Subject: [PATCH] Mark timer_stats as incompatible with multiple pid namespaces
Posted by ebiederm on Wed, 12 Dec 2007 23:10:56 GMT
View Forum Message <> Reply to Message

/proc/timer_stats currently reports the user of a timer by pid,
which is a reasonable approach. However if you are not in
the initial pid namespace the pid that is reported is nonsense.

Therefore until we can make timer_stats pid namespace safe just
disable it in the build if pid namespace support is selected
so we at least know we have a conflict.

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

```
---
 lib/Kconfig.debug |    2 +-
 1 files changed, 1 insertions(+), 1 deletions(-)

diff --git a/lib/Kconfig.debug b/lib/Kconfig.debug
index cc42773..c8e81e6 100644
--- a/lib/Kconfig.debug
+++ b/lib/Kconfig.debug
@@ -146,7 +146,7 @@ config SCHEDSTATS

 config TIMER_STATS
  bool "Collect kernel timers statistics"
- depends on DEBUG_KERNEL && PROC_FS
+ depends on DEBUG_KERNEL && PROC_FS && !PID_NS
  help
    If you say Y here, additional code will be inserted into the
    timer routines to collect statistics about kernel timers being
--
1.5.3.rc6.17.g1911
```

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

Subject: Re: [PATCH] Mark timer_stats as incompatible with multiple pid namespaces
Posted by Ingo Molnar on Thu, 13 Dec 2007 09:01:02 GMT
View Forum Message <> Reply to Message

* Eric W. Biederman <ebiederm@xmission.com> wrote:

> /proc/timer_stats currently reports the user of a timer by pid, which
> is a reasonable approach. However if you are not in the initial pid

> namespace the pid that is reported is nonsense.
>
> Therefore until we can make timer_stats pid namespace safe just
> disable it in the build if pid namespace support is selected so we at
> least know we have a conflict.

What the heck??? Please solve this properly instead of hiding it.
/proc/timer_stats is damn useful and it's a must-have for powertop to
work.

 Ingo

Subject: Re: [PATCH] Mark timer_stats as incompatible with multiple pid namespaces
Posted by ebiederm on Thu, 13 Dec 2007 11:55:22 GMT
View Forum Message <> Reply to Message

Ingo Molnar <mingo@elte.hu> writes:

> * Eric W. Biederman <ebiederm@xmission.com> wrote:
>
>> /proc/timer_stats currently reports the user of a timer by pid, which
>> is a reasonable approach. However if you are not in the initial pid
>> namespace the pid that is reported is nonsense.
>>
>> Therefore until we can make timer_stats pid namespace safe just
>> disable it in the build if pid namespace support is selected so we at
>> least know we have a conflict.
>
> What the heck??? Please solve this properly instead of hiding it.
> /proc/timer_stats is damn useful and it's a must-have for powertop to
> work.

Hmm. Perhaps the dependency conflict should go in the other direction
then.

My goal is to document the issue while a proper fix is being written.
I have known about this for all of about 1 day now. It was added since
last time I went through the kernel and made a thorough sweep of pid users.

What the proper fix is isn't even obvious at this point.
Possibly it is making /proc/timer_stats disappear in child pid namespaces.
  Which we don't currently have the infrastructure fore.

Possibly it is reworking the stats collection so we store a struct pid *
instead of a pid_t value. So we would know if the reader of the value
can even see processes you have collected stats for.

It is going to take a bit to digest what is going on and solve this properly.

In the same vein do we actively have interesting user space programs
using /proc/sched_debug? It is the same class of problem. Yet
another interface talking to user space with pids.

Eric

Subject: Re: [PATCH] Mark timer_stats as incompatible with multiple pid namespaces
Posted by Ingo Molnar on Thu, 13 Dec 2007 13:04:23 GMT

View Forum Message <> Reply to Message

* Eric W. Biederman <ebiederm@xmission.com> wrote:

> > What the heck??? Please solve this properly instead of hiding it.
> > /proc/timer_stats is damn useful and it's a must-have for powertop
> > to work.
>
> Hmm. Perhaps the dependency conflict should go in the other direction
> then.
>
> My goal is to document the issue while a proper fix is being written.
> I have known about this for all of about 1 day now. It was added
> since last time I went through the kernel and made a thorough sweep of
> pid users.
>
> What the proper fix is isn't even obvious at this point. Possibly it
> is making /proc/timer_stats disappear in child pid namespaces.
>   Which we don't currently have the infrastructure fore.
>
> Possibly it is reworking the stats collection so we store a struct pid
> * instead of a pid_t value. So we would know if the reader of the
> value can even see processes you have collected stats for.

the problem is, this interface stores historic PIDs too - i.e. PIDs of
tasks that might have exited already.

