Subject: [PATCH] proc: Fix proc_kill_inodes to kill dentries on all proc superblocks
Posted by ebiederm on Fri, 26 Oct 2007 17:48:55 GMT
View Forum Message <> Reply to Message

It appears we overlooked support for removing generic proc files
when we added support for multiple proc super blocks. Handle
that now.

Since all of the relevant code is already inside of a spin_lock
we can just grab the superblock lock and walk through all of
the instances of proc superblocks.

Heavy weight but it is simple and should work.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 fs/proc/generic.c | 38 ++++++++++++++++++++++++++----------
 fs/proc/internal.h | 2 ++
 fs/proc/root.c | 2 +-
 3 files changed, 24 insertions(+), 18 deletions(-)

diff --git a/fs/proc/generic.c b/fs/proc/generic.c
index 1bdb624..3906770 100644
--- a/fs/proc/generic.c
+++ b/fs/proc/generic.c
@@ -561,28 +561,32 @@ static int proc_register(struct proc_dir_entry * dir, struct proc_dir_entry
* dp
 static void proc_kill_inodes(struct proc_dir_entry *de)
 {
  struct list_head *p;
- struct super_block *sb = proc_mnt->mnt_sb;
+ struct super_block *sb;

  /*
   * Actually it's a partial revoke().
   */
- file_list_lock();
- list_for_each(p, &sb->s_files) {
- struct file * filp = list_entry(p, struct file, f_u.fu_list);
- struct dentry * dentry = filp->f_path.dentry;
- struct inode * inode;
- const struct file_operations *fops;
-
- if (dentry->d_op != &proc_dentry_operations)
-   continue;
- inode = dentry->d_inode;
- if (PDE(inode) != de)
-   continue;

```
-  fops = filp->f_op;
-  filp->f_op = NULL;
-  fops_put(fops);
+ spin_lock(&sb_lock);
+ list_for_each_entry(sb, &proc_fs_type.fs_supers, s_instances) {
+  file_list_lock();
+  list_for_each(p, &sb->s_files) {
+   struct file * filp = list_entry(p, struct file, f_u.fu_list);
+   struct dentry * dentry = filp->f_path.dentry;
+   struct inode * inode;
+   const struct file_operations *fops;
+
+   if (dentry->d_op != &proc_dentry_operations)
+    continue;
+   inode = dentry->d_inode;
+   if (PDE(inode) != de)
+    continue;
+   fops = filp->f_op;
+   filp->f_op = NULL;
+   fops_put(fops);
+  }
+  file_list_unlock();
  }
- file_list_unlock();
+ spin_unlock(&sb_lock);
 }

 static struct proc_dir_entry *proc_create(struct proc_dir_entry **parent,
diff --git a/fs/proc/internal.h b/fs/proc/internal.h
index 1820eb2..1b2b6c6 100644
--- a/fs/proc/internal.h
+++ b/fs/proc/internal.h
@@ -78,3 +78,5 @@ static inline int proc_fd(struct inode *inode)
 {
  return PROC_I(inode)->fd;
 }
+
+extern struct file_system_type proc_fs_type;
diff --git a/fs/proc/root.c b/fs/proc/root.c
index ec9cb3b..1f86bb8 100644
--- a/fs/proc/root.c
+++ b/fs/proc/root.c
@@ -98,7 +98,7 @@ static void proc_kill_sb(struct super_block *sb)
  put_pid_ns(ns);
 }

-static struct file_system_type proc_fs_type = {
+struct file_system_type proc_fs_type = {
```

```
  .name  = "proc",
  .get_sb  = proc_get_sb,
  .kill_sb = proc_kill_sb,
--
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] proc: Fix proc_kill_inodes to kill dentries on all proc superblocks
Posted by Linus Torvalds on Fri, 26 Oct 2007 18:06:52 GMT
View Forum Message <> Reply to Message

On Fri, 26 Oct 2007, Eric W. Biederman wrote:
>
> It appears we overlooked support for removing generic proc files
> when we added support for multiple proc super blocks. Handle
> that now.

There seems to be more users of "proc_mnt" out there. proc_flush_task,
anyone?

  Linus

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

---

Subject: Re: [PATCH] proc: Fix proc_kill_inodes to kill dentries on all proc superblocks
Posted by ebiederm on Fri, 26 Oct 2007 18:36:53 GMT
View Forum Message <> Reply to Message

Linus Torvalds <torvalds@linux-foundation.org> writes:

> On Fri, 26 Oct 2007, Eric W. Biederman wrote:
>>
>> It appears we overlooked support for removing generic proc files
>> when we added support for multiple proc super blocks. Handle
>> that now.
>
> There seems to be more users of "proc_mnt" out there. proc_flush_task,

> anyone?

Yes. I've seen those and I do have some patches in my queue
to address those. Technically none of those causes us to
do the wrong thing in the presence of multiple proc superblocks,
by the usage of proc_mnt. It still looks worth it to see
if we can kill all of the users of proc_mnt.

proc_flush_task is technically ok in it's usage of proc_mnt.
It is just using it as what appears to be an unnecessary optimization.
However the added complexity is currently hiding another bug.
We call proc_flush_task_mnt with the wrong arguments.
Further I haven't convinced myself the call to pid_ns_release_proc
does not have some strange race in there, although there is
nothing obviously wrong there.

Eric

Subject: [PATCH] proc: Simplify and correct proc_flush_task
Posted by ebiederm on Fri, 26 Oct 2007 19:43:28 GMT
View Forum Message <> Reply to Message

Currently we special case when we have only the initial pid namespace.
Unfortunately in doing so the copied case for the other namespaces was
broken so we don't properly flush the thread directories :(

So this patch removes the unnecessary special case (removing a
usage of proc_mnt) and corrects the flushing of the thread directories.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 fs/proc/base.c | 15 ++++++---------
 1 files changed, 6 insertions(+), 9 deletions(-)

diff --git a/fs/proc/base.c b/fs/proc/base.c
index aeaf0d0..a17c268 100644
--- a/fs/proc/base.c
+++ b/fs/proc/base.c
@@ -2328,21 +2328,18 @@ out:

 void proc_flush_task(struct task_struct *task)
 {
- int i, leader;

