Subject: [RFC,PATCH] fix /sbin/init signal handling
Posted by Oleg Nesterov on Sun, 19 Aug 2007 15:08:22 GMT
View Forum Message <> Reply to Message

(Not for inclusion yet, against 2.6.23-rc2, untested)

Currently, /sbin/init is protected from unhandled signals by the
"current == child_reaper(current)" check in get_signal_to_deliver().
This is not enough, we have multiple problems:

- this doesn't work for multi-threaded inits, and we can't
  fix this by simply making this check group-wide.

- /sbin/init and kernel threads are not protected from
  handle_stop_signal(). Minor problem, but not good and
  allows to "steal" SIGCONT or change ->signal->flags.

- /sbin/init is not protected from __group_complete_signal(),
  sig_fatal() can set SIGNAL_GROUP_EXIT and block exec(), kill
  sub-threads, set ->group_stop_count, etc.

Also, with support for multiple pid namespaces, we need an ability to
actually kill the sub-namespace's init from the parent namespace. In
this case it is not possible (without painful and intrusive changes)
to make the "should we honor this signal" decision on the receiver's
side.

Hopefully this patch (adds 43 bytes to kernel/signal.o) can solve
these problems.

Notes:

- Blocked signals are never ignored, so init still can receive
  a pending blocked signal after sigprocmask(SIG_UNBLOCK).
  Easy to fix, but probably we can ignore this issue.

- this patch allows us to simplify de_thread() playing games
  with pid_ns->child_reaper.

(Side note: the current behaviour of things like force_sig_info_fault()
 is not very good, init should not ignore these signals and go to the
 endless loop. Exit + panic is imho better, easy to chamge)

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 +39,35 @@

```
 static struct kmem_cache *sigqueue_cachep;

+static int sig_init_ignore(struct task_struct *tsk)
+{
+ // Currently this check is a bit racy with exec(),
+ // we can _simplify_ de_thread and close the race.
+ if (likely(!is_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))
    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);
 }
```

```
 /*
@@ -569,6 +590,9 @@ 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.
@@ -1841,14 +1865,6 @@ relock:
   if (sig_kernel_ignore(signr)) /* Default is nothing. */
    continue;

-  /*
-   * Init of a pid space gets no signals it doesn't want from
-   * within that pid space. It can of course get signals from
-   * its parent pid space.
-   */
-  if (current == child_reaper(current))
-   continue;
-
   if (sig_kernel_stop(signr)) {
    /*
     * The default action is to stop all threads in
@@ -2300,13 +2316,10 @@ int do_sigaction(int sig, struct k_sigac
  k = &current->sighand->action[sig-1];

  spin_lock_irq(&current->sighand->siglock);
- if (signal_pending(current)) {
-  /*
-   * If there might be a fatal signal pending on multiple
-   * threads, make sure we take it before changing the action.
-   */
+ if (current->signal->flags & SIGNAL_GROUP_EXIT) {
   spin_unlock_irq(&current->sighand->siglock);
-  return -ERESTARTNOINTR;
+  /* The return value doesn't matter, SIGKILL is pending */
+  return -EINTR;
  }

  if (oact)
@@ -2327,8 +2340,7 @@ 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: [RFC,PATCH] fix /sbin/init signal handling
Posted by Sukadev Bhattiprolu on Tue, 21 Aug 2007 07:10:19 GMT
View Forum Message <> Reply to Message

Oleg,

I am still reviewing this patch and will try to plug in the multiple
pid ns code and play with it some more in the next couple of days.

But am curious why we need the in_interrupt() check and that too only
for the container-init process.

Also, maybe a dumb que, are the checks for SIG_IGN and SIG_DFL required
both on sender side and the receiver side (get_signal_to_deliver())

Thanks,

Sukadev

Oleg Nesterov [oleg@tv-sign.ru] wrote:
| (Not for inclusion yet, against 2.6.23-rc2, untested)
|
| Currently, /sbin/init is protected from unhandled signals by the
| "current == child_reaper(current)" check in get_signal_to_deliver().
| This is not enough, we have multiple problems:
|
| - this doesn't work for multi-threaded inits, and we can't
| fix this by simply making this check group-wide.
|
| - /sbin/init and kernel threads are not protected from
| handle_stop_signal(). Minor problem, but not good and
| allows to "steal" SIGCONT or change ->signal->flags.
|
| - /sbin/init is not protected from __group_complete_signal(),
| sig_fatal() can set SIGNAL_GROUP_EXIT and block exec(), kill

