Subject: [patch -mm 00/17] new namespaces and related syscalls
Posted by Cedric Le Goater on Tue, 05 Dec 2006 10:27:52 GMT
View Forum Message <> Reply to Message

All,

Here's a patch serie we've been working on for a while. I'd like to
know which could make it to -mm and which need refinements. Depending
on the feedback, i'll send them on lkml@.

It feels like that the empty net namespace framework could make it
there already.

Patch menu :

* start with user and net namespace. no real feature and no unshare.
  mostly solves conflicts in nsproxy which are becoming painful.
  tasty and harmless

[patch 01/17] net namespace: empty framework
[patch 02/17] user namespace: add the framework

* follow some cleanups and code movement preparing the next drop.
  less tasty but still harmless.

[patch 03/17] namespace : export unshare of namespace and fs_struct
[patch 04/17] nsproxy: externalizes exit_task_namespaces
[patch 05/17] ipc namespace : externalizes unshare_ipcs
[patch 06/17] nsproxy: add extern to nsproxy functions
[patch 07/17] nsproxy: make put_nsproxy an extern

* nsproxy syscalls.
  real meat.

[patch 08/17] nsproxy: add hashtable
[patch 09/17] nsproxy: add namespace flags
[patch 10/17] nsproxy: add unshare_ns and bind_ns syscalls

* use the user namespace for real.
  wine and cheese

[patch 11/17] user namespace: add user_namespace ptr to vfsmount
[patch 12/17] user namespace: hook permission
[patch 13/17] user namespace: implement shared mounts
[patch 14/17] user namespace: maintain user ns for priv_userns mounts to vfsmount

* finish with the stuff that make a real meal : unsharing namespaces

some will surely find these less in sync with their taste, like
dessert without sugar :)

[patch 15/17] pid namespace: add unshare
[patch 16/17] net namespace: add unshare
[patch 17/17] user namespace: add unshare

Cheers,

C.

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

Subject: [patch -mm 01/17] net namespace: empty framework
Posted by Cedric Le Goater on Tue, 05 Dec 2006 10:27:53 GMT
View Forum Message <> Reply to Message

From: Cedric Le Goater <clg@fr.ibm.com>

This patch adds an empty net namespace framework with an API
compatible with the other available namespaces.

It doesn't provide any feature by itself but prepares the ground for
work on L2 and L3 network isolation. It also solves patch conflict
issues in nsproxy, which are painful and boring to resolve.

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

---
include/linux/init_task.h | 2 +
include/linux/net_namespace.h | 49 ++
include/linux/nsproxy.h | 2 +
kernel/nsproxy.c | 12 ++++++++++
net/Kconfig | 8 ++++++
net/core/Makefile | 2 -
net/core/net_namespace.c | 41 ++++++++++++++++++++++++++++++++++
7 files changed, 115 insertions(+), 1 deletion(-)

Index: 2.6.19-rc6-mm2/include/linux/init_task.h
===
--- 2.6.19-rc6-mm2.orig/include/linux/init_task.h
+++ 2.6.19-rc6-mm2/include/linux/init_task.h
@@ -8,6 +8,7 @@
#include <linux/lockdep.h>
#include <linux/ipc.h>

```
 #include <linux/pid_namespace.h>
+#include <linux/net_namespace.h>

 #define INIT_FDTABLE \
 {       \
@@ -78,6 +79,7 @@ extern struct nsproxy init_nsproxy;
  .id  = 0,      \
  .uts_ns  = &init_uts_ns,     \
  .mnt_ns  = NULL,      \
+ INIT_NET_NS(net_ns)      \
  INIT_IPC_NS(ipc_ns)      \
 }

Index: 2.6.19-rc6-mm2/include/linux/net_namespace.h
===================================================================
--- /dev/null
+++ 2.6.19-rc6-mm2/include/linux/net_namespace.h
@@ -0,0 +1,49 @@
+#ifndef _LINUX_NET_NAMESPACE_H
+#define _LINUX_NET_NAMESPACE_H
+
+#include <linux/kref.h>
+#include <linux/nsproxy.h>
+
+struct net_namespace {
+ struct kref kref;
+};
+
+extern struct net_namespace init_net_ns;
+
+#ifdef CONFIG_NET_NS
+
+#define INIT_NET_NS(net_ns) .net_ns = &init_net_ns,
+
+static inline void get_net_ns(struct net_namespace *ns)
+{
+ kref_get(&ns->kref);
+}
+
+extern int copy_net_ns(int flags, struct task_struct *tsk);
+
+extern void free_net_ns(struct kref *kref);
+
+static inline void put_net_ns(struct net_namespace *ns)
+{
+ kref_put(&ns->kref, free_net_ns);
+}
+
```

```
+#else
+
+#define INIT_NET_NS(net_ns)
+
+static inline void get_net_ns(struct net_namespace *ns)
+{
+}
+
+static inline int copy_net_ns(int flags, struct task_struct *tsk)
+{
+ return 0;
+}
+
+static inline void put_net_ns(struct net_namespace *ns)
+{
+}
+#endif
+
+#endif /* _LINUX_NET_NAMESPACE_H */
Index: 2.6.19-rc6-mm2/include/linux/nsproxy.h
===================================================================
--- 2.6.19-rc6-mm2.orig/include/linux/nsproxy.h
+++ 2.6.19-rc6-mm2/include/linux/nsproxy.h
@@ -8,6 +8,7 @@ struct mnt_namespace;
 struct uts_namespace;
 struct ipc_namespace;
 struct pid_namespace;
+struct net_namespace;

 /*
  * A structure to contain pointers to all per-process
@@ -29,6 +30,7 @@ struct nsproxy {
  struct ipc_namespace *ipc_ns;
  struct mnt_namespace *mnt_ns;
  struct pid_namespace *pid_ns;
+ struct net_namespace *net_ns;
 };
 extern struct nsproxy init_nsproxy;

Index: 2.6.19-rc6-mm2/kernel/nsproxy.c
===================================================================
--- 2.6.19-rc6-mm2.orig/kernel/nsproxy.c
+++ 2.6.19-rc6-mm2/kernel/nsproxy.c
@@ -20,6 +20,7 @@
 #include <linux/mnt_namespace.h>
 #include <linux/utsname.h>
 #include <linux/pid_namespace.h>
+#include <linux/net_namespace.h>
```

```
 struct nsproxy init_nsproxy = INIT_NSPROXY(init_nsproxy);

@@ -71,6 +72,8 @@ struct nsproxy *dup_namespaces(struct ns
    get_ipc_ns(ns->ipc_ns);
   if (ns->pid_ns)
    get_pid_ns(ns->pid_ns);
+  if (ns->net_ns)
+   get_net_ns(ns->net_ns);
  }

  return ns;
@@ -118,10 +121,17 @@ int copy_namespaces(int flags, struct ta
  if (err)
   goto out_pid;

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

+out_net:
+ 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);
@@ -147,5 +157,7 @@ void free_nsproxy(struct nsproxy *ns)
   put_ipc_ns(ns->ipc_ns);
  if (ns->pid_ns)
   put_pid_ns(ns->pid_ns);
+ if (ns->net_ns)
+  put_net_ns(ns->net_ns);
  kfree(ns);
 }
Index: 2.6.19-rc6-mm2/net/core/Makefile
===================================================================
--- 2.6.19-rc6-mm2.orig/net/core/Makefile
+++ 2.6.19-rc6-mm2/net/core/Makefile
@@ -3,7 +3,7 @@
 #

 obj-y := sock.o request_sock.o skbuff.o iovec.o datagram.o stream.o scm.o \
-  gen_stats.o gen_estimator.o
+  gen_stats.o gen_estimator.o net_namespace.o
```

```
 obj-$(CONFIG_SYSCTL) += sysctl_net_core.o

Index: 2.6.19-rc6-mm2/net/core/net_namespace.c
===================================================================
--- /dev/null
+++ 2.6.19-rc6-mm2/net/core/net_namespace.c
@@ -0,0 +1,41 @@
+/*
+ *  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/net_namespace.h>
+
+struct net_namespace init_net_ns = {
+ .kref = {
+  .refcount = ATOMIC_INIT(2),
+ },
+};
+
+#ifdef CONFIG_NET_NS
+
+int copy_net_ns(int flags, struct task_struct *tsk)
+{
+ struct net_namespace *old_ns = tsk->nsproxy->net_ns;
+ int err = 0;
+
+ if (!old_ns)
+  return 0;
+
+ get_net_ns(old_ns);
+ return err;
+}
+
+void free_net_ns(struct kref *kref)
+{
+ struct net_namespace *ns;
+
+ ns = container_of(kref, struct net_namespace, kref);
+ kfree(ns);
+}
+
```

```
+#endif /* CONFIG_NET_NS */
Index: 2.6.19-rc6-mm2/net/Kconfig
===================================================================
--- 2.6.19-rc6-mm2.orig/net/Kconfig
+++ 2.6.19-rc6-mm2/net/Kconfig
@@ -67,6 +67,14 @@ source "net/netlabel/Kconfig"

 endif # if INET

+config NET_NS
+ bool "Network Namespaces"
+ help
+   This option enables multiple independent network namespaces,
+   each having own network devices, IP addresses, routes, and so on.
+   If unsure, answer N.
+
+
 config NETWORK_SECMARK
  bool "Security Marking"
  help

--
```

Subject: [patch -mm 02/17] user namespace: add the framework
Posted by Cedric Le Goater on Tue, 05 Dec 2006 10:27:54 GMT
View Forum Message <> Reply to Message

From: Cedric Le Goater <clg@fr.ibm.com>

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 #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>

```
 include/linux/init_task.h      |    2 +
 include/linux/nsproxy.h        |    2 +
 include/linux/sched.h          |    3 +-
 include/linux/user_namespace.h |   49 ++++++++++++++++++++++++++++++++++++++++
 init/Kconfig                   |    8 ++++++
 kernel/Makefile                |    2 -
 kernel/fork.c                  |    2 -
 kernel/nsproxy.c               |   11 +++++++++
 kernel/sys.c                   |    5 ++--
 kernel/user.c                  |   18 +++++++--------
 kernel/user_namespace.c        |   44 ++++++++++++++++++++++++++++++++++++++
 11 files changed, 132 insertions(+), 14 deletions(-)
```

Index: 2.6.19-rc6-mm2/kernel/user.c
```
===================================================================
--- 2.6.19-rc6-mm2.orig/kernel/user.c
+++ 2.6.19-rc6-mm2/kernel/user.c
@@ -14,20 +14,19 @@
 #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 kmem_cache_t *uid_cachep;
-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;
Index: 2.6.19-rc6-mm2/include/linux/nsproxy.h
===================================================================
--- 2.6.19-rc6-mm2.orig/include/linux/nsproxy.h
+++ 2.6.19-rc6-mm2/include/linux/nsproxy.h
@@ -9,6 +9,7 @@ struct uts_namespace;
```

```
 struct ipc_namespace;
 struct pid_namespace;
 struct net_namespace;
+struct user_namespace;

 /*
  * A structure to contain pointers to all per-process
@@ -31,6 +32,7 @@ struct nsproxy {
  struct mnt_namespace *mnt_ns;
  struct pid_namespace *pid_ns;
  struct net_namespace *net_ns;
+ struct user_namespace *user_ns;
 };
 extern struct nsproxy init_nsproxy;

Index: 2.6.19-rc6-mm2/include/linux/sched.h
===================================================================
--- 2.6.19-rc6-mm2.orig/include/linux/sched.h
+++ 2.6.19-rc6-mm2/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
@@ -1286,7 +1287,7 @@ extern struct task_struct *find_task_by_
 extern void __set_special_pids(pid_t session, pid_t pgrp);

 /* per-UID process charging. */
-extern struct user_struct * alloc_uid(uid_t);
+extern struct user_struct * alloc_uid(struct user_namespace *, uid_t);
 static inline struct user_struct *get_uid(struct user_struct *u)
 {
  atomic_inc(&u->__count);
Index: 2.6.19-rc6-mm2/include/linux/init_task.h
===================================================================
--- 2.6.19-rc6-mm2.orig/include/linux/init_task.h
+++ 2.6.19-rc6-mm2/include/linux/init_task.h
@@ -9,6 +9,7 @@
 #include <linux/ipc.h>
 #include <linux/pid_namespace.h>
 #include <linux/net_namespace.h>
+#include <linux/user_namespace.h>

 #define INIT_FDTABLE \
 {       \
```

```
@@ -81,6 +82,7 @@ extern struct nsproxy init_nsproxy;
  .mnt_ns  = NULL,      \
  INIT_NET_NS(net_ns)      \
  INIT_IPC_NS(ipc_ns)      \
+ .user_ns = &init_user_ns,    \
 }

 #define INIT_SIGHAND(sighand) {      \
Index: 2.6.19-rc6-mm2/kernel/sys.c
===================================================================
--- 2.6.19-rc6-mm2.orig/kernel/sys.c
+++ 2.6.19-rc6-mm2/kernel/sys.c
@@ -33,6 +33,7 @@
 #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>
@@ -1062,13 +1063,13 @@ 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;
  }
Index: 2.6.19-rc6-mm2/kernel/fork.c
===================================================================
--- 2.6.19-rc6-mm2.orig/kernel/fork.c
+++ 2.6.19-rc6-mm2/kernel/fork.c
@@ -996,7 +996,7 @@ static struct task_struct *copy_process(
  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;
  }
```

```
Index: 2.6.19-rc6-mm2/kernel/nsproxy.c
===================================================================
--- 2.6.19-rc6-mm2.orig/kernel/nsproxy.c
+++ 2.6.19-rc6-mm2/kernel/nsproxy.c
@@ -74,6 +74,8 @@ struct nsproxy *dup_namespaces(struct ns
    get_pid_ns(ns->pid_ns);
   if (ns->net_ns)
    get_net_ns(ns->net_ns);
+  if (ns->user_ns)
+   get_user_ns(ns->user_ns);
  }

  return ns;
@@ -125,10 +127,17 @@ int copy_namespaces(int flags, struct ta
  if (err)
   goto out_net;

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

+out_user:
+ if (new_ns->net_ns)
+  put_net_ns(new_ns->net_ns);
 out_net:
  if (new_ns->pid_ns)
   put_pid_ns(new_ns->pid_ns);
@@ -159,5 +168,7 @@ void free_nsproxy(struct nsproxy *ns)
   put_pid_ns(ns->pid_ns);
  if (ns->net_ns)
   put_net_ns(ns->net_ns);
+ if (ns->user_ns)
+  put_user_ns(ns->user_ns);
  kfree(ns);
 }
Index: 2.6.19-rc6-mm2/init/Kconfig
===================================================================
--- 2.6.19-rc6-mm2.orig/init/Kconfig
+++ 2.6.19-rc6-mm2/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
Index: 2.6.19-rc6-mm2/include/linux/user_namespace.h
===================================================================
--- /dev/null
+++ 2.6.19-rc6-mm2/include/linux/user_namespace.h
@@ -0,0 +1,49 @@
+#ifndef _LINUX_USER_NAMESPACE_H
+#define _LINUX_USER_NAMESPACE_H
+
+#include <linux/kref.h>
+#include <linux/nsproxy.h>
+
+#define UIDHASH_BITS (CONFIG_BASE_SMALL ? 3 : 8)
+#define UIDHASH_SZ (1 << UIDHASH_BITS)
+
+struct user_namespace {
+ struct kref  kref;
+ struct list_head uidhash_table[UIDHASH_SZ];
+ struct user_struct *root_user;
+};
+
+extern struct user_namespace init_user_ns;
+
+#ifdef CONFIG_USER_NS
+
+static inline void get_user_ns(struct user_namespace *ns)
+{
+ kref_get(&ns->kref);
+}
+
+extern int copy_user_ns(int flags, struct task_struct *tsk);
+extern void free_user_ns(struct kref *kref);
+
+static inline void put_user_ns(struct user_namespace *ns)
+{
+ kref_put(&ns->kref, free_user_ns);
+}
+
```

```
+#else
+
+static inline void get_user_ns(struct user_namespace *ns)
+{
+}
+
+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 */
Index: 2.6.19-rc6-mm2/kernel/user_namespace.c
===================================================================
--- /dev/null
+++ 2.6.19-rc6-mm2/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 */
Index: 2.6.19-rc6-mm2/kernel/Makefile
===================================================================
--- 2.6.19-rc6-mm2.orig/kernel/Makefile
+++ 2.6.19-rc6-mm2/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 \

--
```

Subject: [patch -mm 03/17] namespace : export unshare of namespace and fs_struct

Posted by Cedric Le Goater on Tue, 05 Dec 2006 10:27:55 GMT

View Forum Message <> Reply to Message

From: Cedric Le Goater <clg@fr.ibm.com>

This is needed by the unshare_ns syscall.

```
Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>
---
 include/linux/mnt_namespace.h |    4 ++++
 kernel/fork.c                 |    6 +++---
 2 files changed, 7 insertions(+), 3 deletions(-)

Index: 2.6.19-rc6-mm2/include/linux/mnt_namespace.h
===================================================================
--- 2.6.19-rc6-mm2.orig/include/linux/mnt_namespace.h
+++ 2.6.19-rc6-mm2/include/linux/mnt_namespace.h
@@ -14,6 +14,10 @@ struct mnt_namespace {
  int event;
 };

+extern int unshare_fs(unsigned long unshare_flags, struct fs_struct **new_fsp);
+extern int unshare_mnt_ns(unsigned long unshare_flags,
+      struct mnt_namespace **new_nsp,
+      struct fs_struct *new_fs);
 extern int copy_mnt_ns(int, struct task_struct *);
 extern void __put_mnt_ns(struct mnt_namespace *ns);
 extern struct mnt_namespace *dup_mnt_ns(struct task_struct *,
Index: 2.6.19-rc6-mm2/kernel/fork.c
===================================================================
--- 2.6.19-rc6-mm2.orig/kernel/fork.c
+++ 2.6.19-rc6-mm2/kernel/fork.c
@@ -1498,7 +1498,7 @@ static int unshare_thread(unsigned long
 /*
  * Unshare the filesystem structure if it is being shared
  */
-static int unshare_fs(unsigned long unshare_flags, struct fs_struct **new_fsp)
+extern int unshare_fs(unsigned long unshare_flags, struct fs_struct **new_fsp)
 {
  struct fs_struct *fs = current->fs;

@@ -1515,7 +1515,7 @@ static int unshare_fs(unsigned long unsh
 /*
  * Unshare the mnt_namespace structure if it is being shared
  */
-static int unshare_mnt_namespace(unsigned long unshare_flags,
+extern int unshare_mnt_ns(unsigned long unshare_flags,
    struct mnt_namespace **new_nsp, struct fs_struct *new_fs)
 {
  struct mnt_namespace *ns = current->nsproxy->mnt_ns;
@@ -1634,7 +1634,7 @@ asmlinkage long sys_unshare(unsigned lon
   goto bad_unshare_out;
  if ((err = unshare_fs(unshare_flags, &new_fs)))
   goto bad_unshare_cleanup_thread;
- if ((err = unshare_mnt_namespace(unshare_flags, &new_ns, new_fs)))
```

```
+ if ((err = unshare_mnt_ns(unshare_flags, &new_ns, new_fs)))
   goto bad_unshare_cleanup_fs;
  if ((err = unshare_sighand(unshare_flags, &new_sigh)))
   goto bad_unshare_cleanup_ns;

--
```

_____

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

Subject: [patch -mm 04/17] nsproxy: externalizes exit_task_namespaces
Posted by Cedric Le Goater on Tue, 05 Dec 2006 10:27:56 GMT
View Forum Message <> Reply to Message

From: Cedric Le Goater <clg@fr.ibm.com>

this is required to remove a header dependency in sched.h which breaks
next patches.

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

```
---
 include/linux/nsproxy.h |   15 ++++----------
 kernel/nsproxy.c        |   11 ++++++++++
 2 files changed, 15 insertions(+), 11 deletions(-)

Index: 2.6.19-rc6-mm2/include/linux/nsproxy.h
===================================================================
--- 2.6.19-rc6-mm2.orig/include/linux/nsproxy.h
+++ 2.6.19-rc6-mm2/include/linux/nsproxy.h
@@ -2,7 +2,8 @@
 #define _LINUX_NSPROXY_H

 #include <linux/spinlock.h>
-#include <linux/sched.h>
+
+struct task_struct;

 struct mnt_namespace;
 struct uts_namespace;
@@ -48,14 +49,6 @@ static inline void put_nsproxy(struct ns
  }
 }

-static inline void exit_task_namespaces(struct task_struct *p)
-{
```

```
- struct nsproxy *ns = p->nsproxy;
- if (ns) {
-  task_lock(p);
-  p->nsproxy = NULL;
-  task_unlock(p);
-  put_nsproxy(ns);
- }
-}
+extern void exit_task_namespaces(struct task_struct *p);
+
 #endif
Index: 2.6.19-rc6-mm2/kernel/nsproxy.c
===================================================================
--- 2.6.19-rc6-mm2.orig/kernel/nsproxy.c
+++ 2.6.19-rc6-mm2/kernel/nsproxy.c
@@ -37,6 +37,17 @@ void get_task_namespaces(struct task_str
  }
 }

+void exit_task_namespaces(struct task_struct *p)
+{
+ struct nsproxy *ns = p->nsproxy;
+ if (ns) {
+  task_lock(p);
+  p->nsproxy = NULL;
+  task_unlock(p);
+  put_nsproxy(ns);
+ }
+}
+
 /*
  * creates a copy of "orig" with refcount 1.
  * This does not grab references to the contained namespaces,

--
```

Subject: [patch -mm 05/17] ipc namespace : externalizes unshare_ipcs
Posted by Cedric Le Goater on Tue, 05 Dec 2006 10:27:57 GMT
View Forum Message <> Reply to Message

From: Cedric Le Goater <clg@fr.ibm.com>

This patch externalizes unshare_ipcs when CONFIG_IPC_NS is not

set. This is needed by the unshare_ns syscall. Unfortunately, it could
not be handled with a dummy static inline in ipc.h because of
header dependencies on sched.h.