```
- struct pid *pid, *tgid;
+ int i;
+ struct pid *pid, *tgid = NULL;
  struct upid *upid;

- leader = thread_group_leader(task);
- proc_flush_task_mnt(proc_mnt, task->pid, leader ? task->tgid : 0);
  pid = task_pid(task);
- if (pid->level == 0)
-  return;
+ if (thread_group_leader(task))
+  tgid = task_tgid(task);

- tgid = task_tgid(task);
- for (i = 1; i <= pid->level; i++) {
+ for (i = 0; i <= pid->level; i++) {
   upid = &pid->numbers[i];
   proc_flush_task_mnt(upid->ns->proc_mnt, upid->nr,
-    leader ? 0 : tgid->numbers[i].nr);
+   tgid ? tgid->numbers[i].nr : 0);
  }

  upid = &pid->numbers[pid->level];
--
1.5.3.rc6.17.g1911
```

Subject: [PATCH] pidns: Limit kill -1 and cap_set_all
Posted by ebiederm on Fri, 26 Oct 2007 20:37:48 GMT
View Forum Message <> Reply to Message

This patch implements task_in_pid_ns and uses it to limit cap_set_all
and sys_kill(-1,) to only those tasks in the current pid namespace.

Without this we have a setup for a very nasty surprise.

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

```
 include/linux/pid_namespace.h |    2 ++
 kernel/capability.c           |    3 +++
 kernel/pid.c                  |   11 +++++++++++
 kernel/signal.c               |    5 ++++-
 4 files changed, 20 insertions(+), 1 deletions(-)
```

```
diff --git a/include/linux/pid_namespace.h b/include/linux/pid_namespace.h
index 0227e68..b454678 100644
--- a/include/linux/pid_namespace.h
+++ b/include/linux/pid_namespace.h
@@ -78,4 +78,6 @@ static inline struct task_struct *task_child_reaper(struct task_struct *tsk)
  return tsk->nsproxy->pid_ns->child_reaper;
 }

