Subject: [PATCH] Signal semantics for /sbin/init
Posted by Sukadev Bhattiprolu on Sat, 27 Oct 2007 19:00:10 GMT
View Forum Message <> Reply to Message

(This is Oleg's patch with my tweaks to compile, Oleg pls sign-off).

From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
Subject: [PATCH 1/3] Signal semantics for /sbin/init

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.

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

---
 kernel/signal.c | 44 ++++++++++++++++++++++++++++++++++----------
 1 file changed, 30 insertions(+), 14 deletions(-)

Index: 2.6.23-mm1/kernel/signal.c
===
--- 2.6.23-mm1.orig/kernel/signal.c 2007-10-27 08:48:56.000000000 -0700
+++ 2.6.23-mm1/kernel/signal.c 2007-10-27 08:49:23.000000000 -0700
@@ -26,6 +26,7 @@
 #include <linux/freezer.h>
 #include <linux/pid_namespace.h>
 #include <linux/nsproxy.h>
+#include <linux/hardirq.h>

 #include <asm/param.h>
 #include <asm/uaccess.h>
@@ -39,11 +40,32 @@

 static struct kmem_cache *sigqueue_cachep;

+static int sig_init_ignore(struct task_struct *tsk)
+{

-static int sig_ignored(struct task_struct *t, int sig)
+ // Currently this check is a bit racy with exec(),
+ // we can _simplify_ de_thread and close the race.
+ if (likely(!is_global_init(tsk->group_leader)))
+ return 0;
+
+ return 1;
+}
+
+static int sig_task_ignore(struct task_struct *tsk, int sig)
 {
- void __user * handler;
+ 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)
+{
  /*
   * Tracers always want to know about signals..
   */
@@ -58,10 +80,7 @@ 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);
 }

 /*
@@ -566,6 +585,9 @@ 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.
@@ -1860,12 +1882,6 @@ 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
@@ -2317,6 +2333,7 @@ int do_sigaction(int sig, struct k_sigac
  k = &current->sighand->action[sig-1];

  spin_lock_irq(&current->sighand->siglock);
+
  if (oact)
   *oact = *k;
```

```
@@ -2335,8 +2352,7 @@ 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);
```

Subject: Re: [PATCH] Signal semantics for /sbin/init
Posted by Dave Hansen on Mon, 29 Oct 2007 16:13:37 GMT
View Forum Message <> Reply to Message

On Sat, 2007-10-27 at 12:00 -0700, sukadev@us.ibm.com wrote:
> +static int sig_init_ignore(struct task_struct *tsk)
> +{
> -static int sig_ignored(struct task_struct *t, int sig)
> + // Currently this check is a bit racy with exec(),
> + // we can _simplify_ de_thread and close the race.
> + if (likely(!is_global_init(tsk->group_leader)))
> + return 0;
> +
> + return 1;
> +}

Umm. C++ comments? ;)

Also, I'm not sure an out-of-line function this small really needs a
likely/unlikely hint. Seems like a waste of the bytes to me.

> +static int sig_task_ignore(struct task_struct *tsk, int sig)
> {
> - void __user * handler;
> + void __user * handler = tsk->sighand->action[sig-1].sa.sa_handler;
> +
> + if (handler == SIG_IGN)
> + return 1;

Something simple like:
/*

```
 * ensure that the process is expecting to get this signal,
 * and thus won't get SIG_...
 */
```
Would be helpful to understand your motivation here.

```
> + 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)
> +{
>   /*
>    * Tracers always want to know about signals..
>    */
> @@ -58,10 +80,7 @@ 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);
>  }
```

And technically, your helper patch could be separated out from the rest here. It isn't a huge deal, but it would be nice.

```
>  /*
> @@ -566,6 +585,9 @@ static void handle_stop_signal(int sig,
>     */
>    return;
>
> + if (sig_init_ignore(p))
> +  return;
```

The function name isn't quite good enough to understand why this is here. Quick comment, maybe?

```
>   if (sig_kernel_stop(sig)) {
>    /*
>     * This is a stop signal.  Remove SIGCONT from all queues.
> @@ -1860,12 +1882,6 @@ 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
> @@ -2317,6 +2333,7 @@ int do_sigaction(int sig, struct k_sigac
> k = &current->sighand->action[sig-1];
>
> spin_lock_irq(&current->sighand->siglock);
> +
> if (oact)
> *oact = *k;

Fix that, please.