the proper way would be to also store the namespace address, and to

filter out non-matching entries. (If leakage of this data across
namespace creation is of any concern then flush out existing namespace
data when a namespace is destroyed)

 Ingo

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

Subject: Re: [PATCH] Mark timer_stats as incompatible with multiple pid namespaces
Posted by ebiederm on Thu, 13 Dec 2007 17:14:15 GMT
View Forum Message <> Reply to Message

Ingo Molnar <mingo@elte.hu> writes:

> * Eric W. Biederman <ebiederm@xmission.com> wrote:
>
>> > What the heck??? Please solve this properly instead of hiding it.
>> > /proc/timer_stats is damn useful and it's a must-have for powertop
>> > to work.
>>
>> Hmm. Perhaps the dependency conflict should go in the other direction
>> then.
>>
>> My goal is to document the issue while a proper fix is being written.
>> I have known about this for all of about 1 day now. It was added
>> since last time I went through the kernel and made a thorough sweep of
>> pid users.
>>
>> What the proper fix is isn't even obvious at this point. Possibly it
>> is making /proc/timer_stats disappear in child pid namespaces.
>>   Which we don't currently have the infrastructure fore.
>>
>> Possibly it is reworking the stats collection so we store a struct pid
>> * instead of a pid_t value. So we would know if the reader of the
>> value can even see processes you have collected stats for.
>
> the problem is, this interface stores historic PIDs too - i.e. PIDs of
> tasks that might have exited already.

Well struct pid * works in that case if you grab the reference to it.

> the proper way would be to also store the namespace address, and to
> filter out non-matching entries.

Sounds right. If I have the struct pid I can do something like:

```
pid_t upid = pid_nr_ns(pid, current->nsproxy->pid_ns);
if (!upid)
    continue;
```

If I get a pid value in the users pid namespace then the entry
has not been filtered out and I need to display it.

> (If leakage of this data across
> namespace creation is of any concern then flush out existing namespace
> data when a namespace is destroyed)

Given the structure I don't think flushing out the data makes much sense.
Just filtering it should be fine.