+extern int task_in_pid_ns(struct task_struct *tsk, struct pid_namespace *ns);
+
 #endif /* _LINUX_PID_NS_H */
diff --git a/kernel/capability.c b/kernel/capability.c
index efbd9cd..a801016 100644
--- a/kernel/capability.c
+++ b/kernel/capability.c
@@ -125,6 +125,7 @@ static inline int cap_set_all(kernel_cap_t *effective,
          kernel_cap_t *inheritable,
          kernel_cap_t *permitted)
 {
+     struct pid_namespace *pid_ns = current->nsproxy->pid_ns;
      struct task_struct *g, *target;
      int ret = -EPERM;
      int found = 0;
@@ -132,6 +133,8 @@ static inline int cap_set_all(kernel_cap_t *effective,
      do_each_thread(g, target) {
              if (target == current || is_container_init(target->group_leader))
                      continue;
+             if (!task_in_pid_ns(target, pid_ns))
+       continue;
              found = 1;
       if (security_capset_check(target, effective, inheritable,
       permitted))
diff --git a/kernel/pid.c b/kernel/pid.c
index f815455..1c332ca 100644
--- a/kernel/pid.c
+++ b/kernel/pid.c
@@ -430,6 +430,17 @@ struct pid *find_get_pid(pid_t nr)
  return pid;
 }

+static int pid_in_pid_ns(struct pid *pid, struct pid_namespace *ns)
+{
+ return pid && (ns->level <= pid->level) &&
+  pid->numbers[ns->level].ns == ns;
+}
+
+int task_in_pid_ns(struct task_struct *task, struct pid_namespace *ns)
```

```
+{
+ return pid_in_pid_ns(task_pid(task), ns);
+}
+
 pid_t pid_nr_ns(struct pid *pid, struct pid_namespace *ns)
 {
  struct upid *upid;
diff --git a/kernel/signal.c b/kernel/signal.c
index 1200630..8f5a31f 100644
--- a/kernel/signal.c
+++ b/kernel/signal.c
@@ -1147,10 +1147,13 @@ static int kill_something_info(int sig, struct siginfo *info, int pid)
  } else if (pid == -1) {
   int retval = 0, count = 0;
   struct task_struct * p;
+  struct pid_namespace *ns = current->nsproxy->pid_ns;

   read_lock(&tasklist_lock);
   for_each_process(p) {
-   if (p->pid > 1 && !same_thread_group(p, current)) {
+   if (!is_container_init(p) &&
+       !same_thread_group(p, current) &&
+       task_in_pid_ns(p, ns)) {
     int err = group_send_sig_info(sig, info, p);
     ++count;
     if (err != -EPERM)
--
1.5.3.rc6.17.g1911
```

Subject: Re: [PATCH] pidns: Limit kill -1 and cap_set_all
Posted by Kir Kolyshkin on Fri, 26 Oct 2007 21:06:54 GMT
View Forum Message <> Reply to Message

Eric, this problem is a known one. Currently Pavel and Sukadev are working on a appropriate signal management for namespaces.

If you'd first sent this patch for discussion/review to containers@, you'd know that the patch is very incomplete.

Rgrds,
  Kir.
Sent from my BlackBerry; please reply to kir@openvz.org

-----Original Message-----
From: devel-bounces@openvz.org <devel-bounces@openvz.org>
To: Linus Torvalds <torvalds@linux-foundation.org>
CC: Linux Containers <containers@lists.osdl.org>; Andrew Morton <akpm@linux-foundation.org>; linux-kernel@vger.kernel.org <linux-kernel@vger.kernel.org>; Oleg Nesterov <oleg@tv-sign.ru>
Sent: Fri Oct 26 16:37:48 2007
Subject: [Devel] [PATCH] pidns: Limit kill -1 and cap_set_all

This patch implements task_in_pid_ns and uses it to limit cap_set_all
and sys_kill(-1,) to only those tasks in the current pid namespace.

Without this we have a setup for a very nasty surprise.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 include/linux/pid_namespace.h | 2 ++
 kernel/capability.c | 3 +++
 kernel/pid.c | 11 +++++++++++
 kernel/signal.c | 5 ++++-
 4 files changed, 20 insertions(+), 1 deletions(-)

diff --git a/include/linux/pid_namespace.h b/include/linux/pid_namespace.h
index 0227e68..b454678 100644
--- a/include/linux/pid_namespace.h
+++ b/include/linux/pid_namespace.h
@@ -78,4 +78,6 @@ static inline struct task_struct *task_child_reaper(struct task_struct *tsk)
  return tsk->nsproxy->pid_ns->child_reaper;
 }

+extern int task_in_pid_ns(struct task_struct *tsk, struct pid_namespace *ns);
+
 #endif /* _LINUX_PID_NS_H */
diff --git a/kernel/capability.c b/kernel/capability.c
index efbd9cd..a801016 100644
--- a/kernel/capability.c
+++ b/kernel/capability.c
@@ -125,6 +125,7 @@ static inline int cap_set_all(kernel_cap_t *effective,
          kernel_cap_t *inheritable,
          kernel_cap_t *permitted)
 {
+   struct pid_namespace *pid_ns = current->nsproxy->pid_ns;
      struct task_struct *g, *target;
      int ret = -EPERM;
      int found = 0;
@@ -132,6 +133,8 @@ static inline int cap_set_all(kernel_cap_t *effective,
      do_each_thread(g, target) {

