Subject: [PATCH 3/15] kern_siginfo helper
Posted by Pavel Emelianov on Thu, 26 Jul 2007 14:48:15 GMT
View Forum Message <> Reply to Message

From: Sukadev Bhattiprolu <sukadev@us.ibm.com>

get_signal_to_deliver() checks to ensure that /sbin/init does not
receive any unwanted signals. With implementation of multiple pid
namespaces, we need to extend this check for all "container-init"
processes (processes with pid == 1 in the pid namespace)

IOW, A container-init process, must not receive unwanted signals from
within the container. However, the container-init must receive and
honor any signals it receives from an ancestor pid namespace (i.e it
must appear like a normal process in its parent namespace).

i.e for correct processing of the signal, either:

- the recepient of the signal (in get_signal_to_deliver()) must
  have some information (such as pid namespace) of the sender

- or the sender of the signal (in send_signal()) should know
  whether the signal is an "unwanted" signal from the recepient's
  POV.

This patch provides a simple mechanism, to pass additional information from
the sender to the recepient (from send_signal() to get_signal_to_deliver()).

This patch is just a helper and a follow-on patch will use this infrastructure
to actually pass in information about the sender.

Implementation note:

Most signal interfaces in the kernel operate on 'siginfo_t' data
structure which is user-visible and cannot be easily modified/
extended. So this patch defines a wrapper, 'struct kern_siginfo',
around the 'siginfo_t' and allows extending this wrapper for future
needs.

TODO: This is more an exploratory patch and modifies only interfaces
necessary to implement correct signal semantics in pid namespaces.

If the approach is feasible, we could consistently use 'kern_siginfo'
in other signal interfaces and possibly in 'struct sigqueue'.

We could modify dequeue_signal() to directly work with 'kern_siginfo'
and remove dequeue_signal_kern_info().

Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>

---

 include/linux/signal.h | 7 +++++++
 kernel/signal.c | 38 ++++++++++++++++++++++++++++++++------
 2 files changed, 38 insertions(+), 7 deletions(-)

diff -upr linux-2.6.23-rc1-mm1.orig/include/linux/signal.h
linux-2.6.23-rc1-mm1-7/include/linux/signal.h
--- linux-2.6.23-rc1-mm1.orig/include/linux/signal.h 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/include/linux/signal.h 2007-07-26 16:36:37.000000000 +0400
@@ -7,6 +7,13 @@
 #ifdef __KERNEL__
 #include <linux/list.h>

+struct kern_siginfo {
+ siginfo_t *info;
+ int flags;
+};
+
+#define KERN_SIGINFO_CINIT 0x1 /* True iff signalling container-init */
+
 /*
  * Real Time signals may be queued.
  */
diff -upr linux-2.6.23-rc1-mm1.orig/kernel/signal.c linux-2.6.23-rc1-mm1-7/kernel/signal.c
--- linux-2.6.23-rc1-mm1.orig/kernel/signal.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/kernel/signal.c 2007-07-26 16:36:37.000000000 +0400
@@ -299,10 +299,12 @@ unblock_all_signals(void)
  spin_unlock_irqrestore(&current->sighand->siglock, flags);
 }

-static int collect_signal(int sig, struct sigpending *list, siginfo_t *info)
+static int collect_signal(int sig, struct sigpending *list,
+ struct kern_siginfo *kinfo)
 {
  struct sigqueue *q, *first = NULL;
  int still_pending = 0;
+ siginfo_t *info = kinfo->info;

  if (unlikely(!sigismember(&list->signal, sig)))
   return 0;
@@ -343,7 +348,7 @@ static int collect_signal(int sig, struc
 }

 static int __dequeue_signal(struct sigpending *pending, sigset_t *mask,
-   siginfo_t *info)

```
+   struct kern_siginfo *kinfo)
 {
  int sig = next_signal(pending, mask);

@@ -357,7 +364,7 @@ static int __dequeue_signal(struct sigpe
    }
   }

-  if (!collect_signal(sig, pending, info))
+  if (!collect_signal(sig, pending, kinfo))
    sig = 0;
  }

@@ -370,18 +377,20 @@ static int __dequeue_signal(struct sigpe
  *
  * All callers have to hold the siglock.
  */
-int dequeue_signal(struct task_struct *tsk, sigset_t *mask, siginfo_t *info)
+int dequeue_signal_kern_info(struct task_struct *tsk, sigset_t *mask,
+  struct kern_siginfo *kinfo)
 {
  int signr = 0;
+ siginfo_t *info = kinfo->info;

  /* We only dequeue private signals from ourselves, we don't let
   * signalfd steal them
   */
  if (tsk == current)
-  signr = __dequeue_signal(&tsk->pending, mask, info);
+  signr = __dequeue_signal(&tsk->pending, mask, kinfo);
  if (!signr) {
   signr = __dequeue_signal(&tsk->signal->shared_pending,
-      mask, info);
+       mask, kinfo);
   /*
    * itimer signal ?
    *
@@ -441,6 +450,22 @@ int dequeue_signal(struct task_struct *t
 }

 /*
+ * Dequeue a signal and return the element to the caller, which is
+ * expected to free it.
+ *
+ * All callers have to hold the siglock.
+ */
+int dequeue_signal(struct task_struct *tsk, sigset_t *mask, siginfo_t *info)
+{
```

```
+ struct kern_siginfo kinfo;
+
+ kinfo.info = info;
+ kinfo.flags = 0;
+
+ return dequeue_signal_kern_info(tsk, mask, &kinfo);
+}
+
+/*
  * Tell a process that it has a new active signal..
  *
  * NOTE! we rely on the previous spin_lock to
@@ -1779,6 +1873,10 @@ int get_signal_to_deliver(siginfo_t *inf
 {
  sigset_t *mask = &current->blocked;
  int signr = 0;
+ struct kern_siginfo kinfo;
+
+ kinfo.info = info;
+ kinfo.flags = 0;

  try_to_freeze();

@@ -1791,7 +1889,7 @@ relock:
       handle_group_stop())
    goto relock;

-  signr = dequeue_signal(current, mask, info);
+   signr = dequeue_signal_kern_info(current, mask, &kinfo);

   if (!signr)
    break; /* will return 0 */
```