> @@ -2335,8 +2352,7 @@ 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);
-- Dave

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

---

Subject: Re: [PATCH] Signal semantics for /sbin/init
Posted by ebiederm on Mon, 29 Oct 2007 20:02:37 GMT
View Forum Message <> Reply to Message

My first impression is that this patch is generally good. However
it does not actually address the case of the init in a pid namespace.

Eric

_____
Containers mailing list
Containers@lists.linux-foundation.org

Subject: Re: [PATCH] Signal semantics for /sbin/init
Posted by ebiederm on Tue, 30 Oct 2007 01:24:34 GMT
View Forum Message <> Reply to Message

sukadev@us.ibm.com writes:

> (This is Oleg's patch with my tweaks to compile, Oleg pls sign-off).
> ---
>
> From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
> Subject: [PATCH 1/3] Signal semantics for /sbin/init
>
> 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.
>
> Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>

I kept thinking about the problem and the ignored signal issue
really disturbed me, because we did not have a clean definition
on how things are supposed to work. I do think Oleg was on the right
track.

If we make the rule:
When sending a signal to init. The presence of a signal handler
that is neither SIG_IGN nor SIG_DFL allows the signal to be sent to
init. If the signal is not sent it is silently dropped, without
becoming pending. Further if init specifies it's signal handler as
SIG_IGN or SIG_DFL all pending signals will be dropped.

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 or SIG_IGN, and
then it blocks them.

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. Which means that we can now allow
force_sig_info to send signals to init and that panic the kernel
instead of going into an infinite busy loop taking an exception
sending a signal and then retaking the same exception.

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

Please take a look and tell me what you think.

```
diff --git a/kernel/signal.c b/kernel/signal.c
index 4537bdd..79856eb 100644
--- a/kernel/signal.c
+++ b/kernel/signal.c
@@ -546,6 +546,22 @@ static int check_kill_permission(int sig, struct siginfo *info,
  return security_task_kill(t, info, sig, 0);
 }

+static int sig_available(struct task_struct *tsk, int sig)
```

```
+{
+ void __user * handler;
+ int available = 1;
+
+ if (likely(!is_global_init(tsk)))
+  goto out;
+
+ handler = tsk->sighand->action[sig-1].sa.sa_handler;
+ available = (handler != SIG_IGN) &&
+       (handler != SIG_DFL);
+out:
+ return available;
+}
+
+
 /* forward decl */
 static void do_notify_parent_cldstop(struct task_struct *tsk, int why);

@@ -948,6 +964,9 @@ __group_send_sig_info(int sig, struct siginfo *info, struct task_struct *p)
  int ret = 0;

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

  /* Short-circuit ignored signals.  */
@@ -1379,6 +1398,11 @@ 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_available(p, sig)) {
+  ret = 1;
+  goto out;
+ }
+
  handle_stop_signal(sig, p);

  /* Short-circuit ignored signals.  */
@@ -1860,12 +1884,6 @@ 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
@@ -2246,8 +2264,10 @@ 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_available(p, sig)) {
+    handle_stop_signal(sig, p);
+    error = specific_send_sig_info(sig, &info, p);
+   }
    spin_unlock_irq(&p->sighand->siglock);
   }
  }
@@ -2336,7 +2356,8 @@ int do_sigaction(int sig, struct k_sigaction *act, struct k_sigaction
*oact)
    *   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))) {
+     (act->sa.sa_handler == SIG_DFL && sig_kernel_ignore(sig)) ||
+     !sig_available(current, sig)) {
    struct task_struct *t = current;
    sigemptyset(&mask);
    sigaddset(&mask, sig);
```

Subject: Re: [PATCH] Signal semantics for /sbin/init
Posted by Dave Hansen on Tue, 30 Oct 2007 17:30:32 GMT
View Forum Message <> Reply to Message

On Mon, 2007-10-29 at 19:24 -0600, Eric W. Biederman wrote:
>
> When sending a signal to init. The presence of a signal handler
> that is neither SIG_IGN nor SIG_DFL allows the signal to be sent to
> init. If the signal is not sent it is silently dropped, without
> becoming pending. Further if init specifies it's signal handler as
> SIG_IGN or SIG_DFL all pending signals will be dropped.

