
Subject: [PATCH] namespaces: introduce sys_hijack (v4)

Posted by [serue](#) on Tue, 09 Oct 2007 20:09:28 GMT

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

>From 945fe66259cd0cfdc2fe846287b7821e329a558c Mon Sep 17 00:00:00 2001

From: sergeh@us.ibm.com <hallyn@kernel.(none)>

Date: Tue, 9 Oct 2007 08:30:30 -0700

Subject: [PATCH] namespaces: introduce sys_hijack (v4)

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, ...).

Introduce sys_hijack (for x86 only so far). It is like clone, but in place of a stack pointer (which is assumed null) it accepts a pid. The process identified by that pid is the one which is actually cloned. Some state - include the file table, the signals and sighand (and hence tty), and the ->parent are taken from the calling process.

The effect is a sort of namespace enter. The following program uses sys_hijack to 'enter' all namespaces of the specified pid. For instance in one terminal, do

```
mount -t cgroup -ons /cgroup
hostname
qemu
ns_exec -u /bin/sh
hostname serge
echo $$
1073
cat /proc/$$/cgroup
ns:/node_1073
```

In another terminal then do

```
hostname
qemu
cat /proc/$$/cgroup
ns:/
hijack 1073
hostname
serge
cat /proc/$$/cgroup
ns:/node_1073
```

sys_hijack is arch-dependent and is only implemented for i386 so far.

Changelog:

Aug 23: send a stop signal to the hijacked process
(like ptrace does).

Oct 09: Update for 2.6.23-rc8-mm2 (mainly pidns)

Don't take task_lock under rcu_read_lock

Send hijacked process to cgroup_fork() as
the first argument.

Removed some unneeded task_locks.

```
=====
hijack.c
=====
```

```
int do_clone_task(void)
{
    execl("/bin/sh", "/bin/sh", NULL);
}

int main(int argc, char *argv[])
{
    int pid;
    int ret;
    int status;

    if (argc < 2)
        return 1;
    pid = atoi(argv[1]);

    ret = syscall(327, SIGCHLD, pid, NULL, NULL);

    if (ret == 0) {
        return do_clone_task();
    } else if (ret < 0) {
        perror("sys_hijack");
    } else {
        printf("waiting on cloned process %d\n", ret);
        while (waitpid(ret, &status, __WCLONE) != ret);
        printf("cloned process %d exited with %d\n", ret, status);
    }

    return ret;
}
```

```
=====
```

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

```
arch/i386/kernel/process.c      | 58 ++++++
arch/i386/kernel/syscall_table.S | 1 +
```

```

arch/s390/kernel/process.c      | 12 +++++-
include/asm-i386/unistd.h       | 3 +-
include/linux/cgroup.h          | 5 ++-
include/linux/pid.h             | 2 +-
include/linux/ptrace.h          | 1 +
include/linux/sched.h           | 2 +
include/linux/syscalls.h        | 1 +
kernel/cgroup.c                 | 8 ++--
kernel/fork.c                   | 69 ++++++-----
kernel/pid.c                    | 5 ++-
kernel/ptrace.c                 | 7 ++++
13 files changed, 141 insertions(+), 33 deletions(-)

```

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

```
index bfcd01e..01f4d16 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);
@@ -783,6 +789,54 @@ asmlinkage int sys_clone(struct pt_regs regs)
    return do_fork(clone_flags, newsp, &regs, 0, parent_tidptr, child_tidptr);
}

