oleg Nesterov <oleg@tv-sign.ru> writes:

> OK, if we change the semantics for /sbin/init signals we can avoid
> a lot of problems,

Yes. Otherwise we must track the source of the signals.

>> No. We should treat signals that we process for /sbin/init completely
>> normally.
>
> ... including this one. I am not arguing.
>
>> This gives /sbin/init completely normal signal handling if the signal is
>> ever enqueued. Something trivial to implement and explain.
>
> Well, I am not sure about "explain" though. Unless I missed something
> this makes the semantics a bit special.

Well the semantics are a bit special for init period. I just
make them special in a slightly different way.

> Suppose that init does sigtimedwait() but the handler == SIG_DFL.

Yes that is a bit surprising. However it is still easy to explain.
The signal is never enqueued so sigtimedwait never gets the chance
to do anything with it. Interestingly enough this is not a problem
for the current sysvinit.

sysvinit does this at start up:

```
	/*
	 *	Ignore all signals.
	 */
	for(f = 1; f <= NSIG; f++)
		SETSIG(sa, f, SIG_IGN, SA_RESTART);
```

So everything is initialized to SIG_IGN by userspace, in the common case.
Which means none of this special case logic will actually kick in, except
for SIGKILL and SIGSTOP. The signals we can't change.

Eric

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

Subject: Re: [PATCH 8/9] signal: Drop signals before sending them to init.
Posted by Oleg Nesterov on Sun, 16 Dec 2007 15:52:44 GMT

View Forum Message <> Reply to Message

On 12/13, Eric W. Biederman wrote:
>
> Oleg Nesterov <oleg@tv-sign.ru> writes:
>
> > OK, if we change the semantics for /sbin/init signals we can avoid
> > a lot of problems,
>
> Yes. Otherwise we must track the source of the signals.

Yes.

> > Well, I am not sure about "explain" though. Unless I missed something
> > this makes the semantics a bit special.
>
> Well the semantics are a bit special for init period. I just
> make them special in a slightly different way.

Yes.

But still I personally can't agree with the new behaviour.

> > Suppose that init does sigtimedwait() but the handler == SIG_DFL.
>
> Yes that is a bit surprising. However it is still easy to explain.
> The signal is never enqueued so sigtimedwait never gets the chance
> to do anything with it.

But this precisely means that sigtimedwait() is just broken for init.

Another example. /sbin/init execs, and tries to make sure it doesn't
miss the important signal (say, SIGCHLD). So it blocks SIGCHLD, execs,
and the new program installs the handler. However, the handler == SIG_DFL
right after exec, so signal could be missed.

Eric, I think the blocked signal should be enqueued. If application
blocks the signal, this is the strong indication it does care about
it, we shouldn't throw it out.

Yes, this also has surprises. If init unblocks the signal without
installing the handler, it could be killed. But this can happen with
your patch as well. sig_init_drop() returns false if we have a handler,
but this races with sys_rt_sigaction() which can set SIG_DFL, so init
could be killed.

IOW, I still have a strong feeling that this patch

http://marc.info/?l=linux-kernel&m=118753610515859

is better, and more correct. That said, this all is very subjective,
I can't "prove" this of course.

```
> Interestingly enough this is not a problem
> for the current sysvinit.
>
> sysvinit does this at start up:
>         /*
>          *      Ignore all signals.
>          */
>         for(f = 1; f <= NSIG; f++)
>                 SETSIG(sa, f, SIG_IGN, SA_RESTART);
```

Yes sure, the "good" init shouldn't have any problems with any approach.

Oleg.

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

Subject: Re: [PATCH 8/9] signal: Drop signals before sending them to init.
Posted by ebiederm on Tue, 18 Dec 2007 04:06:52 GMT
View Forum Message <> Reply to Message

Oleg Nesterov <oleg@tv-sign.ru> writes:

> On 12/13, Eric W. Biederman wrote:
>>
>> Oleg Nesterov <oleg@tv-sign.ru> writes:
>>
>> > OK, if we change the semantics for /sbin/init signals we can avoid
>> > a lot of problems,
>>
>> Yes. Otherwise we must track the source of the signals.
>
> Yes.

Good. We have fundamental agreement that we should slightly
change the definition of how signals to init are dropped.