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

---
 include/linux/ipc.h | 2 +-
 kernel/fork.c | 2 +-
 2 files changed, 2 insertions(+), 2 deletions(-)

Index: 2.6.19-rc6-mm2/include/linux/ipc.h
===
--- 2.6.19-rc6-mm2.orig/include/linux/ipc.h
+++ 2.6.19-rc6-mm2/include/linux/ipc.h
@@ -99,7 +99,6 @@ extern struct ipc_namespace init_ipc_ns;
 #ifdef CONFIG_IPC_NS
 extern void free_ipc_ns(struct kref *kref);
 extern int copy_ipcs(unsigned long flags, struct task_struct *tsk);
-extern int unshare_ipcs(unsigned long flags, struct ipc_namespace **ns);
 #else
 static inline int copy_ipcs(unsigned long flags, struct task_struct *tsk)
 {
@@ -107,6 +106,7 @@ static inline int copy_ipcs(unsigned lon
 }
 #endif

+extern int unshare_ipcs(unsigned long flags, struct ipc_namespace **ns);
 static inline struct ipc_namespace *get_ipc_ns(struct ipc_namespace *ns)
 {
 #ifdef CONFIG_IPC_NS
Index: 2.6.19-rc6-mm2/kernel/fork.c
===
--- 2.6.19-rc6-mm2.orig/kernel/fork.c
+++ 2.6.19-rc6-mm2/kernel/fork.c
@@ -1591,7 +1591,7 @@ static int unshare_semundo(unsigned long
 }

 #ifndef CONFIG_IPC_NS
-static inline int unshare_ipcs(unsigned long flags, struct ipc_namespace **ns)
+extern int unshare_ipcs(unsigned long flags, struct ipc_namespace **ns)
 {
  if (flags & CLONE_NEWIPC)
   return -EINVAL;

--

_____
Containers mailing list

Subject: [patch -mm 06/17] nsproxy: add extern to nsproxy functions
Posted by Cedric Le Goater on Tue, 05 Dec 2006 10:27:58 GMT
View Forum Message <> Reply to Message

From: Cedric Le Goater <clg@fr.ibm.com>

Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>
---
include/linux/nsproxy.h | 8 ++++----
1 file changed, 4 insertions(+), 4 deletions(-)

Index: 2.6.19-rc6-mm2/include/linux/nsproxy.h
===
--- 2.6.19-rc6-mm2.orig/include/linux/nsproxy.h
+++ 2.6.19-rc6-mm2/include/linux/nsproxy.h
@@ -37,10 +37,10 @@ struct nsproxy {
};
extern struct nsproxy init_nsproxy;

-struct nsproxy *dup_namespaces(struct nsproxy *orig);
-int copy_namespaces(int flags, struct task_struct *tsk);
-void get_task_namespaces(struct task_struct *tsk);
-void free_nsproxy(struct nsproxy *ns);
+extern struct nsproxy *dup_namespaces(struct nsproxy *orig);
+extern int copy_namespaces(int flags, struct task_struct *tsk);
+extern void get_task_namespaces(struct task_struct *tsk);
+extern void free_nsproxy(struct nsproxy *ns);

static inline void put_nsproxy(struct nsproxy *ns)
{

--

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

Subject: [patch -mm 07/17] nsproxy: make put_nsproxy an extern
Posted by Cedric Le Goater on Tue, 05 Dec 2006 10:27:59 GMT
View Forum Message <> Reply to Message

From: Cedric Le Goater <clg@fr.ibm.com>

This routine will require modifications in the next patches and it
won't be possible to keep it as an inline anymore.

Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>
---
 include/linux/nsproxy.h | 9 +--------
 kernel/nsproxy.c | 9 ++++++++-
 2 files changed, 9 insertions(+), 9 deletions(-)

Index: 2.6.19-rc6-mm2/include/linux/nsproxy.h
===
--- 2.6.19-rc6-mm2.orig/include/linux/nsproxy.h
+++ 2.6.19-rc6-mm2/include/linux/nsproxy.h
@@ -40,14 +40,7 @@ extern struct nsproxy init_nsproxy;
 extern struct nsproxy *dup_namespaces(struct nsproxy *orig);
 extern int copy_namespaces(int flags, struct task_struct *tsk);
 extern void get_task_namespaces(struct task_struct *tsk);
-extern void free_nsproxy(struct nsproxy *ns);
-
-static inline void put_nsproxy(struct nsproxy *ns)
-{
- if (atomic_dec_and_test(&ns->count)) {
- free_nsproxy(ns);
- }
-}
+extern void put_nsproxy(struct nsproxy *ns);

 extern void exit_task_namespaces(struct task_struct *p);

Index: 2.6.19-rc6-mm2/kernel/nsproxy.c
===
--- 2.6.19-rc6-mm2.orig/kernel/nsproxy.c
+++ 2.6.19-rc6-mm2/kernel/nsproxy.c
@@ -167,7 +167,7 @@ out_ns:
  goto out;
 }

-void free_nsproxy(struct nsproxy *ns)
+static void free_nsproxy(struct nsproxy *ns)
 {
  if (ns->mnt_ns)
   put_mnt_ns(ns->mnt_ns);
@@ -183,3 +183,10 @@ void free_nsproxy(struct nsproxy *ns)
  kfree(ns);
 }
+

```
+void put_nsproxy(struct nsproxy *ns)
+{
+ if (atomic_dec_and_test(&ns->count)) {
+  free_nsproxy(ns);
+ }
+}

--
```

_____

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

Subject: [patch -mm 08/17] nsproxy: add hashtable
Posted by Cedric Le Goater on Tue, 05 Dec 2006 10:28:00 GMT
View Forum Message <> Reply to Message

From: Cedric Le Goater <clg@fr.ibm.com>

This patch adds a hashtable of nsproxy using the nsproxy as a key.
init_nsproxy is hashed at init with key 0. This is considered to be
the 'host' nsproxy.

Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>
---
 include/linux/nsproxy.h | 3 +++
 kernel/nsproxy.c | 43 +++
 2 files changed, 46 insertions(+)

Index: 2.6.19-rc6-mm2/include/linux/nsproxy.h
===
--- 2.6.19-rc6-mm2.orig/include/linux/nsproxy.h
+++ 2.6.19-rc6-mm2/include/linux/nsproxy.h
@@ -2,6 +2,7 @@
 #define _LINUX_NSPROXY_H

 #include <linux/spinlock.h>
+#include <linux/list.h>

 struct task_struct;

@@ -34,6 +35,8 @@ struct nsproxy {
  struct pid_namespace *pid_ns;
  struct net_namespace *net_ns;
  struct user_namespace *user_ns;
+
+ struct hlist_node ns_hash_node;

```
 };
 extern struct nsproxy init_nsproxy;

Index: 2.6.19-rc6-mm2/kernel/nsproxy.c
===================================================================
--- 2.6.19-rc6-mm2.orig/kernel/nsproxy.c
+++ 2.6.19-rc6-mm2/kernel/nsproxy.c
@@ -22,6 +22,15 @@
 #include <linux/pid_namespace.h>
 #include <linux/net_namespace.h>

+#define NS_HASH_BITS   3 /* this might need some configuration */
+#define NS_HASH_SIZE  (1 << NS_HASH_BITS)
+#define NS_HASH_MASK  (NS_HASH_SIZE - 1)
+#define ns_hashfn(id)  (((id >> NS_HASH_BITS) + id) & NS_HASH_MASK)
+#define ns_hash_head(id) &ns_hash[ns_hashfn(id)]
+
+static struct hlist_head ns_hash[NS_HASH_SIZE];
+static DEFINE_SPINLOCK(ns_hash_lock);
+
 struct nsproxy init_nsproxy = INIT_NSPROXY(init_nsproxy);

 static inline void get_nsproxy(struct nsproxy *ns)
@@ -190,3 +199,37 @@ void put_nsproxy(struct nsproxy *ns)
   free_nsproxy(ns);
  }
 }
+
+/*
+ * This routine must be called with the ns_hash spin_locked
+ */
+static inline struct nsproxy *ns_hash_find(int id)
+{
+ struct hlist_node *elem;
+ struct nsproxy *ns;
+
+ hlist_for_each_entry(ns, elem, ns_hash_head(id), ns_hash_node) {
+  if (ns->id == id) {
+   get_nsproxy(ns);
+   return ns;
+  }
+ }
+
+ return NULL;
+}
+
+static int __init nshash_init(void)
+{
```

```
+ int i;
+
+ for (i = 0; i < NS_HASH_SIZE; ++i)
+  INIT_HLIST_HEAD(&ns_hash[i]);
+
+ spin_lock_irq(&ns_hash_lock);
+ hlist_add_head(&init_nsproxy.ns_hash_node, ns_hash_head(0));
+ spin_unlock_irq(&ns_hash_lock);
+
+ return 0;
+}
+
+module_init(nshash_init);

--
```

Subject: [patch -mm 09/17] nsproxy: add namespace flags
Posted by Cedric Le Goater on Tue, 05 Dec 2006 10:28:01 GMT
View Forum Message <> Reply to Message

From: Cedric Le Goater <clg@fr.ibm.com>

Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>
---
 include/linux/nsproxy.h | 11 ++++++++++
 1 file changed, 11 insertions(+)

Index: 2.6.19-rc6-mm2/include/linux/nsproxy.h
===
--- 2.6.19-rc6-mm2.orig/include/linux/nsproxy.h
+++ 2.6.19-rc6-mm2/include/linux/nsproxy.h
@@ -14,6 +14,17 @@ struct net_namespace;
 struct user_namespace;

 /*
+ * namespaces flags
+ */
+#define NS_MNT 0x00000001
+#define NS_UTS 0x00000002
+#define NS_IPC 0x00000004
+#define NS_PID 0x00000008
+#define NS_NET 0x00000010
+#define NS_USER 0x00000020

```
+#define NS_ALL  (NS_MNT|NS_UTS|NS_IPC|NS_PID|NS_NET|NS_USER)
+
+/*
  * A structure to contain pointers to all per-process
  * namespaces - fs (mount), uts, network, sysvipc, etc.
  *

--
```

_____

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

Subject: [patch -mm 10/17] nsproxy: add unshare_ns and bind_ns syscalls
Posted by Cedric Le Goater on Tue, 05 Dec 2006 10:28:02 GMT
View Forum Message <> Reply to Message

From: Cedric Le Goater <clg@fr.ibm.com>

The following patch defines 2 new syscalls specific to nsproxy and
namespaces :

* unshare_ns :

 enables a process to unshare one or more namespaces. this
       duplicates the unshare syscall for the moment but we
 expect to diverge when the number of namespaces increases

* bind_ns :

 allows a process to bind
 1 - its nsproxy to some identifier
 2 - to another nsproxy using an identifier or -pid

Here's a sample user space program to use them.

```
#include <stdio.h>
#include <stdlib.h>
#include <sched.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>
#include <libgen.h>

#include <linux/unistd.h>

#if __i386__
```

```
#    define __NR_unshare_ns 324
#    define __NR_bind_ns  325
#elif __ia64__
#    define __NR_unshare_ns 1303
#    define __NR_bind_ns 1304
#elif __powerpc__
#    define __NR_unshare_ns 303
#    define __NR_bind_ns 304
#elif __s390x__
#    define __NR_unshare_ns 313
#    define __NR_bind_ns 314
#elif __x86_64__
#    define __NR_unshare_ns 284
#    define __NR_bind_ns 285
#else
#    error "Architecture not supported"
#endif

static inline _syscall1 (int,  unshare_ns, unsigned long, flags)
static inline _syscall2 (int,  bind_ns, int, id, unsigned long, flags)

#define NS_MNT  0x00000001
#define NS_UTS  0x00000002
#define NS_IPC  0x00000004
#define NS_PID  0x00000008
#define NS_NET  0x00000010
#define NS_USER  0x00000020

static const char* procname;

static void usage(const char *name)
{
 printf("usage: %s [-h] [-I id] [-muiUnp] [command [arg ...]]\n", name);
 printf("\n");
 printf("  -h  this message\n");
 printf("\n");
 printf("  -I <id> bind process to nsproxy <id>\n");
 printf("  -m  mount namespace\n");
 printf("  -u  utsname namespace\n");
 printf("  -i  ipc namespace\n");
 printf("  -U  user namespace\n");
 printf("  -n  net namespace\n");
 printf("  -p  pid namespace\n");
 printf("\n");
 printf("(C) Copyright IBM Corp. 2006\n");
 printf("\n");
 exit(1);
}
```

```
int main(int argc, char *argv[])
{
 int c;
 unsigned long flags = 0;
 int id = -1;

 procname = basename(argv[0]);

 while ((c = getopt(argc, argv, "+muiUnphI:")) != EOF) {
  switch (c) {
  case 'I': if (optarg)
    id = atoi(optarg); break;

  case 'm': flags |= NS_MNT;  break;
  case 'u': flags |= NS_UTS;  break;
  case 'i': flags |= NS_IPC;  break;
  case 'U': flags |= NS_USER; break;
  case 'n': flags |= NS_NET;  break;
  case 'p': flags |= NS_PID;  break;
  case 'h':
  default:
   usage(procname);
  }
 };

 argv = &argv[optind];
 argc = argc - optind;

 if (!strcmp(procname, "unsharens")) {
  if (unshare_ns(flags) == -1) {
   perror("unshare_ns");
   return 1;
  }
 }

 if (bind_ns(id, flags) == -1) {
  perror("bind_ns");
  return 1;
 }

 if (argc) {
  execve(argv[0], argv, __environ);
  perror("execve");
  return 1;
 }

 return 0;
```

```
}

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

---
 arch/i386/kernel/syscall_table.S  |    2
 arch/ia64/kernel/entry.S          |    2
 arch/s390/kernel/compat_wrapper.S |   11 +
 arch/s390/kernel/syscalls.S       |    2
 arch/x86_64/ia32/ia32entry.S      |    2
 include/asm-i386/unistd.h         |    4
 include/asm-ia64/unistd.h         |    4
 include/asm-powerpc/systbl.h      |    2
 include/asm-powerpc/unistd.h      |    4
 include/asm-s390/unistd.h         |    4
 include/asm-x86_64/unistd.h       |    6
 include/linux/syscalls.h          |    3
 kernel/nsproxy.c                  |  325 ++++++++++++++++++++++++++++++++++++++-
 kernel/sys_ni.c                   |    4
 14 files changed, 366 insertions(+), 9 deletions(-)

Index: 2.6.19-rc6-mm2/include/linux/syscalls.h
===================================================================
--- 2.6.19-rc6-mm2.orig/include/linux/syscalls.h
+++ 2.6.19-rc6-mm2/include/linux/syscalls.h
@@ -605,6 +605,9 @@ asmlinkage long sys_set_robust_list(stru
        size_t len);
 asmlinkage long sys_getcpu(unsigned __user *cpu, unsigned __user *node, struct getcpu_cache
__user *cache);

+asmlinkage long sys_unshare_ns(unsigned long unshare_flags);
+asmlinkage long sys_bind_ns(int id, unsigned long unshare_flags);
+
 int kernel_execve(const char *filename, char *const argv[], char *const envp[]);

 asmlinkage long sys_kevent_get_events(int ctl_fd, unsigned int min, unsigned int max,
Index: 2.6.19-rc6-mm2/kernel/nsproxy.c
===================================================================
--- 2.6.19-rc6-mm2.orig/kernel/nsproxy.c
+++ 2.6.19-rc6-mm2/kernel/nsproxy.c
@@ -22,7 +22,11 @@
 #include <linux/pid_namespace.h>
 #include <linux/net_namespace.h>

-#define NS_HASH_BITS   3 /* this might need some configuration */
+/*
+ * nsproxies are stored in a hash but a rbtree might be more
+ * appropriate.
```

```
+ */
+#define NS_HASH_BITS   3
 #define NS_HASH_SIZE  (1 << NS_HASH_BITS)
 #define NS_HASH_MASK  (NS_HASH_SIZE - 1)
 #define ns_hashfn(id)  (((id >> NS_HASH_BITS) + id) & NS_HASH_MASK)
@@ -193,11 +197,26 @@ static void free_nsproxy(struct nsproxy
  kfree(ns);
 }

+/*
+ * put_nsproxy() is similar to free_uid() in kernel/user.c
+ *
+ * the lock can be taken from a tasklet context (task getting freed by
+ * RCU) which requires to be irq safe.
+ */
 void put_nsproxy(struct nsproxy *ns)
 {
- if (atomic_dec_and_test(&ns->count)) {
-  free_nsproxy(ns);
- }
+  unsigned long flags;
+
+  local_irq_save(flags);
+  if (atomic_dec_and_lock(&ns->count, &ns_hash_lock)) {
+   BUG_ON(!ns->id);
+   if (ns->id != -1)
+    hlist_del(&ns->ns_hash_node);
+   spin_unlock_irqrestore(&ns_hash_lock, flags);
+   free_nsproxy(ns);
+  } else {
+   local_irq_restore(flags);
+  }
 }

 /*
@@ -218,6 +237,304 @@ static inline struct nsproxy *ns_hash_fi
  return NULL;
 }

+static int bind_ns(int id, struct nsproxy *ns)
+{
+  struct nsproxy *prev;
+ int ret = 0;
+
+  if (id < 0)
+   return -EINVAL;
+
+  spin_lock_irq(&ns_hash_lock);
```

```
+  prev = ns_hash_find(id);
+  if (!prev) {
+  ns->id = id;
+  hlist_add_head(&ns->ns_hash_node, ns_hash_head(ns->id));
+ }
+ spin_unlock_irq(&ns_hash_lock);
+
+ if (prev) {
+  ret = -EBUSY;
+  put_nsproxy(prev);
+ }
+  return ret;
+}
+
+static int switch_ns(int id, unsigned long flags)
+{
+ int err = 0;
+ struct nsproxy *ns = NULL, *old_ns = NULL, *new_ns = NULL;
+
+ if (flags & ~NS_ALL)
+  return -EINVAL;
+
+ /* Let 0 be a default value ? */
+ if (!flags)
+  flags = NS_ALL;
+
+ if (id < 0) {
+  struct task_struct *p;
+
+  err = -ESRCH;
+  read_lock(&tasklist_lock);
+  p = find_task_by_pid(-id);
+  if (p) {
+   task_lock(p);
+   get_nsproxy(p->nsproxy);
+   ns = p->nsproxy;
+   task_unlock(p);
+  }
+  read_unlock(&tasklist_lock);
+ } else {
+  err = -ENOENT;
+  spin_lock_irq(&ns_hash_lock);
+  ns = ns_hash_find(id);
+  spin_unlock_irq(&ns_hash_lock);
+ }
+
+ if (!ns)
+  goto out;
```

```
+
+ new_ns = ns;
+
+ /*
+  * clone current nsproxy and populate it with the namespaces
+  * chosen by flags.
+  */
+ if (flags != NS_ALL) {
+  new_ns = dup_namespaces(current->nsproxy);
+  if (!new_ns) {
+   err = -ENOMEM;
+   goto out_ns;
+  }
+
+  if (flags & NS_MNT) {
+   put_mnt_ns(new_ns->mnt_ns);
+   get_mnt_ns(ns->mnt_ns);
+   new_ns->mnt_ns = ns->mnt_ns;
+  }
+
+  if (flags & NS_UTS) {
+   put_uts_ns(new_ns->uts_ns);
+   get_uts_ns(ns->uts_ns);
+   new_ns->uts_ns = ns->uts_ns;
+  }
+
+  if (flags & NS_IPC) {
+   put_ipc_ns(new_ns->ipc_ns);
+   new_ns->ipc_ns = get_ipc_ns(ns->ipc_ns);
+  }
+ out_ns:
+  put_nsproxy(ns);
+ }
+
+ task_lock(current);
+ if (new_ns) {
+  old_ns = current->nsproxy;
+  current->nsproxy = new_ns;
+ }
+ task_unlock(current);
+
+ if (old_ns)
+  put_nsproxy(old_ns);
+
+ err = 0;
+out:
+ return err;
+}
```

```
+
+
+/*
+ * bind_ns - bind the nsproxy of a task to an id or bind a task to a
+ *           identified nsproxy
+ *
+ * @id: nsproxy identifier if positive or pid if negative
+ * @flags: identifies the namespaces to bind to
+ *
+ * bind_ns serves 2 purposes.
+ *
+ * The first is to bind the nsproxy of the current task to the
+ * identifier @id. If the identifier is already used, -EBUSY is
+ * returned. If the nsproxy is already bound, -EACCES is returned.
+ * flags is not used in that case.
+ *
+ * The second use is to bind the current task to a subset of
+ * namespaces of an identified nsproxy. If positive, @id is considered
+ * being an nsproxy identifier previously used to bind the nsproxy to
+ * @id. If negative, @id is the pid of a task which is another way to
+ * identify a nsproxy. Switching nsproxy is restricted to tasks within
+ * nsproxy 0, the default nsproxy. If unknown, -ENOENT is returned.
+ * @flags is used to bind the task to the selected namespaces.
+ *
+ * Both uses may return -EINVAL for invalid arguments and -EPERM for
+ * insufficient privileges.
+ *
+ * Returns 0 on success.
+ */
+asmlinkage long sys_bind_ns(int id, unsigned long flags)
+{
+ struct nsproxy *ns = current->nsproxy;
+ int ret = 0;
+
+ /*
+  * ns is being changed by switch_ns(), protect it
+  */
+ get_nsproxy(ns);
+
+ /*
+  * protect ns->id
+  */
+ spin_lock(&ns->nslock);
+ switch (ns->id) {
+ case -1:
+  /*
+   * only an unbound nsproxy can be bound to an id.
+   */
```

```
+  ret = bind_ns(id, ns);
+  break;
+
+ case 0:
+  if (!capable(CAP_SYS_ADMIN)) {
+   ret = -EPERM;
+   goto unlock;
+  }
+
+  /*
+   * only nsproxy 0 can switch nsproxy. if target id is
+   * 0, this is a nop.
+   */
+  if (id)
+   ret = switch_ns(id, flags);
+  break;
+
+ default:
+  /*
+   * current nsproxy is already bound. forbid any
+   * switch.
+   */
+  ret = -EACCES;
+ }
+unlock:
+ spin_unlock(&ns->nslock);
+ put_nsproxy(ns);
+ return ret;
+}
+
+/*
+ * sys_unshare_ns - unshare one or more of namespaces which were
+ *                  originally shared using clone.
+ *
+ * @unshare_ns_flags: identifies the namespaces to unshare
+ *
+ * this is for the moment a pale copy of unshare but we expect to
+ * diverge when the number of namespaces increases. the number of
+ * available clone flags is also very low. This is one way to get
+ * some air.
+ *
+ */
+asmlinkage long sys_unshare_ns(unsigned long unshare_ns_flags)
+{
+  int err = 0;
+  struct nsproxy *new_nsproxy = NULL, *old_nsproxy = NULL;
+  struct fs_struct *fs, *new_fs = NULL;
+  struct mnt_namespace *mnt, *new_mnt = NULL;
```

```
+  struct uts_namespace *uts, *new_uts = NULL;
+  struct ipc_namespace *ipc, *new_ipc = NULL;
+  unsigned long unshare_flags = 0;
+
+  /* Return -EINVAL for all unsupported flags */
+  err = -EINVAL;
+  if (unshare_ns_flags & ~NS_ALL)
+   goto bad_unshare_ns_out;
+
+  /*
+   * compatibility with unshare()/clone() : convert ns flags to
+   * clone flags
+   */
+  if (unshare_ns_flags & NS_MNT)
+   unshare_flags |= CLONE_NEWNS|CLONE_FS;
+  if (unshare_ns_flags & NS_UTS)
+   unshare_flags |= CLONE_NEWUTS;
+  if (unshare_ns_flags & NS_IPC)
+   unshare_flags |= CLONE_NEWIPC;
+
+  if ((err = unshare_fs(unshare_flags, &new_fs)))
+   goto bad_unshare_ns_out;
+  if ((err = unshare_mnt_ns(unshare_flags, &new_mnt, new_fs)))
+   goto bad_unshare_ns_cleanup_fs;
+  if ((err = unshare_utsname(unshare_flags, &new_uts)))
+   goto bad_unshare_ns_cleanup_mnt;
+  if ((err = unshare_ipcs(unshare_flags, &new_ipc)))
+   goto bad_unshare_ns_cleanup_uts;
+
+  if (new_mnt || new_uts || new_ipc) {
+   old_nsproxy = current->nsproxy;
+   new_nsproxy = dup_namespaces(old_nsproxy);
+   if (!new_nsproxy) {
+    err = -ENOMEM;
+    goto bad_unshare_ns_cleanup_ipc;
+   }
+  }
+
+  if (new_fs || new_mnt || new_uts || new_ipc) {
+
+   task_lock(current);
+
+   if (new_nsproxy) {
+    current->nsproxy = new_nsproxy;
+    new_nsproxy = old_nsproxy;
+   }
+
+   if (new_fs) {
```

```
+    fs = current->fs;
+    current->fs = new_fs;
+    new_fs = fs;
+   }
+
+   if (new_mnt) {
+    mnt = current->nsproxy->mnt_ns;
+    current->nsproxy->mnt_ns = new_mnt;
+    new_mnt = mnt;
+   }
+
+   if (new_uts) {
+    uts = current->nsproxy->uts_ns;
+    current->nsproxy->uts_ns = new_uts;
+    new_uts = uts;
+   }
+
+   if (new_ipc) {
+    ipc = current->nsproxy->ipc_ns;
+    current->nsproxy->ipc_ns = new_ipc;
+    new_ipc = ipc;
+   }
+
+   task_unlock(current);
+  }
+
+  if (new_nsproxy)
+   put_nsproxy(new_nsproxy);
+
+bad_unshare_ns_cleanup_ipc:
+  if (new_ipc)
+   put_ipc_ns(new_ipc);
+
+bad_unshare_ns_cleanup_uts:
+  if (new_uts)
+   put_uts_ns(new_uts);
+
+bad_unshare_ns_cleanup_mnt:
+  if (new_mnt)
+   put_mnt_ns(new_mnt);
+
+bad_unshare_ns_cleanup_fs:
+  if (new_fs)
+   put_fs_struct(new_fs);
+
+bad_unshare_ns_out:
+  return err;
+}
```

```
+
 static int __init nshash_init(void)
 {
  int i;
Index: 2.6.19-rc6-mm2/kernel/sys_ni.c
===================================================================
--- 2.6.19-rc6-mm2.orig/kernel/sys_ni.c
+++ 2.6.19-rc6-mm2/kernel/sys_ni.c
@@ -146,3 +146,7 @@ cond_syscall(compat_sys_migrate_pages);
 cond_syscall(sys_bdflush);
 cond_syscall(sys_ioprio_set);
 cond_syscall(sys_ioprio_get);
+
+/* user resources syscalls */
+cond_syscall(sys_unshare_ns);
+cond_syscall(sys_bind_ns);
Index: 2.6.19-rc6-mm2/arch/i386/kernel/syscall_table.S
===================================================================
--- 2.6.19-rc6-mm2.orig/arch/i386/kernel/syscall_table.S
+++ 2.6.19-rc6-mm2/arch/i386/kernel/syscall_table.S
@@ -323,3 +323,5 @@ ENTRY(sys_call_table)
  .long sys_kevent_ctl
  .long sys_kevent_wait
  .long sys_kevent_ring_init
+ .long sys_unshare_ns
+ .long sys_bind_ns  /* 325 */
Index: 2.6.19-rc6-mm2/arch/ia64/kernel/entry.S
===================================================================
--- 2.6.19-rc6-mm2.orig/arch/ia64/kernel/entry.S
+++ 2.6.19-rc6-mm2/arch/ia64/kernel/entry.S
@@ -1610,5 +1610,7 @@ sys_call_table:
  data8 sys_sync_file_range  // 1300
  data8 sys_tee
  data8 sys_vmsplice
+ data8 sys_unshare_ns
+ data8 sys_bind_ns

  .org sys_call_table + 8*NR_syscalls // guard against failures to increase NR_syscalls
Index: 2.6.19-rc6-mm2/arch/s390/kernel/compat_wrapper.S
===================================================================
--- 2.6.19-rc6-mm2.orig/arch/s390/kernel/compat_wrapper.S
+++ 2.6.19-rc6-mm2/arch/s390/kernel/compat_wrapper.S
@@ -1665,3 +1665,14 @@ sys_getcpu_wrapper:
  llgtr %r3,%r3   # unsigned *
  llgtr %r4,%r4   # struct getcpu_cache *
  jg sys_getcpu
+
+ .globl sys_unshare_ns_wrapper
```

```
+sys_unshare_ns_wrapper:
+ llgfr   %r2,%r2              # unsigned long
+ jg      sys_unshare_ns
+
+ .globl sys_bind_ns_wrapper
+sys_bind_ns_wrapper:
+ lgfr    %r2,%r2              # int
+ llgfr   %r3,%r3              # unsigned long
+ jg      sys_bind_ns
Index: 2.6.19-rc6-mm2/arch/s390/kernel/syscalls.S
===================================================================
--- 2.6.19-rc6-mm2.orig/arch/s390/kernel/syscalls.S
+++ 2.6.19-rc6-mm2/arch/s390/kernel/syscalls.S
@@ -321,3 +321,5 @@ SYSCALL(sys_vmsplice,sys_vmsplice,compat
 NI_SYSCALL       /* 310 sys_move_pages */
 SYSCALL(sys_getcpu,sys_getcpu,sys_getcpu_wrapper)
 SYSCALL(sys_epoll_pwait,sys_epoll_pwait,sys_ni_syscall)
+SYSCALL(sys_unshare_ns,sys_unshare_ns,sys_unshare_ns_wrapper)
+SYSCALL(sys_bind_ns,sys_bind_ns,sys_bind_ns_wrapper)
Index: 2.6.19-rc6-mm2/arch/x86_64/ia32/ia32entry.S
===================================================================
--- 2.6.19-rc6-mm2.orig/arch/x86_64/ia32/ia32entry.S
+++ 2.6.19-rc6-mm2/arch/x86_64/ia32/ia32entry.S
@@ -722,4 +722,6 @@ ia32_sys_call_table:
  .quad sys_kevent_ctl  /* 320 */
  .quad sys_kevent_wait
  .quad sys_kevent_ring_init
+ .quad sys_unshare_ns
+ .quad sys_bind_ns
 ia32_syscall_end:
Index: 2.6.19-rc6-mm2/include/asm-i386/unistd.h
===================================================================
--- 2.6.19-rc6-mm2.orig/include/asm-i386/unistd.h
+++ 2.6.19-rc6-mm2/include/asm-i386/unistd.h
@@ -329,10 +329,12 @@
 #define __NR_kevent_ctl  321
 #define __NR_kevent_wait 322
 #define __NR_kevent_ring_init 323
+#define __NR_unshare_ns  324
+#define __NR_bind_ns  325

 #ifdef __KERNEL__

-#define NR_syscalls 324
+#define NR_syscalls 326

 #define __ARCH_WANT_IPC_PARSE_VERSION
 #define __ARCH_WANT_OLD_READDIR
```

```
Index: 2.6.19-rc6-mm2/include/asm-ia64/unistd.h
===================================================================
--- 2.6.19-rc6-mm2.orig/include/asm-ia64/unistd.h
+++ 2.6.19-rc6-mm2/include/asm-ia64/unistd.h
@@ -291,11 +291,13 @@
 #define __NR_sync_file_range  1300
 #define __NR_tee   1301
 #define __NR_vmsplice   1302
+#define __NR_unshare_ns   1303
+#define __NR_bind_ns   1304

 #ifdef __KERNEL__


-#define NR_syscalls   279 /* length of syscall table */
+#define NR_syscalls   281 /* length of syscall table */

 #define __ARCH_WANT_SYS_RT_SIGACTION

Index: 2.6.19-rc6-mm2/include/asm-powerpc/systbl.h
===================================================================
--- 2.6.19-rc6-mm2.orig/include/asm-powerpc/systbl.h
+++ 2.6.19-rc6-mm2/include/asm-powerpc/systbl.h
@@ -305,3 +305,5 @@ SYSCALL_SPU(faccessat)
 COMPAT_SYS_SPU(get_robust_list)
 COMPAT_SYS_SPU(set_robust_list)
 COMPAT_SYS(move_pages)
+SYSCALL_SPU(unshare_ns)
+SYSCALL_SPU(bind_ns)
Index: 2.6.19-rc6-mm2/include/asm-powerpc/unistd.h
===================================================================
--- 2.6.19-rc6-mm2.orig/include/asm-powerpc/unistd.h
+++ 2.6.19-rc6-mm2/include/asm-powerpc/unistd.h
@@ -324,10 +324,12 @@
 #define __NR_get_robust_list 299
 #define __NR_set_robust_list 300
 #define __NR_move_pages  301
+#define __NR_unshare_ns  302
+#define __NR_bind_ns  303

 #ifdef __KERNEL__

-#define __NR_syscalls  302
+#define __NR_syscalls  304

 #define __NR__exit __NR_exit
 #define NR_syscalls __NR_syscalls
Index: 2.6.19-rc6-mm2/include/asm-s390/unistd.h
```

```
===================================================================
--- 2.6.19-rc6-mm2.orig/include/asm-s390/unistd.h
+++ 2.6.19-rc6-mm2/include/asm-s390/unistd.h
@@ -250,8 +250,10 @@
 /* Number 310 is reserved for new sys_move_pages */
 #define __NR_getcpu  311
 #define __NR_epoll_pwait 312
+#define __NR_unshare_ns  313
+#define __NR_bind_ns  314

-#define NR_syscalls 313
+#define NR_syscalls 315

 /*
  * There are some system calls that are not present on 64 bit, some
Index: 2.6.19-rc6-mm2/include/asm-x86_64/unistd.h
===================================================================
--- 2.6.19-rc6-mm2.orig/include/asm-x86_64/unistd.h
+++ 2.6.19-rc6-mm2/include/asm-x86_64/unistd.h
@@ -627,8 +627,12 @@ __SYSCALL(__NR_kevent_ctl, sys_kevent_ct
 __SYSCALL(__NR_kevent_wait, sys_kevent_wait)
 #define __NR_kevent_ring_init 283
 __SYSCALL(__NR_kevent_ring_init, sys_kevent_ring_init)
+#define __NR_unshare_ns  284
+__SYSCALL(__NR_unshare_ns, sys_unshare_ns)
+#define __NR_bind_ns  285
+__SYSCALL(__NR_bind_ns, sys_bind_ns)

-#define __NR_syscall_max __NR_kevent_ring_init
+#define __NR_syscall_max __NR_bind_ns

 #ifndef __NO_STUBS
 #define __ARCH_WANT_OLD_READDIR

--

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers
```

