Subject: userns: targeted capabilities v5
Posted by serge on Thu, 17 Feb 2011 15:02:24 GMT
View Forum Message <> Reply to Message

Here is a repost of my previous user namespace patch, ported onto
last night's git head.

It fixes several things I was doing wrong in the last (v4)
posting, in particular:

1. don't set uts_ns->user_ns to current's when !CLONE_NEWUTS
2. add a ipc_ns->user_ns which owns ipc_ns, and use that to
   decide CAP_IPC_OWNER
3. fix logic flaw caused by bad parantheses
4. allow do_prlimit to current
5. don't always give root full privs to init_user_ns

The expected course of development for user namespaces is laid out
at https://wiki.ubuntu.com/UserNamespace. Bugs aside, this
patchset is supposed to not at all affect systems which are not
actively using user namespaces, and only restrict what tasks in
child user namespace can do. They begin to limit privilege to
a user namespace, so that root in a container cannot kill or
ptrace tasks in the parent user namespace, and can only get
world access rights to files. Since all files currently belong
to the initila user namespace, that means that child user
namespaces can only get world access rights to *all* files.
While this temporarily makes user namespaces bad for system
containers, it starts to get useful for some sandboxing.

I've run the 'runltplite.sh' with and without this patchset and
found no difference. So all in all, this is the first version
of this patchset for which I feel comfortable asking: please
consider applying.

thanks,
-serge

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

Subject: [PATCH 1/9] Add a user_namespace as creator/owner of uts_namespace
Posted by serge on Thu, 17 Feb 2011 15:02:57 GMT
View Forum Message <> Reply to Message

copy_process() handles CLONE_NEWUSER before the rest of the

namespaces. So in the case of clone(CLONE_NEWUSER|CLONE_NEWUTS)
the new uts namespace will have the new user namespace as its
owner. That is what we want, since we want root in that new
userns to be able to have privilege over it.

Changelog:
 Feb 15: don't set uts_ns->user_ns if we didn't create
  a new uts_ns.

Signed-off-by: Serge E. Hallyn <serge.hallyn@canonical.com>
---
 include/linux/utsname.h | 3 +++
 init/version.c | 2 ++
 kernel/nsproxy.c | 5 +++++
 kernel/user.c | 8 ++++++--
 kernel/utsname.c | 4 ++++
 5 files changed, 20 insertions(+), 2 deletions(-)

diff --git a/include/linux/utsname.h b/include/linux/utsname.h
index 69f3997..85171be 100644
--- a/include/linux/utsname.h
+++ b/include/linux/utsname.h
@@ -37,9 +37,12 @@ struct new_utsname {
 #include <linux/nsproxy.h>
 #include <linux/err.h>

+struct user_namespace;
+
 struct uts_namespace {
  struct kref kref;
  struct new_utsname name;
+ struct user_namespace *user_ns;
 };
 extern struct uts_namespace init_uts_ns;

diff --git a/init/version.c b/init/version.c
index adff586..97bb86f 100644
--- a/init/version.c
+++ b/init/version.c
@@ -21,6 +21,7 @@ extern int version_string(LINUX_VERSION_CODE);
 int version_string(LINUX_VERSION_CODE);
 #endif

+extern struct user_namespace init_user_ns;
 struct uts_namespace init_uts_ns = {
  .kref = {
   .refcount = ATOMIC_INIT(2),
@@ -33,6 +34,7 @@ struct uts_namespace init_uts_ns = {

```
   .machine = UTS_MACHINE,
   .domainname = UTS_DOMAINNAME,
  },
+ .user_ns = &init_user_ns,
 };
 EXPORT_SYMBOL_GPL(init_uts_ns);

diff --git a/kernel/nsproxy.c b/kernel/nsproxy.c
index f74e6c0..034dc2e 100644
--- a/kernel/nsproxy.c
+++ b/kernel/nsproxy.c
@@ -74,6 +74,11 @@ static struct nsproxy *create_new_namespaces(unsigned long flags,
   err = PTR_ERR(new_nsp->uts_ns);
   goto out_uts;
  }
+ if (new_nsp->uts_ns != tsk->nsproxy->uts_ns) {
+  put_user_ns(new_nsp->uts_ns->user_ns);
+  new_nsp->uts_ns->user_ns = task_cred_xxx(tsk, user)->user_ns;
+  get_user_ns(new_nsp->uts_ns->user_ns);
+ }

  new_nsp->ipc_ns = copy_ipcs(flags, tsk->nsproxy->ipc_ns);
  if (IS_ERR(new_nsp->ipc_ns)) {
diff --git a/kernel/user.c b/kernel/user.c
index 5c598ca..9e03e9c 100644
--- a/kernel/user.c
+++ b/kernel/user.c
@@ -17,9 +17,13 @@
 #include <linux/module.h>
 #include <linux/user_namespace.h>

+/*
+ * userns count is 1 for root user, 1 for init_uts_ns,
+ * and 1 for... ?
+ */
 struct user_namespace init_user_ns = {
  .kref = {
-  .refcount = ATOMIC_INIT(2),
+  .refcount = ATOMIC_INIT(3),
  },
  .creator = &root_user,
 };
@@ -47,7 +51,7 @@ static struct kmem_cache *uid_cachep;
  */
 static DEFINE_SPINLOCK(uidhash_lock);

-/* root_user.__count is 2, 1 for init task cred, 1 for init_user_ns->creator */
+/* root_user.__count is 2, 1 for init task cred, 1 for init_user_ns->user_ns */
```

```
 struct user_struct root_user = {
  .__count = ATOMIC_INIT(2),
  .processes = ATOMIC_INIT(1),
diff --git a/kernel/utsname.c b/kernel/utsname.c
index 8a82b4b..a7b3a8d 100644
--- a/kernel/utsname.c
+++ b/kernel/utsname.c
@@ -14,6 +14,7 @@
 #include <linux/utsname.h>
 #include <linux/err.h>
 #include <linux/slab.h>
+#include <linux/user_namespace.h>

 static struct uts_namespace *create_uts_ns(void)
 {
@@ -40,6 +41,8 @@ static struct uts_namespace *clone_uts_ns(struct uts_namespace *old_ns)

  down_read(&uts_sem);
  memcpy(&ns->name, &old_ns->name, sizeof(ns->name));
+ ns->user_ns = old_ns->user_ns;
+ get_user_ns(ns->user_ns);
  up_read(&uts_sem);
  return ns;
 }
@@ -71,5 +74,6 @@ void free_uts_ns(struct kref *kref)
  struct uts_namespace *ns;

  ns = container_of(kref, struct uts_namespace, kref);
+ put_user_ns(ns->user_ns);
  kfree(ns);
 }
--
1.7.0.4
```

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

---

Subject: [PATCH 2/9] security: Make capabilities relative to the user namespace.
Posted by serge on Thu, 17 Feb 2011 15:03:06 GMT
View Forum Message <> Reply to Message

- Introduce ns_capable to test for a capability in a non-default
  user namespace.
- Teach cap_capable to handle capabilities in a non-default
  user namespace.