Does this mean that container-init processes are specially treated when
signalled from _outside_ the container for which they are the init?

-- Dave

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

---

Subject: Re: [PATCH] Signal semantics for /sbin/init
Posted by ebiederm on Tue, 30 Oct 2007 18:09:48 GMT
View Forum Message <> Reply to Message

Dave Hansen <haveblue@us.ibm.com> writes:

> On Mon, 2007-10-29 at 19:24 -0600, Eric W. Biederman wrote:
>>
>> When sending a signal to init. The presence of a signal handler
>> that is neither SIG_IGN nor SIG_DFL allows the signal to be sent to
>> init. If the signal is not sent it is silently dropped, without
>> becoming pending. Further if init specifies it's signal handler as
>> SIG_IGN or SIG_DFL all pending signals will be dropped.
>
> Does this mean that container-init processes are specially treated when
> signalled from _outside_ the container for which they are the init?

My proof of concept patch still needs the extra logic from sukas
follow on patch.

The result is that container_init processes will _not_ be treated
specially when signaled from outside the container. They will
be treated like normal processes.

This becomes straight forward to do because we move all of the
decision logic for dropping signals into the sender.

To maintain backwards compatibility we need only drop signals that
have a signal handler of SIG_DFL, not SIG_IGN (oops).

Eric

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

---

Subject: Re: [PATCH] Signal semantics for /sbin/init
Posted by Sukadev Bhattiprolu on Tue, 30 Oct 2007 21:37:10 GMT
View Forum Message <> Reply to Message

| sukadev@us.ibm.com writes:
|
| > (This is Oleg's patch with my tweaks to compile, Oleg pls sign-off).
| > ---
| >
| > From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
| > Subject: [PATCH 1/3] Signal semantics for /sbin/init
| >
| > 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.
| >
| > Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
|
| I kept thinking about the problem and the ignored signal issue
| really disturbed me, because we did not have a clean definition
| on how things are supposed to work. I do think Oleg was on the right
| track.
|
| If we make the rule:
| When sending a signal to init. The presence of a signal handler
| that is neither SIG_IGN nor SIG_DFL allows the signal to be sent to
| init. If the signal is not sent it is silently dropped, without
| becoming pending. Further if init specifies it's signal handler as
| SIG_IGN or SIG_DFL all pending signals will be dropped.
|
| 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 or SIG_IGN, and
| then it blocks them.

Yes, thats a small change in semantics to blocked signals and
may not matter much in practice.

|
| 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. Which means that we can now allow
| force_sig_info to send signals to init and that panic the kernel
| instead of going into an infinite busy loop taking an exception
| sending a signal and then retaking the same exception.
|
| This change also makes it possible to easily implement the
| the desired semantics of /sbin/init for pid namespaces where
| outer processes can kill init but processes inside the pid
| namespace can not.
|
| Please take a look and tell me what you think.

Overall, it looks good, couple of questions below. Will port my
patches and test it out.

|
| diff --git a/kernel/signal.c b/kernel/signal.c
| index 4537bdd..79856eb 100644
| --- a/kernel/signal.c
| +++ b/kernel/signal.c
| @@ -546,6 +546,22 @@ static int check_kill_permission(int sig, struct siginfo *info,

```
|   return security_task_kill(t, info, sig, 0);
|  }
|
| +static int sig_available(struct task_struct *tsk, int sig)
```

Hmm. IMHO, 'available' is not immediately obvious/clear.

```
| +{
| + void __user * handler;
| + int available = 1;
| +
| + if (likely(!is_global_init(tsk)))
| +  goto out;
```

With multiple pid namespaces, I guess, this would become

```
 if (!is_container_init(tsk))
  goto out;

 /* from parent namespace, don't ignore */
 if (!task_in_descendant_ns(tsk))
  goto out;
```

If this is correct, I have a question below re: do_sigaction.

```
| +
| + handler = tsk->sighand->action[sig-1].sa.sa_handler;
| + available = (handler != SIG_IGN) &&
| +      (handler != SIG_DFL);
```