Subject: [patch -mm 11/17] user namespace: add user_namespace ptr to vfsmount
Posted by Cedric Le Goater on Tue, 05 Dec 2006 10:28:03 GMT
View Forum Message <> Reply to Message

From: Serge E. Hallyn <serue@us.ibm.com>

Add user_namespace ptr to vfsmount, and define a helper to compare it

to the task's user_ns.

Signed-off-by: Serge E. Hallyn <serue@us.ibm.com>
---
 fs/namespace.c | 4 ++++
 include/linux/mount.h | 2 ++
 include/linux/sched.h | 12 ++++++++++++
 3 files changed, 18 insertions(+)

Index: 2.6.19-rc6-mm2/fs/namespace.c
===
--- 2.6.19-rc6-mm2.orig/fs/namespace.c
+++ 2.6.19-rc6-mm2/fs/namespace.c
@@ -25,6 +25,7 @@
 #include <linux/security.h>
 #include <linux/mount.h>
 #include <linux/ramfs.h>
+#include <linux/user_namespace.h>
 #include <asm/uaccess.h>
 #include <asm/unistd.h>
 #include "pnode.h"
@@ -56,6 +57,8 @@ struct vfsmount *alloc_vfsmnt(const char
  struct vfsmount *mnt = kmem_cache_alloc(mnt_cache, GFP_KERNEL);
  if (mnt) {
   memset(mnt, 0, sizeof(struct vfsmount));
+ mnt->mnt_user_ns = current->nsproxy->user_ns;
+ get_user_ns(mnt->mnt_user_ns);
   atomic_set(&mnt->mnt_count, 1);
   INIT_LIST_HEAD(&mnt->mnt_hash);
   INIT_LIST_HEAD(&mnt->mnt_child);
@@ -88,6 +91,7 @@ EXPORT_SYMBOL(simple_set_mnt);

 void free_vfsmnt(struct vfsmount *mnt)
 {
+ put_user_ns(mnt->mnt_user_ns);
  kfree(mnt->mnt_devname);
  kmem_cache_free(mnt_cache, mnt);
 }
Index: 2.6.19-rc6-mm2/include/linux/mount.h
===
--- 2.6.19-rc6-mm2.orig/include/linux/mount.h
+++ 2.6.19-rc6-mm2/include/linux/mount.h
@@ -21,6 +21,7 @@ struct super_block;
 struct vfsmount;
 struct dentry;
 struct mnt_namespace;
+struct user_namespace;

```
 #define MNT_NOSUID 0x01
 #define MNT_NODEV 0x02
@@ -53,6 +54,7 @@ struct vfsmount {
  struct list_head mnt_slave; /* slave list entry */
  struct vfsmount *mnt_master; /* slave is on master->mnt_slave_list */
  struct mnt_namespace *mnt_ns; /* containing namespace */
+ struct user_namespace *mnt_user_ns; /* namespace for uid interpretation */
  int mnt_pinned;
 };

Index: 2.6.19-rc6-mm2/include/linux/sched.h
===================================================================
--- 2.6.19-rc6-mm2.orig/include/linux/sched.h
+++ 2.6.19-rc6-mm2/include/linux/sched.h
@@ -83,6 +83,8 @@ struct sched_param {
 #include <linux/timer.h>
 #include <linux/hrtimer.h>
 #include <linux/task_io_accounting.h>
+#include <linux/nsproxy.h>
+#include <linux/mount.h>

 #include <asm/processor.h>

@@ -1589,6 +1591,16 @@ extern int cond_resched(void);
 extern int cond_resched_lock(spinlock_t * lock);
 extern int cond_resched_softirq(void);

+static inline int task_mnt_same_uid(struct task_struct *tsk,
+     struct vfsmount *mnt)
+{
+ if (tsk->nsproxy == init_task.nsproxy)
+  return 1;
+ if (mnt->mnt_user_ns == tsk->nsproxy->user_ns)
+  return 1;
+ return 0;
+}
+
 /*
  * Does a critical section need to be broken due to another
  * task waiting?:

--

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers
```

Subject: [patch -mm 12/17] user namespace: hook permission
Posted by Cedric Le Goater on Tue, 05 Dec 2006 10:28:04 GMT
View Forum Message <> Reply to Message

From: Serge Hallyn <serue@us.ibm.com>

Hook permission to check vfsmnt->user_ns against current.

Signed-off-by: Serge E. Hallyn <serue@us.ibm.com>
---
fs/namei.c | 4 ++++
1 file changed, 4 insertions(+)

Index: 2.6.19-rc6-mm2/fs/namei.c
===
--- 2.6.19-rc6-mm2.orig/fs/namei.c
+++ 2.6.19-rc6-mm2/fs/namei.c
@@ -246,6 +246,8 @@ int permission(struct inode *inode, int
return -EACCES;
}

+ if (nd && !task_mnt_same_uid(current, nd->mnt))
+ return -EACCES;

/*
* MAY_EXEC on regular files requires special handling: We override
@@ -433,6 +435,8 @@ static int exec_permission_lite(struct i
{
umode_t mode = inode->i_mode;

+ if (!task_mnt_same_uid(current, nd->mnt))
+ return -EACCES;
if (inode->i_op && inode->i_op->permission)
return -EAGAIN;

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

Subject: [patch -mm 13/17] user namespace: implement shared mounts
Posted by Cedric Le Goater on Tue, 05 Dec 2006 10:28:05 GMT
View Forum Message <> Reply to Message

From: Serge E. Hallyn <serue@us.ibm.com>

Implement shared and private userns mounts. Private userns mounts
are ones which no process in another user namespace can use.

Here is a sample mount program which only does a bind mount of arg1
onto arg2, but making the destination a private userns mount.

```
int main(int argc, char *argv[])
{
 int type;
 if(argc != 3) {
  fprintf(stderr, "usage: %s src dest", argv[0]);
  return 1;
 }

 fprintf(stdout, "%s %s %s\n", argv[0], argv[1], argv[2]);

 type = MS_PRIV_USERNS | MS_BIND;
 setfsuid(getuid());

 if(mount(argv[1], argv[2], "none", type, "") == -1) {
  perror("mount");
  return 1;
 }
 return 0;
}
```

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

```
 fs/namespace.c        |   16 ++++++++++++----
 fs/pnode.h            |    1 +
 include/linux/fs.h    |    1 +
 include/linux/mount.h |    1 +
 include/linux/sched.h |    2 ++
 5 files changed, 17 insertions(+), 4 deletions(-)

Index: 2.6.19-rc6-mm2/fs/namespace.c
===================================================================
--- 2.6.19-rc6-mm2.orig/fs/namespace.c
+++ 2.6.19-rc6-mm2/fs/namespace.c
@@ -259,6 +259,8 @@ static struct vfsmount *clone_mnt(struct
   }
   if (flag & CL_MAKE_SHARED)
    set_mnt_shared(mnt);
+  if (flag & CL_PRIV_USERNS)
+   mnt->mnt_flags |= MNT_PRIV_USERNS;

   /* stick the duplicate mount on the same expiry list
```

```
    * as the original if that was on one */
@@ -370,6 +372,7 @@ static int show_vfsmnt(struct seq_file *
   { MNT_NOSUID, ",nosuid" },
   { MNT_NODEV, ",nodev" },
   { MNT_NOEXEC, ",noexec" },
+  { MNT_PRIV_USERNS, ",privuserns" },
   { MNT_NOATIME, ",noatime" },
   { MNT_NODIRATIME, ",nodiratime" },
   { 0, NULL }
@@ -901,11 +904,14 @@ static int do_change_type(struct nameida
 /*
  * do loopback mount.
  */
-static int do_loopback(struct nameidata *nd, char *old_name, int recurse)
+static int do_loopback(struct nameidata *nd, char *old_name, int recurse,
+       int uidns_share)
 {
  struct nameidata old_nd;
  struct vfsmount *mnt = NULL;
  int err = mount_is_safe(nd);
+ int flag = (uidns_share ? CL_PRIV_USERNS : 0);
+
  if (err)
   return err;
  if (!old_name || !*old_name)
@@ -924,9 +930,9 @@ static int do_loopback(struct nameidata

  err = -ENOMEM;
  if (recurse)
-  mnt = copy_tree(old_nd.mnt, old_nd.dentry, 0);
+  mnt = copy_tree(old_nd.mnt, old_nd.dentry, flag);
  else
-  mnt = clone_mnt(old_nd.mnt, old_nd.dentry, 0);
+  mnt = clone_mnt(old_nd.mnt, old_nd.dentry, flag);

  if (!mnt)
   goto out;
@@ -1409,6 +1415,8 @@ long do_mount(char *dev_name, char *dir_
   mnt_flags |= MNT_NOATIME;
  if (flags & MS_NODIRATIME)
   mnt_flags |= MNT_NODIRATIME;
+ if (flags & MS_PRIV_USERNS)
+  mnt_flags |= MNT_PRIV_USERNS;

  flags &= ~(MS_NOSUID | MS_NOEXEC | MS_NODEV | MS_ACTIVE |
      MS_NOATIME | MS_NODIRATIME);
@@ -1426,7 +1434,7 @@ long do_mount(char *dev_name, char *dir_
   retval = do_remount(&nd, flags & ~MS_REMOUNT, mnt_flags,
```

```
        data_page);
  else if (flags & MS_BIND)
-  retval = do_loopback(&nd, dev_name, flags & MS_REC);
+  retval = do_loopback(&nd, dev_name, flags & MS_REC, flags & MS_PRIV_USERNS);
  else if (flags & (MS_SHARED | MS_PRIVATE | MS_SLAVE | MS_UNBINDABLE))
   retval = do_change_type(&nd, flags);
  else if (flags & MS_MOVE)
Index: 2.6.19-rc6-mm2/fs/pnode.h
===================================================================
--- 2.6.19-rc6-mm2.orig/fs/pnode.h
+++ 2.6.19-rc6-mm2/fs/pnode.h
@@ -22,6 +22,7 @@
 #define CL_COPY_ALL   0x04
 #define CL_MAKE_SHARED   0x08
 #define CL_PROPAGATION   0x10
+#define CL_PRIV_USERNS   0x20

 static inline void set_mnt_shared(struct vfsmount *mnt)
 {
Index: 2.6.19-rc6-mm2/include/linux/fs.h
===================================================================
--- 2.6.19-rc6-mm2.orig/include/linux/fs.h
+++ 2.6.19-rc6-mm2/include/linux/fs.h
@@ -120,6 +120,7 @@ extern int dir_notify_enable;
 #define MS_PRIVATE (1<<18) /* change to private */
 #define MS_SLAVE (1<<19) /* change to slave */
 #define MS_SHARED (1<<20) /* change to shared */
+#define MS_PRIV_USERNS (1<<21) /* compare user namespaces for permission */
 #define MS_ACTIVE (1<<30)
 #define MS_NOUSER (1<<31)

Index: 2.6.19-rc6-mm2/include/linux/mount.h
===================================================================
--- 2.6.19-rc6-mm2.orig/include/linux/mount.h
+++ 2.6.19-rc6-mm2/include/linux/mount.h
@@ -34,6 +34,7 @@ struct user_namespace;
 #define MNT_SHARED 0x1000 /* if the vfsmount is a shared mount */
 #define MNT_UNBINDABLE 0x2000 /* if the vfsmount is a unbindable mount */
 #define MNT_PNODE_MASK 0x3000 /* propogation flag mask */
+#define MNT_PRIV_USERNS 0x4000  /* compare user namespaces for permission */

 struct vfsmount {
  struct list_head mnt_hash;
Index: 2.6.19-rc6-mm2/include/linux/sched.h
===================================================================
--- 2.6.19-rc6-mm2.orig/include/linux/sched.h
+++ 2.6.19-rc6-mm2/include/linux/sched.h
@@ -1596,6 +1596,8 @@ static inline int task_mnt_same_uid(stru
```

```
 {
  if (tsk->nsproxy == init_task.nsproxy)
   return 1;
+ if (!(mnt->mnt_flags & MNT_PRIV_USERNS))
+  return 1;
  if (mnt->mnt_user_ns == tsk->nsproxy->user_ns)
   return 1;
  return 0;

--
```

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

Subject: [patch -mm 14/17] user namespace: maintain user ns for priv_userns mounts to vfsmount
Posted by Cedric Le Goater on Tue, 05 Dec 2006 10:28:06 GMT
View Forum Message <> Reply to Message

From: Serge E. Hallyn <serue@us.ibm.com>

If a vfsmount is marked priv_userns, then a bind mount should
maintain the old->user_ns. Not doing so presents a simple way
around the priv_userns security mechanism :)

Signed-off-by: Serge E. Hallyn <serue@us.ibm.com>
---
 fs/namespace.c | 2 ++
 1 file changed, 2 insertions(+)

Index: 2.6.19-rc6-mm2/fs/namespace.c
===
--- 2.6.19-rc6-mm2.orig/fs/namespace.c
+++ 2.6.19-rc6-mm2/fs/namespace.c
@@ -240,6 +240,8 @@ static struct vfsmount *clone_mnt(struct

  if (mnt) {
   mnt->mnt_flags = old->mnt_flags;
+ if (mnt->mnt_flags & MNT_PRIV_USERNS)
+   mnt->mnt_user_ns = old->mnt_user_ns;
   atomic_inc(&sb->s_active);
   mnt->mnt_sb = sb;
   mnt->mnt_root = dget(root);

--

Subject: [patch -mm 15/17] pid namespace: add unshare
Posted by Cedric Le Goater on Tue, 05 Dec 2006 10:28:07 GMT
View Forum Message <> Reply to Message

From: Cedric Le Goater <clg@fr.ibm.com>

Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>
---
include/linux/pid_namespace.h | 2 +
kernel/nsproxy.c | 25 ++++++++++++++++++++--
kernel/pid.c | 44 ++
3 files changed, 68 insertions(+), 3 deletions(-)