```

```

+asmlinkage int sys_hijack(struct pt_regs regs)
+{

```

```

+ unsigned long clone_flags;
+ int __user *parent_tidptr, *child_tidptr;
+ pid_t pid;
+ struct task_struct *task;
+ int ret = -EINVAL;
+
+ clone_flags = regs.ebx;
+ pid = regs.ecx;
+ parent_tidptr = (int __user *)regs.edx;
+ child_tidptr = (int __user *)regs.edi;
+
+ rcu_read_lock();
+ task = find_task_by_vpid(pid);
+ if (task)
+   get_task_struct(task);
+ rcu_read_unlock();
+
+ if (task) {
+   task_lock(task);
+   put_task_struct(task);
+ }
+
+ if (task) {
+   if (!ptrace_may_attach_locked(task)) {
+     ret = -EPERM;
+     goto out_put_task;
+   }
+   if (task->ptrace) {
+     ret = -EBUSY;
+     goto out_put_task;
+   }
+   force_sig_specific(SIGSTOP, task);
+
+   task_unlock(task);
+   ret = do_fork_task(task, clone_flags, regs.esp, &regs, 0,
+     parent_tidptr, child_tidptr);
+   wake_up_process(task);
+   task = NULL;
+ }
+
+out_put_task:
+ if (task)
+   task_unlock(task);
+ return ret;
+}
+
+/*

```

* This is trivial, and on the face of it looks like it

```

* could equally well be done in user mode.
diff --git a/arch/i386/kernel/syscall_table.S b/arch/i386/kernel/syscall_table.S
index df6e41e..495930c 100644
--- a/arch/i386/kernel/syscall_table.S
+++ b/arch/i386/kernel/syscall_table.S
@@ -326,3 +326,4 @@ ENTRY(sys_call_table)
    .long sys_fallocate
    .long sys_revoket /* 325 */
    .long sys_frevoke
+ .long sys_hijack
diff --git a/arch/s390/kernel/process.c b/arch/s390/kernel/process.c
index 70c5737..f256e7a 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/asm-i386/unistd.h b/include/asm-i386/unistd.h
index 006c1b3..fe6eeb4 100644
--- a/include/asm-i386/unistd.h
+++ b/include/asm-i386/unistd.h
@@ -332,10 +332,11 @@
#define __NR_fallocate 324

```

```

#define __NR_revokeat 325
#define __NR_frevoked 326
+#define __NR_hijack 327

#ifdef __KERNEL__

-#define NR_syscalls 327
+#define NR_syscalls 328

#define __ARCH_WANT_IPC_PARSE_VERSION
#define __ARCH_WANT_OLD_READDIR
diff --git a/include/linux/cgroup.h b/include/linux/cgroup.h
index 8747932..cb6d335 100644
--- a/include/linux/cgroup.h
+++ b/include/linux/cgroup.h
@@ -26,7 +26,7 @@ extern int cgroup_init(void);
extern void cgroup_init_smp(void);
extern void cgroup_lock(void);
extern void cgroup_unlock(void);
-extern void cgroup_fork(struct task_struct *p);
+extern void cgroup_fork(struct task_struct *parent, struct task_struct *p);
extern void cgroup_fork_callbacks(struct task_struct *p);
extern void cgroup_post_fork(struct task_struct *p);
extern void cgroup_exit(struct task_struct *p, int run_callbacks);
@@ -309,7 +309,8 @@ void cgroup_iter_end(struct cgroup *cont, struct cgroup_iter *it);
static inline int cgroup_init_early(void) { return 0; }
static inline int cgroup_init(void) { return 0; }
static inline void cgroup_init_smp(void) {}
-static inline void cgroup_fork(struct task_struct *p) {}
+static inline void cgroup_fork(struct task_struct *parent,
+ struct task_struct *p) {}
static inline void cgroup_fork_callbacks(struct task_struct *p) {}
static inline void cgroup_post_fork(struct task_struct *p) {}
static inline void cgroup_exit(struct task_struct *p, int callbacks) {}
diff --git a/include/linux/pid.h b/include/linux/pid.h
index e29a900..145dce7 100644
--- a/include/linux/pid.h
+++ b/include/linux/pid.h
@@ -119,7 +119,7 @@ extern struct pid *find_pid(int nr);
extern struct pid *find_get_pid(int nr);
extern struct pid *find_get_pid(int nr, struct pid_namespace *ns);

-extern struct pid *alloc_pid(struct pid_namespace *ns);
+extern struct pid *alloc_pid(struct task_struct *task);
extern void FASTCALL(free_pid(struct pid *pid));
extern void zap_pid_ns_processes(struct pid_namespace *pid_ns);

diff --git a/include/linux/ptrace.h b/include/linux/ptrace.h

