

---

Subject: Re: [PATCH 1/2] namespaces: introduce sys\_hijack (v10)

Posted by [serue](#) on Tue, 27 Nov 2007 16:11:32 GMT

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

---

Quoting Crispin Cowan ([crispin@crispincowan.com](mailto:crispin@crispincowan.com)):

> Just the name "sys\_hijack" makes me concerned.

>

> This post describes a bunch of "what", but doesn't tell us about "why"

> we would want this. What is it for?

Please see my response to Casey's email.

> And I second Casey's concern about careful management of the privilege

> required to "hijack" a process.

Absolutely. We're definately still in RFC territory.

Note that there are currently several proposed (but no upstream) ways to accomplish entering a namespace:

1. `bind_ns()` is a new pair of syscalls proposed by Cedric. An nsproxy is given an integer id. The id can be used to enter an nsproxy, basically a straight `current->nsproxy = target_nsproxy;`

2. I had previously posted a patchset on top of the nsproxy cgroup which allowed entering a nsproxy through the ns cgroup interface.

There are objections to both those patchsets because simply switching a task's nsproxy using a syscall or file write in the middle of running a binary is quite unsafe. Eric Biederman had suggested using ptrace or something like it to accomplish the goal.

Just using ptrace is however not safe either. You are inheriting \*all\* of the target's context, so it shouldn't be difficult for a nefarious container/vserver admin to trick the host admin into running something which gives the container/vserver admin full access to the host.

That's where the hijack idea came from. Yes, I called it hijack to make sure alarm bells went off :) bc it's definately still worrisome. But at this point I believe it is the safest solution suggested so far.

-serge

> Crispin

>

> Mark Nelson wrote:

> > Here's the latest version of sys\_hijack.

> > Apologies for its lateness.  
> >  
> > Thanks!  
> >  
> > Mark.  
> >  
> > Subject: [PATCH 1/2] namespaces: introduce sys\_hijack (v10)  
> >  
> > 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 i386 and s390 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 - including the file  
> > table, the signals and sighand (and hence tty), and the ->parent  
> > are taken from the calling process.  
> >  
> > A process to be hijacked may be identified by process id, in the  
> > case of HIJACK\_PID. Alternatively, in the case of HIJACK\_CG an  
> > open fd for a cgroup 'tasks' file may be specified. The first  
> > available task in that cgroup will then be hijacked.  
> >  
> > HIJACK\_NS is implemented as a third hijack method. The main  
> > purpose is to allow entering an empty cgroup without having  
> > to keep a task alive in the target cgroup. When HIJACK\_NS  
> > is called, only the cgroup and nsproxy are copied from the  
> > cgroup. Security, user, and rootfs info is not retained  
> > in the cgroups and so cannot be copied to the child task.  
> >  
> > In order to hijack a process, the calling process must be  
> > allowed to ptrace the target.  
> >  
> > Sending sigstop to the hijacked task can trick its parent shell  
> > (if it is a shell foreground task) into thinking it should retake  
> > its tty.  
> >  
> > So try not sending SIGSTOP, and instead hold the task\_lock over  
> > the hijacked task throughout the do\_fork\_task() operation.  
> > This is really dangerous. I've fixed cgroup\_fork() to not  
> > task\_lock(task) in the hijack case, but there may well be other  
> > code called during fork which can under "some circumstances"  
> > task\_lock(task).  
> >  
> > Still, this is working for me.  
> >  
> > The effect is a sort of namespace enter. The following program

```

>> uses sys_hijack to 'enter' all namespaces of the specified task.
>> For instance in one terminal, do
>>
>> mount -t cgroup -ons cgroup /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/$$/cgroup
>> ns:/
>> hijack pid 1073
>> hostname
>> serge
>> cat /proc/$$/cgroup
>> ns:/node_1073
>> hijack cgroup /cgroup/node_1073/tasks
>>
>> 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.
>> Oct 16: Fix bug introduced into alloc_pid.
>> Oct 16: Add 'int which' argument to sys_hijack to
>> allow later expansion to use cgroup in place
>> of pid to specify what to hijack.
>> Oct 24: Implement hijack by open cgroup file.
>> Nov 02: Switch copying of task info: do full copy
>> from current, then copy relevant pieces from
>> hijacked task.
>> Nov 06: Verbatim task_struct copy now comes from current,
>> after which copy_hijackable_taskinfo() copies
>> relevant context pieces from the hijack source.
>> Nov 07: Move arch-independent hijack code to kernel/fork.c
>> Nov 07: powerpc and x86_64 support (Mark Nelson)
>> Nov 07: Don't allow hijacking members of same session.