Index: 2.6.19-rc6-mm2/include/linux/pid_namespace.h
===
--- 2.6.19-rc6-mm2.orig/include/linux/pid_namespace.h
+++ 2.6.19-rc6-mm2/include/linux/pid_namespace.h
@@ -29,6 +29,8 @@ static inline void get_pid_ns(struct pid
 kref_get(&ns->kref);
}

+extern int unshare_pid_ns(unsigned long unshare_ns_flags,
+ struct pid_namespace **new_pid);
extern int copy_pid_ns(int flags, struct task_struct *tsk);
extern void free_pid_ns(struct kref *kref);

Index: 2.6.19-rc6-mm2/kernel/pid.c
===
--- 2.6.19-rc6-mm2.orig/kernel/pid.c
+++ 2.6.19-rc6-mm2/kernel/pid.c
@@ -361,6 +361,50 @@ struct pid *find_ge_pid(int nr)
}
EXPORT_SYMBOL_GPL(find_get_pid);

+static struct pid_namespace *clone_pid_ns(struct pid_namespace *old_ns)
+{
+ struct pid_namespace *ns;
+ int i;
+
+ ns = kmalloc(sizeof(struct pid_namespace), GFP_KERNEL);
+ if (!ns)
+ return ns;
+

```
+ kref_init(&ns->kref);
+
+ atomic_set(&ns->pidmap[0].nr_free, BITS_PER_PAGE - 1);
+ ns->pidmap[0].page = kzalloc(PAGE_SIZE, GFP_KERNEL);
+ if (!ns->pidmap[0].page) {
+  kfree(ns);
+  return NULL;
+ }
+
+ set_bit(0, ns->pidmap[0].page);
+
+ for (i = 1; i < PIDMAP_ENTRIES; i++) {
+  atomic_set(&ns->pidmap[i].nr_free, BITS_PER_PAGE);
+  ns->pidmap[i].page = NULL;
+ }
+ ns->last_pid = 0;
+ ns->child_reaper = current;
+ return ns;
+}
+
+int unshare_pid_ns(unsigned long unshare_ns_flags,
+     struct pid_namespace **new_pid)
+{
+ if (unshare_ns_flags & NS_PID) {
+  if (!capable(CAP_SYS_ADMIN))
+   return -EPERM;
+
+  *new_pid = clone_pid_ns(current->nsproxy->pid_ns);
+  if (!*new_pid)
+   return -ENOMEM;
+ }
+
+ return 0;
+}
+
 int copy_pid_ns(int flags, struct task_struct *tsk)
 {
  struct pid_namespace *old_ns = tsk->nsproxy->pid_ns;
Index: 2.6.19-rc6-mm2/kernel/nsproxy.c
===================================================================
--- 2.6.19-rc6-mm2.orig/kernel/nsproxy.c
+++ 2.6.19-rc6-mm2/kernel/nsproxy.c
@@ -324,6 +324,11 @@ static int switch_ns(int id, unsigned lo
    put_ipc_ns(new_ns->ipc_ns);
    new_ns->ipc_ns = get_ipc_ns(ns->ipc_ns);
   }
+  if (flags & NS_PID) {
+   get_pid_ns(ns->pid_ns);
```

```
+   put_pid_ns(new_ns->pid_ns);
+   new_ns->pid_ns = ns->pid_ns;
+  }
  out_ns:
   put_nsproxy(ns);
  }
@@ -440,6 +445,7 @@ asmlinkage long sys_unshare_ns(unsigned
   struct mnt_namespace *mnt, *new_mnt = NULL;
   struct uts_namespace *uts, *new_uts = NULL;
   struct ipc_namespace *ipc, *new_ipc = NULL;
+ struct pid_namespace *pid, *new_pid = NULL;
   unsigned long unshare_flags = 0;

   /* Return -EINVAL for all unsupported flags */
@@ -467,16 +473,19 @@ asmlinkage long sys_unshare_ns(unsigned
   if ((err = unshare_ipcs(unshare_flags, &new_ipc)))
    goto bad_unshare_ns_cleanup_uts;

-  if (new_mnt || new_uts || new_ipc) {
+ if ((err = unshare_pid_ns(unshare_ns_flags, &new_pid)))
+  goto bad_unshare_ns_cleanup_ipc;
+
+ if (new_mnt || new_uts || new_ipc || new_pid) {
    old_nsproxy = current->nsproxy;
    new_nsproxy = dup_namespaces(old_nsproxy);
    if (!new_nsproxy) {
     err = -ENOMEM;
-    goto bad_unshare_ns_cleanup_ipc;
+    goto bad_unshare_ns_cleanup_pid;
    }
   }

-  if (new_fs || new_mnt || new_uts || new_ipc) {
+  if (new_fs || new_mnt || new_uts || new_ipc || new_pid) {

    task_lock(current);

@@ -509,12 +518,22 @@ asmlinkage long sys_unshare_ns(unsigned
     new_ipc = ipc;
    }

+  if (new_pid) {
+   pid = current->nsproxy->pid_ns;
+   current->nsproxy->pid_ns = new_pid;
+   new_pid = pid;
+  }
+
    task_unlock(current);
```

```
  }

  if (new_nsproxy)
   put_nsproxy(new_nsproxy);

+bad_unshare_ns_cleanup_pid:
+ if (new_pid)
+  put_pid_ns(new_pid);
+
 bad_unshare_ns_cleanup_ipc:
  if (new_ipc)
   put_ipc_ns(new_ipc);

--
```

_____

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

Subject: [patch -mm 16/17] net namespace: add unshare
Posted by Cedric Le Goater on Tue, 05 Dec 2006 10:28:08 GMT
View Forum Message <> Reply to Message

From: Cedric Le Goater <clg@fr.ibm.com>

Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>
---
 include/linux/net_namespace.h | 13 ++++++++++++
 kernel/nsproxy.c | 25 ++++++++++++++++++++---
 net/core/net_namespace.c | 35 ++++++++++++++++++++++++++++++++++
 3 files changed, 70 insertions(+), 3 deletions(-)

Index: 2.6.19-rc6-mm2/include/linux/net_namespace.h
===
--- 2.6.19-rc6-mm2.orig/include/linux/net_namespace.h
+++ 2.6.19-rc6-mm2/include/linux/net_namespace.h
@@ -3,6 +3,7 @@

 #include <linux/kref.h>
 #include <linux/nsproxy.h>
+#include <linux/errno.h>

 struct net_namespace {
  struct kref kref;
@@ -19,6 +20,9 @@ static inline void get_net_ns(struct net
  kref_get(&ns->kref);
 }

```
+extern int unshare_net_ns(unsigned long unshare_flags,
+     struct net_namespace **new_net);
+
 extern int copy_net_ns(int flags, struct task_struct *tsk);

 extern void free_net_ns(struct kref *kref);
@@ -36,6 +40,15 @@ static inline void get_net_ns(struct net
 {
 }

+static inline int unshare_net_ns(unsigned long unshare_flags,
+     struct net_namespace **new_net)
+{
+ if (unshare_flags & NS_NET)
+  return -EINVAL;
+
+ return 0;
+}
+
 static inline int copy_net_ns(int flags, struct task_struct *tsk)
 {
  return 0;
Index: 2.6.19-rc6-mm2/kernel/nsproxy.c
===================================================================
--- 2.6.19-rc6-mm2.orig/kernel/nsproxy.c
+++ 2.6.19-rc6-mm2/kernel/nsproxy.c
@@ -329,6 +329,12 @@ static int switch_ns(int id, unsigned lo
    put_pid_ns(new_ns->pid_ns);
    new_ns->pid_ns = ns->pid_ns;
   }
+  if (flags & NS_NET) {
+   get_net_ns(ns->net_ns);
+   put_net_ns(new_ns->net_ns);
+   new_ns->net_ns = ns->net_ns;
+  }
+
  out_ns:
   put_nsproxy(ns);
  }
@@ -446,6 +452,7 @@ asmlinkage long sys_unshare_ns(unsigned
   struct uts_namespace *uts, *new_uts = NULL;
   struct ipc_namespace *ipc, *new_ipc = NULL;
  struct pid_namespace *pid, *new_pid = NULL;
+ struct net_namespace *net, *new_net = NULL;
   unsigned long unshare_flags = 0;

   /* Return -EINVAL for all unsupported flags */
```

```
@@ -475,17 +482,19 @@ asmlinkage long sys_unshare_ns(unsigned

  if ((err = unshare_pid_ns(unshare_ns_flags, &new_pid)))
   goto bad_unshare_ns_cleanup_ipc;
+ if ((err = unshare_net_ns(unshare_ns_flags, &new_net)))
+  goto bad_unshare_ns_cleanup_pid;

- if (new_mnt || new_uts || new_ipc || new_pid) {
+ if (new_mnt || new_uts || new_ipc || new_pid || new_net) {
    old_nsproxy = current->nsproxy;
    new_nsproxy = dup_namespaces(old_nsproxy);
    if (!new_nsproxy) {
     err = -ENOMEM;
-    goto bad_unshare_ns_cleanup_pid;
+    goto bad_unshare_ns_cleanup_net;
    }
   }

-  if (new_fs || new_mnt || new_uts || new_ipc || new_pid) {
+  if (new_fs || new_mnt || new_uts || new_ipc || new_pid || new_net) {

    task_lock(current);

@@ -524,12 +533,22 @@ asmlinkage long sys_unshare_ns(unsigned
    new_pid = pid;
   }

+  if (new_net) {
+   net = current->nsproxy->net_ns;
+   current->nsproxy->net_ns = new_net;
+   new_net = net;
+  }
+
    task_unlock(current);
   }

   if (new_nsproxy)
    put_nsproxy(new_nsproxy);

+bad_unshare_ns_cleanup_net:
+ if (new_net)
+  put_net_ns(new_net);
+
 bad_unshare_ns_cleanup_pid:
  if (new_pid)
   put_pid_ns(new_pid);
Index: 2.6.19-rc6-mm2/net/core/net_namespace.c
===================================================================
```

```
--- 2.6.19-rc6-mm2.orig/net/core/net_namespace.c
+++ 2.6.19-rc6-mm2/net/core/net_namespace.c
@@ -18,6 +18,41 @@ struct net_namespace init_net_ns = {

 #ifdef CONFIG_NET_NS

+/*
+ * Clone a new ns copying an original net ns, setting refcount to 1
+ * @old_ns: namespace to clone
+ * Return NULL on error (failure to kmalloc), new ns otherwise
+ */
+static struct net_namespace *clone_net_ns(struct net_namespace *old_ns)
+{
+ struct net_namespace *ns;
+
+ ns = kmalloc(sizeof(struct net_namespace), GFP_KERNEL);
+ if (!ns)
+  return NULL;
+
+ kref_init(&ns->kref);
+ return ns;
+}
+
+/*
+ * unshare the current process' net namespace.
+ */
+int unshare_net_ns(unsigned long unshare_flags,
+     struct net_namespace **new_net)
+{
+ if (unshare_flags & NS_NET) {
+  if (!capable(CAP_SYS_ADMIN))
+   return -EPERM;
+
+  *new_net = clone_net_ns(current->nsproxy->net_ns);
+  if (!*new_net)
+   return -ENOMEM;
+ }
+
+ return 0;
+}
+
 int copy_net_ns(int flags, struct task_struct *tsk)
 {
  struct net_namespace *old_ns = tsk->nsproxy->net_ns;

--
```

Subject: [patch -mm 17/17] user namespace: add unshare
Posted by Cedric Le Goater on Tue, 05 Dec 2006 10:28:09 GMT
View Forum Message <> Reply to Message

From: Cedric Le Goater <clg@fr.ibm.com>

Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>
---
include/linux/user_namespace.h | 11 +++++++
kernel/nsproxy.c | 26 +++++++++++++++++--
kernel/user_namespace.c | 57 ++
3 files changed, 91 insertions(+), 3 deletions(-)

Index: 2.6.19-rc6-mm2/kernel/nsproxy.c
===
--- 2.6.19-rc6-mm2.orig/kernel/nsproxy.c
+++ 2.6.19-rc6-mm2/kernel/nsproxy.c
@@ -335,6 +335,12 @@ static int switch_ns(int id, unsigned lo
new_ns->net_ns = ns->net_ns;
}

+ if (flags & NS_USER) {
+ get_user_ns(ns->user_ns);
+ put_user_ns(new_ns->user_ns);
+ new_ns->user_ns = ns->user_ns;
+ }
+
out_ns:
put_nsproxy(ns);
}
@@ -453,6 +459,7 @@ asmlinkage long sys_unshare_ns(unsigned
struct ipc_namespace *ipc, *new_ipc = NULL;
struct pid_namespace *pid, *new_pid = NULL;
struct net_namespace *net, *new_net = NULL;
+ struct user_namespace *user, *new_user = NULL;
unsigned long unshare_flags = 0;

/* Return -EINVAL for all unsupported flags */
@@ -484,17 +491,20 @@ asmlinkage long sys_unshare_ns(unsigned
goto bad_unshare_ns_cleanup_ipc;
if ((err = unshare_net_ns(unshare_ns_flags, &new_net)))
goto bad_unshare_ns_cleanup_pid;
+ if ((err = unshare_user_ns(unshare_ns_flags, &new_user)))
+ goto bad_unshare_ns_cleanup_net;

```
- if (new_mnt || new_uts || new_ipc || new_pid || new_net) {
+ if (new_mnt || new_uts || new_ipc || new_pid || new_net || new_user) {
    old_nsproxy = current->nsproxy;
    new_nsproxy = dup_namespaces(old_nsproxy);
    if (!new_nsproxy) {
     err = -ENOMEM;
-    goto bad_unshare_ns_cleanup_net;
+    goto bad_unshare_ns_cleanup_user;
    }
   }

-  if (new_fs || new_mnt || new_uts || new_ipc || new_pid || new_net) {
+  if (new_fs || new_mnt || new_uts || new_ipc || new_pid || new_net ||
+      new_user) {

    task_lock(current);

@@ -539,12 +549,22 @@ asmlinkage long sys_unshare_ns(unsigned
    new_net = net;
   }

+  if (new_user) {
+   user = current->nsproxy->user_ns;
+   current->nsproxy->user_ns = new_user;
+   new_user = user;
+  }
+
    task_unlock(current);
   }

   if (new_nsproxy)
    put_nsproxy(new_nsproxy);

+bad_unshare_ns_cleanup_user:
+ if (new_user)
+  put_user_ns(new_user);
+
 bad_unshare_ns_cleanup_net:
  if (new_net)
   put_net_ns(new_net);
Index: 2.6.19-rc6-mm2/include/linux/user_namespace.h
===================================================================
--- 2.6.19-rc6-mm2.orig/include/linux/user_namespace.h
+++ 2.6.19-rc6-mm2/include/linux/user_namespace.h
@@ -22,6 +22,8 @@ static inline void get_user_ns(struct us
  kref_get(&ns->kref);
 }
```

```
+extern int unshare_user_ns(unsigned long unshare_flags,
+      struct user_namespace **new_user);
 extern int copy_user_ns(int flags, struct task_struct *tsk);
 extern void free_user_ns(struct kref *kref);

@@ -36,6 +38,15 @@ static inline void get_user_ns(struct us
 {
 }

+static inline int unshare_user_ns(unsigned long unshare_flags,
+      struct user_namespace **new_user)
+{
+ if (unshare_flags & NS_USER)
+  return -EINVAL;
+
+ return 0;
+}
+
 static inline int copy_user_ns(int flags, struct task_struct *tsk)
 {
  return 0;
Index: 2.6.19-rc6-mm2/kernel/user_namespace.c
===================================================================
--- 2.6.19-rc6-mm2.orig/kernel/user_namespace.c
+++ 2.6.19-rc6-mm2/kernel/user_namespace.c
@@ -21,6 +21,63 @@ EXPORT_SYMBOL_GPL(init_user_ns);

 #ifdef CONFIG_USER_NS

+/*
+ * Clone a new ns copying an original user ns, setting refcount to 1
+ * @old_ns: namespace to clone
+ * Return NULL on error (failure to kmalloc), new ns otherwise
+ */
+static struct user_namespace *clone_user_ns(struct user_namespace *old_ns)
+{
+ struct user_namespace *ns;
+ struct user_struct *new_user;
+ int n;
+
+ ns = kmalloc(sizeof(struct user_namespace), GFP_KERNEL);
+ if (!ns)
+  return NULL;
+
+ kref_init(&ns->kref);
+
+ for(n = 0; n < UIDHASH_SZ; ++n)
```

```
+  INIT_LIST_HEAD(ns->uidhash_table + n);
+
+ /* Insert new root user.  */
+ ns->root_user = alloc_uid(ns, 0);
+ if (!ns->root_user) {
+  kfree(ns);
+  return NULL;
+ }
+
+ /* Reset current->user with a new one */
+ new_user = alloc_uid(ns, current->uid);
+ if (!new_user) {
+  free_uid(ns->root_user);
+  kfree(ns);
+  return NULL;
+ }
+
+ switch_uid(new_user);
+ return ns;
+}
+
+/*
+ * unshare the current process' user namespace.
+ */
+int unshare_user_ns(unsigned long unshare_flags,
+      struct user_namespace **new_user)
+{
+ if (unshare_flags & NS_USER) {
+  if (!capable(CAP_SYS_ADMIN))
+   return -EPERM;
+
+  *new_user = clone_user_ns(current->nsproxy->user_ns);
+  if (!*new_user)
+   return -ENOMEM;
+ }
+
+ return 0;
+}
+
 int copy_user_ns(int flags, struct task_struct *tsk)
 {
  struct user_namespace *old_ns = tsk->nsproxy->user_ns;

--
```

Subject: Re: [patch -mm 11/17] user namespace: add user_namespace ptr to vfsmount
Posted by serue on Tue, 05 Dec 2006 18:27:44 GMT
View Forum Message <> Reply to Message

Quoting clg@fr.ibm.com (clg@fr.ibm.com):
> From: Serge E. Hallyn <serue@us.ibm.com>
>
> Add user_namespace ptr to vfsmount, and define a helper to compare it
> to the task's user_ns.

Eric,

Patches 11-14 are my next iteration of the patches to tag vfsmounts
with a user_namespace ptr. Would these make the user namespaces
acceptable to you?

These still leave signals unaddressed, but one can argue that signals
are implicitly addressed through pidspaces. If we decide that's
insufficient, we can add user_ns checks to signal permission checks.

thanks,
-serge

>
> Signed-off-by: Serge E. Hallyn <serue@us.ibm.com>
> ---
> fs/namespace.c | 4 ++++
> include/linux/mount.h | 2 ++
> include/linux/sched.h | 12 ++++++++++++
> 3 files changed, 18 insertions(+)
>
> Index: 2.6.19-rc6-mm2/fs/namespace.c
> ===
> --- 2.6.19-rc6-mm2.orig/fs/namespace.c
> +++ 2.6.19-rc6-mm2/fs/namespace.c
> @@ -25,6 +25,7 @@
> #include <linux/security.h>
> #include <linux/mount.h>
> #include <linux/ramfs.h>
> +#include <linux/user_namespace.h>
> #include <asm/uaccess.h>
> #include <asm/unistd.h>
> #include "pnode.h"
> @@ -56,6 +57,8 @@ struct vfsmount *alloc_vfsmnt(const char
> struct vfsmount *mnt = kmem_cache_alloc(mnt_cache, GFP_KERNEL);
> if (mnt) {
> memset(mnt, 0, sizeof(struct vfsmount));
> + mnt->mnt_user_ns = current->nsproxy->user_ns;

> + get_user_ns(mnt->mnt_user_ns);
>   atomic_set(&mnt->mnt_count, 1);
>   INIT_LIST_HEAD(&mnt->mnt_hash);
>   INIT_LIST_HEAD(&mnt->mnt_child);
> @@ -88,6 +91,7 @@ EXPORT_SYMBOL(simple_set_mnt);
>
> void free_vfsmnt(struct vfsmount *mnt)
> {
> + put_user_ns(mnt->mnt_user_ns);
>   kfree(mnt->mnt_devname);
>   kmem_cache_free(mnt_cache, mnt);
> }
> Index: 2.6.19-rc6-mm2/include/linux/mount.h
> ===
> --- 2.6.19-rc6-mm2.orig/include/linux/mount.h
> +++ 2.6.19-rc6-mm2/include/linux/mount.h
> @@ -21,6 +21,7 @@ struct super_block;
> struct vfsmount;
> struct dentry;
> struct mnt_namespace;
> +struct user_namespace;
>
> #define MNT_NOSUID 0x01
> #define MNT_NODEV 0x02
> @@ -53,6 +54,7 @@ struct vfsmount {
>   struct list_head mnt_slave; /* slave list entry */
>   struct vfsmount *mnt_master; /* slave is on master->mnt_slave_list */
>   struct mnt_namespace *mnt_ns; /* containing namespace */
> + struct user_namespace *mnt_user_ns; /* namespace for uid interpretation */
>   int mnt_pinned;
> };
>
> Index: 2.6.19-rc6-mm2/include/linux/sched.h
> ===
> --- 2.6.19-rc6-mm2.orig/include/linux/sched.h
> +++ 2.6.19-rc6-mm2/include/linux/sched.h
> @@ -83,6 +83,8 @@ struct sched_param {
> #include <linux/timer.h>
> #include <linux/hrtimer.h>
> #include <linux/task_io_accounting.h>
> +#include <linux/nsproxy.h>
> +#include <linux/mount.h>
>
> #include <asm/processor.h>
>
> @@ -1589,6 +1591,16 @@ extern int cond_resched(void);
> extern int cond_resched_lock(spinlock_t * lock);
> extern int cond_resched_softirq(void);

>
> +static inline int task_mnt_same_uid(struct task_struct *tsk,
> +   struct vfsmount *mnt)
> +{
> + if (tsk->nsproxy == init_task.nsproxy)
> + return 1;
> + if (mnt->mnt_user_ns == tsk->nsproxy->user_ns)
> + return 1;
> + return 0;
> +}
> +
> /*
>   * Does a critical section need to be broken due to another
>   * task waiting?:
>
> --

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

Subject: Re: [patch -mm 05/17] ipc namespace : externalizes unshare_ipcs
Posted by Herbert Poetzl on Tue, 05 Dec 2006 23:32:11 GMT
View Forum Message <> Reply to Message

On Tue, Dec 05, 2006 at 11:27:57AM +0100, clg@fr.ibm.com wrote:
> From: Cedric Le Goater <clg@fr.ibm.com>
>
> This patch externalizes unshare_ipcs when CONFIG_IPC_NS is not
> set. This is needed by the unshare_ns syscall. Unfortunately, it could
> not be handled with a dummy static inline in ipc.h because of
> header dependencies on sched.h.

hmm, maybe sched.h should be further broken down ...

best,
Herbert

> Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>
>
> ---
> include/linux/ipc.h | 2 +-
> kernel/fork.c | 2 +-
> 2 files changed, 2 insertions(+), 2 deletions(-)
>
> Index: 2.6.19-rc6-mm2/include/linux/ipc.h
> ==========

> --- 2.6.19-rc6-mm2.orig/include/linux/ipc.h
> +++ 2.6.19-rc6-mm2/include/linux/ipc.h
> @@ -99,7 +99,6 @@ extern struct ipc_namespace init_ipc_ns;
> #ifdef CONFIG_IPC_NS
> extern void free_ipc_ns(struct kref *kref);
> extern int copy_ipcs(unsigned long flags, struct task_struct *tsk);
> -extern int unshare_ipcs(unsigned long flags, struct ipc_namespace **ns);
> #else
> static inline int copy_ipcs(unsigned long flags, struct task_struct *tsk)
> {
> @@ -107,6 +106,7 @@ static inline int copy_ipcs(unsigned lon
> }
> #endif
>
> +extern int unshare_ipcs(unsigned long flags, struct ipc_namespace **ns);
> static inline struct ipc_namespace *get_ipc_ns(struct ipc_namespace *ns)
> {
> #ifdef CONFIG_IPC_NS
> Index: 2.6.19-rc6-mm2/kernel/fork.c
> ===
> --- 2.6.19-rc6-mm2.orig/kernel/fork.c
> +++ 2.6.19-rc6-mm2/kernel/fork.c
> @@ -1591,7 +1591,7 @@ static int unshare_semundo(unsigned long
> }
>
> #ifndef CONFIG_IPC_NS
> -static inline int unshare_ipcs(unsigned long flags, struct ipc_namespace **ns)
> +extern int unshare_ipcs(unsigned long flags, struct ipc_namespace **ns)
> {
>   if (flags & CLONE_NEWIPC)
>   return -EINVAL;
>
> --
> ____
> Containers mailing list
> Containers@lists.osdl.org
> https://lists.osdl.org/mailman/listinfo/containers

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

Subject: Re: [patch -mm 08/17] nsproxy: add hashtable
Posted by Herbert Poetzl on Tue, 05 Dec 2006 23:36:03 GMT
View Forum Message <> Reply to Message

On Tue, Dec 05, 2006 at 11:28:00AM +0100, clg@fr.ibm.com wrote:

> From: Cedric Le Goater <clg@fr.ibm.com>
>
> This patch adds a hashtable of nsproxy using the nsproxy as a key.
> init_nsproxy is hashed at init with key 0. This is considered to be
> the 'host' nsproxy.

hmm? how is that going to work?