```

index ae8146a..727a4a9 100644

--- a/include/linux/ptrace.h

+++ b/include/linux/ptrace.h

```
@@ -97,6 +97,7 @@ extern void __ptrace_link(struct task_struct *child,
extern void __ptrace_unlink(struct task_struct *child);
extern void ptrace_untrace(struct task_struct *child);
extern int ptrace_may_attach(struct task_struct *task);
+extern int ptrace_may_attach_locked(struct task_struct *task);
```

```
static inline void ptrace_link(struct task_struct *child,
                               struct task_struct *new_parent)
```

diff --git a/include/linux/sched.h b/include/linux/sched.h

index 4f21af1..d85c3cf 100644

--- a/include/linux/sched.h

+++ b/include/linux/sched.h

```
@@ -1630,6 +1630,7 @@ 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);
```

```
@@ -1645,6 +1646,7 @@ extern int allow_signal(int);
extern int disallow_signal(int);
```

```
extern int do_execve(char *, char __user * __user *, char __user * __user *, struct pt_regs *);
+extern long do_fork_task(struct task_struct *task, unsigned long, unsigned long, struct pt_regs *,
unsigned long, int __user *, int __user *);
extern long do_fork(unsigned long, unsigned long, struct pt_regs *, unsigned long, int __user *,
int __user *);
struct task_struct *fork_idle(int);
```

diff --git a/include/linux/syscalls.h b/include/linux/syscalls.h

index f696874..5bc7384 100644

--- a/include/linux/syscalls.h

+++ b/include/linux/syscalls.h

```
@@ -616,5 +616,6 @@ int kernel_execve(const char *filename, char *const argv[], char *const
envp[]);
```

```
asmlinkage long sys_revoket(int dfd, const char __user *filename);
asmlinkage long sys_frevoket(unsigned int fd);
+asmlinkage long sys_hijack(unsigned long flags, pid_t pid, int __user *ptid, int __user *ctid);
```

#endif

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

index 1e8aa53..e587896 100644

--- a/kernel/cgroup.c

+++ b/kernel/cgroup.c

@@ -2460,12 +2460,12 @@ static struct file_operations proc_cgroupstats_operations = {

* At the point that cgroup_fork() is called, 'current' is the parent

* task, and the passed argument 'child' points to the child task.

*/

-void cgroup_fork(struct task_struct *child)

+void cgroup_fork(struct task_struct *parent, struct task_struct *child)

{

- task_lock(current);

- child->cgroups = current->cgroups;

+ task_lock(parent);

+ child->cgroups = parent->cgroups;

get_css_set(child->cgroups);

- task_unlock(current);

+ task_unlock(parent);

INIT_LIST_HEAD(&child->cg_list);

}

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

index f85731a..ac73f3e 100644

--- a/kernel/fork.c

+++ b/kernel/fork.c

@@ -621,13 +621,14 @@ 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)

{

if (clone_flags & CLONE_FS) {

- atomic_inc(¤t->fs->count);

+ atomic_inc(&src->fs->count);

return 0;

}

- tsk->fs = __copy_fs_struct(current->fs);

+ tsk->fs = __copy_fs_struct(src->fs);

if (!tsk->fs)

return -ENOMEM;

return 0;

@@ -973,7 +974,8 @@ 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,
@@ -1007,15 +1009,17 @@ 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;

      rt_mutex_init_task(p);