```
            if (target == current || is_container_init(target->group_leader))
                  continue;
+            if (!task_in_pid_ns(target, pid_ns))
+       continue;
            found = 1;
       if (security_capset_check(target, effective, inheritable,
       permitted))
diff --git a/kernel/pid.c b/kernel/pid.c
index f815455..1c332ca 100644
--- a/kernel/pid.c
+++ b/kernel/pid.c
@@ -430,6 +430,17 @@ struct pid *find_get_pid(pid_t nr)
  return pid;
 }

+static int pid_in_pid_ns(struct pid *pid, struct pid_namespace *ns)
+{
+ return pid && (ns->level <= pid->level) &&
+  pid->numbers[ns->level].ns == ns;
+}
+
+int task_in_pid_ns(struct task_struct *task, struct pid_namespace *ns)
+{
+ return pid_in_pid_ns(task_pid(task), ns);
+}
+
 pid_t pid_nr_ns(struct pid *pid, struct pid_namespace *ns)
 {
  struct upid *upid;
diff --git a/kernel/signal.c b/kernel/signal.c
index 1200630..8f5a31f 100644
--- a/kernel/signal.c
+++ b/kernel/signal.c
@@ -1147,10 +1147,13 @@ static int kill_something_info(int sig, struct siginfo *info, int pid)
  } else if (pid == -1) {
   int retval = 0, count = 0;
   struct task_struct * p;
+  struct pid_namespace *ns = current->nsproxy->pid_ns;

   read_lock(&tasklist_lock);
   for_each_process(p) {
-   if (p->pid > 1 && !same_thread_group(p, current)) {
+   if (!is_container_init(p) &&
+       !same_thread_group(p, current) &&
+       task_in_pid_ns(p, ns)) {
     int err = group_send_sig_info(sig, info, p);
     ++count;
     if (err != -EPERM)
```

--
1.5.3.rc6.17.g1911

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

_____
Devel mailing list
Devel@openvz.org
https://openvz.org/mailman/listinfo/devel

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

---

Subject: Re: [PATCH] pidns: Limit kill -1 and cap_set_all
Posted by ebiederm on Fri, 26 Oct 2007 22:10:04 GMT
View Forum Message <> Reply to Message

"Kir Kolyshkin" <kir@swsoft.com> writes:

> Eric, this problem is a known one. Currently Pavel and Sukadev are working on a
> appropriate signal management for namespaces.

I'm fairly certain that the signal issue you they are dealing with
is how to keep children from killing the init of a pid namespace.
At least that is what Suka mentioned when I asked him earlier,
and that is the discussion I have at various times on the containers
list.

This is a totally different issue. This is keeping
kill -1 in a child pid namespace from sending a signal to
every process in the system.

It is a bad bug and this patch is a complete fix for it.
To my knowledge Pavel and Suka just overlooked this case.

> If you'd first sent this patch for discussion/review to containers@, you'd know
> that the patch is very incomplete.

I explicitly asked Suka, Pavel, and Cedric how complete they thought
the pid namespace was before I started sending patches. With
Pavel at a conference it makes sense why he did not reply.

For the most part Suka and Cedric did not know about the issues

that I have been finding and fixing.

So since several of them have been fixable with trivial simple fixes
and because we have passed -rc1 and it is now time for bug fixes
and possibly a few cleanups. I have been making bug fixes.

Kir I am happy from feedback from people who can understand the code
and patches. But I am not going to hold up sending simple obviously
correct fixes for discussion.

Eric

Subject: Re: [PATCH] pidns: Limit kill -1 and cap_set_all
Posted by Kirill Korotaev on Mon, 29 Oct 2007 08:36:29 GMT
View Forum Message <> Reply to Message

I dislike this patch:
it's not scalable/efficient to travers all the tasks
while we know the pid namespace we care about.

Kirill

Eric W. Biederman wrote:
> This patch implements task_in_pid_ns and uses it to limit cap_set_all
> and sys_kill(-1,) to only those tasks in the current pid namespace.
>
> Without this we have a setup for a very nasty surprise.
>
> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
> ---
> include/linux/pid_namespace.h | 2 ++
> kernel/capability.c | 3 +++
> kernel/pid.c | 11 +++++++++++
> kernel/signal.c | 5 ++++-
> 4 files changed, 20 insertions(+), 1 deletions(-)
>
> diff --git a/include/linux/pid_namespace.h b/include/linux/pid_namespace.h
> index 0227e68..b454678 100644
> --- a/include/linux/pid_namespace.h
> +++ b/include/linux/pid_namespace.h
> @@ -78,4 +78,6 @@ static inline struct task_struct *task_child_reaper(struct task_struct *tsk)

