
Subject: [PATCH 2/8] user ns: add the framework
Posted by [serue](#) on Tue, 19 Dec 2006 23:00:00 GMT
[View Forum Message](#) <> [Reply to Message](#)

From: Cedric Le Goater <clg@fr.ibm.com>
Subject: [PATCH 2/8] user ns: add the framework

This patch adds the user namespace struct and framework

Basically, it will allow a process to unshare its user_struct table, resetting at the same time its own user_struct and all the associated accounting.

A new root user (uid == 0) is added to the user namespace upon creation. Such root users have full privileges and it seems that theses privileges should be controlled through some means (process capabilities ?)

The unshare is not included in this patch.

Changes since [try #4]:

- Updated get_user_ns and put_user_ns to accept NULL, and get_user_ns to return the namespace.

Changes since [try #3]:

- moved struct user_namespace to files user_namespace.{c,h}

Changes since [try #2]:

- removed struct user_namespace* argument from find_user()

Changes since [try #1]:

- removed struct user_namespace* argument from find_user()
- added a root_user per user namespace

Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>

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

```
include/linux/init_task.h    |  2 ++
include/linux/nsproxy.h      |  1 +
include/linux/sched.h        |  3 ++
include/linux/user_namespace.h | 53 +++++
init/Kconfig                 |  8 +++++
kernel/Makefile              |  2 +-
kernel/fork.c                |  2 +-
kernel/nsproxy.c             | 12 +++++
kernel/sys.c                 |  5 +--
kernel/user.c                | 18 +++++-----
```

kernel/user_namespace.c | 44 ++++++
11 files changed, 136 insertions(+), 14 deletions(-)

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

index a2d95ff..26dc24b 100644

--- a/include/linux/init_task.h

+++ b/include/linux/init_task.h

@@ -8,6 +8,7 @@ #include <linux/utsname.h>

#include <linux/lockdep.h>

#include <linux/ipc.h>

#include <linux/pid_namespace.h>

+ #include <linux/user_namespace.h>

#define INIT_FDTABLE \

{ \

@@ -78,6 +79,7 @@ #define INIT_NS_PROXY(nsproxy) { \

.uts_ns = &init_uts_ns, \

.mnt_ns = NULL, \

INIT_IPC_NS(ipc_ns) \

+ .user_ns = &init_user_ns, \

}

#define INIT_SIGHAND(sighand) { \

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

index 602f3d2..e57920e 100644

--- a/include/linux/nsproxy.h

+++ b/include/linux/nsproxy.h

@@ -29,6 +29,7 @@ struct nsproxy {

struct ipc_namespace *ipc_ns;

struct mnt_namespace *mnt_ns;

struct pid_namespace *pid_ns;

+ struct user_namespace *user_ns;

};

extern struct nsproxy init_nsproxy;

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

index d7e4de1..5a3f630 100644

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

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

@@ -252,6 +252,7 @@ extern signed long schedule_timeout_unin

asmlinkage void schedule(void);

struct nsproxy;

+ struct user_namespace;

/* Maximum number of active map areas.. This is a random (large) number */

#define DEFAULT_MAX_MAP_COUNT 65536

@@ -1285,7 +1286,7 @@ extern struct task_struct *find_task_by_


```

+
+##else
+
+static inline struct user_namespace *get_user_ns(struct user_namespace *ns)
+{
+ return NULL;
+}
+
+static inline int copy_user_ns(int flags, struct task_struct *tsk)
+{
+ return 0;
+}
+
+static inline void put_user_ns(struct user_namespace *ns)
+{
+}
+##endif
+
+##endif /* _LINUX_USER_H */
diff --git a/init/Kconfig b/init/Kconfig
index d1fed9e..c930912 100644
--- a/init/Kconfig
+++ b/init/Kconfig
@@ -222,6 +222,14 @@ config UTS_NS
     vservers, to use uts namespaces to provide different
     uts info for different servers. If unsure, say N.

+config USER_NS
+ bool "User Namespaces"
+ default n
+ help
+   Support user namespaces. This allows containers, i.e.
+   vservers, to use user namespaces to provide different
+   user info for different servers. If unsure, say N.
+
 config AUDIT
     bool "Auditing support"
     depends on NET
diff --git a/kernel/Makefile b/kernel/Makefile
index 839a58b..7c39952 100644
--- a/kernel/Makefile
+++ b/kernel/Makefile
@@ -4,7 +4,7 @@
obj-y = sched.o fork.o exec_domain.o panic.o printk.o profile.o \
        exit.o itimer.o time.o softirq.o resource.o \
-   sysctl.o capability.o ptrace.o timer.o user.o \
+   sysctl.o capability.o ptrace.o timer.o user.o user_namespace.o \