```
|    sub-threads, set ->group_stop_count, etc.
|
| Also, with support for multiple pid namespaces, we need an ability to
| actually kill the sub-namespace's init from the parent namespace. In
| this case it is not possible (without painful and intrusive changes)
| to make the "should we honor this signal" decision on the receiver's
| side.
|
| Hopefully this patch (adds 43 bytes to kernel/signal.o) can solve
| these problems.
|
| Notes:
|
|  - Blocked signals are never ignored, so init still can receive
|    a pending blocked signal after sigprocmask(SIG_UNBLOCK).
|    Easy to fix, but probably we can ignore this issue.
|
|  - this patch allows us to simplify de_thread() playing games
|    with pid_ns->child_reaper.
|
| (Side note: the current behaviour of things like force_sig_info_fault()
|  is not very good, init should not ignore these signals and go to the
|  endless loop. Exit + panic is imho better, easy to chamge)
|
| 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 +39,35 @@
|
|  static struct kmem_cache *sigqueue_cachep;
|
| +static int sig_init_ignore(struct task_struct *tsk)
| +{
| + // Currently this check is a bit racy with exec(),
| + // we can _simplify_ de_thread and close the race.
| + if (likely(!is_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))
|     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);
|  }
|
|  /*
| @@ -569,6 +590,9 @@ 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.
| @@ -1841,14 +1865,6 @@ relock:
|     if (sig_kernel_ignore(signr)) /* Default is nothing. */
|      continue;
|
| -  /*
| -   * Init of a pid space gets no signals it doesn't want from
| -   * within that pid space. It can of course get signals from
| -   * its parent pid space.
```

```
| -   */
| -  if (current == child_reaper(current))
| -   continue;
| -
|    if (sig_kernel_stop(signr)) {
|     /*
|      * The default action is to stop all threads in
| @@ -2300,13 +2316,10 @@ int do_sigaction(int sig, struct k_sigac
|   k = &current->sighand->action[sig-1];
|
|   spin_lock_irq(&current->sighand->siglock);
| - if (signal_pending(current)) {
| -  /*
| -   * If there might be a fatal signal pending on multiple
| -   * threads, make sure we take it before changing the action.
| -   */
| + if (current->signal->flags & SIGNAL_GROUP_EXIT) {
|    spin_unlock_irq(&current->sighand->siglock);
| -  return -ERESTARTNOINTR;
| +  /* The return value doesn't matter, SIGKILL is pending */
| +  return -EINTR;
|   }
|
|   if (oact)
| @@ -2327,8 +2340,7 @@ 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: [RFC,PATCH] fix /sbin/init signal handling
Posted by Pavel Emelianov on Tue, 21 Aug 2007 09:31:24 GMT
View Forum Message <> Reply to Message

Oleg Nesterov wrote:
> (Not for inclusion yet, against 2.6.23-rc2, untested)
>
> Currently, /sbin/init is protected from unhandled signals by the

