Subject: [PATCH 2/2] Warn when container-init defaults fatal signals
Posted by Sukadev Bhattiprolu on Sat, 27 Oct 2007 19:09:28 GMT
View Forum Message <> Reply to Message

From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
Subject: [PATCH 2/2] Warn when container-init defaults fatal signals

Print a warning the first time a container-init (other than global init)
forks a child process without explicitly ignoring or handling a fatal signal.
Comments in the patch below explain the gory background :-)

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

---
 kernel/fork.c | 51 ++++++++++++++++++++++++++++++++++++++++++++++++++++
 1 file changed, 51 insertions(+)

Index: 2.6.23-mm1/kernel/fork.c
===================================================================
--- 2.6.23-mm1.orig/kernel/fork.c 2007-10-27 11:46:38.000000000 -0700
+++ 2.6.23-mm1/kernel/fork.c 2007-10-27 11:48:36.000000000 -0700
@@ -966,6 +966,53 @@ static void rt_mutex_init_task(struct ta
 }

 /*
+ * Container-init process must appear like a normal process to its sibling
+ * in the parent namespace and should be killable (or not) in the usual way.
+ *
+ * But it must be immune to any unwanted signals from within its own namespace.
+ *
+ * At the time of sending the signal, sig_init_ignore() checks and ignores
+ * if receiver is container-init and the signal is unwanted.
+ *
+ * A limitation with the check in sig_init_ignore() is that if the signal is
+ * blocked by the container-init at the time of the check, we cannot ignore
+ * the signal because the container-init may install a handler for the signal
+ * before unblocking it.
+ *
+ * But if the container-init unblocks the signal without installing the handler,
+ * the unwanted signal will still be delivered to the container-init. If the
+ * unwanted signal is fatal (i.e default action is to terminate), we end up
+ * terminating the container-init and hence the container.
+ *
+ * There does not seem to be an easy/clean way to address this blocked-signal
+ * issue in the kernel.  For now, it appears easier to let the container-init
+ * decide what it wants to do with signals i.e have it _explicitly_ ignore or
+ * handle all fatal signals.
+ *

```
+ * Following routine prints a warning if the container-init does not
+ * explicitly ignore or handle fatal signals.
+ *
+ * Return 1 if the warning is printed.  Return 0 otherwise.
+ */
+static int check_fatal_signals(struct task_struct *task)
+{
+ int i;
+
+ if (!is_container_init(task))
+  return 0;
+
+ for (i = 1; i < _NSIG; i++) {
+  if (!sig_fatal(task, i))
+   continue;
+
+  printk(KERN_WARNING "Container init %d does not handle/ignore "
+    "all fatal signals\n", task_pid_nr(task));
+  return 1;
+ }
+ return 0;
+}
+
+/*
  * This creates a new process as a copy of the old one,
  * but does not actually start it yet.
  *
@@ -983,6 +1030,10 @@ static struct task_struct *copy_process(
  int retval;
  struct task_struct *p;
  int cgroup_callbacks_done = 0;
+ static int fatal_signal_warned;
+
+ if (!is_global_init(current) && !fatal_signal_warned)
+  fatal_signal_warned = check_fatal_signals(current);

  if ((clone_flags & (CLONE_NEWNS|CLONE_FS)) == (CLONE_NEWNS|CLONE_FS))
   return ERR_PTR(-EINVAL);
```