For me the really important part is that we have a clean
definition so that we can fix bugs in the implementation.
Rather then declaring a particular implementation is the
definition and getting lost there.

>> > Well, I am not sure about "explain" though. Unless I missed something
>> > this makes the semantics a bit special.
>>
>> Well the semantics are a bit special for init period. I just
>> make them special in a slightly different way.
>
> Yes.
>
> But still I personally can't agree with the new behaviour.

You argument that a program that blocks signals wants them is
compelling, especially since blocking signals is not a common
thing to do.

So I would have no problem with a definition said signals
will be dropped when sent to init if at the time they are
sent the signal is SIG_DFL and unblocked.

>> > Suppose that init does sigtimedwait() but the handler == SIG_DFL.
>>
>> Yes that is a bit surprising. However it is still easy to explain.
>> The signal is never enqueued so sigtimedwait never gets the chance
>> to do anything with it.
>
> But this precisely means that sigtimedwait() is just broken for init.
>
> Another example. /sbin/init execs, and tries to make sure it doesn't
> miss the important signal (say, SIGCHLD). So it blocks SIGCHLD, execs,

> and the new program installs the handler. However, the handler == SIG_DFL
> right after exec, so signal could be missed.
>
> Eric, I think the blocked signal should be enqueued. If application
> blocks the signal, this is the strong indication it does care about
> it, we shouldn't throw it out.

A reasonable argument.

> Yes, this also has surprises. If init unblocks the signal without
> installing the handler, it could be killed. But this can happen with
> your patch as well. sig_init_drop() returns false if we have a handler,
> but this races with sys_rt_sigaction() which can set SIG_DFL, so init
> could be killed.

I am checking under the sighand lock so we should not race,
at least not internally to the kernel.

> IOW, I still have a strong feeling that this patch
>
> http://marc.info/?l=linux-kernel&m=118753610515859
>
> is better, and more correct. That said, this all is very subjective,
> I can't "prove" this of course.

My fundamental problem with that patch is that it drops signals
after we have started processing them, and it modifies the code
of an optimization.

To have a clean definition and clean semantics I think we need
to drop the signal earlier in the path. Which is what I
really object to in your patch.

> Yes sure, the "good" init shouldn't have any problems with any approach.

So the only day to day case we care about are programs that are not
built to be init like /bin/bash running as init. For cases like
this I can see a real advantage in not dropping blocked signals going
to init, they might even use sigtimedwait.

For day to day usage I don't see any practical difference in
dropping or not dropping blocked signals, because the default
is unblocked.

Eric

____
Containers mailing list

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

---

Subject: Re: [PATCH 9/9] signal: Ignore signals sent to the pid namespace init
Posted by ebiederm on Tue, 18 Dec 2007 08:37:40 GMT
View Forum Message <> Reply to Message

---

ebiederm@xmission.com (Eric W. Biederman) writes:

> ebiederm@xmission.com (Eric W. Biederman) writes:
>>
>> Yes. If we are not in process context (in_interrupt) we do infer
>> the sender incorrectly. Duh. I saw something in earlier patches
>> people had posted didn't understand it, and didn't get an answer
>> when I asked about it.
>
> Grr. Rather I didn't see a reply when I asked about it. I just
> spotted Suka's old reply telling me it was to attempt to prevent
> init from getting sigio.   Clearly I have a big fat blind spot
> here you could drive a truck through.
>
> Kernel generated signals are a pain because they are neither fish nor
> fowl. They are sent privileged, but we don't necessarily want to give
> them all curtsies and deliver them. Grr.

I solved this in my thinking earlier but I should really mention it.
There are two kinds of kernel generated signals.

- Services like timers, SIGIO, etc where the kernel is doing something
  on behalf of the process.
- Process management where we send SIGKILL to make a process go away.

For services we clearly want to ignore the signal.
For the kernel flexing's it's muscle we want to accept the signal,
and let init die and accept the consequences. Unfortunately we
don't do that today.

Since most signals are services and since we want to preserve today
semantics we want to drop kernel generated signals.

For those cases where it matters we can have the kernel send a signal
by a different path, that doesn't give init the option of ignoring
the signal.