> "current == child_reaper(current)" check in get_signal_to_deliver().
> This is not enough, we have multiple problems:
>
> - this doesn't work for multi-threaded inits, and we can't
>   fix this by simply making this check group-wide.
>
> - /sbin/init and kernel threads are not protected from
>   handle_stop_signal(). Minor problem, but not good and
>   allows to "steal" SIGCONT or change ->signal->flags.
>
> - /sbin/init is not protected from __group_complete_signal(),
>   sig_fatal() can set SIGNAL_GROUP_EXIT and block exec(), kill
>   sub-threads, set ->group_stop_count, etc.
>
> Also, with support for multiple pid namespaces, we need an ability to
> actually kill the sub-namespace's init from the parent namespace. In
> this case it is not possible (without painful and intrusive changes)
> to make the "should we honor this signal" decision on the receiver's
> side.
>
> Hopefully this patch (adds 43 bytes to kernel/signal.o) can solve
> these problems.
>
> Notes:
>
> - Blocked signals are never ignored, so init still can receive
>   a pending blocked signal after sigprocmask(SIG_UNBLOCK).
>   Easy to fix, but probably we can ignore this issue.
>
> - this patch allows us to simplify de_thread() playing games
>   with pid_ns->child_reaper.
>
> (Side note: the current behaviour of things like force_sig_info_fault()
> is not very good, init should not ignore these signals and go to the
> endless loop. Exit + panic is imho better, easy to chamge)
>
> 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 +39,35 @@
>
> static struct kmem_cache *sigqueue_cachep;
>
> +static int sig_init_ignore(struct task_struct *tsk)
> +{
> + // Currently this check is a bit racy with exec(),
> + // we can _simplify_ de_thread and close the race.

> + if (likely(!is_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);
> +}

These two look like the init ignores "less" than a usual task,
i.e. the decision of whether a task has to ignore a signal depends
on whether the init has and some more. This is... strange :)

> 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))
>   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);
> }
>
> /*
> @@ -569,6 +590,9 @@ static void handle_stop_signal(int sig,
>   */

>   return;
>
> + if (sig_init_ignore(p))
> + return;
> +

Why do we need for explicit stop handling for init? Shouldn't
it be automatically checked in get_signal_to_deliver()?

>   if (sig_kernel_stop(sig)) {
>   /*
>   * This is a stop signal. Remove SIGCONT from all queues.
> @@ -1841,14 +1865,6 @@ relock:
>   if (sig_kernel_ignore(signr)) /* Default is nothing. */
>   continue;
>
> - /*
> -   * Init of a pid space gets no signals it doesn't want from
> -   * within that pid space. It can of course get signals from
> -   * its parent pid space.
> -   */
> - if (current == child_reaper(current))
> -   continue;
> -
>   if (sig_kernel_stop(signr)) {
>   /*
>   * The default action is to stop all threads in
> @@ -2300,13 +2316,10 @@ int do_sigaction(int sig, struct k_sigac
>   k = &current->sighand->action[sig-1];
>
>   spin_lock_irq(&current->sighand->siglock);
> - if (signal_pending(current)) {
> - /*
> -   * If there might be a fatal signal pending on multiple
> -   * threads, make sure we take it before changing the action.
> -   */
> + if (current->signal->flags & SIGNAL_GROUP_EXIT) {
>   spin_unlock_irq(&current->sighand->siglock);
> - return -ERESTARTNOINTR;
> + /* The return value doesn't matter, SIGKILL is pending */
> + return -EINTR;
>   }
>
>   if (oact)
> @@ -2327,8 +2340,7 @@ 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: [RFC,PATCH] fix /sbin/init signal handling
Posted by Oleg Nesterov on Tue, 21 Aug 2007 10:30:02 GMT
View Forum Message <> Reply to Message

On 08/21, sukadev@us.ibm.com wrote:
>
> I am still reviewing this patch and will try to plug in the multiple
> pid ns code and play with it some more in the next couple of days.

Thanks!

> But am curious why we need the in_interrupt() check and that too only
> for the container-init process.

For example, send_sigio(). We shouldn't send this signal to /sbin/init.
But what if the interrupted process is from the parent namespace?