The motivation is to get to the unprivileged creation of new namespaces. It looks like this gets us 90% of the way there, with only potential uid confusion issues left.

I still need to handle getting all caps after creation but otherwise I think I have a good starter patch that achieves all of your goals.

Changelog:
11/05/2010: [serge] add apparmor
12/14/2010: [serge] fix capabilities to created user namespaces
Without this, if user serge creates a user_ns, he won't have capabilities to the user_ns he created. THis is because we were first checking whether his effective caps had the caps he needed and returning -EPERM if not, and THEN checking whether he was the creator. Reverse those checks.
12/16/2010: [serge] security_real_capable needs ns argument in !security case
01/11/2011: [serge] add task_ns_capable helper
01/11/2011: [serge] add nsown_capable() helper per Bastian Blank suggestion
02/16/2011: [serge] fix a logic bug: the root user is always creator of init_user_ns, but should not always have capabilities to it! Fix the check in cap_capable().

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
Signed-off-by: Serge E. Hallyn <serge.hallyn@canonical.com>
---
include/linux/capability.h | 10 ++++++++--
include/linux/security.h | 25 ++++++++++++++----------
kernel/capability.c | 32 ++++++++++++++++++++++++++++++--
security/apparmor/lsm.c | 5 +++--
security/commoncap.c | 40 ++++++++++++++++++++++++++++++++++-------
security/security.c | 16 ++++++++++------
security/selinux/hooks.c | 14 +++++++++-----
7 files changed, 107 insertions(+), 35 deletions(-)

diff --git a/include/linux/capability.h b/include/linux/capability.h
index fb16a36..cb3d2d9 100644
--- a/include/linux/capability.h
+++ b/include/linux/capability.h
@@ -544,7 +544,7 @@ extern const kernel_cap_t __cap_init_eff_set;
*
* Note that this does not set PF_SUPERPRIV on the task.
*/
-#define has_capability(t, cap) (security_real_capable((t), (cap)) == 0)
+#define has_capability(t, cap) (security_real_capable((t), &init_user_ns, (cap)) == 0)