I mean, a simple clone() call (on the host) will
change the nsproxy and spawn new ones, but that
doesn't mean that the process has left the 'host'
it just means that some namespace is not shared
with the 'other' processes ...

best,
Herbert

> Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>
> ---
> include/linux/nsproxy.h | 3 +++
> kernel/nsproxy.c | 43 ++
> 2 files changed, 46 insertions(+)
>
> Index: 2.6.19-rc6-mm2/include/linux/nsproxy.h
> ===
> --- 2.6.19-rc6-mm2.orig/include/linux/nsproxy.h
> +++ 2.6.19-rc6-mm2/include/linux/nsproxy.h
> @@ -2,6 +2,7 @@
> #define _LINUX_NSPROXY_H
>
> #include <linux/spinlock.h>
> +#include <linux/list.h>
>
> struct task_struct;
>
> @@ -34,6 +35,8 @@ struct nsproxy {
>   struct pid_namespace *pid_ns;
>   struct net_namespace *net_ns;
>   struct user_namespace *user_ns;
> +
> + struct hlist_node ns_hash_node;
> };
> extern struct nsproxy init_nsproxy;
>
> Index: 2.6.19-rc6-mm2/kernel/nsproxy.c
> ===
> --- 2.6.19-rc6-mm2.orig/kernel/nsproxy.c
> +++ 2.6.19-rc6-mm2/kernel/nsproxy.c

```
> @@ -22,6 +22,15 @@
>  #include <linux/pid_namespace.h>
>  #include <linux/net_namespace.h>
>
> +#define NS_HASH_BITS   3 /* this might need some configuration */
> +#define NS_HASH_SIZE  (1 << NS_HASH_BITS)
> +#define NS_HASH_MASK  (NS_HASH_SIZE - 1)
> +#define ns_hashfn(id)  (((id >> NS_HASH_BITS) + id) & NS_HASH_MASK)
> +#define ns_hash_head(id) &ns_hash[ns_hashfn(id)]
> +
> +static struct hlist_head ns_hash[NS_HASH_SIZE];
> +static DEFINE_SPINLOCK(ns_hash_lock);
> +
>  struct nsproxy init_nsproxy = INIT_NSPROXY(init_nsproxy);
>
>  static inline void get_nsproxy(struct nsproxy *ns)
> @@ -190,3 +199,37 @@ void put_nsproxy(struct nsproxy *ns)
>    free_nsproxy(ns);
>   }
>  }
> +
> +/*
> + * This routine must be called with the ns_hash spin_locked
> + */
> +static inline struct nsproxy *ns_hash_find(int id)
> +{
> + struct hlist_node *elem;
> + struct nsproxy *ns;
> +
> + hlist_for_each_entry(ns, elem, ns_hash_head(id), ns_hash_node) {
> +  if (ns->id == id) {
> +   get_nsproxy(ns);
> +   return ns;
> +  }
> + }
> +
> + return NULL;
> +}
> +
> +static int __init nshash_init(void)
> +{
> + int i;
> +
> + for (i = 0; i < NS_HASH_SIZE; ++i)
> +  INIT_HLIST_HEAD(&ns_hash[i]);
> +
> + spin_lock_irq(&ns_hash_lock);
> + hlist_add_head(&init_nsproxy.ns_hash_node, ns_hash_head(0));
```

> + spin_unlock_irq(&ns_hash_lock);
> +
> + return 0;
> +}
> +
> +module_init(nshash_init);
>
> --
> _____
> Containers mailing list
> Containers@lists.osdl.org
> https://lists.osdl.org/mailman/listinfo/containers

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

Subject: Re: [patch -mm 09/17] nsproxy: add namespace flags
Posted by Herbert Poetzl on Tue, 05 Dec 2006 23:41:58 GMT
View Forum Message <> Reply to Message

On Tue, Dec 05, 2006 at 11:28:01AM +0100, clg@fr.ibm.com wrote:
> From: Cedric Le Goater <clg@fr.ibm.com>
>
> Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>
> ---
> include/linux/nsproxy.h | 11 ++++++++++
> 1 file changed, 11 insertions(+)
>
> Index: 2.6.19-rc6-mm2/include/linux/nsproxy.h
> ===
> --- 2.6.19-rc6-mm2.orig/include/linux/nsproxy.h
> +++ 2.6.19-rc6-mm2/include/linux/nsproxy.h
> @@ -14,6 +14,17 @@ struct net_namespace;
> struct user_namespace;
>
> /*
> + * namespaces flags
> + */
> +#define NS_MNT 0x00000001
> +#define NS_UTS 0x00000002
> +#define NS_IPC 0x00000004
> +#define NS_PID 0x00000008
> +#define NS_NET 0x00000010
> +#define NS_USER 0x00000020
> +#define NS_ALL (NS_MNT|NS_UTS|NS_IPC|NS_PID|NS_NET|NS_USER)

hmm, why _another_ set of flags to refer to the
namespaces? is the clone()/unshare() set of flags
not sufficient for that? if so, shouldn't we
switch (or even better change? the unshare() too)
to a new set of syscalls?

note: even if they are just for internal purpose
and will never get exposed to userspace, we should
think twice before we create just another set of
flags, and if we do so, please let us change them
all, including certain clone flags (and add a
single compatibility wrapper for the 'old' syscalls)

best,
Herbert

> +
> +/*
> * A structure to contain pointers to all per-process
> * namespaces - fs (mount), uts, network, sysvipc, etc.
> *
>
> --
> _____
> Containers mailing list
> Containers@lists.osdl.org
> https://lists.osdl.org/mailman/listinfo/containers

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

Subject: Re: [patch -mm 05/17] ipc namespace : externalizes unshare_ipcs
Posted by Cedric Le Goater on Thu, 07 Dec 2006 09:00:16 GMT
View Forum Message <> Reply to Message

Herbert Poetzl wrote:
> On Tue, Dec 05, 2006 at 11:27:57AM +0100, clg@fr.ibm.com wrote:
>> From: Cedric Le Goater <clg@fr.ibm.com>
>>
>> This patch externalizes unshare_ipcs when CONFIG_IPC_NS is not
>> set. This is needed by the unshare_ns syscall. Unfortunately, it could
>> not be handled with a dummy static inline in ipc.h because of
>> header dependencies on sched.h.
>
> hmm, maybe sched.h should be further broken down ...

that will require a few thousands patches :) but yes, sched.h is just
so full of everything that it breaks the kernel build real often.

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

Subject: Re: [patch -mm 08/17] nsproxy: add hashtable
Posted by Cedric Le Goater on Thu, 07 Dec 2006 09:06:03 GMT
View Forum Message <> Reply to Message

>> This patch adds a hashtable of nsproxy using the nsproxy as a key.
>> init_nsproxy is hashed at init with key 0. This is considered to be
>> the 'host' nsproxy.
>
> hmm? how is that going to work?
>
> I mean, a simple clone() call (on the host) will
> change the nsproxy and spawn new ones, but that
> doesn't mean that the process has left the 'host'
> it just means that some namespace is not shared
> with the 'other' processes ...

right. the new nsproxy is built with the new and old
namespaces and the id can be identify it.

nsproxy 0 is init_nsproxy which aggregates all the
initial namespaces.

C.

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

Subject: Re: [patch -mm 09/17] nsproxy: add namespace flags
Posted by Cedric Le Goater on Thu, 07 Dec 2006 09:29:28 GMT
View Forum Message <> Reply to Message

>> /*
>> + * namespaces flags
>> + */
>> +#define NS_MNT 0x00000001
>> +#define NS_UTS 0x00000002
>> +#define NS_IPC 0x00000004

>> +#define NS_PID 0x00000008
>> +#define NS_NET 0x00000010
>> +#define NS_USER 0x00000020
>> +#define NS_ALL (NS_MNT|NS_UTS|NS_IPC|NS_PID|NS_NET|NS_USER)
>
> hmm, why _another_ set of flags to refer to the
> namespaces?

well, because namespaces are a new kind in the kernel

> is the clone()/unshare() set of flags not sufficient
> for that?

because we are reaching the limits of the CLONE_ flags.

> if so, shouldn't we switch (or even better change?
> the unshare() too) to a new set of syscalls?

unshare_ns() is a new syscall and we don't really need a
clone anyway. nop ?

we could make the clone flags and namespace flags compatible
with :

#define NS_MNT CLONE_NEWNS
#define NS_UTS CLONE_NEWUTS
#define NS_IPC CLONE_NEWIPC

that shouldn't be a big issue but we could also
remove/deprecate :

#define CLONE_NEWNS 0x00020000 /* New namespace group? */
#define CLONE_NEWUTS 0x04000000 /* New utsname group? */
#define CLONE_NEWIPC 0x08000000 /* New ipcs */

to make sure namespaces can not be unshared using
unshare()

> note: even if they are just for internal purpose
> and will never get exposed to userspace,

they will. unshare_ns() is a syscall.

> we should think twice before we create just another
> set of flags, and if we do so, please let us change
> them all, including certain clone flags (and add a
> single compatibility wrapper for the 'old' syscalls)

so you would keep the unshare as is but change the set
of flags its using, making sure the old ones are still
compatible with the new ones.

something like this :

```
int sys_unshare(int unshare_flags)
{
 int unshare_ns_flags;

 unshare_ns_flags = convert_flags(unshare_flags);

 return sys_unshare_ns(unshare_ns_flags);
}
```

?

C.

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

Subject: Re: [patch -mm 10/17] nsproxy: add unshare_ns and bind_ns syscalls
Posted by ebiederm on Fri, 08 Dec 2006 19:26:49 GMT
View Forum Message <> Reply to Message

clg@fr.ibm.com writes:

> From: Cedric Le Goater <clg@fr.ibm.com>
>
> The following patch defines 2 new syscalls specific to nsproxy and
> namespaces :
>
> * unshare_ns :
>
> enables a process to unshare one or more namespaces. this
> duplicates the unshare syscall for the moment but we
> expect to diverge when the number of namespaces increases

Are we out of clone flags yet? If not this is premature.

> * bind_ns :
>
> allows a process to bind
> 1 - its nsproxy to some identifier
> 2 - to another nsproxy using an identifier or -pid

NAK

Don't use global identifiers. Use pids. i.e. struct pid * for your
identifiers. Is there is a reason pids are unsuitable?

I'm also worried about the security implications of switching namespaces
on a process. That is something that needs to be looked at very closely.

These two changes certainly don't belong in a single patch, and they
certainly use a bit more explanation. syscalls are not something to add lightly.
Because they must be supported forever.

Eric

Subject: Re: [patch -mm 08/17] nsproxy: add hashtable
Posted by ebiederm on Fri, 08 Dec 2006 19:30:31 GMT
View Forum Message <> Reply to Message

clg@fr.ibm.com writes:

> From: Cedric Le Goater <clg@fr.ibm.com>
>
> This patch adds a hashtable of nsproxy using the nsproxy as a key.
> init_nsproxy is hashed at init with key 0. This is considered to be
> the 'host' nsproxy.

NAK. Which namespace do these ids live in?

It sounds like you are setting up to make the 'host' nsproxy special
and have special rules. That also sounds wrong.

Even letting the concept of nsproxy escape to user space sounds wrong.
nsproxy is an internal space optimization. It's not struct container
and I don't think we want it to become that.

Eric

Subject: Re: [patch -mm 09/17] nsproxy: add namespace flags
Posted by ebiederm on Fri, 08 Dec 2006 19:40:33 GMT
View Forum Message <> Reply to Message

Cedric Le Goater <clg@fr.ibm.com> writes:

>>> /*
>>> + * namespaces flags
>>> + */
>>> +#define NS_MNT 0x00000001
>>> +#define NS_UTS 0x00000002
>>> +#define NS_IPC 0x00000004
>>> +#define NS_PID 0x00000008
>>> +#define NS_NET 0x00000010
>>> +#define NS_USER 0x00000020
>>> +#define NS_ALL (NS_MNT|NS_UTS|NS_IPC|NS_PID|NS_NET|NS_USER)
>>
>> hmm, why _another_ set of flags to refer to the
>> namespaces?
>
> well, because namespaces are a new kind in the kernel

Gratuitous incompatibility.

>> is the clone()/unshare() set of flags not sufficient
>> for that?
>
> because we are reaching the limits of the CLONE_ flags.

Not really. There are at least 8 bits that clone cannot use
but that unshare can.

>> if so, shouldn't we switch (or even better change?
>> the unshare() too) to a new set of syscalls?
>
> unshare_ns() is a new syscall and we don't really need a
> clone anyway. nop ?

Huh? Clone should be the primary. There are certain namespaces
that it are very hard to unshare, without creating a new process.

> we could make the clone flags and namespace flags compatible
> with :
>
> #define NS_MNT CLONE_NEWNS
> #define NS_UTS CLONE_NEWUTS
> #define NS_IPC CLONE_NEWIPC
>
> that shouldn't be a big issue but we could also

> remove/deprecate :
>
> #define CLONE_NEWNS 0x00020000 /* New namespace group? */
> #define CLONE_NEWUTS 0x04000000 /* New utsname group? */
> #define CLONE_NEWIPC 0x08000000 /* New ipcs */
>
> to make sure namespaces can not be unshared using
> unshare()

Nope. We got them now they can not be removed. It's part of the ABI.
And since we are not removing the functionality it makes not sense
at all to just change the name of the flags.

>> we should think twice before we create just another
>> set of flags, and if we do so, please let us change
>> them all, including certain clone flags (and add a
>> single compatibility wrapper for the 'old' syscalls)
>
> so you would keep the unshare as is but change the set
> of flags its using, making sure the old ones are still
> compatible with the new ones.
>
> something like this :
>
> int sys_unshare(int unshare_flags)
> {
> int unshare_ns_flags;
>
> unshare_ns_flags = convert_flags(unshare_flags);
>
> return sys_unshare_ns(unshare_ns_flags);
> }
>
> ?

If necessary.

Eric

Subject: Re: [patch -mm 08/17] nsproxy: add hashtable
Posted by serue on Fri, 08 Dec 2006 19:53:06 GMT
View Forum Message <> Reply to Message

Quoting Eric W. Biederman (ebiederm@xmission.com):
> clg@fr.ibm.com writes:
>
> > From: Cedric Le Goater <clg@fr.ibm.com>
> >
> > This patch adds a hashtable of nsproxy using the nsproxy as a key.
> > init_nsproxy is hashed at init with key 0. This is considered to be
> > the 'host' nsproxy.
>
> NAK. Which namespace do these ids live in?
>
> It sounds like you are setting up to make the 'host' nsproxy special
> and have special rules. That also sounds wrong.
>
> Even letting the concept of nsproxy escape to user space sounds wrong.
> nsproxy is an internal space optimization. It's not struct container
> and I don't think we want it to become that.
>
> Eric

So would you advocate referring to containers just by the pid of a
process containing the nsproxy, and letting userspace maintain a mapping
of id's to containers through container create/enter commands? Or is
there some other way you were thinking of doing this?

thanks,
-serge

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

Subject: Re: [patch -mm 04/17] nsproxy: externalizes exit_task_namespaces
Posted by ebiederm on Fri, 08 Dec 2006 20:16:22 GMT
View Forum Message <> Reply to Message

clg@fr.ibm.com writes:

> From: Cedric Le Goater <clg@fr.ibm.com>
>
> this is required to remove a header dependency in sched.h which breaks
> next patches.

This just doesn't feel right.

Why with unshare working now are you needing to rework everything?

Eric

_____

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

Subject: Re: [patch -mm 08/17] nsproxy: add hashtable
Posted by ebiederm on Fri, 08 Dec 2006 20:57:38 GMT
View Forum Message <> Reply to Message

"Serge E. Hallyn" <serue@us.ibm.com> writes:

> Quoting Eric W. Biederman (ebiederm@xmission.com):
>> clg@fr.ibm.com writes:
>>
>> > From: Cedric Le Goater <clg@fr.ibm.com>
>> >
>> > This patch adds a hashtable of nsproxy using the nsproxy as a key.
>> > init_nsproxy is hashed at init with key 0. This is considered to be
>> > the 'host' nsproxy.
>>
>> NAK. Which namespace do these ids live in?
>>
>> It sounds like you are setting up to make the 'host' nsproxy special
>> and have special rules. That also sounds wrong.
>>
>> Even letting the concept of nsproxy escape to user space sounds wrong.
>> nsproxy is an internal space optimization. It's not struct container
>> and I don't think we want it to become that.
>>
>> Eric
>
> So would you advocate referring to containers just by the pid of a
> process containing the nsproxy, and letting userspace maintain a mapping
> of id's to containers through container create/enter commands? Or is
> there some other way you were thinking of doing this?

There are two possible ways.
1) Just use a process using the namespace.
   This is easiest to implement.

2) Have a struct pid reference in the namespace itself, and probably
   an extra pointer in struct pid to find it.
   This is the most stable, because fork/exit won't affect which pid you need
   to use.

Beyond that yes it seems to make sense to let user space maintain any mapping

of containers to ids.

Eric

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

Subject: Re: [patch -mm 08/17] nsproxy: add hashtable
Posted by Herbert Poetzl on Sat, 09 Dec 2006 04:11:10 GMT
View Forum Message <> Reply to Message

On Fri, Dec 08, 2006 at 01:57:38PM -0700, Eric W. Biederman wrote:
> "Serge E. Hallyn" <serue@us.ibm.com> writes:
>
> > Quoting Eric W. Biederman (ebiederm@xmission.com):
> >> clg@fr.ibm.com writes:
> >>
> >> > From: Cedric Le Goater <clg@fr.ibm.com>
> >> >
> >> > This patch adds a hashtable of nsproxy using the nsproxy as a key.
> >> > init_nsproxy is hashed at init with key 0. This is considered to be
> >> > the 'host' nsproxy.
> >>
> >> NAK. Which namespace do these ids live in?

well, I gave a similar answer in another email,
so I fully agree with the NAK here ...

> >> It sounds like you are setting up to make the 'host' nsproxy
> >> special and have special rules. That also sounds wrong.
> >>
> >> Even letting the concept of nsproxy escape to user space sounds
> >> wrong. nsproxy is an internal space optimization. It's not struct
> >> container and I don't think we want it to become that.
> >>
> >> Eric
> >
> > So would you advocate referring to containers just by the pid of
> > a process containing the nsproxy, and letting userspace maintain
> > a mapping of id's to containers through container create/enter
> > commands? Or is there some other way you were thinking of doing
> > this?

> There are two possible ways.
> 1) Just use a process using the namespace.
>   This is easiest to implement.

> 2) Have a struct pid reference in the namespace itself,
> and probably an extra pointer in struct pid to find it.
> This is the most stable, because fork/exit won't affect
> which pid you need to use.

while I agree that nsproxy is definitely the wrong
point to tie a 'context' too, as it can contain a
mixture of spaces from inside and outside a context,
and it would require to forbid doing things like
clone() with the space flags, both inside and outside
a 'container' to allow to use them for actual vps
applications, I think that we have to have some kind
of handle to tie specific sets of namespaces too

that 'can' be an nsproxy or something different, but
I'm absolutely unhappy with tying it to a process,
as I already mentioned several times, that lightweight
'containers' do not use/have an init process, and no
single process might survive the entire life span of
that 'container' ...

> Beyond that yes it seems to make sense to let user space
> maintain any mapping of containers to ids.

I agree with that, but we need something to move
around between the various spaces ...

for example, Linux-VServer ties the namespaces to
the context structure (atm) which allows userspace
to set and enter specific spaces of a guest context
(I assume OpenVZ does similar)

HTC,
Herbert

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

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

Subject: Re: [patch -mm 10/17] nsproxy: add unshare_ns and bind_ns syscalls
Posted by Herbert Poetzl on Sat, 09 Dec 2006 04:18:14 GMT
View Forum Message <> Reply to Message

On Fri, Dec 08, 2006 at 12:26:49PM -0700, Eric W. Biederman wrote:
> clg@fr.ibm.com writes:
>
> > From: Cedric Le Goater <clg@fr.ibm.com>
> >
> > The following patch defines 2 new syscalls specific to nsproxy and
> > namespaces :
> >
> > * unshare_ns :
> >
> > enables a process to unshare one or more namespaces. this
> > duplicates the unshare syscall for the moment but we
> > expect to diverge when the number of namespaces increases
>
> Are we out of clone flags yet? If not this is premature.

no, but a different nevertheless related question:
does anybody, except for 'us' use the unshare() syscall?

because if not, then why not simply extend that one
to 64bit and be done, we probably won't need a clone64()
but if we find we do (at some point) adding that with
the new flags would be trivial ...

OTOH, we could also just add an unshare64() too

anyway, we _will_ run out of flags in the near future

> > * bind_ns :
> >
> > allows a process to bind
> > 1 - its nsproxy to some identifier
> > 2 - to another nsproxy using an identifier or -pid
>
> NAK
>
> Don't use global identifiers. Use pids. i.e. struct pid * for your
> identifiers. Is there is a reason pids are unsuitable?

yes, see my reply in the other mail

> I'm also worried about the security implications of switching
> namespaces on a process.
> That is something that needs to be looked at very closely.

Linux-VServer currently uses a capability to prevent
changing between namespaces (a very generic one) but
it probably makes sense to add something like that
in general ... btw, did I mention that the capability
flags are running out too?

> These two changes certainly don't belong in a single patch,
> and they certainly use a bit more explanation.
> syscalls are not something to add lightly.

well, and they will take ages to get into mainline
for all archs, or has that changed sine we reserved
sys_vserver()?

> Because they must be supported forever.