My first draft of a proper fix is below. I don't recall if del_timer
is required on timers or not. If not then I have a struct_pid leak.
Otherwise this generally works and happens to make timer_stats safe
from misaccounting due to pid wrap around.

```
diff --git a/include/linux/hrtimer.h b/include/linux/hrtimer.h
index 627767b..58ea150 100644
--- a/include/linux/hrtimer.h
+++ b/include/linux/hrtimer.h
@@ -334,7 +334,7 @@ extern void sysrq_timer_list_show(void);
  */
 #ifdef CONFIG_TIMER_STATS

-extern void timer_stats_update_stats(void *timer, pid_t pid, void *startf,
+extern void timer_stats_update_stats(void *timer, struct pid *pid, void *startf,
          void *timerf, char *comm,
          unsigned int timer_flag);

@@ -355,6 +355,8 @@ static inline void timer_stats_hrtimer_set_start_info(struct hrtimer *timer)
 static inline void timer_stats_hrtimer_clear_start_info(struct hrtimer *timer)
 {
  timer->start_site = NULL;
+ pit_pid(timer->start_pid);
+ timer->start_pid = NULL;
 }
 #else
 static inline void timer_stats_account_hrtimer(struct hrtimer *timer)
diff --git a/include/linux/timer.h b/include/linux/timer.h
index de0e713..9d96943 100644
--- a/include/linux/timer.h
+++ b/include/linux/timer.h
```

```
@@ -18,7 +18,7 @@ struct timer_list {
 #ifdef CONFIG_TIMER_STATS
  void *start_site;
  char start_comm[16];
- int start_pid;
+ struct pid *start_pid;
 #endif
 };

@@ -109,6 +109,8 @@ static inline void timer_stats_timer_set_start_info(struct timer_list *timer)
 static inline void timer_stats_timer_clear_start_info(struct timer_list *timer)
 {
  timer->start_site = NULL;
+ put_pid(timer->start_pid);
+ timer->start_pid = NULL;
 }
 #else
 static inline void init_timer_stats(void)
diff --git a/kernel/hrtimer.c b/kernel/hrtimer.c
index e65dd0b..3d2b017 100644
--- a/kernel/hrtimer.c
+++ b/kernel/hrtimer.c
@@ -633,7 +633,7 @@ void __timer_stats_hrtimer_set_start_info(struct hrtimer *timer, void
*addr)

  timer->start_site = addr;
  memcpy(timer->start_comm, current->comm, TASK_COMM_LEN);
- timer->start_pid = current->pid;
+ timer->start_pid = get_pid(task_pid(current));
 }
 #endif

@@ -1005,7 +1005,7 @@ void hrtimer_init(struct hrtimer *timer, clockid_t clock_id,

 #ifdef CONFIG_TIMER_STATS
  timer->start_site = NULL;
- timer->start_pid = -1;
+ timer->start_pid = NULL;
  memset(timer->start_comm, 0, TASK_COMM_LEN);
 #endif
 }
diff --git a/kernel/time/timer_list.c b/kernel/time/timer_list.c
index 12c5f4c..41efe4a 100644
--- a/kernel/time/timer_list.c
+++ b/kernel/time/timer_list.c
@@ -51,6 +51,9 @@ print_timer(struct seq_file *m, struct hrtimer *timer, int idx, u64 now)
 {
 #ifdef CONFIG_TIMER_STATS
```

```
  char tmp[TASK_COMM_LEN + 1];
+ pid_t upid = pid_vnr(timer->start_pid);
+ if (!upid)
+  return;
 #endif
  SEQ_printf(m, " #%d: ", idx);
  print_name_offset(m, timer);
@@ -62,7 +65,7 @@ print_timer(struct seq_file *m, struct hrtimer *timer, int idx, u64 now)
  print_name_offset(m, timer->start_site);
  memcpy(tmp, timer->start_comm, TASK_COMM_LEN);
  tmp[TASK_COMM_LEN] = 0;
- SEQ_printf(m, ", %s/%d", tmp, timer->start_pid);
+ SEQ_printf(m, ", %s/%d", tmp, upid);
 #endif
  SEQ_printf(m, "\n");
  SEQ_printf(m, " # expires at %Lu nsecs [in %Lu nsecs]\n",
diff --git a/kernel/time/timer_stats.c b/kernel/time/timer_stats.c
index 417da8c..203bf56 100644
--- a/kernel/time/timer_stats.c
+++ b/kernel/time/timer_stats.c
@@ -137,6 +137,9 @@ static struct entry *tstat_hash_table[TSTAT_HASH_SIZE]
__read_mostly;

 static void reset_entries(void)
 {
+ unsigned long i;
+ for (i = 0; < nr_entries; i++)
+  put_pid(i);
  nr_entries = 0;
  memset(entries, 0, sizeof(entries));
  memset(tstat_hash_table, 0, sizeof(tstat_hash_table));
@@ -203,6 +206,7 @@ static struct entry *tstat_lookup(struct entry *entry, char *comm)
  curr = alloc_entry();
  if (curr) {
   *curr = *entry;
+  get_pid(curr->pid);
   curr->count = 0;
   curr->next = NULL;
   memcpy(curr->comm, comm, TASK_COMM_LEN);
@@ -231,7 +235,7 @@ static struct entry *tstat_lookup(struct entry *entry, char *comm)
  * When the timer is already registered, then the event counter is
  * incremented. Otherwise the timer is registered in a free slot.
  */
-void timer_stats_update_stats(void *timer, pid_t pid, void *startf,
+void timer_stats_update_stats(void *timer, struct pid *pid, void *startf,
          void *timerf, char *comm,
          unsigned int timer_flag)
 {
```

```
@@ -305,13 +309,19 @@ static int tstats_show(struct seq_file *m, void *v)
    atomic_read(&overflow_count));

  for (i = 0; i < nr_entries; i++) {
+  pid_t upid;
+
   entry = entries + i;
+  upid = pid_vnr(entry->pid);
+  if (!upid)
+   continue;
+
    if (entry->timer_flag & TIMER_STATS_FLAG_DEFERRABLE) {
    seq_printf(m, "%4luD, %5d %-16s ",
-    entry->count, entry->pid, entry->comm);
+    entry->count, upid, entry->comm);
   } else {
    seq_printf(m, " %4lu, %5d %-16s ",
-    entry->count, entry->pid, entry->comm);
+    entry->count, upid, entry->comm);
   }

   print_name_offset(m, (unsigned long)entry->start_func);
diff --git a/kernel/timer.c b/kernel/timer.c
index f9419f2..b5b1495 100644
--- a/kernel/timer.c
+++ b/kernel/timer.c
@@ -302,7 +302,8 @@ void __timer_stats_timer_set_start_info(struct timer_list *timer, void
*addr)

  timer->start_site = addr;
  memcpy(timer->start_comm, current->comm, TASK_COMM_LEN);
- timer->start_pid = current->pid;
+ put_pid(timer->start_pid);
+ timer->start_pid = get_pid(task_pid(current));
 }

 static void timer_stats_account_timer(struct timer_list *timer)
@@ -333,7 +334,7 @@ void fastcall init_timer(struct timer_list *timer)
  timer->base = __raw_get_cpu_var(tvec_bases);
 #ifdef CONFIG_TIMER_STATS
  timer->start_site = NULL;
- timer->start_pid = -1;
+ timer->start_pid = NULL;
  memset(timer->start_comm, 0, TASK_COMM_LEN);
 #endif
 }
```
____
Containers mailing list