/**
* has_capability_noaudit - Determine if a task has a superior capability available (unaudited)

```
@@ -558,9 +558,15 @@ extern const kernel_cap_t __cap_init_eff_set;
  * Note that this does not set PF_SUPERPRIV on the task.
  */
 #define has_capability_noaudit(t, cap) \
- (security_real_capable_noaudit((t), (cap)) == 0)
+ (security_real_capable_noaudit((t), &init_user_ns, (cap)) == 0)

+struct user_namespace;
+extern struct user_namespace init_user_ns;
 extern int capable(int cap);
+extern int ns_capable(struct user_namespace *ns, int cap);
+extern int task_ns_capable(struct task_struct *t, int cap);
+
+#define nsown_capable(cap) (ns_capable(current_user_ns(), (cap)))

 /* audit system wants to get cap info from files as well */
 struct dentry;
diff --git a/include/linux/security.h b/include/linux/security.h
index b2b7f97..6bbee08 100644
--- a/include/linux/security.h
+++ b/include/linux/security.h
@@ -46,13 +46,14 @@

 struct ctl_table;
 struct audit_krule;
+struct user_namespace;

 /*
  * These functions are in security/capability.c and are used
  * as the default capabilities functions
  */
 extern int cap_capable(struct task_struct *tsk, const struct cred *cred,
-        int cap, int audit);
+        struct user_namespace *ns, int cap, int audit);
 extern int cap_settime(struct timespec *ts, struct timezone *tz);
 extern int cap_ptrace_access_check(struct task_struct *child, unsigned int mode);
 extern int cap_ptrace_traceme(struct task_struct *parent);
@@ -1254,6 +1255,7 @@ static inline void security_free_mnt_opts(struct security_mnt_opts
*opts)
  * credentials.
  * @tsk contains the task_struct for the process.
  * @cred contains the credentials to use.
+ *      @ns contains the user namespace we want the capability in
  * @cap contains the capability <include/linux/capability.h>.
  * @audit: Whether to write an audit message or not
  * Return 0 if the capability is granted for @tsk.
@@ -1382,7 +1384,7 @@ struct security_operations {
         const kernel_cap_t *inheritable,
```

```
         const kernel_cap_t *permitted);
  int (*capable) (struct task_struct *tsk, const struct cred *cred,
-   int cap, int audit);
+   struct user_namespace *ns, int cap, int audit);
  int (*sysctl) (struct ctl_table *table, int op);
  int (*quotactl) (int cmds, int type, int id, struct super_block *sb);
  int (*quota_on) (struct dentry *dentry);
@@ -1662,9 +1664,9 @@ int security_capset(struct cred *new, const struct cred *old,
       const kernel_cap_t *effective,
       const kernel_cap_t *inheritable,
       const kernel_cap_t *permitted);
-int security_capable(const struct cred *cred, int cap);
-int security_real_capable(struct task_struct *tsk, int cap);
-int security_real_capable_noaudit(struct task_struct *tsk, int cap);
+int security_capable(struct user_namespace *ns, const struct cred *cred, int cap);
+int security_real_capable(struct task_struct *tsk, struct user_namespace *ns, int cap);
+int security_real_capable_noaudit(struct task_struct *tsk, struct user_namespace *ns, int cap);
 int security_sysctl(struct ctl_table *table, int op);
 int security_quotactl(int cmds, int type, int id, struct super_block *sb);
 int security_quota_on(struct dentry *dentry);
@@ -1856,28 +1858,29 @@ static inline int security_capset(struct cred *new,
  return cap_capset(new, old, effective, inheritable, permitted);
 }

-static inline int security_capable(const struct cred *cred, int cap)
+static inline int security_capable(struct user_namespace *ns,
+       const struct cred *cred, int cap)
 {
- return cap_capable(current, cred, cap, SECURITY_CAP_AUDIT);
+ return cap_capable(current, cred, ns, cap, SECURITY_CAP_AUDIT);
 }

-static inline int security_real_capable(struct task_struct *tsk, int cap)
+static inline int security_real_capable(struct task_struct *tsk, struct user_namespace *ns, int cap)
 {
  int ret;

  rcu_read_lock();
- ret = cap_capable(tsk, __task_cred(tsk), cap, SECURITY_CAP_AUDIT);
+ ret = cap_capable(tsk, __task_cred(tsk), ns, cap, SECURITY_CAP_AUDIT);
  rcu_read_unlock();
  return ret;
 }

 static inline
-int security_real_capable_noaudit(struct task_struct *tsk, int cap)
+int security_real_capable_noaudit(struct task_struct *tsk, struct user_namespace *ns, int cap)
 {
```

```
  int ret;

  rcu_read_lock();
- ret = cap_capable(tsk, __task_cred(tsk), cap,
+ ret = cap_capable(tsk, __task_cred(tsk), ns, cap,
           SECURITY_CAP_NOAUDIT);
  rcu_read_unlock();
  return ret;
diff --git a/kernel/capability.c b/kernel/capability.c
index 9e9385f..916658c 100644
--- a/kernel/capability.c
+++ b/kernel/capability.c
@@ -14,6 +14,7 @@
 #include <linux/security.h>
 #include <linux/syscalls.h>
 #include <linux/pid_namespace.h>
+#include <linux/user_namespace.h>
 #include <asm/uaccess.h>

 /*
@@ -301,15 +302,42 @@ error:
  */
 int capable(int cap)
 {
+ return ns_capable(&init_user_ns, cap);
+}
+EXPORT_SYMBOL(capable);
+
+/**
+ * ns_capable - Determine if the current task has a superior capability in effect
+ * @ns:  The usernamespace we want the capability in
+ * @cap: The capability to be tested for
+ *
+ * Return true if the current task has the given superior capability currently
+ * available for use, false if not.
+ *
+ * This sets PF_SUPERPRIV on the task if the capability is available on the
+ * assumption that it's about to be used.
+ */
+int ns_capable(struct user_namespace *ns, int cap)
+{
  if (unlikely(!cap_valid(cap))) {
   printk(KERN_CRIT "capable() called with invalid cap=%u\n", cap);
   BUG();
  }

- if (security_capable(current_cred(), cap) == 0) {
+ if (security_capable(ns, current_cred(), cap) == 0) {
```

```
   current->flags |= PF_SUPERPRIV;
   return 1;
  }
  return 0;
 }
-EXPORT_SYMBOL(capable);
+EXPORT_SYMBOL(ns_capable);
+
+/*
+ * does current have capability 'cap' to the user namespace of task
+ * 't'.  Return true if it does, false otherwise.
+ */
+int task_ns_capable(struct task_struct *t, int cap)
+{
+ return ns_capable(task_cred_xxx(t, user)->user_ns, cap);
+}
+EXPORT_SYMBOL(task_ns_capable);
diff --git a/security/apparmor/lsm.c b/security/apparmor/lsm.c
index b7106f1..b37c2cd 100644
--- a/security/apparmor/lsm.c
+++ b/security/apparmor/lsm.c
@@ -22,6 +22,7 @@
 #include <linux/ctype.h>
 #include <linux/sysctl.h>
 #include <linux/audit.h>
+#include <linux/user_namespace.h>
 #include <net/sock.h>

 #include "include/apparmor.h"
@@ -136,11 +137,11 @@ static int apparmor_capget(struct task_struct *target, kernel_cap_t
*effective,
 }

 static int apparmor_capable(struct task_struct *task, const struct cred *cred,
-       int cap, int audit)
+       struct user_namespace *ns, int cap, int audit)
 {
  struct aa_profile *profile;
  /* cap_capable returns 0 on success, else -EPERM */
- int error = cap_capable(task, cred, cap, audit);
+ int error = cap_capable(task, cred, ns, cap, audit);
  if (!error) {
   profile = aa_cred_profile(cred);
   if (!unconfined(profile))
diff --git a/security/commoncap.c b/security/commoncap.c