> Also, maybe a dumb que, are the checks for SIG_IGN and SIG_DFL required
> both on sender side and the receiver side (get_signal_to_deliver())

get_signal_to_deliver() checks SIG_IGN/SIG_DFL, but I guess you meant
a special check for init.

Unless the patch is wrong, this is not needed. And, in fact we can't do
this anyway. Because get_signal_to_deliver() can't know if the signal
comes from parent namespace or not.

Oleg.

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

Subject: Re: [RFC,PATCH] fix /sbin/init signal handling
Posted by Oleg Nesterov on Tue, 21 Aug 2007 10:40:42 GMT
View Forum Message <> Reply to Message

On 08/21, Pavel Emelyanov wrote:
>
> >+static int sig_init_ignore(struct task_struct *tsk)
> >+{
> >+ // Currently this check is a bit racy with exec(),
> >+ // we can _simplify_ de_thread and close the race.
> >+ if (likely(!is_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);
> >+}
>
> These two look like the init ignores "less" than a usual task,
> i.e. the decision of whether a task has to ignore a signal depends
> on whether the init has and some more. This is... strange :)

Strange, indeed... Unless you misread the code or I misundertood your
message ;)

Could you clarify? The intended behaviour is: the SIG_DFL signal is
ignored if sig_kernel_ignore(sig) or we are /sbin/init. This means
init ignores "more", not "less". Unless I am terribly confused...

> >@@ -569,6 +590,9 @@ static void handle_stop_signal(int sig,
> >   */
> >   return;
> >
> >+ if (sig_init_ignore(p))

> >+ return;
> >+
>
> Why do we need for explicit stop handling for init? Shouldn't
> it be automatically checked in get_signal_to_deliver()?

Again, I don't quite understand what you mean.

The current behaviour is not good, we shouldn't do things like
rm_from_queue(SIGCONT) or ->signal->flags = 0 for /sbin/init.

This becomes worse with multiple namespaces if /sbin/init is ptraced
from the parent namespace (yes, such a ptracing is questionable).

Oleg.

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

---

Subject: Re: [RFC,PATCH] fix /sbin/init signal handling
Posted by serue on Tue, 21 Aug 2007 16:05:06 GMT
View Forum Message <> Reply to Message

Quoting Oleg Nesterov (oleg@tv-sign.ru):
> (Not for inclusion yet, against 2.6.23-rc2, untested)
>
> Currently, /sbin/init is protected from unhandled signals by the
> "current == child_reaper(current)" check in get_signal_to_deliver().
> This is not enough, we have multiple problems:
>
> - this doesn't work for multi-threaded inits, and we can't
>   fix this by simply making this check group-wide.
>
> - /sbin/init and kernel threads are not protected from
>   handle_stop_signal(). Minor problem, but not good and
>   allows to "steal" SIGCONT or change ->signal->flags.
>
> - /sbin/init is not protected from __group_complete_signal(),
>   sig_fatal() can set SIGNAL_GROUP_EXIT and block exec(), kill
>   sub-threads, set ->group_stop_count, etc.
>
> Also, with support for multiple pid namespaces, we need an ability to
> actually kill the sub-namespace's init from the parent namespace. In
> this case it is not possible (without painful and intrusive changes)
> to make the "should we honor this signal" decision on the receiver's

> side.
>
> Hopefully this patch (adds 43 bytes to kernel/signal.o) can solve
> these problems.
>
> Notes:
>
> - Blocked signals are never ignored, so init still can receive
>   a pending blocked signal after sigprocmask(SIG_UNBLOCK).
>   Easy to fix, but probably we can ignore this issue.
>
> - this patch allows us to simplify de_thread() playing games
>   with pid_ns->child_reaper.
>
> (Side note: the current behaviour of things like force_sig_info_fault()
> is not very good, init should not ignore these signals and go to the
> endless loop. Exit + panic is imho better, easy to chamge)
>
> 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 +39,35 @@
>
> static struct kmem_cache *sigqueue_cachep;
>
> +static int sig_init_ignore(struct task_struct *tsk)
> +{
> + // Currently this check is a bit racy with exec(),
> + // we can _simplify_ de_thread and close the race.
> + if (likely(!is_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);
> +}

Looks good. AFAICS init gets exactly those signals for which it
installed a signal handler.