Eric

Subject: Re: [PATCH 8/9] signal: Drop signals before sending them to init.
Posted by Oleg Nesterov on Tue, 18 Dec 2007 12:22:41 GMT

View Forum Message <> Reply to Message

On 12/17, Eric W. Biederman wrote:
>
> So I would have no problem with a definition said signals
> will be dropped when sent to init if at the time they are
> sent the signal is SIG_DFL and unblocked.

Great!

> > But this can happen with
> > your patch as well. sig_init_drop() returns false if we have a handler,
> > but this races with sys_rt_sigaction() which can set SIG_DFL, so init
> > could be killed.
>
> I am checking under the sighand lock so we should not race,
> at least not internally to the kernel.

Yes, but as soon as we drop ->siglock /sbin/init can set SIG_DFL before
noticing the signal.

> > IOW, I still have a strong feeling that this patch
> >
> > http://marc.info/?l=linux-kernel&m=118753610515859
> >
> > is better, and more correct. That said, this all is very subjective,
> > I can't "prove" this of course.
>
> My fundamental problem with that patch is that it drops signals
> after we have started processing them, and it modifies the code
> of an optimization.
>
> To have a clean definition and clean semantics I think we need
> to drop the signal earlier in the path. Which is what I
> really object to in your patch.

Hmm. Could you look at this patch again? I'm attaching it at the end.
(re-diffed against the current code)

It modifies sig_ignored(), so we drop the signal before we started

processing. And in fact it is more "optimized", because we don't need
to check sa_handler twice.

Btw. I don't think we should change force_sig_info(). Suppose that init
blocks/ignores SIGSEGV and do_page_fault() does force_sig_info_fault().
In that case it is better to die explicitely than go into the endless
loop.

Oleg.

```
--- t/kernel/signal.c~INITSIGS 2007-08-19 14:39:35.000000000 +0400
+++ t/kernel/signal.c 2007-08-19 19:00:27.000000000 +0400
@@ -39,11 +39,33 @@

 static struct kmem_cache *sigqueue_cachep;

+static int sig_init_ignore(struct task_struct *tsk)
+{
+ if (likely(!is_container_init(tsk->group_leader)))
+  return 0;
+
+ // ---------------- Multiple pid namespaces ----------------
+ // if (current is from tsk's parent pid_ns && !in_interrupt())
+ // return 0;
+
+ return 1;
+}
+
+static int sig_task_ignore(struct task_struct *tsk, int sig)
+{
+ void __user * handler = tsk->sighand->action[sig-1].sa.sa_handler;
+
+ if (handler == SIG_IGN)
+  return 1;
+
+ if (handler != SIG_DFL)
+  return 0;
+
+ return sig_kernel_ignore(sig) || sig_init_ignore(tsk);
+}

 static int sig_ignored(struct task_struct *t, int sig)
 {
- void __user * handler;
-
  /*
   * Tracers always want to know about signals..
   */
```

```
@@ -58,10 +82,7 @@ static int sig_ignored(struct task_struc
  if (sigismember(&t->blocked, sig) || sigismember(&t->real_blocked, sig))
   return 0;

- /* Is it explicitly or implicitly ignored? */
- handler = t->sighand->action[sig-1].sa.sa_handler;
- return   handler == SIG_IGN ||
-  (handler == SIG_DFL && sig_kernel_ignore(sig));
+ return sig_task_ignore(t, sig);
 }

 /*
@@ -566,6 +587,9 @@ static void handle_stop_signal(int sig,
    */
   return;

+ if (sig_init_ignore(p))
+  return;
+
  if (sig_kernel_stop(sig)) {
   /*
    * This is a stop signal.  Remove SIGCONT from all queues.
@@ -1786,12 +1810,6 @@ relock:
   if (sig_kernel_ignore(signr)) /* Default is nothing. */
    continue;

-  /*
-   * Global init gets no signals it doesn't want.
-   */
-  if (is_global_init(current))
-   continue;
-
   if (sig_kernel_stop(signr)) {
    /*
     * The default action is to stop all threads in
@@ -2303,8 +2316,7 @@ int do_sigaction(int sig, struct k_sigac
    *   (for example, SIGCHLD), shall cause the pending signal to
    *   be discarded, whether or not it is blocked"
    */
-  if (act->sa.sa_handler == SIG_IGN ||
-     (act->sa.sa_handler == SIG_DFL && sig_kernel_ignore(sig))) {
+  if (sig_task_ignore(current, sig)) {
    struct task_struct *t = current;
    sigemptyset(&mask);
    sigaddset(&mask, sig);
```