Can we use sig_user_defined() for the above checks ? sig_available()
looks like an extension of sig_user_defined() for init. How about
sig_init_user_defined() or reverse the logic and use sig_init_ignore()
like Oleg did ?

```
| +out:
| + return available;
| +}
| +
| +
|  /* forward decl */
|  static void do_notify_parent_cldstop(struct task_struct *tsk, int why);
|
| @@ -948,6 +964,9 @@ __group_send_sig_info(int sig, struct siginfo *info, struct task_struct *p)
|   int ret = 0;
|
|   assert_spin_locked(&p->sighand->siglock);
| + if (!sig_available(p, sig))
```

```
| +  return ret;
| +

|   handle_stop_signal(sig, p);
|
|   /* Short-circuit ignored signals.  */
| @@ -1379,6 +1398,11 @@ 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_available(p, sig)) {
| +  ret = 1;
| +  goto out;
| + }
| +
|   handle_stop_signal(sig, p);
|
|   /* Short-circuit ignored signals.  */
```

Hmm. I see now that Oleg's approach would result in the sig_init_ignore()
check being done twice (once in handle_stop_signal() and once in the following
sig_ignored()).

Is that why you don't fold sig_available() into sig_ignored() ? Or, there
other more important/correctness issues as well ?

```
| @@ -1860,12 +1884,6 @@ 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
| @@ -2246,8 +2264,10 @@ 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_available(p, sig)) {
```

```
| +    handle_stop_signal(sig, p);
| +    error = specific_send_sig_info(sig, &info, p);
| +   }
|      spin_unlock_irq(&p->sighand->siglock);
|    }
|   }
| @@ -2336,7 +2356,8 @@ int do_sigaction(int sig, struct k_sigaction *act, struct k_sigaction
*oact)
|      *   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))) {
| +     (act->sa.sa_handler == SIG_DFL && sig_kernel_ignore(sig)) ||
| +     !sig_available(current, sig)) {
|      struct task_struct *t = current;
|      sigemptyset(&mask);
|      sigaddset(&mask, sig);
```

Not sure about this. Consider that we extend the sig_available() as
I mentioned above for container_init(). Then suppose following sequence
occurs:

- container-init receives a fatal signal say SIGUSR1, from parent-ns
  i.e the signal is pending.

- before processing the pending signal, the container-init sets
  the handler to SIG_DFL (which is to terminate).

  Will we then discard the pending SIGUSR1 even though it was from
  from parent ns ?

Subject: Re: [PATCH] Signal semantics for /sbin/init
Posted by ebiederm on Tue, 30 Oct 2007 22:25:51 GMT
View Forum Message <> Reply to Message

sukadev@us.ibm.com writes:

> | 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. Which means that we can now allow
> | force_sig_info to send signals to init and that panic the kernel
> | instead of going into an infinite busy loop taking an exception

> | sending a signal and then retaking the same exception.
> |
> | This change also makes it possible to easily implement the
> | the desired semantics of /sbin/init for pid namespaces where
> | outer processes can kill init but processes inside the pid
> | namespace can not.
> |
> | Please take a look and tell me what you think.
>
> Overall, it looks good, couple of questions below. Will port my
> patches and test it out.
>
> |
> | diff --git a/kernel/signal.c b/kernel/signal.c
> | index 4537bdd..79856eb 100644
> | --- a/kernel/signal.c
> | +++ b/kernel/signal.c
> | @@ -546,6 +546,22 @@ static int check_kill_permission(int sig, struct siginfo
> *info,
> | return security_task_kill(t, info, sig, 0);
> | }
> |
> | +static int sig_available(struct task_struct *tsk, int sig)
>
> Hmm. IMHO, 'available' is not immediately obvious/clear.

Agreed. It was what I had to work with. I have since renamed it
sig_init_drop. Which is a little better.

> | +{
> | + void __user * handler;
> | + int available = 1;
> | +
> | + if (likely(!is_global_init(tsk)))
> | + goto out;
>
> With multiple pid namespaces, I guess, this would become
>
> if (!is_container_init(tsk))
>   goto out;

Although I need to make that tsk->group_leader.

> /* from parent namespace, don't ignore */
> if (!task_in_descendant_ns(tsk))
>   goto out;
>
> If this is correct, I have a question below re: do_sigaction.