I'm not sure about that, most archs 'reuse' syscalls
when there is no user left ...

best,
Herbert

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

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

Subject: Re: [patch -mm 10/17] nsproxy: add unshare_ns and bind_ns syscalls
Posted by ebiederm on Sat, 09 Dec 2006 07:40:03 GMT
View Forum Message <> Reply to Message

Herbert Poetzl <herbert@13thfloor.at> writes:

> On Fri, Dec 08, 2006 at 12:26:49PM -0700, Eric W. Biederman wrote:
>> clg@fr.ibm.com writes:
>>
>> > From: Cedric Le Goater <clg@fr.ibm.com>
>> >
>> > The following patch defines 2 new syscalls specific to nsproxy and
>> > namespaces :
>> >
>> > * unshare_ns :

>> >
>> > enables a process to unshare one or more namespaces. this
>> >   duplicates the unshare syscall for the moment but we
>> > expect to diverge when the number of namespaces increases
>>
>> Are we out of clone flags yet? If not this is premature.
>
> no, but a different nevertheless related question:
> does anybody, except for 'us' use the unshare() syscall?

The pam_namespace module if I have looked at things properly. I
believe that is what it was added to support.

> because if not, then why not simply extend that one
> to 64bit and be done, we probably won't need a clone64()
> but if we find we do (at some point) adding that with
> the new flags would be trivial ...
>
> OTOH, we could also just add an unshare64() too
>
> anyway, we _will_ run out of flags in the near future

Agreed. Please let's cross that bridge when we come to it.

>> I'm also worried about the security implications of switching
>> namespaces on a process.
>> That is something that needs to be looked at very closely.
>
> Linux-VServer currently uses a capability to prevent
> changing between namespaces (a very generic one) but
> it probably makes sense to add something like that
> in general ... btw, did I mention that the capability
> flags are running out too?

I think they have run out. Not that sys_capability needs a revision
but it appears the format of the data does which is likely just
as bad.

>> These two changes certainly don't belong in a single patch,
>> and they certainly use a bit more explanation.
>> syscalls are not something to add lightly.
>
> well, and they will take ages to get into mainline
> for all archs, or has that changed sine we reserved
> sys_vserver()?

I think it is likely a little better. I'm not certain what
your definition of ages is.

>> Because they must be supported forever.
>
> I'm not sure about that, most archs 'reuse' syscalls
> when there is no user left ...

I haven't seen that on i386. Except for experimental
syscalls I have a hard time believing we have any syscalls
that have had all of their users disappear.

Reusing syscall numbers is in a lot of ways completely irresponsible
once you start supporting a binary interface. Even if you remove
the syscall because there are no users or it makes absolutely no
sense any more.

Eric

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

Subject: Re: [patch -mm 08/17] nsproxy: add hashtable
Posted by ebiederm on Sat, 09 Dec 2006 07:54:26 GMT
View Forum Message <> Reply to Message

Herbert Poetzl <herbert@13thfloor.at> writes:

>> There are two possible ways.
>> 1) Just use a process using the namespace.
>> This is easiest to implement.
>
>> 2) Have a struct pid reference in the namespace itself,
>> and probably an extra pointer in struct pid to find it.
>> This is the most stable, because fork/exit won't affect
>> which pid you need to use.
>
> that 'can' be an nsproxy or something different, but
> I'm absolutely unhappy with tying it to a process,
> as I already mentioned several times, that lightweight
> 'containers' do not use/have an init process, and no
> single process might survive the entire life span of
> that 'container' ...

Herbert think of a session id. That is a pid that is
tied to something besides a single process.

It is easy and recursion safe to tie a pid to a namespace

or anything else that make sense, as I suggested above.

The pid namespace feels like the right place for this kind
of activity.

>> Beyond that yes it seems to make sense to let user space
>> maintain any mapping of containers to ids.
>
> I agree with that, but we need something to move
> around between the various spaces ...

If you have CAP_SYS_PTRACE or you have a child process
in a container you can create another with ptrace.

Now I don't mind optimizing that case, with something like
the proposed bind_ns syscall. But we need to be darn certain
why it is safe, and does not change the security model that
we currently have.

I have not seen that discussion yet, and until I do I have
serious concerns. That discussion needs to be on lkml as
well. Why did Al Viro think this was a bad idea when it
was proposed for the mount namespace?

This is where you are on the edge of some very weird interface
interactions. Without suid programs it would be completely safe
for anyone to unshare their mount namespace. With suid programs
allowed an unprivileged unshare mount namespace unshare is next to
impossible.

> for example, Linux-VServer ties the namespaces to
> the context structure (atm) which allows userspace
> to set and enter specific spaces of a guest context
> (I assume OpenVZ does similar)

Yep, and we certainly need to find a way to fulfill this usage
requirement.

Eric

_____

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

Subject: Re: [patch -mm 10/17] nsproxy: add unshare_ns and bind_ns syscalls
Posted by Herbert Poetzl on Sat, 09 Dec 2006 08:22:02 GMT

On Sat, Dec 09, 2006 at 12:40:03AM -0700, Eric W. Biederman wrote:
> Herbert Poetzl <herbert@13thfloor.at> writes:
>
> > On Fri, Dec 08, 2006 at 12:26:49PM -0700, Eric W. Biederman wrote:
> >> clg@fr.ibm.com writes:
> >>
> >> > From: Cedric Le Goater <clg@fr.ibm.com>
> >> >
> >> > The following patch defines 2 new syscalls specific to nsproxy and
> >> > namespaces :
> >> >
> >> > * unshare_ns :
> >> >
> >> > enables a process to unshare one or more namespaces. this
> >> >   duplicates the unshare syscall for the moment but we
> >> > expect to diverge when the number of namespaces increases
> >>
> >> Are we out of clone flags yet? If not this is premature.
> >
> > no, but a different nevertheless related question:
> > does anybody, except for 'us' use the unshare() syscall?
>
> The pam_namespace module if I have looked at things properly.
> I believe that is what it was added to support.

hmm, too bad then ...

> > because if not, then why not simply extend that one
> > to 64bit and be done, we probably won't need a clone64()
> > but if we find we do (at some point) adding that with
> > the new flags would be trivial ...
> >
> > OTOH, we could also just add an unshare64() too
> >
> > anyway, we _will_ run out of flags in the near future
>
> Agreed. Please let's cross that bridge when we come to it.

well, personally I'd prefer to 'know' that the
interfaces we introduce now to handle the 'new'
namespaces will still work in a few months (or
maybe years?) and not require _another_ change

let me give an example here:

we now change our userspace tools to support
new clone() flags, we can do that with a little

support from the kernel quite easily without
changing the tools with every kernel release and
more important, without breaking backwards
compatibility in the Linux-VServer ABI (and API)

if we decide to switch to a completely different
API in, lets say six month from now, 'we' have
two options:

 - add _another_ compatibility layer to handle
   the 'old' (i.e. now introduced) ABI
 - accept that older tools will fail and/or
   produce strange results because 'our' ABI
   isn't supported anymore ...

so naturally, I'm not very excited to introduce
and/or utilize and interface which we all _know_
is not able to satisfy the upcoming demands ...

nevertheless, that isn't considered a major issue
here, so feel free to ignore my personal feelings

> >> I'm also worried about the security implications of switching
> >> namespaces on a process.
> >> That is something that needs to be looked at very closely.
> >
> > Linux-VServer currently uses a capability to prevent
> > changing between namespaces (a very generic one) but
> > it probably makes sense to add something like that
> > in general ... btw, did I mention that the capability
> > flags are running out too?
>
> I think they have run out. Not that sys_capability needs a
> revision but it appears the format of the data does which
> is likely just as bad.

gladly not in 2.6.19, as we absolutely need to
add one capability (CAP_CONTEXT) in Linux-VServer
(the one allowing to utilize sys_vserver)

but once that is taken by mainline, we have to
add another workaround ...

sidenote: I made several suggestions to extend
the capability system (in number of caps and
functionality) of which none was even considered

> >> These two changes certainly don't belong in a single patch,

> >> and they certainly use a bit more explanation.
> >> syscalls are not something to add lightly.
> >
> > well, and they will take ages to get into mainline
> > for all archs, or has that changed sine we reserved
> > sys_vserver()?
>
> I think it is likely a little better. I'm not certain what
> your definition of ages is.

took slightly more than a year, IIRC ...

> >> Because they must be supported forever.
> >
> > I'm not sure about that, most archs 'reuse' syscalls
> > when there is no user left ...
>
> I haven't seen that on i386. Except for experimental
> syscalls I have a hard time believing we have any syscalls
> that have had all of their users disappear.

okay, so be it ...

> Reusing syscall numbers is in a lot of ways completely
> irresponsible once you start supporting a binary interface.
> Even if you remove the syscall because there are no users
> or it makes absolutely no sense any more.

okay, no problem with keeping them around ...

best,
Herbert

> Eric

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

Subject: Re: [patch -mm 10/17] nsproxy: add unshare_ns and bind_ns syscalls
Posted by Cedric Le Goater on Mon, 11 Dec 2006 15:21:05 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> clg@fr.ibm.com writes:
>
>> From: Cedric Le Goater <clg@fr.ibm.com>

>>
>> The following patch defines 2 new syscalls specific to nsproxy and
>> namespaces :
>>
>> * unshare_ns :
>>
>> enables a process to unshare one or more namespaces. this
>>   duplicates the unshare syscall for the moment but we
>> expect to diverge when the number of namespaces increases
>
> Are we out of clone flags yet? If not this is premature.
>
>> * bind_ns :
>>
>> allows a process to bind
>> 1 - its nsproxy to some identifier
>> 2 - to another nsproxy using an identifier or -pid
>
> NAK
>
> Don't use global identifiers. Use pids. i.e. struct pid * for your
> identifiers. Is there is a reason pids are unsuitable?

(1) gives a little more freedom to the sysadmin managing its
(2) uses pids. do you also nak it ?

do you always have access to pid ?

> I'm also worried about the security implications of switching namespaces
> on a process.   That is something that needs to be looked at very closely.

this is required by at least 3 products I know of.

> These two changes certainly don't belong in a single patch, and they
> certainly use a bit more explanation. syscalls are not something to
> add lightly. Because they must be supported forever.

agree.

c.

Subject: Re: [patch -mm 08/17] nsproxy: add hashtable
Posted by Cedric Le Goater on Mon, 11 Dec 2006 15:23:29 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> clg@fr.ibm.com writes:
>
>> From: Cedric Le Goater <clg@fr.ibm.com>
>>
>> This patch adds a hashtable of nsproxy using the nsproxy as a key.
>> init_nsproxy is hashed at init with key 0. This is considered to be
>> the 'host' nsproxy.
>
> NAK. Which namespace do these ids live in?
>
> It sounds like you are setting up to make the 'host' nsproxy special
> and have special rules.

exactly.

> That also sounds wrong.

sounds very nice to me and a few others.

> Even letting the concept of nsproxy escape to user space sounds wrong.
> nsproxy is an internal space optimization. It's not struct container
> and I don't think we want it to become that.

i don't agree here. we need that, so does openvz, vserver, people working
on resource management.

C.

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

Subject: Re: [patch -mm 04/17] nsproxy: externalizes exit_task_namespaces
Posted by Cedric Le Goater on Mon, 11 Dec 2006 15:26:57 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> clg@fr.ibm.com writes:
>
>> From: Cedric Le Goater <clg@fr.ibm.com>
>>
>> this is required to remove a header dependency in sched.h which breaks

>> next patches.
>
> This just doesn't feel right.
>
> Why with unshare working now are you needing to rework everything?

This is not everything. this is just 3 lines compared to millions.

The issue is not here anyway.

c.

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

Subject: Re: [patch -mm 09/17] nsproxy: add namespace flags
Posted by Cedric Le Goater on Mon, 11 Dec 2006 15:27:16 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Cedric Le Goater <clg@fr.ibm.com> writes:
>
>>>> /*
>>>> + * namespaces flags
>>>> + */
>>>> +#define NS_MNT 0x00000001
>>>> +#define NS_UTS 0x00000002
>>>> +#define NS_IPC 0x00000004
>>>> +#define NS_PID 0x00000008
>>>> +#define NS_NET 0x00000010
>>>> +#define NS_USER 0x00000020
>>>> +#define NS_ALL (NS_MNT|NS_UTS|NS_IPC|NS_PID|NS_NET|NS_USER)
>>> hmm, why _another_ set of flags to refer to the
>>> namespaces?
>> well, because namespaces are a new kind in the kernel
>
> Gratuitous incompatibility.

?

>>> is the clone()/unshare() set of flags not sufficient
>>> for that?
>> because we are reaching the limits of the CLONE_ flags.
>
> Not really. There are at least 8 bits that clone cannot use
> but that unshare can.

please, could you list them ?

>>> if so, shouldn't we switch (or even better change?
>>> the unshare() too) to a new set of syscalls?
>> unshare_ns() is a new syscall and we don't really need a
>> clone anyway. nop ?
>
> Huh? Clone should be the primary.   There are certain namespaces
> that it are very hard to unshare, without creating a new process.

You just said above that clone had less available flags than
unshare ...

anyway, could you elaborate a bit more ? I have the opposite
feeling and you gave me that impression also a few month ago.

No problem for me, i just want a way to use this stuff without

>>> we should think twice before we create just another
>>> set of flags, and if we do so, please let us change
>>> them all, including certain clone flags (and add a
>>> single compatibility wrapper for the 'old' syscalls)
>> so you would keep the unshare as is but change the set
>> of flags its using, making sure the old ones are still
>> compatible with the new ones.
>>
>> something like this :
>>
>> int sys_unshare(int unshare_flags)
>> {
>> int unshare_ns_flags;
>>
>> unshare_ns_flags = convert_flags(unshare_flags);
>>
>> return sys_unshare_ns(unshare_ns_flags);
>> }
>>
>> ?
>
> If necessary.

ok good. will check it out.

C.

_____
Containers mailing list

Subject: Re: [patch -mm 08/17] nsproxy: add hashtable
Posted by serue on Mon, 11 Dec 2006 15:29:35 GMT
View Forum Message <> Reply to Message

Quoting Eric W. Biederman (ebiederm@xmission.com):
> Herbert Poetzl <herbert@13thfloor.at> writes:
>
> >> There are two possible ways.
> >> 1) Just use a process using the namespace.
> >>   This is easiest to implement.
> >
> >> 2) Have a struct pid reference in the namespace itself,
> >>   and probably an extra pointer in struct pid to find it.
> >>   This is the most stable, because fork/exit won't affect
> >>   which pid you need to use.
> >
> > that 'can' be an nsproxy or something different, but
> > I'm absolutely unhappy with tying it to a process,
> > as I already mentioned several times, that lightweight
> > 'containers' do not use/have an init process, and no
> > single process might survive the entire life span of
> > that 'container' ...
>
> Herbert think of a session id. That is a pid that is
> tied to something besides a single process.
>
> It is easy and recursion safe to tie a pid to a namespace
> or anything else that make sense, as I suggested above.

Recursion safe, but limiting in that you can only descend one pid
namespace at a time. That limitation aside, providing task
notifiers for all unshares, plus a syscall to jump into all
namespaces belonging to a process known to you by a particular
pid, could be a good approach. Now you can have a userspace
daemon keeping namespace id's tied to processes, giving you the
ability to say

```
 ns_exec -a -l ns12 my_prog
  (unshare all namespaces and run my_prog in
  a container known as 'ns12')
 ns_enter -l ns12 /bin/ps
  (jump into ns12 and run /bin/ps)
```

The likely requirement to run a namespace tracking daemon in

each pid namespace that wants such functionality could become
a resource hog, but that may be just a theoretical problem, since
you'll only need that if you want to play with namespaces,
meaning that for it to be a problem you'd have to have lots of
virtual servers each maintaining namespaces to either do
process migration or spawn more virtual servers (which each
maintain namespaces to...)

> The pid namespace feels like the right place for this kind
> of activity.
>
> >> Beyond that yes it seems to make sense to let user space
> >> maintain any mapping of containers to ids.
> >
> > I agree with that, but we need something to move
> > around between the various spaces ...
>
> If you have CAP_SYS_PTRACE or you have a child process
> in a container you can create another with ptrace.
>
> Now I don't mind optimizing that case, with something like
> the proposed bind_ns syscall. But we need to be darn certain
> why it is safe, and does not change the security model that
> we currently have.

Sigh, and that's going to have to be a discussion per namespace.

> I have not seen that discussion yet, and until I do I have
> serious concerns. That discussion needs to be on lkml as
> well. Why did Al Viro think this was a bad idea when it
> was proposed for the mount namespace?
>
> This is where you are on the edge of some very weird interface
> interactions. Without suid programs it would be completely safe
> for anyone to unshare their mount namespace. With suid programs
> allowed an unprivileged unshare mount namespace unshare is next to
> impossible.
>
> > for example, Linux-VServer ties the namespaces to
> > the context structure (atm) which allows userspace
> > to set and enter specific spaces of a guest context
> > (I assume OpenVZ does similar)
>
> Yep, and we certainly need to find a way to fulfill this usage
> requirement.

_____

Containers mailing list
Containers@lists.osdl.org

Subject: Re: [patch -mm 08/17] nsproxy: add hashtable
Posted by serue on Mon, 11 Dec 2006 15:56:22 GMT
View Forum Message <> Reply to Message

Quoting Serge E. Hallyn (serue@us.ibm.com):
> Quoting Eric W. Biederman (ebiederm@xmission.com):
> > Herbert Poetzl <herbert@13thfloor.at> writes:
> > >> Beyond that yes it seems to make sense to let user space
> > >> maintain any mapping of containers to ids.
> > >
> > > I agree with that, but we need something to move
> > > around between the various spaces ...
> >
> > If you have CAP_SYS_PTRACE or you have a child process
> > in a container you can create another with ptrace.
> >
> > Now I don't mind optimizing that case, with something like
> > the proposed bind_ns syscall. But we need to be darn certain
> > why it is safe, and does not change the security model that
> > we currently have.
>
> Sigh, and that's going to have to be a discussion per namespace.

Well, assuming that we're using pids as identifiers, that means
we can only enter decendent namespaces, which means 'we' must
have created them. So anything we could do by entering the ns,
we could have done by creating it as well, right?

-serge

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

Subject: Re: [patch -mm 08/17] nsproxy: add hashtable
Posted by Cedric Le Goater on Mon, 11 Dec 2006 16:09:33 GMT
View Forum Message <> Reply to Message

Herbert Poetzl wrote:
> On Fri, Dec 08, 2006 at 01:57:38PM -0700, Eric W. Biederman wrote:
>> "Serge E. Hallyn" <serue@us.ibm.com> writes:
>>
>>> Quoting Eric W. Biederman (ebiederm@xmission.com):

>>>> clg@fr.ibm.com writes:
>>>>
>>>>> From: Cedric Le Goater <clg@fr.ibm.com>
>>>>>
>>>>> This patch adds a hashtable of nsproxy using the nsproxy as a key.
>>>>> init_nsproxy is hashed at init with key 0. This is considered to be
>>>>> the 'host' nsproxy.
>>>> NAK. Which namespace do these ids live in?
>
> well, I gave a similar answer in another email,
> so I fully agree with the NAK here ...

hmm, I wasn't that clear to me. OK, let's dig :)

>>>> It sounds like you are setting up to make the 'host' nsproxy
>>>> special and have special rules. That also sounds wrong.
>>>>
>>>> Even letting the concept of nsproxy escape to user space sounds
>>>> wrong. nsproxy is an internal space optimization. It's not struct
>>>> container and I don't think we want it to become that.
>>>>
>>>> Eric
>>> So would you advocate referring to containers just by the pid of
>>> a process containing the nsproxy, and letting userspace maintain
>>> a mapping of id's to containers through container create/enter
>>> commands? Or is there some other way you were thinking of doing
>>> this?
>
>> There are two possible ways.
>> 1) Just use a process using the namespace.
>>   This is easiest to implement.
>
>> 2) Have a struct pid reference in the namespace itself,
>>   and probably an extra pointer in struct pid to find it.
>>   This is the most stable, because fork/exit won't affect
>>   which pid you need to use.
>
> while I agree that nsproxy is definitely the wrong
> point to tie a 'context' too, as it can contain a
> mixture of spaces from inside and outside a context,
> and it would require to forbid doing things like
> clone() with the space flags, both inside and outside
> a 'container' to allow to use them for actual vps
> applications, I think that we have to have some kind
> of handle to tie specific sets of namespaces too

this is nsproxy ...

> that 'can' be an nsproxy or something different, but
> I'm absolutely unhappy with tying it to a process,

hmm, what do you mean ? nsproxy survives the death of any
process. It's not tied to any process in particular. One
process creates it with an unshare but that's all.

the ->nsproxy in task_struct is a way to find it.

> as I already mentioned several times, that lightweight
> 'containers' do not use/have an init process, and no
> single process might survive the entire life span of
> that 'container' ...

I think there is a misunderstanding here. a 'container'
or 'nsproxy' or what ever is a set of namespaces which
are not tied to a process.

you can do that today on 2.6.19 with utsname.

>> Beyond that yes it seems to make sense to let user space
>> maintain any mapping of containers to ids.
>
> I agree with that, but we need something to move
> around between the various spaces ...

the bind_ns syscall lets the user specify the mapping. this
is not done by the kernel.

I had to introduce some rules, like giving more capabilities to
some processes, but that can be changed. For the
moment, they have to live in "init_proxy".

> for example, Linux-VServer ties the namespaces to
> the context structure (atm) which allows userspace
> to set and enter specific spaces of a guest context
> (I assume OpenVZ does similar)

What's the big difference with nsproxy ?

C.

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

Subject: Re: [patch -mm 10/17] nsproxy: add unshare_ns and bind_ns syscalls
Posted by Cedric Le Goater on Mon, 11 Dec 2006 17:05:16 GMT
View Forum Message <> Reply to Message

Herbert Poetzl wrote:
> On Fri, Dec 08, 2006 at 12:26:49PM -0700, Eric W. Biederman wrote:
>> clg@fr.ibm.com writes:
>>
>>> From: Cedric Le Goater <clg@fr.ibm.com>
>>>
>>> The following patch defines 2 new syscalls specific to nsproxy and
>>> namespaces :
>>>
>>> * unshare_ns :
>>>
>>> enables a process to unshare one or more namespaces. this
>>> duplicates the unshare syscall for the moment but we
>>> expect to diverge when the number of namespaces increases
>> Are we out of clone flags yet? If not this is premature.
>
> no, but a different nevertheless related question:
> does anybody, except for 'us' use the unshare() syscall?
>
> because if not, then why not simply extend that one
> to 64bit and be done, we probably won't need a clone64()
> but if we find we do (at some point) adding that with
> the new flags would be trivial ...
>
> OTOH, we could also just add an unshare64() too
>
> anyway, we _will_ run out of flags in the near future

yes. that's probably the way to go. I'll rework unshare_ns() in a
unshare64(). it will give some air to the 32bits clone() and unshare() and
will let us use the >32bits flags for namespaces.

thanks,

C.

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

Subject: Re: [patch -mm 08/17] nsproxy: add hashtable
Posted by ebiederm on Mon, 11 Dec 2006 19:35:12 GMT
View Forum Message <> Reply to Message

"Serge E. Hallyn" <serue@us.ibm.com> writes:

> Quoting Serge E. Hallyn (serue@us.ibm.com):
>> Quoting Eric W. Biederman (ebiederm@xmission.com):
>> > Herbert Poetzl <herbert@13thfloor.at> writes:
>> > >> Beyond that yes it seems to make sense to let user space
>> > >> maintain any mapping of containers to ids.
>> > >
>> > > I agree with that, but we need something to move
>> > > around between the various spaces ...
>> >
>> > If you have CAP_SYS_PTRACE or you have a child process
>> > in a container you can create another with ptrace.
>> >
>> > Now I don't mind optimizing that case, with something like
>> > the proposed bind_ns syscall. But we need to be darn certain
>> > why it is safe, and does not change the security model that
>> > we currently have.
>>
>> Sigh, and that's going to have to be a discussion per namespace.
>
> Well, assuming that we're using pids as identifiers, that means
> we can only enter decendent namespaces, which means 'we' must
> have created them. So anything we could do by entering the ns,
> we could have done by creating it as well, right?