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

Subject: Re: [PATCH 8/9] signal: Drop signals before sending them to init.
Posted by ebiederm on Tue, 18 Dec 2007 13:36:55 GMT
View Forum Message <> Reply to Message

Oleg Nesterov <oleg@tv-sign.ru> writes:

> On 12/17, Eric W. Biederman wrote:
>>
>> So I would have no problem with a definition said signals
>> will be dropped when sent to init if at the time they are
>> sent the signal is SIG_DFL and unblocked.
>
> Great!

My only issue is that with including the blocked signals in
the definition is that it makes things a little harder to
explain and understand.

>> > But this can happen with
>> > your patch as well. sig_init_drop() returns false if we have a handler,
>> > but this races with sys_rt_sigaction() which can set SIG_DFL, so init
>> > could be killed.
>>
>> I am checking under the sighand lock so we should not race,
>> at least not internally to the kernel.
>
> Yes, but as soon as we drop ->siglock /sbin/init can set SIG_DFL before
> noticing the signal.

I guess I don't see this as a race. The definition in my head is
all about permission to send a signal to init. That permission happens
if init shows interest in the incoming signal. So it is an
instant in time decision.

At another instant the permissions may have changed and we are
allowed to send the signal.

You seem to be implying that something is wrong. If something
is wrong if something can be wrong either we have the definition wrong
or the implementation wrong. If we can not implement things to be
correct with our new definition then it is the wrong definition.
So races are totally NOT ok.

I don't think you have made the switch to the permission check
mentality that I am proposing in my definition.

Can you restate from the perspective of allowing or denying
a signal how looking at the blocked state of the signal
is better then just looking at SIG_DFL?

>> > IOW, I still have a strong feeling that this patch
>> >
>> > http://marc.info/?l=linux-kernel&m=118753610515859
>> >
>> > is better, and more correct. That said, this all is very subjective,
>> > I can't "prove" this of course.
>>
>> My fundamental problem with that patch is that it drops signals
>> after we have started processing them, and it modifies the code
>> of an optimization.
>>
>> To have a clean definition and clean semantics I think we need
>> to drop the signal earlier in the path. Which is what I
>> really object to in your patch.
>
> Hmm. Could you look at this patch again? I'm attaching it at the end.
> (re-diffed against the current code)
>
> It modifies sig_ignored(), so we drop the signal before we started
> processing. And in fact it is more "optimized", because we don't need
> to check sa_handler twice.

Yes. It is more "optimized" but from what I can tell less correct.
It makes it really easy to get the definition wrong.   The big
problem is you allow all signals through in the case of ptrace.
Which is so totally wrong. For an optimization (which sig_ignored is)
that is fine. Since this new behavior is not an optimization we
just can't do that.

The definition needs to be something that guarantees the signal
is not sent to init (not ignored by init) if the conditions are
correct.

Semantically not sent is hugely different from ignored.

Doing it in sig_ignored is just a little to late in the signal sending
pathway, and I believe it will be a maintenance nightmare.

> Btw. I don't think we should change force_sig_info(). Suppose that init
> blocks/ignores SIGSEGV and do_page_fault() does force_sig_info_fault().

> In that case it is better to die explicitely than go into the endless
> loop.

Totally and my patch would be smaller if we did. However I think we should change which set of signals are dropped in a separate patch, which is why definition of where signals are dropped and then we should explicitly let signals through in a separate patch.