```

```

> > Nov 07: introduce cgroup_may_hijack, and may_hijack hook to
> > cgroup subsystems. The ns subsystem uses this to
> > enforce the rule that one may only hijack descendent
> > namespaces.
> > Nov 07: s390 support
> > Nov 08: don't send SIGSTOP to hijack source task
> > Nov 10: cache reference to nsproxy in ns cgroup for use in
> > hijacking an empty cgroup.
> > Nov 10: allow partial hijack of empty cgroup
> > Nov 13: don't double-get cgroup for hijack_ns
> > find_css_set() actually returns the set with a
> > reference already held, so cgroup_fork_fromcgroup()
> > by doing a get_css_set() was getting a second
> > reference. Therefore after exiting the hijack
> > task we could not rmdir the csgroup.
> > Nov 22: temporarily remove x86_64 and powerpc support
> > Nov 27: rebased on 2.6.24-rc3
> >
> > =====
> > hijack.c
> > =====
> > /*
> > * Your options are:
> > * hijack pid 1078
> > * hijack cgroup /cgroup/node_1078/tasks
> > * hijack ns /cgroup/node_1078/tasks
> > */
> >
> > #define _BSD_SOURCE
> > #include <unistd.h>
> > #include <sys/syscall.h>
> > #include <sys/types.h>
> > #include <sys/wait.h>
> > #include <sys/stat.h>
> > #include <fcntl.h>
> > #include <sched.h>
> > #include <stdio.h>
> > #include <stdlib.h>
> > #include <string.h>
> >
> > #if __i386__
> > #   define __NR_hijack 325
> > #elif __s390x__
> > #   define __NR_hijack 319
> > #else
> > #   error "Architecture not supported"
> > #endif
> >

```

```

> > #ifndef CLONE_NEWUTS
> > #define CLONE_NEWUTS 0x04000000
> > #endif
> >
> > void usage(char *me)
> > {
> > printf("Usage: %s pid <pid>\n", me);
> > printf("    | %s cgroup <cgroup_tasks_file>\n", me);
> > printf("    | %s ns <cgroup_tasks_file>\n", me);
> > exit(1);
> > }
> >
> > int exec_shell(void)
> > {
> > execl("/bin/sh", "/bin/sh", NULL);
> > }
> >
> > #define HIJACK_PID 1
> > #define HIJACK_CG 2
> > #define HIJACK_NS 3
> >
> > int main(int argc, char *argv[])
> > {
> > int id;
> > int ret;
> > int status;
> > int which_hijack;
> >
> > if (argc < 3 || !strcmp(argv[1], "-h"))
> >     usage(argv[0]);
> > if (strcmp(argv[1], "cgroup") == 0)
> >     which_hijack = HIJACK_CG;
> > else if (strcmp(argv[1], "ns") == 0)
> >     which_hijack = HIJACK_NS;
> > else
> >     which_hijack = HIJACK_PID;
> >
> > switch(which_hijack) {
> > case HIJACK_PID:
> >     id = atoi(argv[2]);
> >     printf("hijacking pid %d\n", id);
> >     break;
> > case HIJACK_CG:
> > case HIJACK_NS:
> >     id = open(argv[2], O_RDONLY);
> >     if (id == -1) {
> >         perror("cgroup open");
> >         return 1;

```

```

>> }
>> break;
>> }
>>
>> ret = syscall(__NR_hijack, SIGCHLD, which_hijack, (unsigned long)id);
>>
>> if (which_hijack != HIJACK_PID)
>> close(id);
>> if (ret == 0) {
>> return exec_shell();
>> } else if (ret < 0) {
>> perror("sys_hijack");
>> } else {
>> printf("waiting on cloned process %d\n", ret);
>> while(waitpid(-1, &status, __WALL) != -1)
>> ;
>> printf("cloned process exited with %d (waitpid ret %d)\n",
>> status, ret);
>> }
>>
>> return ret;
>> }
>> =====
>>
>> Signed-off-by: Serge Hallyn <serue@us.ibm.com>
>> Signed-off-by: Mark Nelson <markn@au1.ibm.com>
>> ---
>> Documentation/cgroups.txt      | 9 +
>> arch/s390/kernel/process.c     | 21 +++
>> arch/x86/kernel/process_32.c   | 24 ++++
>> arch/x86/kernel/syscall_table_32.S | 1
>> include/asm-x86/unistd_32.h    | 3
>> include/linux/cgroup.h        | 28 ++++-
>> include/linux/nsproxy.h       | 12 +-
>> include/linux/ptrace.h        | 1
>> include/linux/sched.h         | 19 +++
>> include/linux/syscalls.h      | 2
>> kernel/cgroup.c               | 133 ++++++
>> kernel/fork.c                  | 201 ++++++
>> kernel/ns_cgroup.c            | 88 ++++++
>> kernel/nsproxy.c              | 4
>> kernel/ptrace.c                | 7 +
>> 15 files changed, 523 insertions(+), 30 deletions(-)
>>
>> Index: upstream/arch/s390/kernel/process.c
>> =====
>> --- upstream.orig/arch/s390/kernel/process.c
>> +++ upstream/arch/s390/kernel/process.c