>
> 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))
>   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);
> }

Looks good.

>
> /*
> @@ -569,6 +590,9 @@ 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.
> @@ -1841,14 +1865,6 @@ relock:
>   if (sig_kernel_ignore(signr)) /* Default is nothing. */
>   continue;
>
> - /*
> -   * Init of a pid space gets no signals it doesn't want from
> -   * within that pid space. It can of course get signals from

> - * its parent pid space.
> - */
> - if (current == child_reaper(current))
> - continue;
> -

Ok, so the idea is that this will now be caught when the signal is sent,
using sig_ignored(), (i.e at send_sigqueue, send_group_sigqueue,
specific_send_sig_info, and __group_send_sig_info) and so doesn't need
to be checked here?

I was hoping that meant that sig_init_ignore() would always be called
with current as the sending process, but I guess that's not the case?
At least in get_signal_to_deliver() we might resend a signal, though
I guess we assume the signal comes from current->parent, so maybe we
can pass that as an argument...

>   if (sig_kernel_stop(signr)) {
>   /*
>   * The default action is to stop all threads in
> @@ -2300,13 +2316,10 @@ int do_sigaction(int sig, struct k_sigac
>   k = &current->sighand->action[sig-1];
>
>   spin_lock_irq(&current->sighand->siglock);
> - if (signal_pending(current)) {
> - /*
> -   * If there might be a fatal signal pending on multiple
> -   * threads, make sure we take it before changing the action.
> -   */
> + if (current->signal->flags & SIGNAL_GROUP_EXIT) {
>   spin_unlock_irq(&current->sighand->siglock);
> - return -ERESTARTNOINTR;
> + /* The return value doesn't matter, SIGKILL is pending */
> + return -EINTR;
>   }

Looks right, based on the original comment.

>
>   if (oact)
> @@ -2327,8 +2340,7 @@ 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);

Haven't tested, but the patch reads good.

thanks,
-serge

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

---

Subject: Re: [RFC,PATCH] fix /sbin/init signal handling
Posted by Oleg Nesterov on Tue, 21 Aug 2007 17:04:43 GMT
View Forum Message <> Reply to Message

On 08/21, Serge E. Hallyn wrote:
>
> Quoting Oleg Nesterov (oleg@tv-sign.ru):
> > @@ -1841,14 +1865,6 @@ relock:
> > if (sig_kernel_ignore(signr)) /* Default is nothing. */
> > continue;
> >
> > - /*
> > - * Init of a pid space gets no signals it doesn't want from
> > - * within that pid space. It can of course get signals from
> > - * its parent pid space.
> > - */
> > - if (current == child_reaper(current))
> > - continue;
> > -
>
> Ok, so the idea is that this will now be caught when the signal is sent,
> using sig_ignored(), (i.e at send_sigqueue, send_group_sigqueue,
> specific_send_sig_info, and __group_send_sig_info) and so doesn't need
> to be checked here?

Yes.

> I was hoping that meant that sig_init_ignore() would always be called
> with current as the sending process, but I guess that's not the case?

Usually current == sender, but if the signal was sent from interrupt
context, current is some random process.

> At least in get_signal_to_deliver() we might resend a signal, though
> I guess we assume the signal comes from current->parent, so maybe we

> can pass that as an argument...

get_signal_to_deliver() might resend a signal, but only when current is ptraced. In that case the signal will be delivered even if we are init, no problem. (except that ptracing of sub-namespace init is problem by itself).

Thanks for looking at this!

Oleg.

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