> | +
> | + handler = tsk->sighand->action[sig-1].sa.sa_handler;
> | + available = (handler != SIG_IGN) &&
> | +   (handler != SIG_DFL);
>
> Can we use sig_user_defined() for the above checks ? sig_available()
> looks like an extension of sig_user_defined() for init.

Well the SIG_IGN check is actually wrong here. We should just check
for SIG_DFL if we want to maintain the most possible compatibility.

> How about
> sig_init_user_defined() or reverse the logic and use sig_init_ignore()
> like Oleg did ?

That could work.

> | +out:
> | + return available;
> | +}
> | +
> | +
> | /* forward decl */
> | static void do_notify_parent_cldstop(struct task_struct *tsk, int why);
> |
> | @@ -948,6 +964,9 @@ __group_send_sig_info(int sig, struct siginfo *info,
> struct task_struct *p)
> | int ret = 0;
> |
> | assert_spin_locked(&p->sighand->siglock);
> | + if (!sig_available(p, sig))
> | + return ret;
> | +
>
> | handle_stop_signal(sig, p);
> |
> | /* Short-circuit ignored signals. */
> | @@ -1379,6 +1398,11 @@ 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_available(p, sig)) {
> | + ret = 1;
> | + goto out;
> | + }

> | +
> | handle_stop_signal(sig, p);
> |
> | /* Short-circuit ignored signals. */
>
> Hmm. I see now that Oleg's approach would result in the sig_init_ignore()
> check being done twice (once in handle_stop_signal() and once in the following
> sig_ignored()).
>
> Is that why you don't fold sig_available() into sig_ignored() ? Or, there
> other more important/correctness issues as well ?

It is a very slight but subtle point.
I have modified the definition so that signals with a handler SIG_DFL never
reach the init process not even to the pending mask. Essentially
this means we should drop them before any attempts at processing them.

So dropping the signals before handle_stop_signal is important.

As for not folding the signals into sig_ignore. We need to drop the signal
even if the signal is masked. So it looks like an ugly special case.

In addition by not dropping the signal to init everywhere (in particular
on the paths where we force a signal) we allow the kernel to kill init
and panic the system if /sbin/init does something nasty instead of looping
forever.

> | @@ -1860,12 +1884,6 @@ 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
> | @@ -2246,8 +2264,10 @@ 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_available(p, sig)) {

> | +   handle_stop_signal(sig, p);
> | +   error = specific_send_sig_info(sig, &info, p);
> | +   }
> | spin_unlock_irq(&p->sighand->siglock);
> | }
> | }
> | @@ -2336,7 +2356,8 @@ int do_sigaction(int sig, struct k_sigaction *act,
> struct k_sigaction *oact)
> | *   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))) {
> | +   (act->sa.sa_handler == SIG_DFL && sig_kernel_ignore(sig)) ||
> | +   !sig_available(current, sig)) {
> | struct task_struct *t = current;
> | sigemptyset(&mask);
> | sigaddset(&mask, sig);
>
> Not sure about this. Consider that we extend the sig_available() as
> I mentioned above for container_init(). Then suppose following sequence
> occurs:
>
> - container-init receives a fatal signal say SIGUSR1, from parent-ns
>   i.e the signal is pending.
>
> - before processing the pending signal, the container-init sets
>   the handler to SIG_DFL (which is to terminate).
>
>   Will we then discard the pending SIGUSR1 even though it was from
>   from parent ns ?

Good question.

I guess if a signal from a child is already pending we have limited
responsibility for dropping it, because we have already allowed it
through...

My thought had been especially after I saw that part of Oleg's patch
that it was the right thing to do.

Given my attempt at well defined semantics for dropping the signal
from the sender. The rule would be if the signal makes it to the init
process we should not treat it specially.

The historical behavior would have been that if the signal was from a
child the signal would have been dropped just after this point when it
was delivered.

However if someone wants to prevent that case we can use sigwait, to
remove blocked signals. Further different rules for signal handling
for init I think are largely problematic for authors of different
inits because they are hard to remember.

Further sysvinit doesn't ever set a signal handler to SIG_DFL except
after it forks and just before it execs a process so the common case
will not be affected.

Therefore I think you are right. We don't need this case and it
is likely to be more problematic then useful.

Eric

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

---

Subject: Re: [PATCH] Signal semantics for /sbin/init
Posted by Sukadev Bhattiprolu on Fri, 02 Nov 2007 21:00:47 GMT
View Forum Message <> Reply to Message

Eric,

Can you send out your modified patch for this - I can port mine on top
and resend ?