```

```

    signal.o sys.o kmod.o workqueue.o pid.o \
    rcupdate.o extable.o params.o posix-timers.o \
    kthread.o wait.o kfifo.o sys_ni.o posix-cpu-timers.o mutex.o \
diff --git a/kernel/fork.c b/kernel/fork.c
index a6ad544..deafa6e 100644
--- a/kernel/fork.c
+++ b/kernel/fork.c
@@ -996,7 +996,7 @@ @@ #endif
    if (atomic_read(&p->user->processes) >=
        p->signal->rlim[RLIMIT_NPROC].rlim_cur) {
        if (!capable(CAP_SYS_ADMIN) && !capable(CAP_SYS_RESOURCE) &&
-       p->user != &root_user)
+       p->user != current->nsproxy->user_ns->root_user)
            goto bad_fork_free;
    }

```

```

diff --git a/kernel/nsproxy.c b/kernel/nsproxy.c
index f11bbbf..cf43e06 100644
--- a/kernel/nsproxy.c
+++ b/kernel/nsproxy.c
@@ -80,6 +80,8 @@ @@ struct nsproxy *dup_namespaces(struct ns
    get_ipc_ns(ns->ipc_ns);
    if (ns->pid_ns)
        get_pid_ns(ns->pid_ns);
+   if (ns->user_ns)
+   get_user_ns(ns->user_ns);
}

```

```

    return ns;
@@ -127,10 +129,18 @@ @@ int copy_namespaces(int flags, struct ta
    if (err)
        goto out_pid;

+   err = copy_user_ns(flags, tsk);
+   if (err)
+       goto out_user;
+
    out:
        put_nsproxy(old_ns);
        return err;

+out_user:
+   if (new_ns->pid_ns)
+       put_pid_ns(new_ns->pid_ns);
+
    out_pid:
        if (new_ns->ipc_ns)
            put_ipc_ns(new_ns->ipc_ns);

```

```

@@ -156,5 +166,7 @@ void free_nsproxy(struct nsproxy *ns)
    put_ipc_ns(ns->ipc_ns);
    if (ns->pid_ns)
        put_pid_ns(ns->pid_ns);
+ if (ns->user_ns)
+ put_user_ns(ns->user_ns);
    kfree(ns);
}
diff --git a/kernel/sys.c b/kernel/sys.c
index 5590255..26bdd84 100644
--- a/kernel/sys.c
+++ b/kernel/sys.c
@@ -33,6 +33,7 @@ #include <linux/getcpu.h>
#include <linux/compat.h>
#include <linux/syscalls.h>
#include <linux/kprobes.h>
+#include <linux/user_namespace.h>

#include <asm/uaccess.h>
#include <asm/io.h>
@@ -1070,13 +1071,13 @@ static int set_user(uid_t new_ruid, int
{
    struct user_struct *new_user;

- new_user = alloc_uid(new_ruid);
+ new_user = alloc_uid(current->nsproxy->user_ns, new_ruid);
    if (!new_user)
        return -EAGAIN;

    if (atomic_read(&new_user->processes) >=
        current->signal->rlim[RLIMIT_NPROC].rlim_cur &&
- new_user != &root_user) {
+ new_user != current->nsproxy->user_ns->root_user) {
        free_uid(new_user);
        return -EAGAIN;
    }
}
diff --git a/kernel/user.c b/kernel/user.c
index 4869563..98b8250 100644
--- a/kernel/user.c
+++ b/kernel/user.c
@@ -14,20 +14,19 @@ #include <linux/slab.h>
#include <linux/bitops.h>
#include <linux/key.h>
#include <linux/interrupt.h>
+#include <linux/module.h>
+#include <linux/user_namespace.h>

/*