>
> Oleg.
>

```
> --- t/kernel/signal.c~INITSIGS 2007-08-19 14:39:35.000000000 +0400
> +++ t/kernel/signal.c 2007-08-19 19:00:27.000000000 +0400
> @@ -39,11 +39,33 @@
>
>  static struct kmem_cache *sigqueue_cachep;
>
> +static int sig_init_ignore(struct task_struct *tsk)
> +{
> + if (likely(!is_container_init(tsk->group_leader)))
> +  return 0;
> +
> + // ---------------- Multiple pid namespaces ----------------
> + // if (current is from tsk's parent pid_ns && !in_interrupt())
> + // return 0;
```

We need to test siginfo to see if the signal is a signal from the kernel not in_interrupt(). So (before handling namespaces this would be)

```
static int sig_init_ignrore(struct task_struct *tsk, siginfo_t *info,
       struct pid *sender)
{
        /* Grumble we should look at the TGID and not need to
         * pass in group_leader.
         */
 if (likely(!is_container_init(tsk->group_leader)))
        return 0;



 /* Ignore signals from the kernel */
 if ((!is_si_special(info) && SI_FROM_KERNEL(info)) ||
     (info != SEND_SIG_NOINFO))
  return 1;

 /* If the kernel didn't send the signal figure out who did */
   if (!sender)
  sender = task_tgid(current);

 /* Don't drop user mode signals from an outer pid
  * namespace.
  */
 if (!pid_in_ns(sender, task_active_pid_ns(task)))
  return 0;

 return 1;
}
```

```
> + return 1;
> +}
> +
> +static int sig_task_ignore(struct task_struct *tsk, int sig)
> +{
> + void __user * handler = tsk->sighand->action[sig-1].sa.sa_handler;
> +
> + if (handler == SIG_IGN)
> +  return 1;
> +
> + if (handler != SIG_DFL)
> +  return 0;
> +
> + return sig_kernel_ignore(sig) || sig_init_ignore(tsk);
> +}
>
>  static int sig_ignored(struct task_struct *t, int sig)
>  {
> - void __user * handler;
> -
>   /*
>    * Tracers always want to know about signals..
>    */
```

Since we are dropping the signal before it is sent, tracers
should never see the signal.

```
> @@ -58,10 +82,7 @@ static int sig_ignored(struct task_struc
>   if (sigismember(&t->blocked, sig) || sigismember(&t->real_blocked, sig))
>    return 0;
>
> - /* Is it explicitly or implicitly ignored? */
> - handler = t->sighand->action[sig-1].sa.sa_handler;
> - return   handler == SIG_IGN ||
> -  (handler == SIG_DFL && sig_kernel_ignore(sig));
> + return sig_task_ignore(t, sig);
>  }
>
>  /*
> @@ -566,6 +587,9 @@ static void handle_stop_signal(int sig,
>     */
>    return;
>
> + if (sig_init_ignore(p))
> +  return;
> +
>   if (sig_kernel_stop(sig)) {
>    /*
>     * This is a stop signal.  Remove SIGCONT from all queues.
> @@ -1786,12 +1810,6 @@ relock:
>    if (sig_kernel_ignore(signr)) /* Default is nothing. */
>     continue;
>
> -  /*
> -   * Global init gets no signals it doesn't want.
> -   */
> -  if (is_global_init(current))
> -   continue;
> -
>    if (sig_kernel_stop(signr)) {
>     /*
>      * The default action is to stop all threads in
> @@ -2303,8 +2316,7 @@ int do_sigaction(int sig, struct k_sigac
>     *   (for example, SIGCHLD), shall cause the pending signal to
>     *   be discarded, whether or not it is blocked"
>     */
> -  if (act->sa.sa_handler == SIG_IGN ||
> -     (act->sa.sa_handler == SIG_DFL && sig_kernel_ignore(sig))) {
> +  if (sig_task_ignore(current, sig)) {
>     struct task_struct *t = current;
>     sigemptyset(&mask);
>     sigaddset(&mask, sig);
```

Any time you start ignoring the signal here you are not thinking
in terms of never sending the signal or not. Once a signal is sent
we must treat it like normal (to have a clean definition).

In the namespace case we can not look at a pending signal and decide
if we should drop it or not. So changing sigaction is impossible.

