
Subject: Re: [RFC PATCH 3/5] use next syscall data to predefine process ids
Posted by [serue](#) on Tue, 08 Jul 2008 19:49:26 GMT

[View Forum Message](#) <> [Reply to Message](#)

Quoting Nadia.Derbey@bull.net (Nadia.Derbey@bull.net):

> [PATCH 03/05]

>

> This patch uses the value written into the next_syscall_data proc file

> as a target upid nr for the next process to be created.

> The following syscalls have a new behavior if next_syscall_data is set:

> . fork()

> . vfork()

> . clone()

>

> In the current version, if the process belongs to nested namespaces, only

> the upper namespace level upid nr is allowed to be predefined, since there

> is not yet a way to take a snapshot of upid nrs at all namespaces levels.

>

> But this can easily be extended in the future.

>

> Signed-off-by: Nadia Derbey <Nadia.Derbey@bull.net>

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

thanks,

-serge

>
> ---
> include/linux/next_syscall_data.h | 2
> kernel/fork.c | 5 -
> kernel/pid.c | 116 ++++++-----
> 3 files changed, 102 insertions(+), 21 deletions(-)
>
> Index: linux-2.6.26-rc8-mm1/kernel/pid.c
> ======
> --- linux-2.6.26-rc8-mm1.orig/kernel/pid.c 2008-07-08 12:12:39.000000000 +0200
> +++ linux-2.6.26-rc8-mm1/kernel/pid.c 2008-07-08 12:24:04.000000000 +0200
> @@ -122,6 +122,26 @@ static void free_pidmap(struct upid *upi
> atomic_inc(&map->nr_free);
> }
>
> +static inline int alloc_pidmap_page(struct pidmap *map)
> +{
> + if (unlikely(!map->page)) {
> + void *page = kzalloc(PAGE_SIZE, GFP_KERNEL);
> + /*
> + * Free the page if someone raced with us

```

> + * installing it:
> +
> + spin_lock_irq(&pidmap_lock);
> + if (map->page)
> + kfree(page);
> + else
> + map->page = page;
> + spin_unlock_irq(&pidmap_lock);
> + if (unlikely(!map->page))
> + return -1;
> +
> + }
> + return 0;
> +
> +
> static int alloc_pidmap(struct pid_namespace *pid_ns)
> {
>     int i, offset, max_scan, pid, last = pid_ns->last_pid;
> @@ -134,21 +154,8 @@ static int alloc_pidmap(struct pid_names
>     map = &pid_ns->pidmap[pid/BITS_PER_PAGE];
>     max_scan = (pid_max + BITS_PER_PAGE - 1)/BITS_PER_PAGE - !offset;
>     for (i = 0; i <= max_scan; ++i) {
> -     if (unlikely(!map->page)) {
> -         void *page = kzalloc(PAGE_SIZE, GFP_KERNEL);
> -         /*
> -          * Free the page if someone raced with us
> -          * installing it:
> -         */
> -         spin_lock_irq(&pidmap_lock);
> -         if (map->page)
> -             kfree(page);
> -         else
> -             map->page = page;
> -         spin_unlock_irq(&pidmap_lock);
> -         if (unlikely(!map->page))
> -             break;
> -     }
> +     if (unlikely(alloc_pidmap_page(map)))
> +     break;
>     if (likely(atomic_read(&map->nr_free))) {
>         do {
>             if (!test_and_set_bit(offset, map->page)) {
> @@ -182,6 +189,33 @@ static int alloc_pidmap(struct pid_names
>         return -1;
>     }
>     /*
> + * Return 0 if successful (i.e. next_nr could be assigned as a upid nr).
> + * -errno else

```

```

> + */
> +static int alloc_fixed_pidmap(struct pid_namespace *pid_ns, int next_nr)
> +{
> + int offset;
> + struct pidmap *map;
> +
> + if (next_nr < RESERVED_PIDS || next_nr >= pid_max)
> + return -EINVAL;
> +
> + map = &pid_ns->pidmap[next_nr / BITS_PER_PAGE];
> +
> + if (unlikely(alloc_pidmap_page(map)))
> + return -ENOMEM;
> +
> + offset = next_nr & BITS_PER_PAGE_MASK;
> + if (test_and_set_bit(offset, map->page))
> + return -EBUSY;
> +
> + atomic_dec(&map->nr_free);
> + pid_ns->last_pid = max(pid_ns->last_pid, next_nr);
> +
> + return 0;
> +}
> +
> int next_pidmap(struct pid_namespace *pid_ns, int last)
> {
> + int offset;
> @@ -239,6 +273,24 @@ void free_pid(struct pid *pid)
> + call_rcu(&pid->rcu, delayed_put_pid);
> +}
>
> +/*
> + * Sets a predefined upid nr for the process' upper namespace level
> + */
> +static int set_predefined_pid(struct pid_namespace *ns, struct pid *pid,
> + int next_nr)
> +{
> + int i = ns->level;
> + int rc;
> +
> + rc = alloc_fixed_pidmap(ns, next_nr);
> + if (rc < 0)
> + return rc;
> +
> + pid->numbers[i].nr = next_nr;
> + pid->numbers[i].ns = ns;
> + return 0;
> +}