Suka

Eric W. Biederman [ebiederm@xmission.com] wrote:
| sukadev@us.ibm.com writes:
|
| > | 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. Which means that we can now allow
| > | force_sig_info to send signals to init and that panic the kernel
| > | instead of going into an infinite busy loop taking an exception
| > | sending a signal and then retaking the same exception.
| > |
| > | This change also makes it possible to easily implement the
| > | the desired semantics of /sbin/init for pid namespaces where
| > | outer processes can kill init but processes inside the pid
| > | namespace can not.
| > |
| > | Please take a look and tell me what you think.
| >

| > Overall, it looks good, couple of questions below. Will port my
| > patches and test it out.
| >
| > |
| > | diff --git a/kernel/signal.c b/kernel/signal.c
| > | index 4537bdd..79856eb 100644
| > | --- a/kernel/signal.c
| > | +++ b/kernel/signal.c
| > | @@ -546,6 +546,22 @@ static int check_kill_permission(int sig, struct siginfo
| > *info,
| > | return security_task_kill(t, info, sig, 0);
| > | }
| > |
| > | +static int sig_available(struct task_struct *tsk, int sig)
| >
| > Hmm. IMHO, 'available' is not immediately obvious/clear.
|
| Agreed. It was what I had to work with. I have since renamed it
| sig_init_drop. Which is a little better.
|
| > | +{
| > | + void __user * handler;
| > | + int available = 1;
| > | +
| > | + if (likely(!is_global_init(tsk)))
| > | + goto out;
| >
| > With multiple pid namespaces, I guess, this would become
| >
| > if (!is_container_init(tsk))
| >   goto out;
|
| Although I need to make that tsk->group_leader.
|
| > /* from parent namespace, don't ignore */
| > if (!task_in_descendant_ns(tsk))
| >   goto out;
| >
| > If this is correct, I have a question below re: do_sigaction.
|
| > | +
| > | + handler = tsk->sighand->action[sig-1].sa.sa_handler;
| > | + available = (handler != SIG_IGN) &&
| > | +   (handler != SIG_DFL);
| >
| > Can we use sig_user_defined() for the above checks ? sig_available()
| > looks like an extension of sig_user_defined() for init.
|

| Well the SIG_IGN check is actually wrong here. We should just check
| for SIG_DFL if we want to maintain the most possible compatibility.
|
|
| > How about
| > sig_init_user_defined() or reverse the logic and use sig_init_ignore()
| > like Oleg did ?
|
| That could work.
|
| > | +out:
| > | + return available;
| > | +}
| > | +
| > | +
| > | /* forward decl */
| > | static void do_notify_parent_cldstop(struct task_struct *tsk, int why);
| > |
| > | @@ -948,6 +964,9 @@ __group_send_sig_info(int sig, struct siginfo *info,
| > struct task_struct *p)
| > | int ret = 0;
| > |
| > | assert_spin_locked(&p->sighand->siglock);
| > | + if (!sig_available(p, sig))
| > | + return ret;
| > | +
| >
| > | handle_stop_signal(sig, p);
| > |
| > | /* Short-circuit ignored signals. */
| > | @@ -1379,6 +1398,11 @@ 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_available(p, sig)) {
| > | + ret = 1;
| > | + goto out;
| > | + }
| > | +
| > | handle_stop_signal(sig, p);
| > |
| > | /* Short-circuit ignored signals. */
| >
| > Hmm. I see now that Oleg's approach would result in the sig_init_ignore()
| > check being done twice (once in handle_stop_signal() and once in the following
| > sig_ignored()).
| >

