Subject: [RFC] [PATCH 0/2] namespace enter through hijack
Posted by serue on Wed, 29 Aug 2007 20:04:20 GMT
View Forum Message <> Reply to Message

This patchset implements namespace entering by forcing a task in
the target namespace to clone itself. This has some advantages
over just replacing a random task's namespace pointers to the
target ones. In particular

1. If switching pid namespaces, the stack of upids is
   automatically correctly generated.
2. Security context is inherited from the target
   task. Assuming a security module which labels
   data based on the task security context, like selinux,
   this may prevent severe mislabeling of container data by
   an inadvertant host system administrator action.
   Whether that works or not will still depend on the
   policy and the task cloned.

While this version takes a pid of a process to clone (for
convenience of prototyping) we may prefer to use a ns_container
name and choose one of it's tasks, to prevent pid wraparound.

Tested and 'works for me', but at the moment I'm just sending
this out for discussion.

Alternatives to this include bind_ns()+switch_ns() by Cedric
and the ns_container namespace entering enhancements I've
previously sent.

thanks,
-serge

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

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,
+long do_fork_task(struct task_struct *task,
+  unsigned long clone_flags,
        unsigned long stack_start,
        struct pt_regs *regs,
        unsigned long stack_size,
@@ -1393,13 +1406,19 @@ long do_fork(unsigned long clone_flags,
  int trace = 0;
  long nr;

+ if (task != current) {
```

```
+  /* sanity checks */
+  /* we only want to allow hijacking the simplest cases */
+  if (clone_flags & CLONE_SYSVSEM)
+   return -EINVAL;
+ }
  if (unlikely(current->ptrace)) {
   trace = fork_traceflag (clone_flags);
   if (trace)
    clone_flags |= CLONE_PTRACE;
  }

- p = copy_process(clone_flags, stack_start, regs, stack_size,
+ p = copy_process(task, clone_flags, stack_start, regs, stack_size,
    parent_tidptr, child_tidptr, NULL);
  /*
   * Do this prior waking up the new thread - the thread pointer
@@ -1448,6 +1467,23 @@ long do_fork(unsigned long clone_flags,
  return nr;
 }

+/*
+ *  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.
+ */
+long do_fork(unsigned long clone_flags,
+       unsigned long stack_start,
+       struct pt_regs *regs,
+       unsigned long stack_size,
+       int __user *parent_tidptr,
+       int __user *child_tidptr)
+{
+ return do_fork_task(current, clone_flags, stack_start,
+  regs, stack_size, parent_tidptr, child_tidptr);
+}
+
 #ifndef ARCH_MIN_MMSTRUCT_ALIGN
 #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

---

Subject: [RFC] [PATCH 2/2] namespace enter: introduce sys_hijack (v3)
Posted by serue on Wed, 29 Aug 2007 20:05:15 GMT
View Forum Message <> Reply to Message

>From aec05999084bf3a94add66e98462652ed9408f86 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 2/2] namespace enter: introduce sys_hijack (v3)

Introduce sys_hijack (for x86 only). 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

```
 hostname
   qemu
 ns_exec -u /bin/sh
   hostname ab
```

```
        echo $$
          1073
```

In another terminal then do

```
 hostname
   qemu
 hijack 1073
   hostname
     ab
```

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

==========
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);
  ret = waitpid(ret, &status, __WALL);
  printf("cloned process exited with %d (waitpid ret %d)\n",
    status, ret);
 }

 return ret;
```

```
}
============================================================

Signed-off-by: sergeh@us.ibm.com <hallyn@kernel.(none)>
---
 arch/i386/kernel/process.c       |   51 ++++++++++++++++++++++++++++++++++++++
 arch/i386/kernel/syscall_table.S |    1 +
 include/asm-i386/unistd.h        |    3 +-
 include/linux/ptrace.h           |    2 +
 include/linux/sched.h            |    1 +
 include/linux/syscalls.h         |    1 +
 kernel/ptrace.c                  |   10 +++++++
 7 files changed, 68 insertions(+), 1 deletions(-)

diff --git a/arch/i386/kernel/process.c b/arch/i386/kernel/process.c
index e01ddac..b13de30 100644
--- a/arch/i386/kernel/process.c
+++ b/arch/i386/kernel/process.c
@@ -789,6 +789,57 @@ 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_pid_type(PIDTYPE_PID, pid);
+ if (task)
+  task_lock(task);
+ rcu_read_unlock();
+
+/* Serge: I'm not clear on this.  Do I need to grab a write
+ * lock to tasklist_lock, like ptrace does? */
+
+ if (task) {
+  if (!ptrace_may_attach_locked(task)) {
+   ret = -EPERM;
+   goto out_put_task;
```