```

```

>> @@ -321,6 +321,27 @@ asmlinkage long sys_clone(void)
>>     parent_tidptr, child_tidptr);
>> }
>>
>> +asmlinkage long sys_hijack(void)
>> +{
>> + struct pt_regs *regs = task_pt_regs(current);
>> + unsigned long sp = regs->orig_gpr2;
>> + unsigned long clone_flags = regs->gprs[3];
>> + int which = regs->gprs[4];
>> + unsigned int fd;
>> + pid_t pid;
>> +
>> + switch (which) {
>> + case HIJACK_PID:
>> + pid = regs->gprs[5];
>> + return hijack_pid(pid, clone_flags, *regs, sp);
>> + case HIJACK_CGROUP:
>> + fd = (unsigned int) regs->gprs[5];
>> + return hijack_cgroup(fd, clone_flags, *regs, sp);
>> + default:
>> + return -EINVAL;
>> + }
>> +}
>> +
>> /*
>> * This is trivial, and on the face of it looks like it
>> * could equally well be done in user mode.
>> Index: upstream/arch/x86/kernel/process_32.c
>> =====
>> --- upstream.orig/arch/x86/kernel/process_32.c
>> +++ upstream/arch/x86/kernel/process_32.c
>> @@ -37,6 +37,7 @@
>> #include <linux/personality.h>
>> #include <linux/tick.h>
>> #include <linux/percpu.h>
>> +#include <linux/cgroup.h>
>>
>> #include <asm/uaccess.h>
>> #include <asm/pgtable.h>
>> @@ -781,6 +782,29 @@ asmlinkage int sys_clone(struct pt_regs
>>     return do_fork(clone_flags, newsp, &regs, 0, parent_tidptr, child_tidptr);
>> }
>>
>> +asmlinkage int sys_hijack(struct pt_regs regs)
>> +{
>> + unsigned long sp = regs.esp;
>> + unsigned long clone_flags = regs.ebx;

```

```

>> + int which = regs.ecx;
>> + unsigned int fd;
>> + pid_t pid;
>> +
>> + switch (which) {
>> + case HIJACK_PID:
>> + pid = regs.edx;
>> + return hijack_pid(pid, clone_flags, regs, sp);
>> + case HIJACK_CGROUP:
>> + fd = (unsigned int) regs.edx;
>> + return hijack_cgroup(fd, clone_flags, regs, sp);
>> + case HIJACK_NS:
>> + fd = (unsigned int) regs.edx;
>> + return hijack_ns(fd, clone_flags, regs, sp);
>> + default:
>> + return -EINVAL;
>> + }
>> +}
>> +
>> /*
>> * This is trivial, and on the face of it looks like it
>> * could equally well be done in user mode.
>> Index: upstream/arch/x86/kernel/syscall_table_32.S
>> =====
>> --- upstream.orig/arch/x86/kernel/syscall_table_32.S
>> +++ upstream/arch/x86/kernel/syscall_table_32.S
>> @@ -324,3 +324,4 @@ ENTRY(sys_call_table)
>> .long sys_timerfd
>> .long sys_eventfd
>> .long sys_fallocate
>> + .long sys_hijack /* 325 */
>> Index: upstream/Documentation/cgroups.txt
>> =====
>> --- upstream.orig/Documentation/cgroups.txt
>> +++ upstream/Documentation/cgroups.txt
>> @@ -495,6 +495,15 @@ LL=cgroup_mutex
>> Called after the task has been attached to the cgroup, to allow any
>> post-attachment activity that requires memory allocations or blocking.
>>
>> +int may_hijack(struct cgroup_subsys *ss, struct cgroup *cont,
>> + struct task_struct *task)
>> +LL=cgroup_mutex
>> +
>> +Called prior to hijacking a task. Current is cloning a new child
>> +which is hijacking cgroup, namespace, and security context from
>> +the target task. Called with the hijacked task locked. Return
>> +0 to allow.
>> +

```



```

>> void fork(struct cgroup_subsys *ss, struct task_struct *task)
>> LL=callback_mutex, maybe read_lock(tasklist_lock)
>>
>> Index: upstream/include/asm-x86/unistd_32.h
>> =====
>> --- upstream.orig/include/asm-x86/unistd_32.h
>> +++ upstream/include/asm-x86/unistd_32.h
>> @@ -330,10 +330,11 @@
>> #define __NR_timerfd 322
>> #define __NR_eventfd 323
>> #define __NR_fallocate 324
>> +#define __NR_hijack 325
>>
>> #ifdef __KERNEL__
>>
>> -#define NR_syscalls 325
>> +#define NR_syscalls 326
>>
>> #define __ARCH_WANT_IPC_PARSE_VERSION
>> #define __ARCH_WANT_OLD_READDIR
>> Index: upstream/include/linux/cgroup.h
>> =====
>> --- upstream.orig/include/linux/cgroup.h
>> +++ upstream/include/linux/cgroup.h
>> @@ -14,19 +14,23 @@
>> #include <linux/nodemask.h>
>> #include <linux/rcupdate.h>
>> #include <linux/cgroupstats.h>
>> +#include <linux/err.h>
>>
>> #ifdef CONFIG_CGROUPS
>>
>> struct cgroupfs_root;
>> struct cgroup_subsys;
>> struct inode;
>> +struct cgroup;
>>
>> extern int cgroup_init_early(void);
>> 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_fromcgroup(struct cgroup *new_cg,
>> + struct task_struct *child);
>> 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);
>> @@ -236,6 +240,8 @@ struct cgroup_subsys {
>> void (*destroy)(struct cgroup_subsys *ss, struct cgroup *cont);
>> int (*can_attach)(struct cgroup_subsys *ss,
>>     struct cgroup *cont, struct task_struct *tsk);
>> + int (*may_hijack)(struct cgroup_subsys *ss,
>> +     struct cgroup *cont, struct task_struct *tsk);
>> void (*attach)(struct cgroup_subsys *ss, struct cgroup *cont,
>>     struct cgroup *old_cont, struct task_struct *tsk);
>> void (*fork)(struct cgroup_subsys *ss, struct task_struct *task);
>> @@ -304,12 +310,21 @@ struct task_struct *cgroup_iter_next(str
>>     struct cgroup_iter *it);
>> void cgroup_iter_end(struct cgroup *cont, struct cgroup_iter *it);
>>
>> +struct cgroup *cgroup_from_fd(unsigned int fd);
>> +struct task_struct *task_from_cgroup_fd(unsigned int fd);
>> +int cgroup_may_hijack(struct task_struct *tsk);
>> #else /* !CONFIG_CGROUPS */
>> +struct cgroup {
>> +};
>>
>> 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_fromcgroup(struct cgroup *new_cg,
>> +     struct task_struct *child) {}
>> +
>> 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) {}
>> @@ -322,6 +337,15 @@ static inline int cgroupstats_build(stru
>> return -EINVAL;
>> }
>>
>> +static inline struct cgroup *cgroup_from_fd(unsigned int fd) { return NULL; }
>> +static inline struct task_struct *task_from_cgroup_fd(unsigned int fd)
>> +{
>> + return ERR_PTR(-EINVAL);
>> +}
>> +static inline int cgroup_may_hijack(struct task_struct *tsk)
>> +{
>> + return 0;
>> +}
>> #endif /* !CONFIG_CGROUPS */
>>