| > Is that why you don't fold sig_available() into sig_ignored() ? Or, there
| > other more important/correctness issues as well ?
|
| It is a very slight but subtle point.
| I have modified the definition so that signals with a handler SIG_DFL never
| reach the init process not even to the pending mask. Essentially
| this means we should drop them before any attempts at processing them.
|
| So dropping the signals before handle_stop_signal is important.
|
| As for not folding the signals into sig_ignore. We need to drop the signal
| even if the signal is masked. So it looks like an ugly special case.
|
| In addition by not dropping the signal to init everywhere (in particular
| on the paths where we force a signal) we allow the kernel to kill init
| and panic the system if /sbin/init does something nasty instead of looping
| forever.
|
|
| > | @@ -1860,12 +1884,6 @@ 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
| > | @@ -2246,8 +2264,10 @@ 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_available(p, sig)) {
| > | +   handle_stop_signal(sig, p);
| > | +   error = specific_send_sig_info(sig, &info, p);
| > | +   }
| > | spin_unlock_irq(&p->sighand->siglock);
| > | }
| > | }
| > | @@ -2336,7 +2356,8 @@ int do_sigaction(int sig, struct k_sigaction *act,
| > struct k_sigaction *oact)
| > | *   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))) {
| > | +   (act->sa.sa_handler == SIG_DFL && sig_kernel_ignore(sig)) ||
| > | +   !sig_available(current, sig)) {
| > | struct task_struct *t = current;
| > | sigemptyset(&mask);
| > | sigaddset(&mask, sig);
| >
| > Not sure about this. Consider that we extend the sig_available() as
| > I mentioned above for container_init(). Then suppose following sequence
| > occurs:
| >
| > - container-init receives a fatal signal say SIGUSR1, from parent-ns
| >   i.e the signal is pending.
| >
| > - before processing the pending signal, the container-init sets
| >   the handler to SIG_DFL (which is to terminate).
| >
| >   Will we then discard the pending SIGUSR1 even though it was from
| >   from parent ns ?
|
| Good question.
|
| I guess if a signal from a child is already pending we have limited
| responsibility for dropping it, because we have already allowed it
| through...
|
| My thought had been especially after I saw that part of Oleg's patch
| that it was the right thing to do.
|
| Given my attempt at well defined semantics for dropping the signal
| from the sender. The rule would be if the signal makes it to the init
| process we should not treat it specially.
|
| The historical behavior would have been that if the signal was from a
| child the signal would have been dropped just after this point when it
| was delivered.
|
| However if someone wants to prevent that case we can use sigwait, to
| remove blocked signals. Further different rules for signal handling
| for init I think are largely problematic for authors of different
| inits because they are hard to remember.
|
| Further sysvinit doesn't ever set a signal handler to SIG_DFL except
| after it forks and just before it execs a process so the common case
| will not be affected.
|

| Therefore I think you are right. We don't need this case and it
| is likely to be more problematic then useful.
|
| Eric

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

---

Subject: Re: [PATCH] Signal semantics for /sbin/init
Posted by ebiederm on Sun, 04 Nov 2007 04:20:53 GMT
View Forum Message <> Reply to Message

sukadev@us.ibm.com writes:

> Eric,
>
> Can you send out your modified patch for this - I can port mine on top
> and resend ?

Yes. I will see send that version in just a bit.

Eric

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

---

Subject: Re: [PATCH] Signal semantics for /sbin/init
Posted by Sukadev Bhattiprolu on Sat, 17 Nov 2007 17:38:22 GMT
View Forum Message <> Reply to Message

Eric W. Biederman [ebiederm@xmission.com] wrote:
| sukadev@us.ibm.com writes:
|
| > Eric,
| >
| > Can you send out your modified patch for this - I can port mine on top
| > and resend ?
|
| Yes. I will see send that version in just a bit.

Eric, can you send this when you get a chance.

_____
Containers mailing list

Subject: Re: [PATCH] Signal semantics for /sbin/init
Posted by Sukadev Bhattiprolu on Wed, 12 Dec 2007 00:49:36 GMT
View Forum Message <> Reply to Message

Eric, can you send out your modified signal semantics patch or
would you like me take a stab at addressing the couple of comments
that came up against the patches ? I feel like we were pretty
close to fix this.

Suka

sukadev@us.ibm.com [sukadev@us.ibm.com] wrote:
| Eric W. Biederman [ebiederm@xmission.com] wrote:
| | sukadev@us.ibm.com writes:
| |
| | > Eric,
| | >
| | > Can you send out your modified patch for this - I can port mine on top
| | > and resend ?
| |
| | Yes. I will see send that version in just a bit.
|
| Eric, can you send this when you get a chance.
| _____
| Containers mailing list
| Containers@lists.linux-foundation.org
| https://lists.linux-foundation.org/mailman/listinfo/containers

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