```

```

* UID task count cache, to get fast user lookup in "alloc_uid"
* when changing user ID's (ie setuid() and friends).
*/

#define UIDHASH_BITS (CONFIG_BASE_SMALL ? 3 : 8)
#define UIDHASH_SZ (1 << UIDHASH_BITS)
#define UIDHASH_MASK (UIDHASH_SZ - 1)
#define __uidhashfn(uid) (((uid >> UIDHASH_BITS) + uid) & UIDHASH_MASK)
#define uidhashentry(uid) (uidhash_table + __uidhashfn((uid)))
#define uidhashentry(ns, uid) ((ns)->uidhash_table + __uidhashfn((uid)))

static struct kmem_cache *uid_cache;
static struct list_head uidhash_table[UIDHASH_SZ];

/*
 * The uidhash_lock is mostly taken from process context, but it is
@@ -94,9 +93,10 @@ struct user_struct *find_user(uid_t uid)
{
    struct user_struct *ret;
    unsigned long flags;
+ struct user_namespace *ns = current->nsproxy->user_ns;

    spin_lock_irqsave(&uidhash_lock, flags);
- ret = uid_hash_find(uid, uidhashentry(uid));
+ ret = uid_hash_find(uid, uidhashentry(ns, uid));
    spin_unlock_irqrestore(&uidhash_lock, flags);
    return ret;
}
@@ -120,9 +120,9 @@ void free_uid(struct user_struct *up)
}
}

-struct user_struct * alloc_uid(uid_t uid)
+struct user_struct * alloc_uid(struct user_namespace *ns, uid_t uid)
{
- struct list_head *hashent = uidhashentry(uid);
+ struct list_head *hashent = uidhashentry(ns, uid);
    struct user_struct *up;

    spin_lock_irq(&uidhash_lock);
@@ -211,11 +211,11 @@ static int __init uid_cache_init(void)
    0, SLAB_HWCACHE_ALIGN|SLAB_PANIC, NULL, NULL);

    for(n = 0; n < UIDHASH_SZ; ++n)
- INIT_LIST_HEAD(uidhash_table + n);
+ INIT_LIST_HEAD(init_user_ns.uidhash_table + n);

    /* Insert the root user immediately (init already runs as root) */

```

```

    spin_lock_irq(&uidhash_lock);
- uid_hash_insert(&root_user, uidhashentry(0));
+ uid_hash_insert(&root_user, uidhashentry(&init_user_ns, 0));
    spin_unlock_irq(&uidhash_lock);

    return 0;
diff --git a/kernel/user_namespace.c b/kernel/user_namespace.c
new file mode 100644
index 0000000..368f8da
--- /dev/null
+++ b/kernel/user_namespace.c
@@ -0,0 +1,44 @@
+/*
+ * This program is free software; you can redistribute it and/or
+ * modify it under the terms of the GNU General Public License as
+ * published by the Free Software Foundation, version 2 of the
+ * License.
+ */
+
+#include <linux/module.h>
+#include <linux/version.h>
+#include <linux/nsproxy.h>
+#include <linux/user_namespace.h>
+
+struct user_namespace init_user_ns = {
+ .kref = {
+ .refcount = ATOMIC_INIT(2),
+ },
+ .root_user = &root_user,
+};
+
+EXPORT_SYMBOL_GPL(init_user_ns);
+
+#ifdef CONFIG_USER_NS
+
+int copy_user_ns(int flags, struct task_struct *tsk)
+{
+ struct user_namespace *old_ns = tsk->nsproxy->user_ns;
+ int err = 0;
+
+ if (!old_ns)
+ return 0;
+
+ get_user_ns(old_ns);
+ return err;
+}
+
+void free_user_ns(struct kref *kref)

```

```
+{  
+ struct user_namespace *ns;  
+  
+ ns = container_of(kref, struct user_namespace, kref);  
+ kfree(ns);  
+}  
+  
+ #endif /* CONFIG_USER_NS */  
--  
1.4.1
```

Containers mailing list
Containers@lists.osdl.org
<https://lists.osdl.org/mailman/listinfo/containers>