```
>   return tsk->nsproxy->pid_ns->child_reaper;
>  }
>
> +extern int task_in_pid_ns(struct task_struct *tsk, struct pid_namespace *ns);
> +
>  #endif /* _LINUX_PID_NS_H */
> diff --git a/kernel/capability.c b/kernel/capability.c
> index efbd9cd..a801016 100644
> --- a/kernel/capability.c
> +++ b/kernel/capability.c
> @@ -125,6 +125,7 @@ static inline int cap_set_all(kernel_cap_t *effective,
>            kernel_cap_t *inheritable,
>            kernel_cap_t *permitted)
>  {
> +     struct pid_namespace *pid_ns = current->nsproxy->pid_ns;
>       struct task_struct *g, *target;
>       int ret = -EPERM;
>       int found = 0;
> @@ -132,6 +133,8 @@ static inline int cap_set_all(kernel_cap_t *effective,
>       do_each_thread(g, target) {
>               if (target == current || is_container_init(target->group_leader))
>                       continue;
> +             if (!task_in_pid_ns(target, pid_ns))
> +       continue;
>               found = 1;
>        if (security_capset_check(target, effective, inheritable,
>        permitted))
> diff --git a/kernel/pid.c b/kernel/pid.c
> index f815455..1c332ca 100644
> --- a/kernel/pid.c
> +++ b/kernel/pid.c
> @@ -430,6 +430,17 @@ struct pid *find_get_pid(pid_t nr)
>   return pid;
>  }
>
> +static int pid_in_pid_ns(struct pid *pid, struct pid_namespace *ns)
> +{
> + return pid && (ns->level <= pid->level) &&
> +  pid->numbers[ns->level].ns == ns;
> +}
> +
> +int task_in_pid_ns(struct task_struct *task, struct pid_namespace *ns)
> +{
> + return pid_in_pid_ns(task_pid(task), ns);
> +}
> +
>  pid_t pid_nr_ns(struct pid *pid, struct pid_namespace *ns)
>  {
```

>   struct upid *upid;
> diff --git a/kernel/signal.c b/kernel/signal.c
> index 1200630..8f5a31f 100644
> --- a/kernel/signal.c
> +++ b/kernel/signal.c
> @@ -1147,10 +1147,13 @@ static int kill_something_info(int sig, struct siginfo *info, int pid)
>   } else if (pid == -1) {
>   int retval = 0, count = 0;
>   struct task_struct * p;
> + struct pid_namespace *ns = current->nsproxy->pid_ns;
>
>   read_lock(&tasklist_lock);
>   for_each_process(p) {
> -   if (p->pid > 1 && !same_thread_group(p, current)) {
> +   if (!is_container_init(p) &&
> +   !same_thread_group(p, current) &&
> +   task_in_pid_ns(p, ns)) {
>   int err = group_send_sig_info(sig, info, p);
>   ++count;
>   if (err != -EPERM)

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

---

Subject: Re: [PATCH] pidns: Limit kill -1 and cap_set_all
Posted by Dave Hansen on Mon, 29 Oct 2007 16:02:18 GMT
View Forum Message <> Reply to Message

On Fri, 2007-10-26 at 14:37 -0600, Eric W. Biederman wrote:
>
> +static int pid_in_pid_ns(struct pid *pid, struct pid_namespace *ns)
> +{
> +   return pid && (ns->level <= pid->level) &&
> +   pid->numbers[ns->level].ns == ns;
> +}

Could we blow this out a little bit? (I think the blown-out version
lends itself to being better commented, and easier to read.) Also, can
we think of any better name for this? It seems a bit funky that:

 pid_in_pid_ns(mypid, &init_pid_ns);

would _ever_ return 0. So, it isn't truly a test for belonging *in* a
namespace, but having that namespace be the lowest level one. I think
Suka toyed with calling it an "active" or "primary" pid namespace. That

differentiated mere membership in a pid namespace from the one that
actually molds that pid's view of the world.