Subject: Re: [PATCH] Mark timer_stats as incompatible with multiple pid namespaces
Posted by Ingo Molnar on Thu, 13 Dec 2007 20:26:21 GMT

View Forum Message <> Reply to Message

* Eric W. Biederman <ebiederm@xmission.com> wrote:

> > the problem is, this interface stores historic PIDs too - i.e. PIDs
> > of tasks that might have exited already.
>
> Well struct pid * works in that case if you grab the reference to it.

but the display of the stats might happen much later. The point of this
API is to save pid+comm, which gives users a good idea about what caused
the events in the past - without having to pin any resource of that
task.

```
>  {
>    timer->start_site = NULL;
> + pit_pid(timer->start_pid);
> + timer->start_pid = NULL;
```

s/pit/put, right?

Ingo

Subject: Re: [PATCH] Mark timer_stats as incompatible with multiple pid namespaces
Posted by ebiederm on Thu, 13 Dec 2007 21:29:09 GMT

View Forum Message <> Reply to Message

Ingo Molnar <mingo@elte.hu> writes:

> * Eric W. Biederman <ebiederm@xmission.com> wrote:
>
>> > the problem is, this interface stores historic PIDs too - i.e. PIDs
>> > of tasks that might have exited already.
>>

>> Well struct pid * works in that case if you grab the reference to it.
>
> but the display of the stats might happen much later. The point of this
> API is to save pid+comm, which gives users a good idea about what caused
> the events in the past - without having to pin any resource of that
> task.

Likewise struct pid is designed not to be a problem if pinned. It is a
little heavier then it used to be with the addition of pid namespace
support but not much. And if it is to heavy struct pid needs to be
fixed.

Holding the struct pid very much does not pin the task struct, and it
shouldn't pin any other resources. I agree 64bytes or so is a bit
more to pin then 4 bytes but it really isn't a lot.

```
>>  {
>>   timer->start_site = NULL;
>> + pit_pid(timer->start_pid);
>> + timer->start_pid = NULL;
>
> s/pit/put, right?
```

Yes. Clearly I haven't tested it yet.

Eric

____

Subject: Re: [PATCH] Mark timer_stats as incompatible with multiple pid namespaces
Posted by Ingo Molnar on Thu, 13 Dec 2007 21:37:11 GMT
View Forum Message <> Reply to Message

* Eric W. Biederman <ebiederm@xmission.com> wrote:

> >> Well struct pid * works in that case if you grab the reference to
> >> it.
> >
> > but the display of the stats might happen much later. The point of
> > this API is to save pid+comm, which gives users a good idea about
> > what caused the events in the past - without having to pin any
> > resource of that task.

>
> Likewise struct pid is designed not to be a problem if pinned. It is a
> little heavier then it used to be with the addition of pid namespace
> support but not much. And if it is to heavy struct pid needs to be
> fixed.
>
> Holding the struct pid very much does not pin the task struct, and it
> shouldn't pin any other resources. I agree 64bytes or so is a bit
> more to pin then 4 bytes but it really isn't a lot.

yeah, and i have no conceptual objections - i just wanted to outline the
thinking behind /proc/timer_stats.

 Ingo

____

Subject: Re: [PATCH] Mark timer_stats as incompatible with multiple pid namespaces
Posted by ebiederm on Thu, 13 Dec 2007 21:48:31 GMT
View Forum Message <> Reply to Message

Ingo Molnar <mingo@elte.hu> writes:

> * Eric W. Biederman <ebiederm@xmission.com> wrote:
>
>> >> Well struct pid * works in that case if you grab the reference to
>> >> it.
>> >
>> > but the display of the stats might happen much later. The point of
>> > this API is to save pid+comm, which gives users a good idea about
>> > what caused the events in the past - without having to pin any
>> > resource of that task.
>>
>> Likewise struct pid is designed not to be a problem if pinned. It is a
>> little heavier then it used to be with the addition of pid namespace
>> support but not much. And if it is to heavy struct pid needs to be
>> fixed.
>>
>> Holding the struct pid very much does not pin the task struct, and it
>> shouldn't pin any other resources. I agree 64bytes or so is a bit
>> more to pin then 4 bytes but it really isn't a lot.
>
> yeah, and i have no conceptual objections - i just wanted to outline the
> thinking behind /proc/timer_stats.

Sure. Appreciated. Outlining the thinking in the other direction
struct pid is supposed to be the pid representation in the kernel.
All of the pid_t stuff really should be pushed as close to the
kernel/user boundary as possible. The closer I get to that ideal
the more cases I find that we need to handle like /proc/timer_stats
and the closer we get to having a complete pid namespace. <pant pant pant>

A struct pid is also now all you pin (besides the inevitable file,
dentry, inode trio) when you open a directory in /proc. So the
task can be freed. Which removed some low-mem exhaustion scenarios
when we introduced it.

Eric