Eric

____
Containers mailing list

Subject: Re: [PATCH 8/9] signal: Drop signals before sending them to init.
Posted by Oleg Nesterov on Tue, 18 Dec 2007 15:30:07 GMT
View Forum Message <> Reply to Message

On 12/18, Eric W. Biederman wrote:
>
> Oleg Nesterov <oleg@tv-sign.ru> writes:
>
> > On 12/17, Eric W. Biederman wrote:
> >>
> >> So I would have no problem with a definition said signals
> >> will be dropped when sent to init if at the time they are
> >> sent the signal is SIG_DFL and unblocked.
> >
> > Great!
>
> My only issue is that with including the blocked signals in
> the definition is that it makes things a little harder to
> explain and understand.

Perhaps yes.

> >> > But this can happen with
> >> > your patch as well. sig_init_drop() returns false if we have a handler,
> >> > but this races with sys_rt_sigaction() which can set SIG_DFL, so init
> >> > could be killed.
> >>
> >> I am checking under the sighand lock so we should not race,
> >> at least not internally to the kernel.
> >
> > Yes, but as soon as we drop ->siglock /sbin/init can set SIG_DFL before
> > noticing the signal.
>
> I guess I don't see this as a race. The definition in my head is
> all about permission to send a signal to init. That permission happens
> if init shows interest in the incoming signal. So it is an
> instant in time decision.

Agreed.

> You seem to be implying that something is wrong. If something
> is wrong if something can be wrong either we have the definition wrong
> or the implementation wrong.

I think that the resulting behaviour is not right. /sbin/init should not
die after signal(SIG_DFL) if it has a pending !sig_kernel_ignore() signal.

Of course this doesn't depend on "should we look at the blocked state or not"
issue.

> >> My fundamental problem with that patch is that it drops signals
> >> after we have started processing them, and it modifies the code
> >> of an optimization.
> >>
> >> To have a clean definition and clean semantics I think we need
> >> to drop the signal earlier in the path. Which is what I
> >> really object to in your patch.
> >
> > Hmm. Could you look at this patch again? I'm attaching it at the end.
> > (re-diffed against the current code)
> >
> > It modifies sig_ignored(), so we drop the signal before we started
> > processing. And in fact it is more "optimized", because we don't need
> > to check sa_handler twice.
>
> Yes. It is more "optimized" but from what I can tell less correct.
> It makes it really easy to get the definition wrong.   The big
> problem is you allow all signals through in the case of ptrace.
> Which is so totally wrong.

Well yes, but I don't think this is "totally wrong". The global init can't
be ptraced, so this doesn't matter. Ptracing of the sub-namespace init from
the parent namespace is special anyway, and currently is not fully supported.

```
> > --- t/kernel/signal.c~INITSIGS 2007-08-19 14:39:35.000000000 +0400
> > +++ t/kernel/signal.c 2007-08-19 19:00:27.000000000 +0400
> > @@ -39,11 +39,33 @@
> >
> >  static struct kmem_cache *sigqueue_cachep;
> >
> > +static int sig_init_ignore(struct task_struct *tsk)
> > +{
> > + if (likely(!is_container_init(tsk->group_leader)))
> > +  return 0;
> > +
> > + // ---------------- Multiple pid namespaces ----------------
> > + // if (current is from tsk's parent pid_ns && !in_interrupt())
> > + // return 0;
```
>
> We need to test siginfo to see if the signal is a signal from
> the kernel not in_interrupt(). So (before handling namespaces
> this would be)

We can siginfo to the list of arguments. But this is another issue.
in_interrupt() just means we should not check namespace.

```
> static int sig_init_ignrore(struct task_struct *tsk, siginfo_t *info,
>       struct pid *sender)
> {
>        /* Grumble we should look at the TGID and not need to
>         * pass in group_leader.
>         */
>  if (likely(!is_container_init(tsk->group_leader)))
>         return 0;
>
>
>  /* Ignore signals from the kernel */
>  if ((!is_si_special(info) && SI_FROM_KERNEL(info)) ||
>     (info != SEND_SIG_NOINFO))
>   return 1;
>
>  /* If the kernel didn't send the signal figure out who did */
>    if (!sender)
>   sender = task_tgid(current);
```