```
static int pid_in_pid_ns(struct pid *pid, struct pid_namespace *ns)
{
 if (!pid)
  return 0;
 if (ns->level > pid->level)
  return 0;
 if (pid->numbers[ns->level].ns != ns)
  return 0;
 return 1;
}
```

-- Dave

Subject: Re: [PATCH] pidns: Limit kill -1 and cap_set_all
Posted by ebiederm on Mon, 29 Oct 2007 17:59:48 GMT
View Forum Message <> Reply to Message

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

> On Fri, 2007-10-26 at 14:37 -0600, Eric W. Biederman wrote:
>>
>> +static int pid_in_pid_ns(struct pid *pid, struct pid_namespace *ns)
>> +{
>> +	return pid && (ns->level <= pid->level) &&
>> +		pid->numbers[ns->level].ns == ns;
>> +}
>
> Could we blow this out a little bit? (I think the blown-out version
> lends itself to being better commented, and easier to read.) Also, can
> we think of any better name for this? It seems a bit funky that:
>
> pid_in_pid_ns(mypid, &init_pid_ns);
>
> would _ever_ return 0.

It can't.

> So, it isn't truly a test for belonging *in* a
> namespace, but having that namespace be the lowest level one.

No. It is precisely a test for being in a namespace.
We first check ns->level to make certain it doesn't fall out
of the array, and then we check to see if the namespace we
are looking for is at that level.

pid->numbers[0].ns == &init_pid_ns.

> I think
> Suka toyed with calling it an "active" or "primary" pid namespace. That
> differentiated mere membership in a pid namespace from the one that
> actually molds that pid's view of the world.

What we want for the test is a test for membership.

> static int pid_in_pid_ns(struct pid *pid, struct pid_namespace *ns)
> {
> if (!pid)
>   return 0;
> if (ns->level > pid->level)
>   return 0;
> if (pid->numbers[ns->level].ns != ns)
>   return 0;
> return 1;
> }

I don't have a problem with that. The rest of the checks for
this in kernel/pid.c are in the same form.

Eric

Subject: Re: [PATCH] pidns: Limit kill -1 and cap_set_all
Posted by Dave Hansen on Mon, 29 Oct 2007 18:07:27 GMT
View Forum Message <> Reply to Message

On Mon, 2007-10-29 at 11:59 -0600, Eric W. Biederman wrote:
> ier to read.) Also, can
> > we think of any better name for this? It seems a bit funky that:
> >
> > pid_in_pid_ns(mypid, &init_pid_ns);
> >

> > would _ever_ return 0.
>
> It can't.
>
> > So, it isn't truly a test for belonging *in* a
> > namespace, but having that namespace be the lowest level one.
>
> No. It is precisely a test for being in a namespace.
> We first check ns->level to make certain it doesn't fall out
> of the array, and then we check to see if the namespace we
> are looking for is at that level.
>
> pid->numbers[0].ns == &init_pid_ns.

Ahhh. I misparsed the:

 pid->numbers[ns->level].ns == ns;

line to be checking at the pid level. You're right, it works fine as it
stands.

-- Dave

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

---

Subject: Re: [PATCH] pidns: Limit kill -1 and cap_set_all
Posted by ebiederm on Mon, 29 Oct 2007 18:07:29 GMT
View Forum Message <> Reply to Message

Kirill Korotaev <dev@openvz.org> writes:

> I dislike this patch:
> it's not scalable/efficient to travers all the tasks
> while we know the pid namespace we care about.

Well the unix way is to implement it simple and stupid and then to
optimize, where needed. We don't currently have a per pid namespace
list of processes or tasks.

This is a trivial bug fix patch, and I wanted the review to be as
simple as possible.

I don't expect people are doing kill -1 all that frequently as it is a
bit rude.

I have no problem doing find_ge_pid and then looking in the hash table
like we do in /proc. It likely will have fewer conflicts, but that
is a bit harder to review, and is actually more code.

Eric

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

---

Subject: Re: [PATCH] pidns: Limit kill -1 and cap_set_all
Posted by serge on Wed, 31 Oct 2007 21:43:33 GMT
View Forum Message <> Reply to Message

Quoting Eric W. Biederman (ebiederm@xmission.com):
>
> This patch implements task_in_pid_ns and uses it to limit cap_set_all
> and sys_kill(-1,) to only those tasks in the current pid namespace.
>
> Without this we have a setup for a very nasty surprise.
>
> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

Thanks Eric. Seems one of the LTP tests being written to test pidns
signaling did just catch that this week. Hopefully the testcases will
be ready to push to ltp "soon".

So despite some complaints raised by others,

Acked-by: Serge Hallyn <serue@us.ibm.com>

for looking correct and very much needed.

thanks,
-serge

> ---
> include/linux/pid_namespace.h | 2 ++
> kernel/capability.c | 3 +++
> kernel/pid.c | 11 +++++++++++
> kernel/signal.c | 5 ++++-
> 4 files changed, 20 insertions(+), 1 deletions(-)
>
> diff --git a/include/linux/pid_namespace.h b/include/linux/pid_namespace.h
> index 0227e68..b454678 100644
> --- a/include/linux/pid_namespace.h