```

```

>> #endif /* _LINUX_CGROUP_H */
>> Index: upstream/include/linux/nsproxy.h
>> =====
>> --- upstream.orig/include/linux/nsproxy.h
>> +++ upstream/include/linux/nsproxy.h
>> @@ -3,6 +3,7 @@
>>
>> #include <linux/spinlock.h>
>> #include <linux/sched.h>
>> +#include <linux/err.h>
>>
>> struct mnt_namespace;
>> struct uts_namespace;
>> @@ -81,10 +82,17 @@ static inline void get_nsproxy(struct ns
>> atomic_inc(&ns->count);
>> }
>>
>> +struct cgroup;
>> #ifdef CONFIG_CGROUP_NS
>> -int ns_cgroup_clone(struct task_struct *tsk);
>> +int ns_cgroup_clone(struct task_struct *tsk, struct nsproxy *nsproxy);
>> +int ns_cgroup_verify(struct cgroup *cgroup);
>> +void copy_hijack_nsproxy(struct task_struct *tsk, struct cgroup *cgroup);
>> #else
>> -static inline int ns_cgroup_clone(struct task_struct *tsk) { return 0; }
>> +static inline int ns_cgroup_clone(struct task_struct *tsk,
>> + struct nsproxy *nsproxy) { return 0; }
>> +static inline int ns_cgroup_verify(struct cgroup *cgroup) { return 0; }
>> +static inline void copy_hijack_nsproxy(struct task_struct *tsk,
>> +      struct cgroup *cgroup) {}
>> #endif
>>
>> #endif
>> Index: upstream/include/linux/ptrace.h
>> =====
>> --- upstream.orig/include/linux/ptrace.h
>> +++ upstream/include/linux/ptrace.h
>> @@ -97,6 +97,7 @@ extern void __ptrace_link(struct task_st
>> 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)
>> Index: upstream/include/linux/sched.h
>> =====
>> --- upstream.orig/include/linux/sched.h

```

```

>> +++ upstream/include/linux/sched.h
>> @@ -29,6 +29,13 @@
>> #define CLONE_NEWNET 0x40000000 /* New network namespace */
>>
>> /*
>> + * Hijack flags
>> + */
>> +#define HIJACK_PID 1 /* 'id' is a pid */
>> +#define HIJACK_CGROUP 2 /* 'id' is an open fd for a cgroup dir */
>> +#define HIJACK_NS 3 /* 'id' is an open fd for a cgroup dir */
>> +
>> +/*
>> * Scheduling policies
>> */
>> #define SCHED_NORMAL 0
>> @@ -1693,9 +1700,19 @@ 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(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);
>>
>> +extern int hijack_task(struct task_struct *task, unsigned long clone_flags,
>> + struct pt_regs regs, unsigned long sp);
>> +extern int hijack_pid(pid_t pid, unsigned long clone_flags, struct pt_regs regs,
>> + unsigned long sp);
>> +extern int hijack_cgroup(unsigned int fd, unsigned long clone_flags,
>> + struct pt_regs regs, unsigned long sp);
>> +extern int hijack_ns(unsigned int fd, unsigned long clone_flags,
>> + struct pt_regs regs, unsigned long sp);
>> +
>> extern void set_task_comm(struct task_struct *tsk, char *from);
>> extern void get_task_comm(char *to, struct task_struct *tsk);
>>
>> Index: upstream/include/linux/syscalls.h
>> =====
>> --- upstream.orig/include/linux/syscalls.h
>> +++ upstream/include/linux/syscalls.h
>> @@ -614,4 +614,6 @@ asmlinkage long sys_fallocate(int fd, in
>>
>> int kernel_execve(const char *filename, char *const argv[], char *const envp[]);
>>
>> +asmlinkage long sys_hijack(unsigned long flags, int which, unsigned long id);
>> +
>> #endif