index 64c2ed9..51fa9ec 100644
--- a/security/commoncap.c
+++ b/security/commoncap.c
```

```
@@ -27,6 +27,7 @@
 #include <linux/sched.h>
 #include <linux/prctl.h>
 #include <linux/securebits.h>
+#include <linux/user_namespace.h>

 /*
  * If a non-root user executes a setuid-root binary in
@@ -68,6 +69,7 @@ EXPORT_SYMBOL(cap_netlink_recv);
  * cap_capable - Determine whether a task has a particular effective capability
  * @tsk: The task to query
  * @cred: The credentials to use
+ * @ns:  The user namespace in which we need the capability
  * @cap: The capability to check for
  * @audit: Whether to write an audit message or not
  *
@@ -79,10 +81,32 @@ EXPORT_SYMBOL(cap_netlink_recv);
  * cap_has_capability() returns 0 when a task has a capability, but the
  * kernel's capable() and has_capability() returns 1 for this case.
  */
-int cap_capable(struct task_struct *tsk, const struct cred *cred, int cap,
-  int audit)
+int cap_capable(struct task_struct *tsk, const struct cred *cred,
+  struct user_namespace *targ_ns, int cap, int audit)
 {
- return cap_raised(cred->cap_effective, cap) ? 0 : -EPERM;
+ for (;;) {
+  /* The creator of the user namespace has all caps. */
+  if (targ_ns != &init_user_ns && targ_ns->creator == cred->user)
+   return 0;
+
+  /* Do we have the necessary capabilities? */
+  if (targ_ns == cred->user->user_ns)
+   return cap_raised(cred->cap_effective, cap) ? 0 : -EPERM;
+
+  /* Have we tried all of the parent namespaces? */
+  if (targ_ns == &init_user_ns)
+   return -EPERM;
+
+  /* If you have the capability in a parent user ns you have it
+   * in the over all children user namespaces as well, so see
+   * if this process has the capability in the parent user
+   * namespace.
+   */
+  targ_ns = targ_ns->creator->user_ns;
+ }
+
+ /* We never get here */
```

```
+ return -EPERM;
 }

 /**
@@ -177,7 +201,8 @@ static inline int cap_inh_is_capped(void)
  /* they are so limited unless the current task has the CAP_SETPCAP
   * capability
   */
- if (cap_capable(current, current_cred(), CAP_SETPCAP,
+ if (cap_capable(current, current_cred(),
+   current_cred()->user->user_ns, CAP_SETPCAP,
    SECURITY_CAP_AUDIT) == 0)
   return 0;
  return 1;
@@ -829,7 +854,8 @@ int cap_task_prctl(int option, unsigned long arg2, unsigned long arg3,
        & (new->securebits ^ arg2))   /*[1]*/
       || ((new->securebits & SECURE_ALL_LOCKS & ~arg2)) /*[2]*/
       || (arg2 & ~(SECURE_ALL_LOCKS | SECURE_ALL_BITS)) /*[3]*/
-      || (cap_capable(current, current_cred(), CAP_SETPCAP,
+      || (cap_capable(current, current_cred(),
+        current_cred()->user->user_ns, CAP_SETPCAP,
         SECURITY_CAP_AUDIT) != 0)  /*[4]*/
    /*
     * [1] no changing of bits that are locked
@@ -894,7 +920,7 @@ int cap_vm_enough_memory(struct mm_struct *mm, long pages)
 {
  int cap_sys_admin = 0;

- if (cap_capable(current, current_cred(), CAP_SYS_ADMIN,
+ if (cap_capable(current, current_cred(), &init_user_ns, CAP_SYS_ADMIN,
    SECURITY_CAP_NOAUDIT) == 0)
   cap_sys_admin = 1;
  return __vm_enough_memory(mm, pages, cap_sys_admin);
@@ -921,7 +947,7 @@ int cap_file_mmap(struct file *file, unsigned long reqprot,
  int ret = 0;

  if (addr < dac_mmap_min_addr) {
-  ret = cap_capable(current, current_cred(), CAP_SYS_RAWIO,
+  ret = cap_capable(current, current_cred(), &init_user_ns, CAP_SYS_RAWIO,
       SECURITY_CAP_AUDIT);
   /* set PF_SUPERPRIV if it turns out we allow the low mmap */
   if (ret == 0)
diff --git a/security/security.c b/security/security.c
index 7b7308a..7a6a0d0 100644
--- a/security/security.c
+++ b/security/security.c
@@ -154,29 +154,33 @@ int security_capset(struct cred *new, const struct cred *old,
         effective, inheritable, permitted);
```

```
 }

-int security_capable(const struct cred *cred, int cap)
+int security_capable(struct user_namespace *ns, const struct cred *cred,
+       int cap)
 {
- return security_ops->capable(current, cred, cap, SECURITY_CAP_AUDIT);
+ return security_ops->capable(current, cred, ns, cap,
+         SECURITY_CAP_AUDIT);
 }

-int security_real_capable(struct task_struct *tsk, int cap)
+int security_real_capable(struct task_struct *tsk, struct user_namespace *ns,
+     int cap)
 {
  const struct cred *cred;
  int ret;

  cred = get_task_cred(tsk);
- ret = security_ops->capable(tsk, cred, cap, SECURITY_CAP_AUDIT);
+ ret = security_ops->capable(tsk, cred, ns, cap, SECURITY_CAP_AUDIT);
  put_cred(cred);
  return ret;
 }

-int security_real_capable_noaudit(struct task_struct *tsk, int cap)
+int security_real_capable_noaudit(struct task_struct *tsk,
+      struct user_namespace *ns, int cap)
 {
  const struct cred *cred;
  int ret;

  cred = get_task_cred(tsk);
- ret = security_ops->capable(tsk, cred, cap, SECURITY_CAP_NOAUDIT);
+ ret = security_ops->capable(tsk, cred, ns, cap, SECURITY_CAP_NOAUDIT);
  put_cred(cred);
  return ret;
 }
diff --git a/security/selinux/hooks.c b/security/selinux/hooks.c
index c8d6992..6dcda48 100644
--- a/security/selinux/hooks.c
+++ b/security/selinux/hooks.c
@@ -77,6 +77,7 @@
 #include <linux/mutex.h>
 #include <linux/posix-timers.h>
 #include <linux/syslog.h>
+#include <linux/user_namespace.h>
```

```
 #include "avc.h"
 #include "objsec.h"
@@ -1423,6 +1424,7 @@ static int current_has_perm(const struct task_struct *tsk,
 /* Check whether a task is allowed to use a capability. */
 static int task_has_capability(struct task_struct *tsk,
           const struct cred *cred,
+          struct user_namespace *ns,
           int cap, int audit)
 {
  struct common_audit_data ad;
@@ -1851,15 +1853,15 @@ static int selinux_capset(struct cred *new, const struct cred *old,
  */

 static int selinux_capable(struct task_struct *tsk, const struct cred *cred,
-       int cap, int audit)
+       struct user_namespace *ns, int cap, int audit)
 {
  int rc;

- rc = cap_capable(tsk, cred, cap, audit);
+ rc = cap_capable(tsk, cred, ns, cap, audit);
  if (rc)
   return rc;

- return task_has_capability(tsk, cred, cap, audit);
+ return task_has_capability(tsk, cred, ns, cap, audit);
 }

 static int selinux_sysctl_get_sid(ctl_table *table, u16 tclass, u32 *sid)
@@ -2012,7 +2014,8 @@ static int selinux_vm_enough_memory(struct mm_struct *mm, long
pages)
 {
  int rc, cap_sys_admin = 0;

- rc = selinux_capable(current, current_cred(), CAP_SYS_ADMIN,
+ rc = selinux_capable(current, current_cred(),
+        &init_user_ns, CAP_SYS_ADMIN,
         SECURITY_CAP_NOAUDIT);
  if (rc == 0)
   cap_sys_admin = 1;
@@ -2829,7 +2832,8 @@ static int selinux_inode_getsecurity(const struct inode *inode, const
char *name
   * and lack of permission just means that we fall back to the
   * in-core context value, not a denial.
   */
- error = selinux_capable(current, current_cred(), CAP_MAC_ADMIN,
+ error = selinux_capable(current, current_cred(),
+    &init_user_ns, CAP_MAC_ADMIN,
```

```
    SECURITY_CAP_NOAUDIT);
  if (!error)
   error = security_sid_to_context_force(isec->sid, &context,
--
1.7.0.4
```