It isn't strict descendents who we can see. i.e. init can create
the thing, and we could have just logged into the network but init
and us still share the same pid namespace.

But yes it would be we can only enter descendent namespaces, for
some definition of enter.

There are two issues.
1) We may have a namespace we want to create and then remove the ability
   for the sysadmin to fiddle with, so it can play with encrypted data or
   something like that safely. Not quite unix but it is certainly worth
   considering.

2) When we only partially enter a namespace it is very easy for additional
   properties to enter that namespace. For example we enter the pid
   namespace and the mount namespace, but keep our current working directory
   in the previous namespace. Then a process in the restricted namespace
   can get out by cd into /proc/<?>/cwd.

If someones permissions to various objects does not depend on the namespace
they are in quite possibly this is a non-issue. If we actually depend on
the isolation to keep things secure enter is a setup for a first rate escape.

Eric

_____

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

Subject: Re: [patch -mm 09/17] nsproxy: add namespace flags
Posted by ebiederm on Mon, 11 Dec 2006 20:02:15 GMT
View Forum Message <> Reply to Message

Cedric Le Goater <clg@fr.ibm.com> writes:

> Eric W. Biederman wrote:
>> Cedric Le Goater <clg@fr.ibm.com> writes:
>>
>>>>> /*
>>>>> + * namespaces flags
>>>>> + */
>>>>> +#define NS_MNT 0x00000001
>>>>> +#define NS_UTS 0x00000002
>>>>> +#define NS_IPC 0x00000004
>>>>> +#define NS_PID 0x00000008
>>>>> +#define NS_NET 0x00000010
>>>>> +#define NS_USER 0x00000020
>>>>> +#define NS_ALL (NS_MNT|NS_UTS|NS_IPC|NS_PID|NS_NET|NS_USER)
>>>> hmm, why _another_ set of flags to refer to the
>>>> namespaces?
>>> well, because namespaces are a new kind in the kernel
>>
>> Gratuitous incompatibility.
>
> ?

Changing the numbers for no good reason. We can easily keep the existing numbers.

>>>> is the clone()/unshare() set of flags not sufficient
>>>> for that?
>>> because we are reaching the limits of the CLONE_ flags.
>>
>> Not really. There are at least 8 bits that clone cannot use
>> but that unshare can.
>
> please, could you list them ?

CSIGNAL. There are a several others as well that will never mean anything
in an unshare context:

I believe the 10 flags below are also nonsense from an unshare perspective.
CLONE_PTRACE 0x00002000
CLONE_VFORK 0x00004000
CLONE_PARENT 0x00008000
CLONE_SETTLS 0x00080000
CLONE_PARENT_SETTID 0x00100000
CLONE_CHILD_CLEARTID 0x00200000
CLONE_DETACHED 0x00400000
CLONE_UNTRACED 0x00800000
CLONE_CHILD_SETTID 0x01000000
CLONE_STOPPED 0x02000000

>>>> if so, shouldn't we switch (or even better change?
>>>> the unshare() too) to a new set of syscalls?
>>> unshare_ns() is a new syscall and we don't really need a
>>> clone anyway. nop ?
>>
>> Huh? Clone should be the primary. There are certain namespaces
>> that it are very hard to unshare, without creating a new process.
>
> You just said above that clone had less available flags than
> unshare ...

I did. It is just easier to support clone than unshare.

> anyway, could you elaborate a bit more ? I have the opposite
> feeling and you gave me that impression also a few month ago.
>
> No problem for me, i just want a way to use this stuff without

My feeling is basically that there are some things that we
can do much more cleanly at process creation time.

>From an implementation standpoint unshare is fairly nasty because
it keeps things from being invariants across the lifetime of a
process. Which means it contains more races and is harder to support.
That's why we can't unshare we can share with clone right now.

Eric

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

Subject: Re: [patch -mm 08/17] nsproxy: add hashtable

Quoting Eric W. Biederman (ebiederm@xmission.com):
> "Serge E. Hallyn" <serue@us.ibm.com> writes:
>
> > Quoting Serge E. Hallyn (serue@us.ibm.com):
> >> Quoting Eric W. Biederman (ebiederm@xmission.com):
> >> > Herbert Poetzl <herbert@13thfloor.at> writes:
> >> > >> Beyond that yes it seems to make sense to let user space
> >> > >> maintain any mapping of containers to ids.
> >> > >
> >> > > I agree with that, but we need something to move
> >> > > around between the various spaces ...
> >> >
> >> > If you have CAP_SYS_PTRACE or you have a child process
> >> > in a container you can create another with ptrace.
> >> >
> >> > Now I don't mind optimizing that case, with something like
> >> > the proposed bind_ns syscall. But we need to be darn certain
> >> > why it is safe, and does not change the security model that
> >> > we currently have.
> >>
> >> Sigh, and that's going to have to be a discussion per namespace.
> >
> > Well, assuming that we're using pids as identifiers, that means
> > we can only enter decendent namespaces, which means 'we' must
> > have created them. So anything we could do by entering the ns,
> > we could have done by creating it as well, right?
>
> It isn't strict descendents who we can see. i.e. init can create
> the thing, and we could have just logged into the network but init
> and us still share the same pid namespace.
>
> But yes it would be we can only enter descendent namespaces, for
> some definition of enter.
>
> There are two issues.
> 1) We may have a namespace we want to create and then remove the ability
> for the sysadmin to fiddle with, so it can play with encrypted data or
> something like that safely. Not quite unix but it is certainly worth
> considering.

Yeah, that occurred to me, but it doesn't seem like we can possibly make
sufficient guarantees to the client to make this worthwhile.

I'd love to be wrong about that, but if nothing else we can't prove to
the client that they're running on an unhacked host. So the host admin
will always have to be trusted.

> 2) When we only partially enter a namespace it is very easy for additional
>   properties to enter that namespace. For example we enter the pid
>   namespace and the mount namespace, but keep our current working directory
>   in the previous namespace. Then a process in the restricted namespace
>   can get out by cd into /proc/<?>/cwd.

Yup, entering existing namespaces should be all-or-nothing.

> If someones permissions to various objects does not depend on the namespace
> they are in quite possibly this is a non-issue. If we actually depend on
> the isolation to keep things secure enter is a setup for a first rate escape.

I don't believe the isolation can be effective between two namespaces
where one is an ancestor of another. It can be so long as one isn't
the ancestor of another, but then we're not allowing either to enter
the other namespace. So it's not a problem.

The bind_ns() proposed by Cedric is stricter, only allowing nsid 0 to
switch namespaces. So it may be overly restrictive, and does introduce
a new global namespace, but it is safe.

-serge

Subject: Re: [patch -mm 08/17] nsproxy: add hashtable
Posted by ebiederm on Mon, 11 Dec 2006 20:34:40 GMT
View Forum Message <> Reply to Message

"Serge E. Hallyn" <serue@us.ibm.com> writes:

> Quoting Eric W. Biederman (ebiederm@xmission.com):
>
> Yeah, that occurred to me, but it doesn't seem like we can possibly make
> sufficient guarantees to the client to make this worthwhile.
>
> I'd love to be wrong about that, but if nothing else we can't prove to
> the client that they're running on an unhacked host. So the host admin
> will always have to be trusted.

To some extent that is true. Although all security models we have
currently fall down if you hack the kernel, or run your kernel
in a hacked virtual environment. It would be nice if under normal
conditions you could mount an encrypted filesystem only in a container

and not have concerns of those files escaping.

Which would probably be a matter of having a separate uid_ns and not
allowing process outside of your container to have any permissions in
that filesystem.

>> 2) When we only partially enter a namespace it is very easy for additional
>>   properties to enter that namespace. For example we enter the pid
>>   namespace and the mount namespace, but keep our current working directory
>>   in the previous namespace. Then a process in the restricted namespace
>>   can get out by cd into /proc/<?>/cwd.
>
> Yup, entering existing namespaces should be all-or-nothing.

A truly all-or-nothing has the problem that there is no external
input into the container, and a very controlled external input
to the existing container is what this is about.

>> If someones permissions to various objects does not depend on the namespace
>> they are in quite possibly this is a non-issue. If we actually depend on
>> the isolation to keep things secure enter is a setup for a first rate escape.
>
> I don't believe the isolation can be effective between two namespaces
> where one is an ancestor of another. It can be so long as one isn't
> the ancestor of another, but then we're not allowing either to enter
> the other namespace. So it's not a problem.

Reasonable.

> The bind_ns() proposed by Cedric is stricter, only allowing nsid 0 to
> switch namespaces. So it may be overly restrictive, and does introduce
> a new global namespace, but it is safe.

I will look a little more. There are a lot patches out there that need
review.   What disturbs a little is that with ptrace we have an existing
mechanism that can do everything we want enter or bind_ns to be able to do.

I actually have code that will let me fork a process in a new namespace today
with out needing bind_ns. What is more I don't even have to be root
to use it.

I would very much prefer to see us optimizing our debugging and
control interfaces so they are efficient then see us implement
something completely new that is problem domain specific.

Eric

_____
Containers mailing list

Subject: Re: [patch -mm 08/17] nsproxy: add hashtable
Posted by serue on Mon, 11 Dec 2006 22:01:15 GMT
View Forum Message <> Reply to Message

Quoting Eric W. Biederman (ebiederm@xmission.com):
> "Serge E. Hallyn" <serue@us.ibm.com> writes:
>
> > Quoting Eric W. Biederman (ebiederm@xmission.com):
> >
> > Yeah, that occurred to me, but it doesn't seem like we can possibly make
> > sufficient guarantees to the client to make this worthwhile.
> >
> > I'd love to be wrong about that, but if nothing else we can't prove to
> > the client that they're running on an unhacked host. So the host admin
> > will always have to be trusted.
>
> To some extent that is true. Although all security models we have
> currently fall down if you hack the kernel, or run your kernel
> in a hacked virtual environment. It would be nice if under normal
> conditions you could mount an encrypted filesystem only in a container
> and not have concerns of those files escaping.

Hmm, well perhaps I'm being overly pessimistic - IBM research did have a
demo based on TPM of remote attestation, which may be usable for
ensuring that you're connecting to a service on your virtual machine on
a certain (unhacked) kernel on particular hardware, in which case what
you're talking about may be possible - given a stringent initial
environment (i.e. not the 'gimme $20/month for a hosted partition in
arizona' environment).

Given that, perhaps having a virtual machine with access to encrypted
storage - safe from the host machine admins - may not be unattainable
after all. And given that, it would be worth designing the ns_enter()
system call so that a parent cannot enter some child namespace.

> Which would probably be a matter of having a separate uid_ns and not
> allowing process outside of your container to have any permissions in
> that filesystem.

Yup. Or even just a separate uid_ns and an ecryptfs partition, so
that the host can back up the encrypted data incrementally (per file,
i.e. not just the whole dmcrypted loop file).

-serge

Subject: Re: [patch -mm 08/17] nsproxy: add hashtable
Posted by serue on Mon, 11 Dec 2006 22:18:34 GMT
View Forum Message <> Reply to Message

Quoting Eric W. Biederman (ebiederm@xmission.com):
> I actually have code that will let me fork a process in a new namespace today
> with out needing bind_ns. What is more I don't even have to be root
> to use it.

Can you elaborate? The user namespace patches don't enforce ptrace
yet, so you could unshare as root, become uid 500, then as uid 500
in the original namespace ptrace the process in the new namespace.
Is that what you're doing? If (when) ptrace enforces the uid namespace,
will that stop what you're doing?

-serge

Subject: Re: [patch -mm 08/17] nsproxy: add hashtable
Posted by Dave Hansen on Mon, 11 Dec 2006 22:53:08 GMT
View Forum Message <> Reply to Message

On Mon, 2006-12-11 at 16:23 +0100, Cedric Le Goater wrote:
> > Even letting the concept of nsproxy escape to user space sounds wrong.
> > nsproxy is an internal space optimization. It's not struct container
> > and I don't think we want it to become that.
>
> i don't agree here. we need that, so does openvz, vserver, people working
> on resource management.

I think what those projects need is _some_ way to group tasks. I'm not
sure they actually need nsproxies.

Two tasks in the same container could very well have different
nsproxies. The nsproxy defines how the pid namespace, and pid<->task
mappings happen for a given task. The init process for a container is
special and might actually appear in more than one pid namespace, while

its children might only appear in one. That means that this init
process's nsproxy can and should actually be different from its
children's. This is despite the fact that they are in the same
container.

If we really need this 'container' grouping, it can easily be something
pointed to _by_ the nsproxy, but it shouldn't _be_ the nsproxy.

-- Dave

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

Subject: Re: [patch -mm 08/17] nsproxy: add hashtable
Posted by ebiederm on Tue, 12 Dec 2006 03:28:40 GMT
View Forum Message <> Reply to Message

"Serge E. Hallyn" <serue@us.ibm.com> writes:

> Quoting Eric W. Biederman (ebiederm@xmission.com):
>> I actually have code that will let me fork a process in a new namespace today
>> with out needing bind_ns. What is more I don't even have to be root
>> to use it.
>
> Can you elaborate? The user namespace patches don't enforce ptrace
> yet, so you could unshare as root, become uid 500, then as uid 500
> in the original namespace ptrace the process in the new namespace.
> Is that what you're doing? If (when) ptrace enforces the uid namespace,
> will that stop what you're doing?

sys_ptrace is allowed in 2 situations.
- The user and group identities are the same.
- The calling process has CAP_SYS_PTRACE capability.

So currently if the uid namespace enforces the user and group checks
that will prevent the first case, and is very desirable. But it won't
stop someone with CAP_SYS_PTRACE. Which given the normal case seems
reasonable.

Getting to the point where you can't trace what a process is doing
would probably require some additional interprocess firewalling
from something like selinux.

Eric

_____

Subject: Re: [patch -mm 08/17] nsproxy: add hashtable
Posted by Cedric Le Goater on Tue, 12 Dec 2006 07:09:45 GMT
View Forum Message <> Reply to Message

Dave Hansen wrote:
> On Mon, 2006-12-11 at 16:23 +0100, Cedric Le Goater wrote:
>>> Even letting the concept of nsproxy escape to user space sounds wrong.
>>> nsproxy is an internal space optimization. It's not struct container
>>> and I don't think we want it to become that.
>> i don't agree here. we need that, so does openvz, vserver, people working
>> on resource management.
>
> I think what those projects need is _some_ way to group tasks. I'm not
> sure they actually need nsproxies.

not only tasks. ipc, fs, etc.

> Two tasks in the same container could very well have different
> nsproxies. The nsproxy defines how the pid namespace, and pid<->task
> mappings happen for a given task.

not only. there are other namespaces in nsproxy.

> The init process for a container is
> special and might actually appear in more than one pid namespace, while
> its children might only appear in one. That means that this init
> process's nsproxy can and should actually be different from its
> children's. This is despite the fact that they are in the same
> container.
>
> If we really need this 'container' grouping, it can easily be something
> pointed to _by_ the nsproxy, but it shouldn't _be_ the nsproxy.

ok so let's add a container object, containing a nsproxy and add
another indirection ...

C.

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

Subject: Re: [patch -mm 08/17] nsproxy: add hashtable
Posted by Cedric Le Goater on Tue, 12 Dec 2006 07:11:33 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting Serge E. Hallyn (serue@us.ibm.com):
>> Quoting Eric W. Biederman (ebiederm@xmission.com):
>>> Herbert Poetzl <herbert@13thfloor.at> writes:
>>>>> Beyond that yes it seems to make sense to let user space
>>>>> maintain any mapping of containers to ids.
>>>> I agree with that, but we need something to move
>>>> around between the various spaces ...
>>> If you have CAP_SYS_PTRACE or you have a child process
>>> in a container you can create another with ptrace.
>>>
>>> Now I don't mind optimizing that case, with something like
>>> the proposed bind_ns syscall. But we need to be darn certain
>>> why it is safe, and does not change the security model that
>>> we currently have.
>> Sigh, and that's going to have to be a discussion per namespace.
>
> Well, assuming that we're using pids as identifiers, that means

we can't because a process could die while the namespace is still
referenced by an other subsystem. We need some kind of id.

> we can only enter decendent namespaces, which means 'we' must
> have created them. So anything we could do by entering the ns,
> we could have done by creating it as well, right?

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

Subject: Re: [patch -mm 08/17] nsproxy: add hashtable
Posted by ebiederm on Tue, 12 Dec 2006 07:52:14 GMT
View Forum Message <> Reply to Message

Cedric Le Goater <clg@fr.ibm.com> writes:

> Serge E. Hallyn wrote:
>> Well, assuming that we're using pids as identifiers, that means
>
> we can't because a process could die while the namespace is still
> referenced by an other subsystem. We need some kind of id.

Think of a session think of a process group heck think of threads
a pid is not tied to one task struct. It is absolutely not a problem
for a namespace to do get_pid(...) when it is initialized and put_pid(...)
just before it is freed.

All of the mechanisms for using pids for something like this are already
in place.

What we don't have is a fast pid to namespace transfer. But that is just
an extra pointer in struct pid. Really that is a trivial patch.
Giving every namespace a pid pointer in struct pid takes a little more
space then I would like but it is not a big deal.

Eric

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

Subject: Re: [patch -mm 08/17] nsproxy: add hashtable
Posted by ebiederm on Tue, 12 Dec 2006 08:37:48 GMT
View Forum Message <> Reply to Message

Cedric Le Goater <clg@fr.ibm.com> writes:

> Dave Hansen wrote:
>> On Mon, 2006-12-11 at 16:23 +0100, Cedric Le Goater wrote:
>>>> Even letting the concept of nsproxy escape to user space sounds wrong.
>>>> nsproxy is an internal space optimization. It's not struct container
>>>> and I don't think we want it to become that.
>>> i don't agree here. we need that, so does openvz, vserver, people working
>>> on resource management.
>>
>> I think what those projects need is _some_ way to group tasks. I'm not
>> sure they actually need nsproxies.
>
> not only tasks. ipc, fs, etc.

What is the important aspect that you need to group. What concept
are you trying to convey?

How do you describe a container in which someone is using the
pam_namespace module? So different tasks in the container have
a different mount namespace?

>> Two tasks in the same container could very well have different
>> nsproxies. The nsproxy defines how the pid namespace, and pid<->task

>> mappings happen for a given task.
>
> not only. there are other namespaces in nsproxy.

The point is that there is not a one to one mapping between containers
and nsproxies. There are likely to be more nsproxies than containers.

>> The init process for a container is
>> special and might actually appear in more than one pid namespace, while
>> its children might only appear in one. That means that this init
>> process's nsproxy can and should actually be different from its
>> children's. This is despite the fact that they are in the same
>> container.
>>
>> If we really need this 'container' grouping, it can easily be something
>> pointed to _by_ the nsproxy, but it shouldn't _be_ the nsproxy.
>
> ok so let's add a container object, containing a nsproxy and add
> another indirection ...

Well that isn't what Dave suggested, and I don't think it will give
you what you want.

Eric

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

Subject: Re: [patch -mm 08/17] nsproxy: add hashtable
Posted by dev on Tue, 12 Dec 2006 08:43:38 GMT
View Forum Message <> Reply to Message

>>>Even letting the concept of nsproxy escape to user space sounds wrong.
>>>nsproxy is an internal space optimization. It's not struct container
>>>and I don't think we want it to become that.
>>
>>i don't agree here. we need that, so does openvz, vserver, people working
>>on resource management.
>
>
> I think what those projects need is _some_ way to group tasks. I'm not
> sure they actually need nsproxies.
>
> Two tasks in the same container could very well have different
> nsproxies.
what is container then from your POV?

> The nsproxy defines how the pid namespace, and pid<->task
> mappings happen for a given task. The init process for a container is
> special and might actually appear in more than one pid namespace, while
> its children might only appear in one. That means that this init
> process's nsproxy can and should actually be different from its
> children's. This is despite the fact that they are in the same
> container.
nsproxy has references to all namespaces, not just pid namespace.
Thus it is a container "view" effectively.
If container is something different, then please define it.

> If we really need this 'container' grouping, it can easily be something
> pointed to _by_ the nsproxy, but it shouldn't _be_ the nsproxy.
You can add another indirection if really want it so much...
But is it required?
We created nsproxy which adds another level of indirection, but from performance POV
it is questinable. I can say that we had a nice experience, when adding
a single dereference in TCP code resulted in ~0.5% performance degradation.

Thanks,
Kirill

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

Subject: Re: [patch -mm 08/17] nsproxy: add hashtable
Posted by ebiederm on Tue, 12 Dec 2006 08:57:48 GMT
View Forum Message <> Reply to Message

Kirill Korotaev <dev@sw.ru> writes:

>>
>> I think what those projects need is _some_ way to group tasks. I'm not
>> sure they actually need nsproxies.
>>
>> Two tasks in the same container could very well have different
>> nsproxies.
> what is container then from your POV?

A nested instance of user space. User space may unshare things
such as the mount namespace so it can give users the ability to
control their own mounts and the like.

>> The nsproxy defines how the pid namespace, and pid<->task

>> mappings happen for a given task. The init process for a container is
>> special and might actually appear in more than one pid namespace, while
>> its children might only appear in one. That means that this init
>> process's nsproxy can and should actually be different from its
>> children's. This is despite the fact that they are in the same
>> container.
> nsproxy has references to all namespaces, not just pid namespace.
> Thus it is a container "view" effectively.
> If container is something different, then please define it.

nsproxy has exactly one instance of all namespaces. A container
in the general case can hold other containers, and near containers
(like processes with separate mount namespaces). As well as
processes.

So nsproxy currently captures the common case for containers but not
the general case.

>> If we really need this 'container' grouping, it can easily be something
>> pointed to _by_ the nsproxy, but it shouldn't _be_ the nsproxy.
> You can add another indirection if really want it so much...
> But is it required?
> We created nsproxy which adds another level of indirection, but from performance
> POV
> it is questinable. I can say that we had a nice experience, when adding
> a single dereference in TCP code resulted in ~0.5% performance degradation.

I totally agree with that, nsproxy is something we need to watch from
a performance point of view. nsproxy is primarily a space
optimization to keep from bloating task struct, and possibly a fork
time optimization. At least at the point we added it no one could
measure overhead from using it.

That is one of the reasons I don't want nsproxy to become explicit and
be exported to user space. So if it is a performance problem we can
change the implementation without affecting users.

Eric

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

Subject: Re: [patch -mm 08/17] nsproxy: add hashtable
Posted by serue on Tue, 12 Dec 2006 15:29:12 GMT
View Forum Message <> Reply to Message

Quoting Eric W. Biederman (ebiederm@xmission.com):
> "Serge E. Hallyn" <serue@us.ibm.com> writes:
>
> > Quoting Eric W. Biederman (ebiederm@xmission.com):
> >> I actually have code that will let me fork a process in a new namespace today
> >> with out needing bind_ns. What is more I don't even have to be root
> >> to use it.
> >
> > Can you elaborate? The user namespace patches don't enforce ptrace
> > yet, so you could unshare as root, become uid 500, then as uid 500
> > in the original namespace ptrace the process in the new namespace.
> > Is that what you're doing? If (when) ptrace enforces the uid namespace,
> > will that stop what you're doing?
>
> sys_ptrace is allowed in 2 situations.
> - The user and group identities are the same.
> - The calling process has CAP_SYS_PTRACE capability.
>
> So currently if the uid namespace enforces the user and group checks
> that will prevent the first case, and is very desirable. But it won't
> stop someone with CAP_SYS_PTRACE. Which given the normal case seems
> reasonable.