```

```

> > Index: upstream/kernel/cgroup.c
> > =====
> > --- upstream.orig/kernel/cgroup.c
> > +++ upstream/kernel/cgroup.c
> > @@ -44,6 +44,7 @@
> > #include <linux/kmod.h>
> > #include <linux/delayacct.h>
> > #include <linux/cgroupstats.h>
> > +#include <linux/file.h>
> >
> > #include <asm/atomic.h>
> >
> > @@ -2442,15 +2443,25 @@ static struct file_operations proc_cgrou
> > * 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;
> > + if (parent == current)
> > + task_lock(parent);
> > + child->cgroups = parent->cgroups;
> > get_css_set(child->cgroups);
> > - task_unlock(current);
> > + if (parent == current)
> > + task_unlock(parent);
> > INIT_LIST_HEAD(&child->cg_list);
> > }
> >
> > +void cgroup_fork_fromcgroup(struct cgroup *new_cg, struct task_struct *child)
> > +{
> > + mutex_lock(&cgroup_mutex);
> > + child->cgroups = find_css_set(child->cgroups, new_cg);
> > + INIT_LIST_HEAD(&child->cg_list);
> > + mutex_unlock(&cgroup_mutex);
> > +}
> > +
> > /**
> > * cgroup_fork_callbacks - called on a new task very soon before
> > * adding it to the tasklist. No need to take any locks since no-one
> > @@ -2801,3 +2812,117 @@ static void cgroup_release_agent(struct
> > spin_unlock(&release_list_lock);
> > mutex_unlock(&cgroup_mutex);
> > }
> > +
> > +static inline int task_available(struct task_struct *task)

```

```

>> +{
>> + if (task == current)
>> + return 0;
>> + if (task_session(task) == task_session(current))
>> + return 0;
>> + switch (task->state) {
>> + case TASK_RUNNING:
>> + case TASK_INTERRUPTIBLE:
>> + return 1;
>> + default:
>> + return 0;
>> + }
>> +}
>> +
>> +struct cgroup *cgroup_from_fd(unsigned int fd)
>> +{
>> + struct file *file;
>> + struct cgroup *cgroup = NULL;;
>> +
>> + file = fget(fd);
>> + if (!file)
>> + return NULL;
>> +
>> + if (!file->f_dentry || !file->f_dentry->d_sb)
>> + goto out_fput;
>> + if (file->f_dentry->d_parent->d_sb->s_magic != CGROUP_SUPER_MAGIC)
>> + goto out_fput;
>> + if (strcmp(file->f_dentry->d_name.name, "tasks"))
>> + goto out_fput;
>> +
>> + cgroup = __d_cgrp(file->f_dentry->d_parent);
>> +
>> +out_fput:
>> + fput(file);
>> + return cgroup;
>> +}
>> +
>> +/*
>> + * Takes an integer which is a open fd in current for a valid
>> + * cgroupfs file. Returns a task in that cgroup, with its
>> + * refcount bumped.
>> + * Since we have an open file on the cgroup tasks file, we
>> + * at least don't have to worry about the cgroup being freed
>> + * in the middle of this.
>> + */
>> +struct task_struct *task_from_cgroup_fd(unsigned int fd)
>> +{
>> + struct cgroup *cgroup;

```

```

>> + struct cgroup_iter it;
>> + struct task_struct *task = NULL;
>> +
>> + cgroup = cgroup_from_fd(fd);
>> + if (!cgroup)
>> + return NULL;
>> +
>> + rcu_read_lock();
>> + cgroup_iter_start(cgroup, &it);
>> + do {
>> + task = cgroup_iter_next(cgroup, &it);
>> + if (task)
>> + printk(KERN_NOTICE "task %d state %lx\n",
>> + task->pid, task->state);
>> + } while (task && !task_available(task));
>> + cgroup_iter_end(cgroup, &it);
>> + if (task)
>> + get_task_struct(task);
>> + rcu_read_unlock();
>> + return task;
>> +}
>> +
>> +/*
>> + * is current allowed to hijack tsk?
>> + * permission will also be denied elsewhere if
>> + * current may not ptrace tsk
>> + * security_task_alloc(new_task, tsk) returns -EPERM
>> + * Here we are only checking whether current may attach
>> + * to tsk's cgroup. If you can't enter the cgroup, you can't
>> + * hijack it.
>> + *
>> + * XXX TODO This means that ns_cgroup.c will need to allow
>> + * entering all descendent cgroups, not just the immediate
>> + * child.
>> + */
>> +int cgroup_may_hijack(struct task_struct *tsk)
>> +{
>> + int ret = 0;
>> + struct cgroupfs_root *root;
>> +
>> + mutex_lock(&cgroup_mutex);
>> + for_each_root(root) {
>> + struct cgroup_subsys *ss;
>> + struct cgroup *cgroup;
>> + int subsys_id;
>> +
>> + /* Skip this hierarchy if it has no active subsystems */
>> + if (!root->actual_subsys_bits)