I don't really understand why do we need this "sender". It is always
current, unless in_interrupt(). do_notify_parent() is special, yes,
but I don't see any problems here, we still can use current, no?

```
> >  static int sig_ignored(struct task_struct *t, int sig)
> >  {
> > - void __user * handler;
> > -
> >   /*
> >    * Tracers always want to know about signals..
> >    */
>
> Since we are dropping the signal before it is sent, tracers
> should never see the signal.
```

please see above.

```
> > @@ -2303,8 +2316,7 @@ int do_sigaction(int sig, struct k_sigac
> >     *   (for example, SIGCHLD), shall cause the pending signal to
> >     *   be discarded, whether or not it is blocked"
> >     */
> > -  if (act->sa.sa_handler == SIG_IGN ||
> > -     (act->sa.sa_handler == SIG_DFL && sig_kernel_ignore(sig))) {
> > +  if (sig_task_ignore(current, sig)) {
> >     struct task_struct *t = current;
> >     sigemptyset(&mask);
> >     sigaddset(&mask, sig);
>
```

> Any time you start ignoring the signal here you are not thinking
> in terms of never sending the signal or not. Once a signal is sent
> we must treat it like normal (to have a clean definition).

We don't agree here.

> In the namespace case we can not look at a pending signal and decide
> if we should drop it or not. So changing sigaction is impossible.

You mean that it is possible that this signal has come from the parent
namespace, and so we should die but not just discard the signal.

I think we can ignore this problem. If we had a handler before (when
the signal was sent), this is - imho - the correct behaviour. If not
then yes, /sbin/init can "accidentally" survive. But the parent namespace
can always use SIGKILL to really kill us.

But yes I agree, this changes one corner case to another. And let me
repeat, I don't claim that "I am right and you are not", and I can't
really prove that my approach is "technically" better. Just a personal
feeling about the "better" tradeoff. And I already said my taste is
perverted ;)

Oleg.

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

Subject: Re: [PATCH 8/9] signal: Drop signals before sending them to init.
Posted by ebiederm on Tue, 18 Dec 2007 21:34:26 GMT

View Forum Message <> Reply to Message

Oleg Nesterov <oleg@tv-sign.ru> writes:

```
>
>> > @@ -2303,8 +2316,7 @@ int do_sigaction(int sig, struct k_sigac
>> >     *   (for example, SIGCHLD), shall cause the pending signal to
>> >     *   be discarded, whether or not it is blocked"
>> >     */
>> > -  if (act->sa.sa_handler == SIG_IGN ||
>> > -      (act->sa.sa_handler == SIG_DFL && sig_kernel_ignore(sig))) {
>> > +  if (sig_task_ignore(current, sig)) {
>> >     struct task_struct *t = current;
```

>> >   sigemptyset(&mask);
>> >   sigaddset(&mask, sig);
>>
>> Any time you start ignoring the signal here you are not thinking
>> in terms of never sending the signal or not. Once a signal is sent
>> we must treat it like normal (to have a clean definition).
>
> We don't agree here.

Which seems to be the heart of the matter.

>> In the namespace case we can not look at a pending signal and decide
>> if we should drop it or not. So changing sigaction is impossible.
>
> You mean that it is possible that this signal has come from the parent
> namespace, and so we should die but not just discard the signal.

Yes.

> I think we can ignore this problem. If we had a handler before (when
> the signal was sent), this is - imho - the correct behaviour. If not
> then yes, /sbin/init can "accidentally" survive. But the parent namespace
> can always use SIGKILL to really kill us.

Only because we can't change SIGKILL to SIG_DFL.

Think of force_siginfo and what happens when we stop dropping signals
on that path. We send the signal and then before we process it
user space does signal(SIG_DFL), and we drop SIGSEGV. Ouch!

> But yes I agree, this changes one corner case to another. And let me
> repeat, I don't claim that "I am right and you are not", and I can't
> really prove that my approach is "technically" better. Just a personal
> feeling about the "better" tradeoff. And I already said my taste is
> perverted ;)

In this instance I can prove that my choice is better.

When the code is called into question and we must decided if
a code behavior is a bug or not we require a definition of
what the code is supposed to do.

Given our technical constraints of not being able to track
the source of the signal, and needing to appear as a normal
process to signal senders outside of the pid namespace we
don't have many choices of definition. The definition that
I can see is:

  Signals sent to init will be silently dropped without
  ever being sent to init, when init has the signal
  handler set to SIG_DFL.

