
Subject: [RFC] [PATCH 1/2] namespace enter: introduce do_fork_task()
Posted by [serue](#) on Wed, 29 Aug 2007 20:04:47 GMT
[View Forum Message](#) <> [Reply to Message](#)

>From 9dbd5b00c5aa0707e3e2ed6e6784f93b396f57ec Mon Sep 17 00:00:00 2001
From: sergeh@us.ibm.com <sergeh@us.ibm.com>
Date: Wed, 22 Aug 2007 15:03:57 -0700
Subject: [RFC] [PATCH 1/2] namespace enter: introduce do_fork_task()

Move most of do_fork() into a new do_fork_task() which acts on a new argument, task, rather than on current. do_fork() becomes a call to do_fork_task(current, ...).

Do the same thing to copy_process and a few other places.

Change security_task_alloc() to take the task being cloned as an argument, since it can no longer be assumed to be current.

Note that the check for CLONE_SYSVSEM needs to be extended to check for all namespaces. We don't allow cloning namespaces on top of a hijack right now.

Signed-off-by: sergeh@us.ibm.com <hallyn@kernel.(none)>

```
arch/i386/kernel/process.c | 10 ++++-
arch/s390/kernel/process.c | 12 +++++-
include/linux/pid.h        | 2 +-
include/linux/sched.h      | 1 +
include/linux/security.h   | 9 +++--
kernel/fork.c              | 80 ++++++-----
kernel/pid.c               | 4 +-
security/dummy.c           | 3 +-
security/security.c        | 4 +-
security/selinux/hooks.c   | 5 +-
10 files changed, 93 insertions(+), 37 deletions(-)
```

diff --git a/arch/i386/kernel/process.c b/arch/i386/kernel/process.c

index 7bdc459..e01ddac 100644

--- a/arch/i386/kernel/process.c

+++ b/arch/i386/kernel/process.c

```
@ @ -455,8 +455,15 @ @ int copy_thread(int nr, unsigned long clone_flags, unsigned long esp,
    unsigned long unused,
    struct task_struct * p, struct pt_regs * regs)
{
+ return copy_a_thread(current, nr, clone_flags, esp, unused,
+ p, regs);
+}
+
```

```

+int copy_a_thread(struct task_struct *tsk, int nr, unsigned long clone_flags,
+ unsigned long esp, unsigned long unused,
+ struct task_struct * p, struct pt_regs * regs)
+{
+    struct pt_regs * childregs;
- struct task_struct *tsk;
+    int err;

+    childregs = task_pt_regs(p);
@@ -471,7 +478,6 @@ int copy_thread(int nr, unsigned long clone_flags, unsigned long esp,

+    savesegment(gs,p->thread.gs);

- tsk = current;
+ if (unlikely(test_tsk_thread_flag(tsk, TIF_IO_BITMAP))) {
+     p->thread.io_bitmap_ptr = kmemdup(tsk->thread.io_bitmap_ptr,
+         IO_BITMAP_BYTES, GFP_KERNEL);
diff --git a/arch/s390/kernel/process.c b/arch/s390/kernel/process.c
index abb447a..5390a4f 100644
--- a/arch/s390/kernel/process.c
+++ b/arch/s390/kernel/process.c
@@ -223,6 +223,14 @@ int copy_thread(int nr, unsigned long clone_flags, unsigned long
new_stackp,
+    unsigned long unused,
+    struct task_struct * p, struct pt_regs * regs)
+    {
+ return copy_a_thread(current, nr, clone_flags, new_stackp, unused,
+     p, regs);
+ }
+
+int copy_a_thread(struct task_struct *task, int nr, unsigned long clone_flags,
+ unsigned long new_stackp, unsigned long unused,
+     struct task_struct * p, struct pt_regs * regs)
+{
+    struct fake_frame
+    {
+        struct stack_frame sf;
@@ -251,8 +259,8 @@ int copy_thread(int nr, unsigned long clone_flags, unsigned long
new_stackp,
+    * save fprs to current->thread.fp_regs to merge them with
+    * the emulated registers and then copy the result to the child.
+    */
- save_fp_regs(&current->thread.fp_regs);
- memcpy(&p->thread.fp_regs, &current->thread.fp_regs,
+ save_fp_regs(&task->thread.fp_regs);
+ memcpy(&p->thread.fp_regs, &task->thread.fp_regs,
+     sizeof(s390_fp_regs));
+    p->thread.user_seg = __pa((unsigned long) p->mm->pgd) | _SEGMENT_TABLE;