```

```

>> + continue;
>> + get_first_subsys(&root->top_cgroup, NULL, &subsys_id);
>> + cgroup = task_cgroup(tsk, subsys_id);
>> + for_each_subsys(root, ss) {
>> +   if (ss->may_hijack) {
>> +     ret = ss->may_hijack(ss, cgroup, tsk);
>> +     if (ret)
>> +       goto out_unlock;
>> +   }
>> + }
>> + }
>> + }
>> +
>> +out_unlock:
>> + mutex_unlock(&cgroup_mutex);
>> + return ret;
>> +}
>> Index: upstream/kernel/fork.c
>> =====
>> --- upstream.orig/kernel/fork.c
>> +++ upstream/kernel/fork.c
>> @@ -189,7 +189,7 @@ static struct task_struct *dup_task_stru
>>   return NULL;
>> }
>>
>> - setup_thread_stack(tsk, orig);
>> + setup_thread_stack(tsk, current);
>>
>> #ifdef CONFIG_CC_STACKPROTECTOR
>>   tsk->stack_canary = get_random_int();
>> @@ -616,13 +616,14 @@ struct fs_struct *copy_fs_struct(struct
>>
>> EXPORT_SYMBOL_GPL(copy_fs_struct);
>>
>> -static 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(&current->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;
>> @@ -962,6 +963,42 @@ static void rt_mutex_init_task(struct ta

```



```

>> #endif
>> }
>>
>> +void copy_hijackable_taskinfo(struct task_struct *p,
>> + struct task_struct *task)
>> +{
>> + p->uid = task->uid;
>> + p->euid = task->euid;
>> + p->suid = task->suid;
>> + p->fsuid = task->fsuid;
>> + p->gid = task->gid;
>> + p->egid = task->egid;
>> + p->sgid = task->sgid;
>> + p->fsgid = task->fsgid;
>> + p->cap_effective = task->cap_effective;
>> + p->cap_inheritable = task->cap_inheritable;
>> + p->cap_permitted = task->cap_permitted;
>> + p->keep_capabilities = task->keep_capabilities;
>> + p->user = task->user;
>> + /*
>> + * should keys come from parent or hijack-src?
>> + */
>> + #ifdef CONFIG_SYSVIPC
>> + p->sysvsem = task->sysvsem;
>> + #endif
>> + p->fs = task->fs;
>> + p->nsproxy = task->nsproxy;
>> + }
>> +
>> + #define HIJACK_SOURCE_TASK 1
>> + #define HIJACK_SOURCE_CG 2
>> + struct hijack_source_info {
>> + char type;
>> + union hijack_source_union {
>> + struct task_struct *task;
>> + struct cgroup *cgroup;
>> + } u;
>> + };
>> +
>> + /*
>> + * This creates a new process as a copy of the old one,
>> + * but does not actually start it yet.
>> + @@ -970,7 +1007,8 @@ static void rt_mutex_init_task(struct ta
>> + * 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 hijack_source_info *src,

```

```

>> + unsigned long clone_flags,
>> unsigned long stack_start,
>> struct pt_regs *regs,
>> unsigned long stack_size,
>> @@ -980,6 +1018,12 @@ static struct task_struct *copy_process(
>> int retval;
>> struct task_struct *p;
>> int cgroup_callbacks_done = 0;
>> + struct task_struct *task;
>> +
>> + if (src->type == HIJACK_SOURCE_TASK)
>> + task = src->u.task;
>> + else
>> + task = current;
>>
>> if ((clone_flags & (CLONE_NEWNS|CLONE_FS)) == (CLONE_NEWNS|CLONE_FS))
>> return ERR_PTR(-EINVAL);
>> @@ -1007,6 +1051,10 @@ static struct task_struct *copy_process(
>> p = dup_task_struct(current);
>> if (!p)
>> goto fork_out;
>> + if (current != task)
>> + copy_hijackable_taskinfo(p, task);
>> + else if (src->type == HIJACK_SOURCE_CG)
>> + copy_hijack_nsproxy(p, src->u.cgroup);
>>
>> rt_mutex_init_task(p);
>>
>> @@ -1084,7 +1132,10 @@ static struct task_struct *copy_process(
>> #endif
>> p->io_context = NULL;
>> p->audit_context = NULL;
>> - cgroup_fork(p);
>> + if (src->type == HIJACK_SOURCE_CG)
>> + cgroup_fork_fromcgroup(src->u.cgroup, p);
>> + else
>> + cgroup_fork(task, p);
>> #ifdef CONFIG_NUMA
>> p->mempolicy = mpol_copy(p->mempolicy);
>> if (IS_ERR(p->mempolicy)) {
>> @@ -1135,7 +1186,7 @@ static struct task_struct *copy_process(
>> 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;
>> @@ -1167,7 +1218,7 @@ static struct task_struct *copy_process(
>> 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;
>> /*
>> @@ -1378,8 +1429,12 @@ struct task_struct * __cpuinit fork_idle
>> {
>> struct task_struct *task;
>> struct pt_regs regs;
>> + struct hijack_source_info src;
>>
>> - task = copy_process(CLONE_VM, 0, idle_regs(&regs), 0, NULL,
>> + src.type = HIJACK_SOURCE_TASK;
>> + src.u.task = current;
>> +
>> + task = copy_process(&src, CLONE_VM, 0, idle_regs(&regs), 0, NULL,
>> &init_struct_pid);
>> if (!IS_ERR(task))
>> init_idle(task, cpu);
>> @@ -1404,29 +1459,43 @@ static int fork_traceflag(unsigned clone
>> }
>>
>> /*
>> - * 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 hijack_source_info *src,
>> + unsigned long clone_flags,
>> unsigned long stack_start,
>> struct pt_regs *regs,
>> unsigned long stack_size,
>> int __user *parent_tidptr,
>> int __user *child_tidptr)
>> {
>> - struct task_struct *p;
>> + struct task_struct *p, *task;
>> int trace = 0;