```
> +++ b/include/linux/pid_namespace.h
> @@ -78,4 +78,6 @@ static inline struct task_struct *task_child_reaper(struct task_struct *tsk)
>   return tsk->nsproxy->pid_ns->child_reaper;
>  }
>
> +extern int task_in_pid_ns(struct task_struct *tsk, struct pid_namespace *ns);
> +
>  #endif /* _LINUX_PID_NS_H */
> diff --git a/kernel/capability.c b/kernel/capability.c
> index efbd9cd..a801016 100644
> --- a/kernel/capability.c
> +++ b/kernel/capability.c
> @@ -125,6 +125,7 @@ static inline int cap_set_all(kernel_cap_t *effective,
>            kernel_cap_t *inheritable,
>            kernel_cap_t *permitted)
>  {
> +     struct pid_namespace *pid_ns = current->nsproxy->pid_ns;
>       struct task_struct *g, *target;
>       int ret = -EPERM;
>       int found = 0;
> @@ -132,6 +133,8 @@ static inline int cap_set_all(kernel_cap_t *effective,
>       do_each_thread(g, target) {
>               if (target == current || is_container_init(target->group_leader))
>                       continue;
> +             if (!task_in_pid_ns(target, pid_ns))
> +       continue;
>               found = 1;
>        if (security_capset_check(target, effective, inheritable,
>        permitted))
> diff --git a/kernel/pid.c b/kernel/pid.c
> index f815455..1c332ca 100644
> --- a/kernel/pid.c
> +++ b/kernel/pid.c
> @@ -430,6 +430,17 @@ struct pid *find_get_pid(pid_t nr)
>   return pid;
>  }
>
> +static int pid_in_pid_ns(struct pid *pid, struct pid_namespace *ns)
> +{
> + return pid && (ns->level <= pid->level) &&
> +  pid->numbers[ns->level].ns == ns;
> +}
> +
> +int task_in_pid_ns(struct task_struct *task, struct pid_namespace *ns)
> +{
> + return pid_in_pid_ns(task_pid(task), ns);
> +}
> +
```

> pid_t pid_nr_ns(struct pid *pid, struct pid_namespace *ns)
> {
>   struct upid *upid;
> diff --git a/kernel/signal.c b/kernel/signal.c
> index 1200630..8f5a31f 100644
> --- a/kernel/signal.c
> +++ b/kernel/signal.c
> @@ -1147,10 +1147,13 @@ static int kill_something_info(int sig, struct siginfo *info, int pid)
>   } else if (pid == -1) {
>   int retval = 0, count = 0;
>   struct task_struct * p;
> + struct pid_namespace *ns = current->nsproxy->pid_ns;
>
>   read_lock(&tasklist_lock);
>   for_each_process(p) {
> -   if (p->pid > 1 && !same_thread_group(p, current)) {
> +   if (!is_container_init(p) &&
> +   !same_thread_group(p, current) &&
> +   task_in_pid_ns(p, ns)) {
>   int err = group_send_sig_info(sig, info, p);
>   ++count;
>   if (err != -EPERM)
> --
> 1.5.3.rc6.17.g1911
>
> -
> To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
> the body of a message to majordomo@vger.kernel.org
> More majordomo info at http://vger.kernel.org/majordomo-info.html
> Please read the FAQ at http://www.tux.org/lkml/

_____

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