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

---

Subject: [PATCH 3/9] allow sethostname in a container
Posted by serge on Thu, 17 Feb 2011 15:03:16 GMT
View Forum Message <> Reply to Message

Signed-off-by: Serge E. Hallyn <serge.hallyn@canonical.com>
---
kernel/sys.c | 2 +-
1 files changed, 1 insertions(+), 1 deletions(-)

```
diff --git a/kernel/sys.c b/kernel/sys.c
index 18da702..7a1bbad 100644
--- a/kernel/sys.c
+++ b/kernel/sys.c
@@ -1177,7 +1177,7 @@ SYSCALL_DEFINE2(sethostname, char __user *, name, int, len)
  int errno;
  char tmp[__NEW_UTS_LEN];

- if (!capable(CAP_SYS_ADMIN))
+ if (!ns_capable(current->nsproxy->uts_ns->user_ns, CAP_SYS_ADMIN))
   return -EPERM;
  if (len < 0 || len > __NEW_UTS_LEN)
   return -EINVAL;
--
1.7.0.4
```

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

---

Subject: [PATCH 4/9] allow killing tasks in your own or child userns
Posted by serge on Thu, 17 Feb 2011 15:03:25 GMT
View Forum Message <> Reply to Message

```
Changelog:
 Dec  8: Fixed bug in my check_kill_permission pointed out by
         Eric Biederman.
 Dec 13: Apply Eric's suggestion to pass target task into kill_ok_by_cred()
         for clarity
 Dec 31: address comment by Eric Biederman:
  don't need cred/tcred in check_kill_permission.
 Jan  1: use const cred struct.
 Jan 11: Per Bastian Blank's advice, clean up kill_ok_by_cred().
 Feb 16: kill_ok_by_cred: fix bad parentheses

Signed-off-by: Serge E. Hallyn <serge.hallyn@canonical.com>
---
 kernel/signal.c |   30 ++++++++++++++++++++++++--------
 1 files changed, 22 insertions(+), 8 deletions(-)

diff --git a/kernel/signal.c b/kernel/signal.c
index 4e3cff1..ffe4bdf 100644
--- a/kernel/signal.c
+++ b/kernel/signal.c
@@ -636,13 +636,33 @@ static inline bool si_fromuser(const struct siginfo *info)
 }

 /*
+ * called with RCU read lock from check_kill_permission()
+ */
+static inline int kill_ok_by_cred(struct task_struct *t)
+{
+ const struct cred *cred = current_cred();
+ const struct cred *tcred = __task_cred(t);
+
+ if (cred->user->user_ns == tcred->user->user_ns &&
+     (cred->euid == tcred->suid ||
+      cred->euid == tcred->uid ||
+      cred->uid  == tcred->suid ||
+      cred->uid  == tcred->uid))
+  return 1;
+
+ if (ns_capable(tcred->user->user_ns, CAP_KILL))
+  return 1;
+
+ return 0;
+}
+
+/*
  * Bad permissions for sending the signal
  * - the caller must hold the RCU read lock
  */
```

```
 static int check_kill_permission(int sig, struct siginfo *info,
      struct task_struct *t)
 {
- const struct cred *cred, *tcred;
  struct pid *sid;
  int error;

@@ -656,14 +676,8 @@ static int check_kill_permission(int sig, struct siginfo *info,
  if (error)
   return error;

- cred = current_cred();
- tcred = __task_cred(t);
  if (!same_thread_group(current, t) &&
-     (cred->euid ^ tcred->suid) &&
-     (cred->euid ^ tcred->uid) &&
-     (cred->uid  ^ tcred->suid) &&
-     (cred->uid  ^ tcred->uid) &&
-     !capable(CAP_KILL)) {
+     !kill_ok_by_cred(t)) {
    switch (sig) {
    case SIGCONT:
     sid = task_session(t);
--
1.7.0.4
```

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

Subject: [PATCH 5/9] Allow ptrace from non-init user namespaces
Posted by serge on Thu, 17 Feb 2011 15:03:33 GMT
View Forum Message <> Reply to Message

ptrace is allowed to tasks in the same user namespace according to
the usual rules (i.e. the same rules as for two tasks in the init
user namespace). ptrace is also allowed to a user namespace to
which the current task the has CAP_SYS_PTRACE capability.

Changelog:
 Dec 31: Address feedback by Eric:
  . Correct ptrace uid check
  . Rename may_ptrace_ns to ptrace_capable
  . Also fix the cap_ptrace checks.
 Jan 1: Use const cred struct
 Jan 11: use task_ns_capable() in place of ptrace_capable().

Signed-off-by: Serge E. Hallyn <serge.hallyn@canonical.com>
---
 include/linux/capability.h | 2 +
 include/linux/user_namespace.h | 9 +++++++
 kernel/ptrace.c | 27 ++++++++++++----------
 kernel/user_namespace.c | 16 +++++++++++++
 security/commoncap.c | 48 ++++++++++++++++++++++++++++++++++------
 5 files changed, 82 insertions(+), 20 deletions(-)

diff --git a/include/linux/capability.h b/include/linux/capability.h
index cb3d2d9..bc0f262 100644
--- a/include/linux/capability.h
+++ b/include/linux/capability.h
@@ -546,6 +546,8 @@ extern const kernel_cap_t __cap_init_eff_set;
  */
 #define has_capability(t, cap) (security_real_capable((t), &init_user_ns, (cap)) == 0)

+#define has_ns_capability(t, ns, cap) (security_real_capable((t), (ns), (cap)) == 0)
+
 /**
  * has_capability_noaudit - Determine if a task has a superior capability available (unaudited)
  * @t: The task in question
diff --git a/include/linux/user_namespace.h b/include/linux/user_namespace.h
index faf4679..862fc59 100644
--- a/include/linux/user_namespace.h
+++ b/include/linux/user_namespace.h
@@ -39,6 +39,9 @@ static inline void put_user_ns(struct user_namespace *ns)
 uid_t user_ns_map_uid(struct user_namespace *to, const struct cred *cred, uid_t uid);
 gid_t user_ns_map_gid(struct user_namespace *to, const struct cred *cred, gid_t gid);

+int same_or_ancestor_user_ns(struct task_struct *task,
+ struct task_struct *victim);
+
 #else

 static inline struct user_namespace *get_user_ns(struct user_namespace *ns)
@@ -66,6 +69,12 @@ static inline gid_t user_ns_map_gid(struct user_namespace *to,
  return gid;
 }