```

```

>> long nr;
>>
>> + if (src->type == HIJACK_SOURCE_TASK)
>> + task = src->u.task;
>> + else
>> + task = current;
>> + if (task != current) {
>> + /* sanity checks */
>> + /* we only want to allow hijacking the simplest cases */
>> + if (clone_flags & CLONE_SYSVSEM)
>> + return -EINVAL;
>> + if (current->ptrace)
>> + return -EPERM;
>> + if (task->ptrace)
>> + 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(src, clone_flags, stack_start, regs, stack_size,
>> child_tidptr, NULL);
>> /*
>> * Do this prior waking up the new thread - the thread pointer
>> @@ -1484,6 +1553,106 @@ 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)
>> +{
>> + struct hijack_source_info src = {
>> + .type = HIJACK_SOURCE_TASK,
>> + .u = { .task = current, },
>> + };

```

```

>> + return do_fork_task(&src, clone_flags, stack_start,
>> + regs, stack_size, parent_tidptr, child_tidptr);
>> +}
>> +
>> +/*
>> + * Called with task count bumped, drops task count before returning
>> + */
>> +int hijack_task(struct task_struct *task, unsigned long clone_flags,
>> + struct pt_regs regs, unsigned long sp)
>> +{
>> + int ret = -EPERM;
>> + struct hijack_source_info src = {
>> + .type = HIJACK_SOURCE_TASK,
>> + .u = { .task = task, },
>> + };
>> +
>> + task_lock(task);
>> + put_task_struct(task);
>> + if (!ptrace_may_attach_locked(task))
>> + goto out_unlock_task;
>> + if (task == current)
>> + goto out_unlock_task;
>> + ret = cgroup_may_hijack(task);
>> + if (ret)
>> + goto out_unlock_task;
>> + if (task->ptrace) {
>> + ret = -EBUSY;
>> + goto out_unlock_task;
>> + }
>> + ret = do_fork_task(&src, clone_flags, sp, &regs, 0, NULL, NULL);
>> +
>> +out_unlock_task:
>> + task_unlock(task);
>> + return ret;
>> +}
>> +
>> +int hijack_pid(pid_t pid, unsigned long clone_flags, struct pt_regs regs,
>> + unsigned long sp)
>> +{
>> + struct task_struct *task;
>> +
>> + rcu_read_lock();
>> + task = find_task_by_vpid(pid);
>> + if (task)
>> + get_task_struct(task);
>> + rcu_read_unlock();
>> +
>> + if (!task)

```

```

>> + return -EINVAL;
>> +
>> + return hijack_task(task, clone_flags, regs, sp);
>> +}
>> +
>> +int hijack_cgroup(unsigned int fd, unsigned long clone_flags,
>> + struct pt_regs regs, unsigned long sp)
>> +{
>> + struct task_struct *task;
>> +
>> + task = task_from_cgroup_fd(fd);
>> + if (!task)
>> + return -EINVAL;
>> +
>> + return hijack_task(task, clone_flags, regs, sp);
>> +}
>> +
>> +int hijack_ns(unsigned int fd, unsigned long clone_flags,
>> + struct pt_regs regs, unsigned long sp)
>> +{
>> + struct hijack_source_info src;
>> + struct cgroup *cgroup;
>> +
>> + cgroup = cgroup_from_fd(fd);
>> + if (!cgroup)
>> + return -EINVAL;
>> +
>> + if (!ns_cgroup_verify(cgroup))
>> + return -EINVAL;
>> +
>> + src.type = HIJACK_SOURCE_CG;
>> + src.u.cgroup = cgroup;
>> + return do_fork_task(&src, clone_flags, sp, &regs, 0, NULL, NULL);
>> +}
>> +
>> #ifndef ARCH_MIN_MMSTRUCT_ALIGN
>> #define ARCH_MIN_MMSTRUCT_ALIGN 0
>> #endif
>> Index: upstream/kernel/ns_cgroup.c
>> =====
>> --- upstream.orig/kernel/ns_cgroup.c
>> +++ upstream/kernel/ns_cgroup.c
>> @@ -7,9 +7,11 @@
>> #include <linux/module.h>
>> #include <linux/cgroup.h>
>> #include <linux/fs.h>
>> +#include <linux/nsproxy.h>
>>

```

```

>> struct ns_cgroup {
>> struct cgroup_subsys_state css;
>> + struct nsproxy *nsproxy;
>> spinlock_t lock;
>> };
>>
>> @@ -22,9 +24,51 @@ static inline struct ns_cgroup *cgroup_t
>> struct ns_cgroup, css);
>> }
>>
>> -int ns_cgroup_clone(struct task_struct *task)
>> +int ns_cgroup_clone(struct task_struct *task, struct nsproxy *nsproxy)
>> {
>> - return cgroup_clone(task, &ns_subsys);
>> + struct cgroup *cgroup;
>> + struct ns_cgroup *ns_cgroup;
>> + int ret = cgroup_clone(task, &ns_subsys);
>> +
>> + if (ret)
>> + return ret;
>> +
>> + cgroup = task_cgroup(task, ns_subsys_id);
>> + ns_cgroup = cgroup_to_ns(cgroup);
>> + ns_cgroup->nsproxy = nsproxy;
>> + get_nsproxy(nsproxy);
>> +
>> + return 0;
>> +}
>> +
>> +int ns_cgroup_verify(struct cgroup *cgroup)
>> +{
>> + struct cgroup_subsys_state *css;
>> + struct ns_cgroup *ns_cgroup;
>> +
>> + css = cgroup_subsys_state(cgroup, ns_subsys_id);
>> + if (!css)
>> + return 0;
>> + ns_cgroup = container_of(css, struct ns_cgroup, css);
>> + if (!ns_cgroup->nsproxy)
>> + return 0;
>> + return 1;
>> +}
>> +
>> +/*
>> + * this shouldn't be called unless ns_cgroup_verify() has
>> + * confirmed that there is a ns_cgroup in this cgroup
>> + *
>> + * tsk is not yet running, and has not yet taken a reference