Yes, I was forgetting that intra-container ptrace is generally
inhibited by lack of a handle to processes in the other container.
So:

 . in checkpoint/restart usage, the normal CAP_SYS_PTRACE
   semantics is fine
 . inside a vserver, the normal CAP_SYS_PTRACE is fine
 . in general, a process inside one vserver cannot reference
   a process in another vserver, so we don't need to worry
   about ptrace permissions at all
 . however, if we want to (as per emails yesterday) provide
   some bit of enforcement of limits from parent namespaces
   to child namespaces - where a pid is in fact available for
   at least the init process (and, depending on our final
   implementation, perhaps all processes) - then we need
   something more.

As you say, selinux permissions would be one way to obtain this.

> Getting to the point where you can't trace what a process is doing
> would probably require some additional interprocess firewalling
> from something like selinux.

Yup.

thanks,
-serge

_____

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

Subject: Re: [patch -mm 08/17] nsproxy: add hashtable
Posted by serue on Tue, 12 Dec 2006 15:45:07 GMT
View Forum Message <> Reply to Message

Quoting Cedric Le Goater (clg@fr.ibm.com):
> Dave Hansen wrote:
> > On Mon, 2006-12-11 at 16:23 +0100, Cedric Le Goater wrote:
> >>> Even letting the concept of nsproxy escape to user space sounds wrong.
> >>> nsproxy is an internal space optimization. It's not struct container
> >>> and I don't think we want it to become that.
> >> i don't agree here. we need that, so does openvz, vserver, people working
> >> on resource management.
> >
> > I think what those projects need is _some_ way to group tasks. I'm not
> > sure they actually need nsproxies.
>
> not only tasks. ipc, fs, etc.
>
> > Two tasks in the same container could very well have different
> > nsproxies. The nsproxy defines how the pid namespace, and pid<->task
> > mappings happen for a given task.
>
> not only. there are other namespaces in nsproxy.

Right, and as Eric has pointed out, you may well want to use one id to
refer to several nsproxies - for instance if you are using unshare
to provide per-user private mount namespaces using pam_namespace.so
(that's mostly for LSPP systems right now, but I do this on my laptop
too). All my accounts are in the same 'container', but have different
mount namespaces, hence different nsproxies.

> > The init process for a container is
> > special and might actually appear in more than one pid namespace, while
> > its children might only appear in one. That means that this init
> > process's nsproxy can and should actually be different from its
> > children's. This is despite the fact that they are in the same
> > container.
> >
> > If we really need this 'container' grouping, it can easily be something
> > pointed to _by_ the nsproxy, but it shouldn't _be_ the nsproxy.

>
> ok so let's add a container object, containing a nsproxy and add
> another indirection ...

No thanks.

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

Subject: Re: [patch -mm 08/17] nsproxy: add hashtable
Posted by Cedric Le Goater on Tue, 12 Dec 2006 18:29:08 GMT
View Forum Message <> Reply to Message

>>> If someones permissions to various objects does not depend on the namespace
>>> they are in quite possibly this is a non-issue. If we actually depend on
>>> the isolation to keep things secure enter is a setup for a first rate escape.
>> I don't believe the isolation can be effective between two namespaces
>> where one is an ancestor of another. It can be so long as one isn't
>> the ancestor of another, but then we're not allowing either to enter
>> the other namespace. So it's not a problem.
>
> Reasonable.
>
>> The bind_ns() proposed by Cedric is stricter, only allowing nsid 0 to
>> switch namespaces. So it may be overly restrictive, and does introduce
>> a new global namespace, but it is safe.
>
> I will look a little more. There are a lot patches out there that need
> review. What disturbs a little is that with ptrace we have an existing
> mechanism that can do everything we want enter or bind_ns to be able to do.

Eric, you have this habit of flooding us with email whenever a patchset is
sent on this topic. It is a bad habit. Please take some time to look at it
before. There is work behind it and it tries to address some issues.

This patchset has been sent on container@ as a proposal for -mm. I'll try
to make a summary of how we can improve next one to move forward.

I still need to read all your emails :)

thanks,

C.

_____
Containers mailing list
Containers@lists.osdl.org

Subject: Re: [patch -mm 08/17] nsproxy: add hashtable
Posted by Herbert Poetzl on Tue, 12 Dec 2006 23:22:22 GMT
View Forum Message <> Reply to Message

On Mon, Dec 11, 2006 at 04:01:15PM -0600, Serge E. Hallyn wrote:
> Quoting Eric W. Biederman (ebiederm@xmission.com):
> > "Serge E. Hallyn" <serue@us.ibm.com> writes:
> >
> > > Quoting Eric W. Biederman (ebiederm@xmission.com):
> > >
> > > Yeah, that occurred to me, but it doesn't seem like we can possibly make
> > > sufficient guarantees to the client to make this worthwhile.
> > >
> > > I'd love to be wrong about that, but if nothing else we can't prove to
> > > the client that they're running on an unhacked host. So the host admin
> > > will always have to be trusted.
> >
> > To some extent that is true. Although all security models we have
> > currently fall down if you hack the kernel, or run your kernel
> > in a hacked virtual environment. It would be nice if under normal
> > conditions you could mount an encrypted filesystem only in a container
> > and not have concerns of those files escaping.
>
> Hmm, well perhaps I'm being overly pessimistic - IBM research did have a
> demo based on TPM of remote attestation, which may be usable for
> ensuring that you're connecting to a service on your virtual machine on
> a certain (unhacked) kernel on particular hardware, in which case what
> you're talking about may be possible - given a stringent initial
> environment (i.e. not the 'gimme $20/month for a hosted partition in
> arizona' environment).

interesting, how would you _ensure_ from inside
such an environment, that nobody tampered with
the kernel you are running on?

> Given that, perhaps having a virtual machine with access to encrypted
> storage - safe from the host machine admins - may not be unattainable
> after all. And given that, it would be worth designing the ns_enter()
> system call so that a parent cannot enter some child namespace.

we currently call this Context Privacy, and it
is partially implemented, but of course, it
does only work if the kernel is known good

> > Which would probably be a matter of having a separate uid_ns and not

> > allowing process outside of your container to have any permissions in
> > that filesystem.
>
> Yup. Or even just a separate uid_ns and an ecryptfs partition, so
> that the host can back up the encrypted data incrementally (per file,
> i.e. not just the whole dmcrypted loop file).

it's simple to avoid access to certain 'tagged'
devices and/or filesystems, it's hard to handle
kernel modifications or even simple things like
reading the kernel memory ...

best,
Herbert

> -serge

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

Subject: Re: [patch -mm 08/17] nsproxy: add hashtable
Posted by Herbert Poetzl on Wed, 13 Dec 2006 04:55:58 GMT
View Forum Message <> Reply to Message

On Tue, Dec 12, 2006 at 11:43:38AM +0300, Kirill Korotaev wrote:
> >>>Even letting the concept of nsproxy escape to user space sounds wrong.
> >>>nsproxy is an internal space optimization. It's not struct container
> >>>and I don't think we want it to become that.

> >>i don't agree here. we need that, so does openvz, vserver, people
> >>working on resource management.
> >
> >
> > I think what those projects need is _some_ way to group tasks. I'm
> > not sure they actually need nsproxies.
> >
> > Two tasks in the same container could very well have different
> > nsproxies.

and typically, they will ...

> what is container then from your POV?

from my PoV, a container is something keeping
processes _inside_ which basically requires
the following elements:

- isolation from other containers
- virtualization of unique elements
- limitation on resources
- policy on all interfaces

the current spaces mostly address the isolation
and to some degree, the virtualization, which
is a good thing, but the container also requires
the resource limitation and the policy, to handle
interfaces to the outside (should not be new to
you, actually :)

so the container (may it be represented by a
structure or not), may reference an nsproxy
(as we do in the 2.6.19 versions of Linux-VServer)
but an nsproxy is not the proper element to
define a container ..

we also want to be able to have sub spaces inside
a container, as long as they do not interfere or
overcome the limitations and policy

> > The nsproxy defines how the pid namespace, and pid<->task
> > mappings happen for a given task. The init process for a container is
> > special and might actually appear in more than one pid namespace, while
> > its children might only appear in one. That means that this init
> > process's nsproxy can and should actually be different from its
> > children's. This is despite the fact that they are in the same
> > container.

> nsproxy has references to all namespaces, not just pid namespace.
> Thus it is a container "view" effectively.

it is a view into the world of one or more processes,
but not necessarily the view of all processes inside
a container :)

> If container is something different, then please define it.

see above ...

> > If we really need this 'container' grouping, it can easily be something
> > pointed to _by_ the nsproxy, but it shouldn't _be_ the nsproxy.

> You can add another indirection if really want it so much...
> But is it required?
> We created nsproxy which adds another level of indirection, but from

> performance POV it is questinable.

I'm not very happy with the nsproxy abstraction,
as I think it would be better handled per task,
and I still have no real world test results what
overhead the nsproxy indirection causes

> I can say that we had a nice experience, when adding a single
> dereference in TCP code resulted in ~0.5% performance degradation.

yes, that is what I fear is happening right now
with the nsproxy ... but I think we need to test
that, and if it makes sense, switch to task direct
spaces (as we had before), just more of them ...

best,
Herbert

> Thanks,
> Kirill
>
> _____
> Containers mailing list
> Containers@lists.osdl.org
> https://lists.osdl.org/mailman/listinfo/containers

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

Subject: Re: [patch -mm 08/17] nsproxy: add hashtable
Posted by Cedric Le Goater on Wed, 13 Dec 2006 14:44:06 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Cedric Le Goater <clg@fr.ibm.com> writes:
>
>> Serge E. Hallyn wrote:
>>> Well, assuming that we're using pids as identifiers, that means
>> we can't because a process could die while the namespace is still
>> referenced by an other subsystem. We need some kind of id.
>
> Think of a session think of a process group heck think of threads
> a pid is not tied to one task struct. It is absolutely not a problem
> for a namespace to do get_pid(...) when it is initialized and put_pid(...)
> just before it is freed.
>

> All of the mechanisms for using pids for something like this are already
> in place.
>
> What we don't have is a fast pid to namespace transfer. But that is just
> an extra pointer in struct pid. Really that is a trivial patch.
> Giving every namespace a pid pointer in struct pid takes a little more
> space then I would like but it is not a big deal.

I'm not sure I understand how you want to do this.

Let me try : you would add a 'struct pid pid' field to all namespaces and assign that 'pid' field with the struct pid of the task creating the namespace ?

C.

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

Subject: Re: [patch -mm 08/17] nsproxy: add hashtable
Posted by Cedric Le Goater on Wed, 13 Dec 2006 15:00:35 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting Cedric Le Goater (clg@fr.ibm.com):
>> Dave Hansen wrote:
>>> On Mon, 2006-12-11 at 16:23 +0100, Cedric Le Goater wrote:
>>>>> Even letting the concept of nsproxy escape to user space sounds wrong.
>>>>> nsproxy is an internal space optimization. It's not struct container
>>>>> and I don't think we want it to become that.
>>>> i don't agree here. we need that, so does openvz, vserver, people working
>>>> on resource management.
>>> I think what those projects need is _some_ way to group tasks. I'm not
>>> sure they actually need nsproxies.
>> not only tasks. ipc, fs, etc.
>>
>>> Two tasks in the same container could very well have different
>>> nsproxies. The nsproxy defines how the pid namespace, and pid<->task
>>> mappings happen for a given task.
>> not only. there are other namespaces in nsproxy.
>
> Right, and as Eric has pointed out, you may well want to use one id to
> refer to several nsproxies - for instance if you are using unshare
> to provide per-user private mount namespaces using pam_namespace.so
> (that's mostly for LSPP systems right now, but I do this on my laptop
> too). All my accounts are in the same 'container', but have different

> mount namespaces, hence different nsproxies.

I think we have definition issue here : what is a 'container' ?

I don't see any issue with the above scenario. unsharing mount namespace
results in the creation of a new nsproxy which will require a new identifier
in order to find this new mount namespace.

so yes, different mount namespaces, hence different nsproxies, hence
different ids if you want to find that new mount namespace.

>>> The init process for a container is
>>> special and might actually appear in more than one pid namespace, while
>>> its children might only appear in one. That means that this init
>>> process's nsproxy can and should actually be different from its
>>> children's. This is despite the fact that they are in the same
>>> container.
>>>
>>> If we really need this 'container' grouping, it can easily be something
>>> pointed to _by_ the nsproxy, but it shouldn't _be_ the nsproxy.
>> ok so let's add a container object, containing a nsproxy and add
>> another indirection ...
>
> No thanks.

exactly.

C.

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

Subject: Re: [patch -mm 08/17] nsproxy: add hashtable
Posted by Cedric Le Goater on Wed, 13 Dec 2006 15:02:56 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Cedric Le Goater <clg@fr.ibm.com> writes:
>
>> Dave Hansen wrote:
>>> On Mon, 2006-12-11 at 16:23 +0100, Cedric Le Goater wrote:
>>>>> Even letting the concept of nsproxy escape to user space sounds wrong.
>>>>> nsproxy is an internal space optimization. It's not struct container
>>>>> and I don't think we want it to become that.

>>>> i don't agree here. we need that, so does openvz, vserver, people working
>>>> on resource management.
>>> I think what those projects need is _some_ way to group tasks. I'm not
>>> sure they actually need nsproxies.
>> not only tasks. ipc, fs, etc.
>
> What is the important aspect that you need to group. What concept
> are you trying to convey?
>
> How do you describe a container in which someone is using the
> pam_namespace module? So different tasks in the container have
> a different mount namespace?

let's define a container first. I'm not sure for what you are using that
term.

>>> Two tasks in the same container could very well have different
>>> nsproxies. The nsproxy defines how the pid namespace, and pid<->task
>>> mappings happen for a given task.
>> not only. there are other namespaces in nsproxy.
>
> The point is that there is not a one to one mapping between containers
> and nsproxies. There are likely to be more nsproxies than containers.

again. please explain the difference that you see between a container
and a nsproxy. I don't get it and i might be missing something important
doing this short cut.

C.

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

Subject: Re: [patch -mm 08/17] nsproxy: add hashtable
Posted by Cedric Le Goater on Wed, 13 Dec 2006 15:17:39 GMT
View Forum Message <> Reply to Message

Herbert Poetzl wrote:
> On Tue, Dec 12, 2006 at 11:43:38AM +0300, Kirill Korotaev wrote:
>>>>> Even letting the concept of nsproxy escape to user space sounds wrong.
>>>>> nsproxy is an internal space optimization. It's not struct container
>>>>> and I don't think we want it to become that.
>
>>>> i don't agree here. we need that, so does openvz, vserver, people
>>>> working on resource management.
>>>

>>> I think what those projects need is _some_ way to group tasks. I'm
>>> not sure they actually need nsproxies.
>>>
>>> Two tasks in the same container could very well have different
>>> nsproxies.
>
> and typically, they will ...

that means we are missing a container object then, a vps, a vcontext, a
vsomething. nop ?

>> what is container then from your POV?
>
> from my PoV, a container is something keeping
> processes _inside_ which basically requires
> the following elements:
>
> - isolation from other containers
> - virtualization of unique elements
> - limitation on resources
> - policy on all interfaces
>
> the current spaces mostly address the isolation
> and to some degree, the virtualization, which
> is a good thing, but the container also requires
> the resource limitation and the policy, to handle
> interfaces to the outside (should not be new to
> you, actually :)
>
> so the container (may it be represented by a
> structure or not), may reference an nsproxy
> (as we do in the 2.6.19 versions of Linux-VServer)
> but an nsproxy is not the proper element to
> define a container ..

agree. it's not complete.

should we address that by introducing a new object ?
could that be done on per-product basis ? I mean like
in a driver model.

> we also want to be able to have sub spaces inside
> a container, as long as they do not interfere or
> overcome the limitations and policy
>
>>> The nsproxy defines how the pid namespace, and pid<->task
>>> mappings happen for a given task. The init process for a container is
>>> special and might actually appear in more than one pid namespace, while

>>> its children might only appear in one. That means that this init
>>> process's nsproxy can and should actually be different from its
>>> children's. This is despite the fact that they are in the same
>>> container.
>
>> nsproxy has references to all namespaces, not just pid namespace.
>> Thus it is a container "view" effectively.
>
> it is a view into the world of one or more processes,
> but not necessarily the view of all processes inside
> a container :)
>
>> If container is something different, then please define it.
>
> see above ...
>
>>> If we really need this 'container' grouping, it can easily be something
>>> pointed to _by_ the nsproxy, but it shouldn't _be_ the nsproxy.
>
>> You can add another indirection if really want it so much...
>> But is it required?
>> We created nsproxy which adds another level of indirection, but from
>> performance POV it is questinable.
>
> I'm not very happy with the nsproxy abstraction,
> as I think it would be better handled per task,
> and I still have no real world test results what
> overhead the nsproxy indirection causes
>
>> I can say that we had a nice experience, when adding a single
>> dereference in TCP code resulted in ~0.5% performance degradation.
>
> yes, that is what I fear is happening right now
> with the nsproxy ... but I think we need to test
> that, and if it makes sense, switch to task direct
> spaces (as we had before), just more of them ...

getting some figures would be nice and we might also be able
to improve the current nsproxy model.

C.

_____

Subject: Re: [patch -mm 08/17] nsproxy: add hashtable
Posted by ebiederm on Wed, 13 Dec 2006 18:53:56 GMT
View Forum Message <> Reply to Message

Cedric Le Goater <clg@fr.ibm.com> writes:

> Eric W. Biederman wrote:
>>
>> What we don't have is a fast pid to namespace transfer. But that is just
>> an extra pointer in struct pid. Really that is a trivial patch.
>> Giving every namespace a pid pointer in struct pid takes a little more
>> space then I would like but it is not a big deal.
>
> I'm not sure I understand how you want to do this.
>
> Let me try : you would add a 'struct pid pid' field to all namespaces and
> assign that 'pid' field with the struct pid of the task creating the
> namespace ?

Yes a struct pid *pid field, that we did the proper reference counting
on.

As for which pid to assign, that is a little trickier. The struct pid
of the task creating the namespace is the obvious choice and that will
always work for clone. For unshare that would only work if we added
the restriction you can't unshare if someone already has used that pid
for that kind of namespace.

Eric

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

Subject: Re: [patch -mm 08/17] nsproxy: add hashtable
Posted by Cedric Le Goater on Thu, 14 Dec 2006 13:17:47 GMT
View Forum Message <> Reply to Message

>> Let me try : you would add a 'struct pid pid' field to all namespaces and
>> assign that 'pid' field with the struct pid of the task creating the
>> namespace ?
>
> Yes a struct pid *pid field, that we did the proper reference counting
> on.

sure.

> As for which pid to assign, that is a little trickier. The struct pid
> of the task creating the namespace is the obvious choice and that will
> always work for clone. For unshare that would only work if we added
> the restriction you can't unshare if someone already has used that pid
> for that kind of namespace.

clone will also be an issue if more than one namespace is unshared. Do
you use the same 'struct pid*' for each namespace ? hmm, it feels also
wrong.

having an id field in the namespace and using a bind_ns like syscall
to let the user assign whatever id he wants to, doesn't seem to be
such a bad idea.

C.

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

Subject: Re: [patch -mm 08/17] nsproxy: add hashtable
Posted by ebiederm on Thu, 14 Dec 2006 21:08:58 GMT
View Forum Message <> Reply to Message

Cedric Le Goater <clg@fr.ibm.com> writes:

>>> Let me try : you would add a 'struct pid pid' field to all namespaces and
>>> assign that 'pid' field with the struct pid of the task creating the
>>> namespace ?
>>
>> Yes a struct pid *pid field, that we did the proper reference counting
>> on.
>
> sure.
>
>> As for which pid to assign, that is a little trickier. The struct pid
>> of the task creating the namespace is the obvious choice and that will
>> always work for clone. For unshare that would only work if we added
>> the restriction you can't unshare if someone already has used that pid
>> for that kind of namespace.
>
> clone will also be an issue if more than one namespace is unshared. Do
> you use the same 'struct pid*' for each namespace ? hmm, it feels also
> wrong.
>
> having an id field in the namespace and using a bind_ns like syscall
> to let the user assign whatever id he wants to, doesn't seem to be

> such a bad idea.

I think I would probably have suggested simply taking the next
available id in that case in practice. Partly it depends on exactly
what we are trying to do with these.

But I do agree getting that last little details right so some corner
case doesn't feel wrong is hard. That is why I try and put off
this kind of things until as much is known about how we are going
to use it as possible. So we can make a good decision and solve
practical problems, and not theoretical ones.

Eric

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

Subject: Re: [patch -mm 08/17] nsproxy: add hashtable
Posted by serue on Wed, 20 Dec 2006 06:12:03 GMT
View Forum Message <> Reply to Message

Quoting Herbert Poetzl (herbert@13thfloor.at):
> On Mon, Dec 11, 2006 at 04:01:15PM -0600, Serge E. Hallyn wrote:
> > Quoting Eric W. Biederman (ebiederm@xmission.com):
> > > "Serge E. Hallyn" <serue@us.ibm.com> writes:
> > >
> > > > Quoting Eric W. Biederman (ebiederm@xmission.com):
> > > >
> > > > Yeah, that occurred to me, but it doesn't seem like we can possibly make
> > > > sufficient guarantees to the client to make this worthwhile.
> > > >
> > > > I'd love to be wrong about that, but if nothing else we can't prove to
> > > > the client that they're running on an unhacked host. So the host admin
> > > > will always have to be trusted.
> > >
> > > To some extent that is true. Although all security models we have
> > > currently fall down if you hack the kernel, or run your kernel
> > > in a hacked virtual environment. It would be nice if under normal
> > > conditions you could mount an encrypted filesystem only in a container
> > > and not have concerns of those files escaping.
> >
> > Hmm, well perhaps I'm being overly pessimistic - IBM research did have a
> > demo based on TPM of remote attestation, which may be usable for
> > ensuring that you're connecting to a service on your virtual machine on
> > a certain (unhacked) kernel on particular hardware, in which case what
> > you're talking about may be possible - given a stringent initial

> > environment (i.e. not the 'gimme $20/month for a hosted partition in
> > arizona' environment).
>
> interesting, how would you _ensure_ from inside
> such an environment, that nobody tampered with
> the kernel you are running on?

Sorry, took awhile to find the best reference, but I guess this would be
it:

http://domino.research.ibm.com/library/cyberdig.nsf/1e4115aea78b6e7c85256b360066f0d4/459e9a2b1f668aee85256f330067589f

Another description is

http://domino.research.ibm.com/comm/research_people.nsf/pages/sailer.ima.html

I guess it was in 2004 that they did a demo of remote attestion at the
RSA conference, as described in the third to last paragraph in
http://cio.co.nz/cio.nsf/0/03943645293DB008CC256E47005D8EA2?OpenDocument
and in
http://domino.research.ibm.com/comm/pr.nsf/pages/news.20040218_linux.html

-serge

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