```

```

/* Set a new TLS ? */
diff --git a/include/linux/pid.h b/include/linux/pid.h
index 1e0e4e3..0d0117a 100644
--- a/include/linux/pid.h
+++ b/include/linux/pid.h
@@ -95,7 +95,7 @@ extern struct pid *FASTCALL(find_pid(int nr));
extern struct pid *find_get_pid(int nr);
extern struct pid *find_ge_pid(int nr);

-extern struct pid *alloc_pid(void);
+extern struct pid *alloc_pid(struct task_struct *task);
extern void FASTCALL(free_pid(struct pid *pid));

static inline pid_t pid_nr(struct pid *pid)
diff --git a/include/linux/sched.h b/include/linux/sched.h
index 734dd58..7fa6710 100644
--- a/include/linux/sched.h
+++ b/include/linux/sched.h
@@ -1518,6 +1518,7 @@ extern struct mm_struct *get_task_mm(struct task_struct *task);
extern void mm_release(struct task_struct *, struct mm_struct *);

extern int copy_thread(int, unsigned long, unsigned long, unsigned long, struct task_struct *,
struct pt_regs *);
+extern int copy_a_thread(struct task_struct *, int, unsigned long, unsigned long, unsigned long,
struct task_struct *, struct pt_regs *);
extern void flush_thread(void);
extern void exit_thread(void);

diff --git a/include/linux/security.h b/include/linux/security.h
index e38230f..f64d286 100644
--- a/include/linux/security.h
+++ b/include/linux/security.h
@@ -530,6 +530,7 @@ struct request_sock;
* @clone_flags contains the flags indicating what should be shared.
* Return 0 if permission is granted.
* @task_alloc_security:
+ * @orig contains the task_struct for the process being cloned.
* @p contains the task_struct for child process.
* Allocate and attach a security structure to the p->security field. The
* security field is initialized to NULL when the task structure is
@@ -1277,7 +1278,8 @@ struct security_operations {
int (*file_receive) (struct file * file);

int (*task_create) (unsigned long clone_flags);
- int (*task_alloc_security) (struct task_struct * p);
+ int (*task_alloc_security) (struct task_struct *orig,
+ struct task_struct * p);
void (*task_free_security) (struct task_struct * p);

```

```

int (*task_setuid) (uid_t id0, uid_t id1, uid_t id2, int flags);
int (*task_post_setuid) (uid_t old_ruid /* or fsuid */ ,
@@ -1528,7 +1530,7 @@ int security_file_send_sigiotask(struct task_struct *tsk,
    struct fown_struct *fown, int sig);
int security_file_receive(struct file *file);
int security_task_create(unsigned long clone_flags);
-int security_task_alloc(struct task_struct *p);
+int security_task_alloc(struct task_struct *orig, struct task_struct *p);
void security_task_free(struct task_struct *p);
int security_task_setuid(uid_t id0, uid_t id1, uid_t id2, int flags);
int security_task_post_setuid(uid_t old_ruid, uid_t old_euid,
@@ -1995,7 +1997,8 @@ static inline int security_task_create (unsigned long clone_flags)
    return 0;
}

```

```

-static inline int security_task_alloc (struct task_struct *p)
+static inline int security_task_alloc (struct task_struct *orig,
+    struct task_struct *p)
{
    return 0;
}

```

diff --git a/kernel/fork.c b/kernel/fork.c

index dd9d4fd..7759c13 100644

--- a/kernel/fork.c

+++ b/kernel/fork.c

```

@@ -605,16 +605,25 @@ struct fs_struct *copy_fs_struct(struct fs_struct *old)

```

```

EXPORT_SYMBOL_GPL(copy_fs_struct);

```

```

-static inline int copy_fs(unsigned long clone_flags, struct task_struct * tsk)
+static inline int copy_fs(unsigned long clone_flags,
+    struct task_struct * src, struct task_struct * tsk)
{
+    int ret = 0;
+
+    if (src != current)
+        task_lock(src);
+    if (clone_flags & CLONE_FS) {
-        atomic_inc(&current->fs->count);
-        return 0;
+        atomic_inc(&src->fs->count);
+        goto out;
    }
-    tsk->fs = __copy_fs_struct(current->fs);
+    tsk->fs = __copy_fs_struct(src->fs);
    if (!tsk->fs)
-        return -ENOMEM;
-    return 0;

```

```

+ ret = -ENOMEM;
+
+out:
+ if (src != current)
+ task_unlock(src);
+ return ret;
}

static int count_open_files(struct fdtable *fdt)
@@ -957,7 +966,8 @@ static inline void rt_mutex_init_task(struct task_struct *p)
 * parts of the process environment (as per the clone
 * flags). The actual kick-off is left to the caller.
 */
-static struct task_struct *copy_process(unsigned long clone_flags,
+static struct task_struct *copy_process(struct task_struct *task,
+ unsigned long clone_flags,
+ unsigned long stack_start,
+ struct pt_regs *regs,
+ unsigned long stack_size,
@@ -992,7 +1002,7 @@ static struct task_struct *copy_process(unsigned long clone_flags,
 goto fork_out;

retval = -ENOMEM;
- p = dup_task_struct(current);
+ p = dup_task_struct(task);
if (!p)
goto fork_out;

@@ -1029,7 +1039,7 @@ static struct task_struct *copy_process(unsigned long clone_flags,
 goto bad_fork_cleanup_put_domain;

if (pid != &init_struct_pid) {
- pid = alloc_pid();
+ pid = alloc_pid(task);
if (!pid)
goto bad_fork_put_binfmt_module;
}
@@ -1116,18 +1126,17 @@ static struct task_struct *copy_process(unsigned long clone_flags,

p->tgid = p->pid;
if (clone_flags & CLONE_THREAD)
- p->tgid = current->tgid;
+ p->tgid = task->tgid;

- if ((retval = security_task_alloc(p)))
+ if ((retval = security_task_alloc(task, p)))
goto bad_fork_cleanup_policy;
if ((retval = audit_alloc(p)))