+static inline int same_or_ancestor_user_ns(struct task_struct *task,
+ struct task_struct *victim)
+{
+ return 1;
+}
+
 #endif

```
 #endif /* _LINUX_USER_H */
diff --git a/kernel/ptrace.c b/kernel/ptrace.c
index 1708b1e..cde4655 100644
--- a/kernel/ptrace.c
+++ b/kernel/ptrace.c
@@ -134,21 +134,24 @@ int __ptrace_may_access(struct task_struct *task, unsigned int mode)
   return 0;
  rcu_read_lock();
  tcred = __task_cred(task);
- if ((cred->uid != tcred->euid ||
-      cred->uid != tcred->suid ||
-      cred->uid != tcred->uid  ||
-      cred->gid != tcred->egid ||
-      cred->gid != tcred->sgid ||
-      cred->gid != tcred->gid) &&
-     !capable(CAP_SYS_PTRACE)) {
-  rcu_read_unlock();
-  return -EPERM;
- }
+ if (cred->user->user_ns == tcred->user->user_ns &&
+     (cred->uid == tcred->euid &&
+      cred->uid == tcred->suid &&
+      cred->uid == tcred->uid  &&
+      cred->gid == tcred->egid &&
+      cred->gid == tcred->sgid &&
+      cred->gid == tcred->gid))
+  goto ok;
+ if (ns_capable(tcred->user->user_ns, CAP_SYS_PTRACE))
+  goto ok;
+ rcu_read_unlock();
+ return -EPERM;
+ok:
  rcu_read_unlock();
  smp_rmb();
  if (task->mm)
   dumpable = get_dumpable(task->mm);
- if (!dumpable && !capable(CAP_SYS_PTRACE))
+ if (!dumpable && !task_ns_capable(task, CAP_SYS_PTRACE))
   return -EPERM;

  return security_ptrace_access_check(task, mode);
@@ -198,7 +201,7 @@ int ptrace_attach(struct task_struct *task)
   goto unlock_tasklist;

  task->ptrace = PT_PTRACED;
- if (capable(CAP_SYS_PTRACE))
+ if (task_ns_capable(task, CAP_SYS_PTRACE))
```

```
   task->ptrace |= PT_PTRACE_CAP;

  __ptrace_link(task, current);
diff --git a/kernel/user_namespace.c b/kernel/user_namespace.c
index 9da289c..0ef2258 100644
--- a/kernel/user_namespace.c
+++ b/kernel/user_namespace.c
@@ -129,6 +129,22 @@ gid_t user_ns_map_gid(struct user_namespace *to, const struct cred
*cred, gid_t
  return overflowgid;
 }

+int same_or_ancestor_user_ns(struct task_struct *task,
+ struct task_struct *victim)
+{
+ struct user_namespace *u1 = task_cred_xxx(task, user)->user_ns;
+ struct user_namespace *u2 = task_cred_xxx(victim, user)->user_ns;
+ for (;;) {
+  if (u1 == u2)
+   return 1;
+  if (u1 == &init_user_ns)
+   return 0;
+  u1 = u1->creator->user_ns;
+ }
+ /* We never get here */
+ return 0;
+}
+
 static __init int user_namespaces_init(void)
 {
  user_ns_cachep = KMEM_CACHE(user_namespace, SLAB_PANIC);
diff --git a/security/commoncap.c b/security/commoncap.c
index 51fa9ec..12ff65c 100644
--- a/security/commoncap.c
+++ b/security/commoncap.c
@@ -130,18 +130,34 @@ int cap_settime(struct timespec *ts, struct timezone *tz)
  * @child: The process to be accessed
  * @mode: The mode of attachment.
  *
+ * If we are in the same or an ancestor user_ns and have all the target
+ * task's capabilities, then ptrace access is allowed.
+ * If we have the ptrace capability to the target user_ns, then ptrace
+ * access is allowed.
+ * Else denied.
+ *
  * Determine whether a process may access another, returning 0 if permission
  * granted, -ve if denied.
  */
```

```
 int cap_ptrace_access_check(struct task_struct *child, unsigned int mode)
 {
  int ret = 0;
+ const struct cred *cred, *tcred;

  rcu_read_lock();
- if (!cap_issubset(__task_cred(child)->cap_permitted,
-     current_cred()->cap_permitted) &&
-     !capable(CAP_SYS_PTRACE))
-  ret = -EPERM;
+ cred = current_cred();
+ tcred = __task_cred(child);
+ /*
+  * The ancestor user_ns check may be gratuitous, as I think
+  * we've already guaranteed that in kernel/ptrace.c.
+  */
+ if (same_or_ancestor_user_ns(current, child) &&
+     cap_issubset(tcred->cap_permitted, cred->cap_permitted))
+  goto out;
+ if (ns_capable(tcred->user->user_ns, CAP_SYS_PTRACE))
+  goto out;
+ ret = -EPERM;
+out:
  rcu_read_unlock();
  return ret;
 }
@@ -150,18 +166,34 @@ int cap_ptrace_access_check(struct task_struct *child, unsigned int
mode)
  * cap_ptrace_traceme - Determine whether another process may trace the current
  * @parent: The task proposed to be the tracer
  *
+ * If parent is in the same or an ancestor user_ns and has all current's
+ * capabilities, then ptrace access is allowed.
+ * If parent has the ptrace capability to current's user_ns, then ptrace
+ * access is allowed.
+ * Else denied.
+ *
  * Determine whether the nominated task is permitted to trace the current
  * process, returning 0 if permission is granted, -ve if denied.
  */
 int cap_ptrace_traceme(struct task_struct *parent)
 {
  int ret = 0;
+ const struct cred *cred, *tcred;

  rcu_read_lock();
- if (!cap_issubset(current_cred()->cap_permitted,
-     __task_cred(parent)->cap_permitted) &&
```

```
-     !has_capability(parent, CAP_SYS_PTRACE))
-  ret = -EPERM;
+ cred = __task_cred(parent);
+ tcred = current_cred();
+ /*
+  * The ancestor user_ns check may be gratuitous, as I think
+  * we've already guaranteed that in kernel/ptrace.c.
+  */
+ if (same_or_ancestor_user_ns(parent, current) &&
+     cap_issubset(tcred->cap_permitted, cred->cap_permitted))
+  goto out;
+ if (has_ns_capability(parent, tcred->user->user_ns, CAP_SYS_PTRACE))
+  goto out;
+ ret = -EPERM;
+out:
  rcu_read_unlock();
  return ret;
 }
--
1.7.0.4
```

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

---

Subject: [PATCH 6/9] user namespaces: convert all capable checks in kernel/sys.c
Posted by serge on Thu, 17 Feb 2011 15:03:42 GMT
View Forum Message <> Reply to Message

This allows setuid/setgid in containers. It also fixes some
corner cases where kernel logic foregoes capability checks when
uids are equivalent. The latter will need to be done throughout
the whole kernel.