With that definition then any time we process a signal
in handle_stop_signal or allow the signal to be processed
in because of ptrace or anything else. We are doing the
wrong thing.

That is why I drop the signal earlier. That is why I don't
do things in sigaction.

Oleg if you can show me a definition that permits the behavior
in your patch we can look at it. Currently I don't believe
that is possible.

The behavior of your code seems to be about picking convenient spots
in the code today and throwing away the signal there. Doing what is
most convenient now tends to fails miserably when you try for a
completely different implementation.

My basic contention is that without a solid definition the code
is unmaintainable, because we can't tell bugs from features.

Oleg does picking a definition and sticking to it make sense to you?
Do you at least see where I am coming from?

Eric

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

---

Subject: Re: [PATCH 8/9] signal: Drop signals before sending them to init.
Posted by Oleg Nesterov on Wed, 19 Dec 2007 13:42:46 GMT

View Forum Message <> Reply to Message

On 12/18, Eric W. Biederman wrote:
>
> Oleg Nesterov <oleg@tv-sign.ru> writes:
> >
> >> In the namespace case we can not look at a pending signal and decide
> >> if we should drop it or not. So changing sigaction is impossible.
> >
> > You mean that it is possible that this signal has come from the parent
> > namespace, and so we should die but not just discard the signal.
>

> Yes.
>
> > I think we can ignore this problem. If we had a handler before (when
> > the signal was sent), this is - imho - the correct behaviour. If not
> > then yes, /sbin/init can "accidentally" survive. But the parent namespace
> > can always use SIGKILL to really kill us.
>
> Only because we can't change SIGKILL to SIG_DFL.
>
> Think of force_siginfo and what happens when we stop dropping signals
> on that path. We send the signal and then before we process it
> user space does signal(SIG_DFL), and we drop SIGSEGV. Ouch!

Not a problem.

First of all, this has nothing to do with init's problems, any application
can can do this with signal(SIG_IGN).

The most important case is SIGSEGV sent from do_trap/do_page_fault/etc.
Another sub-thread can "steal" the signal, but this is harmless. The signal
will be re-generated when application returns from the exception and restarts
the faulting instruction.

> > But yes I agree, this changes one corner case to another. And let me
> > repeat, I don't claim that "I am right and you are not", and I can't
> > really prove that my approach is "technically" better. Just a personal
> > feeling about the "better" tradeoff. And I already said my taste is
> > perverted ;)
>
> In this instance I can prove that my choice is better.
>
> When the code is called into question and we must decided if
> a code behavior is a bug or not we require a definition of
> what the code is supposed to do.
>
> Given our technical constraints of not being able to track
> the source of the signal, and needing to appear as a normal
> process to signal senders outside of the pid namespace we
> don't have many choices of definition. The definition that
> I can see is:
>
>   Signals sent to init will be silently dropped without
>   ever being sent to init, when init has the signal
>   handler set to SIG_DFL.
>
> With that definition then any time we process a signal
> in handle_stop_signal or allow the signal to be processed
> in because of ptrace or anything else. We are doing the

> wrong thing.

I never argued, you propose the very simple and understandable definition.

But this simple rule leads to non-obvious and not good consequences.
Imho, of course.

sigtimedwait() is broken, init can lost the signal during exec,
signal(sighandler) is safe but signal(SIG_DFL) is not.

And speaking about ptrace, it is very special anyway. Just look at
get_signal_to_deliver() which re-sends the signal after ptrace_stop().

> That is why I drop the signal earlier.

I can't understand this part of the discussion. What do you mean "earlier" ?
Note that the patch I showed changes handle_stop_signal(). Because I believe
it should be changed anyway to filter out kernel threads at least.

Regardless of which rules we use to drop the signal, I think it is more
natural to modify sig_ignored(), this also makes the patch smaller.

> Oleg if you can show me a definition that permits the behavior
> in your patch we can look at it. Currently I don't believe
> that is possible.
>
> My basic contention is that without a solid definition the code
> is unmaintainable, because we can't tell bugs from features.

No, I can't show.

Eric, let's stop here. I don't believe we can convince each other.
This happens, and of course it is OK to have different opinions.
And in any case, I am happy with this discussion ;)

Let's go with your approach. In any case it solves the real problems
we have.

Oleg.

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