```

```

>> + * to it's previous ->nsproxy, so we just do a simple assignment
>> + * rather than switch_task_namespaces()
>> + */
>> +void copy_hijack_nsproxy(struct task_struct *tsk, struct cgroup *cgroup)
>> +{
>> + struct ns_cgroup *ns_cgroup;
>> +
>> + ns_cgroup = cgroup_to_ns(cgroup);
>> + tsk->nsproxy = ns_cgroup->nsproxy;
>> }
>>
>> /*
>> @@ -60,6 +104,42 @@ static int ns_can_attach(struct cgroup_s
>> return 0;
>> }
>>
>> +static void ns_attach(struct cgroup_subsys *ss,
>> + struct cgroup *cgroup, struct cgroup *oldcgroup,
>> + struct task_struct *tsk)
>> +{
>> + struct ns_cgroup *ns_cgroup = cgroup_to_ns(cgroup);
>> +
>> + if (likely(ns_cgroup->nsproxy))
>> + return;
>> +
>> + spin_lock(&ns_cgroup->lock);
>> + if (!ns_cgroup->nsproxy) {
>> + ns_cgroup->nsproxy = tsk->nsproxy;
>> + get_nsproxy(ns_cgroup->nsproxy);
>> + }
>> + spin_unlock(&ns_cgroup->lock);
>> +}
>> +
>> +/*
>> + * only allow hijacking child namespaces
>> + * Q: is it crucial to prevent hijacking a task in your same cgroup?
>> + */
>> +static int ns_may_hijack(struct cgroup_subsys *ss,
>> + struct cgroup *new_cgroup, struct task_struct *task)
>> +{
>> + if (current == task)
>> + return -EINVAL;
>> +
>> + if (!capable(CAP_SYS_ADMIN))
>> + return -EPERM;
>> +
>> + if (!cgroup_is_descendant(new_cgroup))
>> + return -EPERM;

```



```

>> +
>> + return 0;
>> +}
>> +
>> /*
>> * Rules: you can only create a cgroup if
>> * 1. you are capable(CAP_SYS_ADMIN)
>> @@ -88,12 +168,16 @@ static void ns_destroy(struct cgroup_sub
>> struct ns_cgroup *ns_cgroup;
>>
>> ns_cgroup = cgroup_to_ns(cgroup);
>> + if (ns_cgroup->nsproxy)
>> + put_nsproxy(ns_cgroup->nsproxy);
>> kfree(ns_cgroup);
>> }
>>
>> struct cgroup_subsys ns_subsys = {
>> .name = "ns",
>> .can_attach = ns_can_attach,
>> + .attach = ns_attach,
>> + .may_hijack = ns_may_hijack,
>> .create = ns_create,
>> .destroy = ns_destroy,
>> .subsys_id = ns_subsys_id,
>> Index: upstream/kernel/nsproxy.c
>> =====
>> --- upstream.orig/kernel/nsproxy.c
>> +++ upstream/kernel/nsproxy.c
>> @@ -144,7 +144,7 @@ int copy_namespaces(unsigned long flags,
>> goto out;
>> }
>>
>> - err = ns_cgroup_clone(tsk);
>> + err = ns_cgroup_clone(tsk, new_ns);
>> if (err) {
>> put_nsproxy(new_ns);
>> goto out;
>> @@ -196,7 +196,7 @@ int unshare_nsproxy_namespaces(unsigned
>> goto out;
>> }
>>
>> - err = ns_cgroup_clone(current);
>> + err = ns_cgroup_clone(current, *new_nsp);
>> if (err)
>> put_nsproxy(*new_nsp);
>>
>> Index: upstream/kernel/ptrace.c
>> =====

```

```
> > --- upstream.orig/kernel/ptrace.c
> > +++ upstream/kernel/ptrace.c
> > @@ -159,6 +159,13 @@ int ptrace_may_attach(struct task_struct
> > 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;
> > -
> > To unsubscribe from this list: send the line "unsubscribe linux-security-module" in
> > the body of a message to majordomo@vger.kernel.org
> > More majordomo info at http://vger.kernel.org/majordomo-info.html
> >
> >
>
> --
> Crispin Cowan, Ph.D.      http://crispincowan.com/~crispin
> CEO, Mercenary Linux    http://mercenarylinux.com/
> Itanium. Vista. GPLv3. Complexity at work
>
> -
> To unsubscribe from this list: send the line "unsubscribe linux-security-module" in
> the body of a message to majordomo@vger.kernel.org
> More majordomo info at http://vger.kernel.org/majordomo-info.html
```

---

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

---