Changelog:
 Jan 11: Use nsown_capable() as suggested by Bastian Blank.
 Jan 11: Fix logic errors in uid checks pointed out by Bastian.
 Feb 15: allow prlimit to current (was regression in previous version)

Signed-off-by: Serge E. Hallyn <serge.hallyn@canonical.com>
---
 kernel/sys.c | 74 ++++++

```
--- a/kernel/sys.c
+++ b/kernel/sys.c
@@ -118,17 +118,29 @@ EXPORT_SYMBOL(cad_pid);

 void (*pm_power_off_prepare)(void);

+/* called with rcu_read_lock, creds are safe */
+static inline int set_one_prio_perm(struct task_struct *p)
+{
+ const struct cred *cred = current_cred(), *pcred = __task_cred(p);
+
+ if (pcred->user->user_ns == cred->user->user_ns &&
+     (pcred->uid  == cred->euid ||
+      pcred->euid == cred->euid))
+  return 1;
+ if (ns_capable(pcred->user->user_ns, CAP_SYS_NICE))
+  return 1;
+ return 0;
+}
+
 /*
  * set the priority of a task
  * - the caller must hold the RCU read lock
  */
 static int set_one_prio(struct task_struct *p, int niceval, int error)
 {
- const struct cred *cred = current_cred(), *pcred = __task_cred(p);
  int no_nice;

- if (pcred->uid  != cred->euid &&
-     pcred->euid != cred->euid && !capable(CAP_SYS_NICE)) {
+ if (!set_one_prio_perm(p)) {
   error = -EPERM;
   goto out;
  }
@@ -502,7 +514,7 @@ SYSCALL_DEFINE2(setregid, gid_t, rgid, gid_t, egid)
  if (rgid != (gid_t) -1) {
   if (old->gid == rgid ||
       old->egid == rgid ||
-      capable(CAP_SETGID))
+      nsown_capable(CAP_SETGID))
    new->gid = rgid;
   else
    goto error;
@@ -511,7 +523,7 @@ SYSCALL_DEFINE2(setregid, gid_t, rgid, gid_t, egid)
   if (old->gid == egid ||
       old->egid == egid ||
       old->sgid == egid ||
```

```
-      capable(CAP_SETGID))
+      nsown_capable(CAP_SETGID))
     new->egid = egid;
   else
     goto error;
@@ -546,7 +558,7 @@ SYSCALL_DEFINE1(setgid, gid_t, gid)
  old = current_cred();

  retval = -EPERM;
- if (capable(CAP_SETGID))
+ if (nsown_capable(CAP_SETGID))
   new->gid = new->egid = new->sgid = new->fsgid = gid;
  else if (gid == old->gid || gid == old->sgid)
   new->egid = new->fsgid = gid;
@@ -613,7 +625,7 @@ SYSCALL_DEFINE2(setreuid, uid_t, ruid, uid_t, euid)
   new->uid = ruid;
   if (old->uid != ruid &&
       old->euid != ruid &&
-      !capable(CAP_SETUID))
+      !nsown_capable(CAP_SETUID))
     goto error;
  }

@@ -622,7 +634,7 @@ SYSCALL_DEFINE2(setreuid, uid_t, ruid, uid_t, euid)
   if (old->uid != euid &&
       old->euid != euid &&
       old->suid != euid &&
-      !capable(CAP_SETUID))
+      !nsown_capable(CAP_SETUID))
     goto error;
  }

@@ -670,7 +682,7 @@ SYSCALL_DEFINE1(setuid, uid_t, uid)
  old = current_cred();

  retval = -EPERM;
- if (capable(CAP_SETUID)) {
+ if (nsown_capable(CAP_SETUID)) {
   new->suid = new->uid = uid;
   if (uid != old->uid) {
     retval = set_user(new);
@@ -712,7 +724,7 @@ SYSCALL_DEFINE3(setresuid, uid_t, ruid, uid_t, euid, uid_t, suid)
  old = current_cred();

  retval = -EPERM;
- if (!capable(CAP_SETUID)) {
+ if (!nsown_capable(CAP_SETUID)) {
   if (ruid != (uid_t) -1 && ruid != old->uid &&
```

```
       ruid != old->euid  && ruid != old->suid)
    goto error;
@@ -776,7 +788,7 @@ SYSCALL_DEFINE3(setresgid, gid_t, rgid, gid_t, egid, gid_t, sgid)
  old = current_cred();

  retval = -EPERM;
- if (!capable(CAP_SETGID)) {
+ if (!nsown_capable(CAP_SETGID)) {
   if (rgid != (gid_t) -1 && rgid != old->gid &&
       rgid != old->egid  && rgid != old->sgid)
    goto error;
@@ -836,7 +848,7 @@ SYSCALL_DEFINE1(setfsuid, uid_t, uid)

  if (uid == old->uid  || uid == old->euid  ||
      uid == old->suid || uid == old->fsuid ||
-     capable(CAP_SETUID)) {
+     nsown_capable(CAP_SETUID)) {
   if (uid != old_fsuid) {
    new->fsuid = uid;
    if (security_task_fix_setuid(new, old, LSM_SETID_FS) == 0)
@@ -869,7 +881,7 @@ SYSCALL_DEFINE1(setfsgid, gid_t, gid)

  if (gid == old->gid  || gid == old->egid  ||
      gid == old->sgid || gid == old->fsgid ||
-     capable(CAP_SETGID)) {
+     nsown_capable(CAP_SETGID)) {
   if (gid != old_fsgid) {
    new->fsgid = gid;
    goto change_okay;
@@ -1177,8 +1189,11 @@ SYSCALL_DEFINE2(sethostname, char __user *, name, int, len)
  int errno;
  char tmp[__NEW_UTS_LEN];

- if (!ns_capable(current->nsproxy->uts_ns->user_ns, CAP_SYS_ADMIN))
+ if (!ns_capable(current->nsproxy->uts_ns->user_ns, CAP_SYS_ADMIN)) {
+  printk(KERN_NOTICE "%s: did not have CAP_SYS_ADMIN\n", __func__);
   return -EPERM;
+ }
+ printk(KERN_NOTICE "%s: did have CAP_SYS_ADMIN\n", __func__);
  if (len < 0 || len > __NEW_UTS_LEN)
   return -EINVAL;
  down_write(&uts_sem);
@@ -1226,7 +1241,7 @@ SYSCALL_DEFINE2(setdomainname, char __user *, name, int, len)
  int errno;
  char tmp[__NEW_UTS_LEN];

- if (!capable(CAP_SYS_ADMIN))
+ if (!ns_capable(current->nsproxy->uts_ns->user_ns, CAP_SYS_ADMIN))
```

```
   return -EPERM;
  if (len < 0 || len > __NEW_UTS_LEN)
   return -EINVAL;
@@ -1341,6 +1356,8 @@ int do_prlimit(struct task_struct *tsk, unsigned int resource,
  rlim = tsk->signal->rlim + resource;
  task_lock(tsk->group_leader);
  if (new_rlim) {
+  /* Keep the capable check against init_user_ns until
+     cgroups can contain all limits */
   if (new_rlim->rlim_max > rlim->rlim_max &&
     !capable(CAP_SYS_RESOURCE))
    retval = -EPERM;
@@ -1384,19 +1401,22 @@ static int check_prlimit_permission(struct task_struct *task)
 {
  const struct cred *cred = current_cred(), *tcred;

- tcred = __task_cred(task);
- if (current != task &&
-     (cred->uid != tcred->euid ||
-      cred->uid != tcred->suid ||
-      cred->uid != tcred->uid  ||
-      cred->gid != tcred->egid ||
-      cred->gid != tcred->sgid ||
-      cred->gid != tcred->gid) &&
-      !capable(CAP_SYS_RESOURCE)) {
-  return -EPERM;
- }
+ if (current == task)
+  return 0;

- return 0;
+ tcred = __task_cred(task);
+ if (cred->user->user_ns == tcred->user->user_ns &&
+     (cred->uid == tcred->euid &&
+      cred->uid == tcred->suid &&
+      cred->uid == tcred->uid  &&
+      cred->gid == tcred->egid &&
+      cred->gid == tcred->sgid &&
+      cred->gid == tcred->gid))
+  return 0;
+ if (ns_capable(tcred->user->user_ns, CAP_SYS_RESOURCE))
+  return 0;
+
+ return -EPERM;
 }

 SYSCALL_DEFINE4(prlimit64, pid_t, pid, unsigned int, resource,
--
```