```

```

    goto bad_fork_cleanup_security;
- /* copy all the process information */
  if ((retval = copy_semundo(clone_flags, p)))
    goto bad_fork_cleanup_audit;
  if ((retval = copy_files(clone_flags, p)))
    goto bad_fork_cleanup_semundo;
- if ((retval = copy_fs(clone_flags, p)))
+ if ((retval = copy_fs(clone_flags, task, p)))
    goto bad_fork_cleanup_files;
  if ((retval = copy_sighand(clone_flags, p)))
    goto bad_fork_cleanup_fs;
@@ -1139,7 +1148,11 @@ static struct task_struct *copy_process(unsigned long clone_flags,
    goto bad_fork_cleanup_mm;
  if ((retval = copy_namespaces(clone_flags, p)))
    goto bad_fork_cleanup_keys;
- retval = copy_thread(0, clone_flags, stack_start, stack_size, p, regs);
+ if (task != current)
+   task_lock(task);
+ retval = copy_a_thread(task, 0, clone_flags, stack_start, stack_size, p, regs);
+ if (task != current)
+   task_unlock(task);
  if (retval)
    goto bad_fork_cleanup_namespaces;

@@ -1352,8 +1365,8 @@ struct task_struct * __cpuinit fork_idle(int cpu)
  struct task_struct *task;
  struct pt_regs regs;

- task = copy_process(CLONE_VM, 0, idle_regs(&regs), 0, NULL, NULL,
-   &init_struct_pid);
+ task = copy_process(current, CLONE_VM, 0, idle_regs(&regs), 0, NULL,
+   NULL, &init_struct_pid);
  if (!IS_ERR(task))
    init_idle(task, cpu);

@@ -1377,12 +1390,12 @@ static inline int fork_traceflag (unsigned clone_flags)
}

/*
- * Ok, this is the main fork-routine.
- *
- * It copies the process, and if successful kick-starts
- * it and waits for it to finish using the VM if required.
+ * if called with task!=current, then caller must ensure that
+ * 1. it has a reference to task
+ * 2. current must have ptrace permission to task
*/
-long do_fork(unsigned long clone_flags,

```



```

#define ARCH_MIN_MMSTRUCT_ALIGN 0
#endif
diff --git a/kernel/pid.c b/kernel/pid.c
index bb07851..c4a3182 100644
--- a/kernel/pid.c
+++ b/kernel/pid.c
@@ -212,14 +212,14 @@ fastcall void free_pid(struct pid *pid)
    call_rcu(&pid->rcu, delayed_put_pid);
}

-struct pid *alloc_pid(void)
+struct pid *alloc_pid(struct task_struct *task)
{
    struct pid *pid;
    enum pid_type type;
    int nr = -1;
    struct pid_namespace *ns;

- ns = task_active_pid_ns(current);
+ ns = task_active_pid_ns(task);
    pid = kmem_cache_alloc(ns->pid_cachep, GFP_KERNEL);
    if (!pid)
        goto out;
diff --git a/security/dummy.c b/security/dummy.c
index 5839faa..183bc86 100644
--- a/security/dummy.c
+++ b/security/dummy.c
@@ -477,7 +477,8 @@ static int dummy_task_create (unsigned long clone_flags)
    return 0;
}

-static int dummy_task_alloc_security (struct task_struct *p)
+static int dummy_task_alloc_security (struct task_struct *orig,
+    struct task_struct *p)
{
    return 0;
}
diff --git a/security/security.c b/security/security.c
index 335e1e1..3bba21e 100644
--- a/security/security.c
+++ b/security/security.c
@@ -616,9 +616,9 @@ int security_task_create(unsigned long clone_flags)
    return security_ops->task_create(clone_flags);
}

-int security_task_alloc(struct task_struct *p)
+int security_task_alloc(struct task_struct *orig, struct task_struct *p)
{

```

```

- return security_ops->task_alloc_security(p);
+ return security_ops->task_alloc_security(orig, p);
}

void security_task_free(struct task_struct *p)
diff --git a/security/selinux/hooks.c b/security/selinux/hooks.c
index 850087c..e37f531 100644
--- a/security/selinux/hooks.c
+++ b/security/selinux/hooks.c
@@ -2739,12 +2739,13 @@ static int selinux_task_create(unsigned long clone_flags)
    return task_has_perm(current, current, PROCESS__FORK);
}

-static int selinux_task_alloc_security(struct task_struct *tsk)
+static int selinux_task_alloc_security(struct task_struct *orig,
+    struct task_struct *tsk)
{
    struct task_security_struct *tsec1, *tsec2;
    int rc;

- tsec1 = current->security;
+ tsec1 = orig->security;

    rc = task_alloc_security(tsk);
    if (rc)
--
1.5.1

```

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