```
+  }
+  if (task->ptrace || (task->state & PF_EXITING)) {
+   ret = -EBUSY;
+   goto out_put_task;
+  }
+  task->ptrace |= PT_HIJACKED;
+  force_sig_specific(SIGSTOP, task);
+  task_unlock(task);
+
+  ret = do_fork_task(task, clone_flags, regs.esp, &regs, 0,
+   parent_tidptr, child_tidptr);
+
+  task_lock(task);
+  task->ptrace = 0;
+  task_unlock(task);
+  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_revokeat  /* 325 */
  .long sys_frevoke
+ .long sys_hijack
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_frevoke  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/ptrace.h b/include/linux/ptrace.h
index ae8146a..ca953ab 100644
--- a/include/linux/ptrace.h
+++ b/include/linux/ptrace.h
@@ -68,6 +68,7 @@
 #define PT_TRACE_VFORK_DONE 0x00000100
 #define PT_TRACE_EXIT 0x00000200
 #define PT_ATTACHED 0x00000400 /* parent != real_parent */
+#define PT_HIJACKED 0x00000800 /* not ptrace, but hijack ongoing */

 #define PT_TRACE_MASK 0x000003f4

@@ -97,6 +98,7 @@ 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 7fa6710..b46ae34 100644
--- a/include/linux/sched.h
+++ b/include/linux/sched.h
@@ -1534,6 +1534,7 @@ 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_revokeat(int dfd, const char __user *filename);
```

```
 asmlinkage long sys_frevoke(unsigned int fd);
+asmlinkage long sys_hijack(unsigned long flags, pid_t pid, int __user *ptid, int __user *ctid);