```

```

> +
> struct pid *alloc_pid(struct pid_namespace *ns)
> {
>     struct pid *pid;
> @@ -248,14 +300,41 @@ struct pid *alloc_pid(struct pid_namespa
>     struct upid *upid;
>
>     pid = kmem_cache_alloc(ns->pid_cachep, GFP_KERNEL);
> - if (!pid)
> + if (!pid) {
> +     pid = ERR_PTR(-ENOMEM);
>     goto out;
> +
>     tmp = ns;
> - for (i = ns->level; i >= 0; i--) {
> + i = ns->level;
> + if (unlikely(next_data_set(current))) {
> + /*
> + * There is a upid nr specified, use it instead of letting
> + * the kernel chose it for us.
> + */
> + int next_nr = get_next_data(current);
> + int rc;
> +
> + reset_next_syscall_data(current);
> + rc = set_predefined_pid(tmp, pid, next_nr);
> + if (rc < 0) {
> +     pid = ERR_PTR(rc);
> +     goto out_free;
> + }
> + /* Go up one level */
> + tmp = tmp->parent;
> + i--;
> +
> + /*
> + * Let the lower levels upid nrs be automatically allocated
> + */
> + for ( ; i >= 0; i--) {
>     nr = alloc_pidmap(tmp);
> - if (nr < 0)
> + if (nr < 0) {
> +     pid = ERR_PTR(-ENOMEM);
>     goto out_free;
> +
>     pid->numbers[i].nr = nr;

```

```

> pid->numbers[i].ns = tmp;
> @@ -284,7 +363,6 @@ out_free:
>   free_pidmap(pid->numbers + i);
>
>   kmem_cache_free(ns->pid_cachep, pid);
> - pid = NULL;
>   goto out;
> }
>
> Index: linux-2.6.26-rc8-mm1/kernel/fork.c
> =====
> --- linux-2.6.26-rc8-mm1.orig/kernel/fork.c 2008-07-08 12:12:39.000000000 +0200
> +++ linux-2.6.26-rc8-mm1/kernel/fork.c 2008-07-08 12:22:47.000000000 +0200
> @@ -1118,10 +1118,11 @@ static struct task_struct *copy_process(
>   goto bad_fork_cleanup_io;
>
>   if (pid != &init_struct_pid) {
> -   retval = -ENOMEM;
>   pid = alloc_pid(task_active_pid_ns(p));
> -   if (!pid)
> +   if (IS_ERR(pid)) {
> +   retval = PTR_ERR(pid);
>   goto bad_fork_cleanup_io;
> +
>
>   if (clone_flags & CLONE_NEWPID) {
>     retval = pid_ns_prepare_proc(task_active_pid_ns(p));
> Index: linux-2.6.26-rc8-mm1/include/linux/next_syscall_data.h
> =====
> --- linux-2.6.26-rc8-mm1.orig/include/linux/next_syscall_data.h 2008-07-08 12:12:39.000000000 +0200
> +++ linux-2.6.26-rc8-mm1/include/linux/next_syscall_data.h 2008-07-08 12:22:47.000000000 +0200
> @@ -5,6 +5,7 @@
>   * following is supported today:
>   *   . object creation with a predefined id
>   *       . for a sysv ipc object
> + *       . for a process
> +
>
> #ifndef _LINUX_NEXT_SYSCALL_DATA_H
> @@ -18,6 +19,7 @@
>   * For example, it can be used to pre-set the id of the object to be created
>   * by next syscall. The following syscalls support this feature:
>   *   . msgget(), semget(), shmget()
> + *   . fork(), vfork(), clone()
> +
> struct next_syscall_data {

```

```
> int ndata;  
>  
> --
```

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