#ifdef CONFIG_TRACE_IRQFLAGS
- DEBUG_LOCKS_WARN_ON(!p->hardirqs_enabled);
- DEBUG_LOCKS_WARN_ON(!p->softirqs_enabled);
+ if (task == current) {
+   DEBUG_LOCKS_WARN_ON(!p->hardirqs_enabled);
+   DEBUG_LOCKS_WARN_ON(!p->softirqs_enabled);
+ }
#endif
      retval = -EAGAIN;
      if (atomic_read(&p->user->processes) >=
@@ -1084,7 +1088,7 @@ static struct task_struct *copy_process(unsigned long clone_flags,
#endif
      p->io_context = NULL;
      p->audit_context = NULL;
- cgroup_fork(p);
+ cgroup_fork(task, p);
#ifdef CONFIG_NUMA
      p->mempolicy = mpol_copy(p->mempolicy);
      if (IS_ERR(p->mempolicy)) {
@@ -1132,7 +1136,7 @@ static struct task_struct *copy_process(unsigned long clone_flags,
          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;
@@ -1144,13 +1148,13 @@ 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);

```

```

+ retval = copy_a_thread(task, 0, clone_flags, stack_start, stack_size, p, regs);
if (retval)
    goto bad_fork_cleanup_namespaces;

if (pid != &init_struct_pid) {
    retval = -ENOMEM;
- pid = alloc_pid(task_active_pid_ns(p));
+ pid = alloc_pid(task);
    if (!pid)
        goto bad_fork_cleanup_namespaces;

@@ -1164,7 +1168,7 @@ static struct task_struct *copy_process(unsigned long clone_flags,
    p->pid = pid_nr(pid);
    p->tgid = p->pid;
    if (clone_flags & CLONE_THREAD)
- p->tgid = current->tgid;
+ p->tgid = task->tgid;

    p->set_child_tid = (clone_flags & CLONE_CHILD_SETTID) ? child_tidptr : NULL;
/*
@@ -1380,7 +1384,7 @@ 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,
+ task = copy_process(current, CLONE_VM, 0, idle_regs(&regs), 0, NULL,
    &init_struct_pid);
    if (!IS_ERR(task))
        init_idle(task, cpu);
@@ -1405,12 +1409,12 @@ 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,
+long do_fork_task(struct task_struct *task,
+ unsigned long clone_flags,
    unsigned long stack_start,
    struct pt_regs *regs,
    unsigned long stack_size,
@@ -1421,13 +1425,23 @@ long do_fork(unsigned long clone_flags,

```



```

diff --git a/kernel/pid.c b/kernel/pid.c
index d7388d7..b887a6a 100644
--- a/kernel/pid.c
+++ b/kernel/pid.c
@@ -238,14 +238,15 @@ fastcall void free_pid(struct pid *pid)
    call_rcu(&pid->rcu, delayed_put_pid);
}

-struct pid *alloc_pid(struct pid_namespace *ns)
+struct pid *alloc_pid(struct task_struct *srctsk)
{
    struct pid *pid;
    enum pid_type type;
    int i, nr;
- struct pid_namespace *tmp;
+ struct pid_namespace *tmp, *ns;
    struct upid *upid;

+ ns = task_active_pid_ns(srctsk);
    pid = kmem_cache_alloc(ns->pid_cachep, GFP_KERNEL);
    if (!pid)
        goto out;
diff --git a/kernel/ptrace.c b/kernel/ptrace.c
index 7c76f2f..c65c9fe 100644
--- a/kernel/ptrace.c
+++ b/kernel/ptrace.c
@@ -159,6 +159,13 @@ int ptrace_may_attach(struct task_struct *task)
    return !err;
}

+int ptrace_may_attach_locked(struct task_struct *task)
+{
+ int err;
+ err = may_attach(task);
+ return !err;
+}
+
+int ptrace_attach(struct task_struct *task)
+{
    int retval;
--
1.5.1

```

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