 #endif
diff --git a/kernel/ptrace.c b/kernel/ptrace.c
index 085943d..4f5c6a0 100644
--- a/kernel/ptrace.c
+++ b/kernel/ptrace.c
@@ -158,6 +158,13 @@ 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;
@@ -195,6 +202,9 @@ repeat:
  /* the same process cannot be attached many times */
  if (task->ptrace & PT_PTRACED)
   goto bad;
+ /* ... or ptraced while being hijacked */
+ if (task->ptrace & PT_HIJACKED)
+  goto bad;
  retval = may_attach(task);
  if (retval)
   goto bad;
--
1.5.1
```

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

---

Subject: Re: [RFC] [PATCH 2/2] namespace enter: introduce sys_hijack (v3)
Posted by Paul Menage on Mon, 03 Sep 2007 08:02:46 GMT
View Forum Message <> Reply to Message

On 8/29/07, Serge E. Hallyn <serue@us.ibm.com> wrote:
> >From aec05999084bf3a94add66e98462652ed9408f86 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 2/2] namespace enter: introduce sys_hijack (v3)
>
> Introduce sys_hijack (for x86 only). 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.

What do you do if there are no processes in a particular container?

I prefer your suggestion of tying this to the nsproxy subsystem - that
would allow you to spawn a child with a given set of namespaces, even
if there were no appropriate process to hijack.

Paul

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

---

Subject: Re: [RFC] [PATCH 2/2] namespace enter: introduce sys_hijack (v3)
Posted by serue on Tue, 04 Sep 2007 12:50:19 GMT
View Forum Message <> Reply to Message

Quoting Paul Menage (menage@google.com):
> On 8/29/07, Serge E. Hallyn <serue@us.ibm.com> wrote:
> > >From aec05999084bf3a94add66e98462652ed9408f86 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 2/2] namespace enter: introduce sys_hijack (v3)
> >
> > Introduce sys_hijack (for x86 only). 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.
>
> What do you do if there are no processes in a particular container?

The nsproxy will have been released so you couldn't enter it anyway.

> I prefer your suggestion of tying this to the nsproxy subsystem - that
> would allow you to spawn a child with a given set of namespaces, even
> if there were no appropriate process to hijack.

I can resend my original ns_container entering patchset (maybe next week
when everyone is back from summits) and we can discuss whether or not it
is safe, or how to improve it if it is not.

thanks,
-serge

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

---

Subject: Re: [RFC] [PATCH 2/2] namespace enter: introduce sys_hijack (v3)
Posted by Dave Hansen on Tue, 04 Sep 2007 19:06:18 GMT
View Forum Message <> Reply to Message

On Tue, 2007-09-04 at 07:50 -0500, Serge E. Hallyn wrote:
> > What do you do if there are no processes in a particular container?
>
> The nsproxy will have been released so you couldn't enter it anyway.

Yeah, we'd need some kind of other object to keep the nsproxy around and
hold a reference to it.

But, it also begs other questions about how we define the namespace
boundaries vs. containers. What if we have a normal container with
chroot'd process inside of it? Two such processes will not share an
nsproxy because the chroot'd one has switched filesystem namespaces.

Who is to say that the "container" is represented by one process's
nsproxy more than another?

-- Dave

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

---

Subject: Re: [RFC] [PATCH 2/2] namespace enter: introduce sys_hijack (v3)
Posted by serue on Tue, 04 Sep 2007 19:32:56 GMT
View Forum Message <> Reply to Message

Quoting Dave Hansen (haveblue@us.ibm.com):
> On Tue, 2007-09-04 at 07:50 -0500, Serge E. Hallyn wrote:

> > > What do you do if there are no processes in a particular container?
> >
> > The nsproxy will have been released so you couldn't enter it anyway.
>
> Yeah, we'd need some kind of other object to keep the nsproxy around and
> hold a reference to it.

We could of course have the ns_container subsystem do that. The ns_container generally stick around until the admin does a manual rm on its directory, so this way we could keep the nsproxy around.

> But, it also begs other questions about how we define the namespace
> boundaries vs. containers. What if we have a normal container with
> chroot'd process inside of it? Two such processes will not share an
> nsproxy because the chroot'd one has switched filesystem namespaces.

But then a chroot isn't really anything to do with a namespace. An equivalent would be clone(CLONE_NEWNS)+pivot_root(new_root,put_old). And that would cause a new namespace for the child. Which is why I think we need to be able to define a container as a set of nsproxies, either by introducing CLONE_NEWCONTAINER or using the ns_container subsystem.

Then a namespace enter would always be done into the namespace init process' nsproxy.

> Who is to say that the "container" is represented by one process's
> nsproxy more than another?

The admin who defined the container I guess :)

One major ugliness is that the definition of namespace boundaries is different with each ns.

Uts namespaces are completely distinct and creates as copies of the original.

Mounts namespaces are by default isolated, and created as copies of the original. But sharing can be done using mounts propagation.

Pid namespaces are hierarchical, with processes being visible in all ancestor namespaces.

IPC namespaces are isolated and created empty.

Network namespaces will be isolated and created empty, with network devices being shared between a parent and child namespace?

In the face of that straight namespace entering is the simplest
way to administer a 'container'. Without namespace entering, every
namespace may need to be administrated somewhat differently, if it is
even possible. (i.e. utsname0

-serge

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

---

Subject: Re: [RFC] [PATCH 2/2] namespace enter: introduce sys_hijack (v3)
Posted by Paul Menage on Wed, 05 Sep 2007 16:15:11 GMT
View Forum Message <> Reply to Message

On 9/4/07, Serge E. Hallyn <serue@us.ibm.com> wrote:
> We could of course have the ns_container subsystem do that. The
> ns_container generally stick around until the admin does a manual rm on
> its directory, so this way we could keep the nsproxy around.

So how about taking sys_hijack() even further and integrate it with
control groups too? So when you do sys_hijack() (or maybe an
alternative name would be sys_fork_in()?) you create a task that
inherits all the control groups of the target task, as well as the
namespaces.

Paul

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

---

Subject: Re: [RFC] [PATCH 2/2] namespace enter: introduce sys_hijack (v3)
Posted by serue on Mon, 10 Sep 2007 19:17:43 GMT
View Forum Message <> Reply to Message

Quoting Paul Menage (menage@google.com):
> On 9/4/07, Serge E. Hallyn <serue@us.ibm.com> wrote:
> > We could of course have the ns_container subsystem do that. The
> > ns_container generally stick around until the admin does a manual rm on
> > its directory, so this way we could keep the nsproxy around.
>
> So how about taking sys_hijack() even further and integrate it with
> control groups too? So when you do sys_hijack() (or maybe an
> alternative name would be sys_fork_in()?) you create a task that

> inherits all the control groups of the target task, as well as the
> namespaces.
>
> Paul

Sorry don't know why i haven't replied to this.

Good point. I see container_fork(p) takes the container from current.
I can change that to container_fork(src, dest) in my next posting.

Is there any reason why we wouldn't want to do that? For instance a
container admin could impose some restrictions which would keep the host
admin from doing something through sys_hijack()? (Not sure that's a big
worry since the restrictions would apply to the container admin as well)

thanks,
-serge

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