1.7.0.4

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

Subject: [PATCH 7/9] add a user namespace owner of ipc ns
Posted by serge on Thu, 17 Feb 2011 15:03:49 GMT
View Forum Message <> Reply to Message

Changelog:
Feb 15: Don't set new ipc->user_ns if we didn't create a new
ipc_ns.

Signed-off-by: Serge E. Hallyn <serge.hallyn@canonical.com>
---
include/linux/ipc_namespace.h | 3 +++
ipc/msgutil.c | 3 +++
ipc/namespace.c | 9 +++++++--
kernel/nsproxy.c | 5 +++++
4 files changed, 18 insertions(+), 2 deletions(-)

```
diff --git a/include/linux/ipc_namespace.h b/include/linux/ipc_namespace.h
index 5195298..46d2eb4 100644
--- a/include/linux/ipc_namespace.h
+++ b/include/linux/ipc_namespace.h
@@ -24,6 +24,7 @@ struct ipc_ids {
  struct idr ipcs_idr;
 };

+struct user_namespace;
 struct ipc_namespace {
  atomic_t count;
  struct ipc_ids ids[3];
@@ -56,6 +57,8 @@ struct ipc_namespace {
  unsigned int    mq_msg_max;      /* initialized to DFLT_MSGMAX */
  unsigned int    mq_msgsize_max;  /* initialized to DFLT_MSGSIZEMAX */

+ /* user_ns which owns the ipc ns */
+ struct user_namespace *user_ns;
 };

 extern struct ipc_namespace init_ipc_ns;
diff --git a/ipc/msgutil.c b/ipc/msgutil.c
index f095ee2..d91ff4b 100644
--- a/ipc/msgutil.c
```

```
+++ b/ipc/msgutil.c
@@ -20,6 +20,8 @@

 DEFINE_SPINLOCK(mq_lock);

+extern struct user_namespace init_user_ns;
+
 /*
  * The next 2 defines are here bc this is the only file
  * compiled when either CONFIG_SYSVIPC and CONFIG_POSIX_MQUEUE
@@ -32,6 +34,7 @@ struct ipc_namespace init_ipc_ns = {
  .mq_msg_max      = DFLT_MSGMAX,
  .mq_msgsize_max  = DFLT_MSGSIZEMAX,
 #endif
+ .user_ns = &init_user_ns,
 };

 atomic_t nr_ipc_ns = ATOMIC_INIT(1);
diff --git a/ipc/namespace.c b/ipc/namespace.c
index a1094ff..aa18899 100644
--- a/ipc/namespace.c
+++ b/ipc/namespace.c
@@ -11,10 +11,11 @@
 #include <linux/slab.h>
 #include <linux/fs.h>
 #include <linux/mount.h>
+#include <linux/user_namespace.h>

 #include "util.h"

-static struct ipc_namespace *create_ipc_ns(void)
+static struct ipc_namespace *create_ipc_ns(struct ipc_namespace *old_ns)
 {
  struct ipc_namespace *ns;
  int err;
@@ -43,6 +44,9 @@ static struct ipc_namespace *create_ipc_ns(void)
  ipcns_notify(IPCNS_CREATED);
  register_ipcns_notifier(ns);

+ ns->user_ns = old_ns->user_ns;
+ get_user_ns(ns->user_ns);
+
  return ns;
 }

@@ -50,7 +54,7 @@ struct ipc_namespace *copy_ipcs(unsigned long flags, struct
ipc_namespace *ns)
 {
```

```
  if (!(flags & CLONE_NEWIPC))
   return get_ipc_ns(ns);
- return create_ipc_ns();
+ return create_ipc_ns(ns);
 }

 /*
@@ -105,6 +109,7 @@ static void free_ipc_ns(struct ipc_namespace *ns)
   * order to have a correct value when recomputing msgmni.
   */
  ipcns_notify(IPCNS_REMOVED);
+ put_user_ns(ns->user_ns);
 }

 /*
diff --git a/kernel/nsproxy.c b/kernel/nsproxy.c
index 034dc2e..b6dbff2 100644
--- a/kernel/nsproxy.c
+++ b/kernel/nsproxy.c
@@ -85,6 +85,11 @@ static struct nsproxy *create_new_namespaces(unsigned long flags,
   err = PTR_ERR(new_nsp->ipc_ns);
   goto out_ipc;
  }
+ if (new_nsp->ipc_ns != tsk->nsproxy->ipc_ns) {
+  put_user_ns(new_nsp->ipc_ns->user_ns);
+  new_nsp->ipc_ns->user_ns = task_cred_xxx(tsk, user)->user_ns;
+  get_user_ns(new_nsp->ipc_ns->user_ns);
+ }

  new_nsp->pid_ns = copy_pid_ns(flags, task_active_pid_ns(tsk));
  if (IS_ERR(new_nsp->pid_ns)) {
--
1.7.0.4
```

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

---

Subject: [PATCH 8/9] user namespaces: convert several capable() calls
Posted by serge on Thu, 17 Feb 2011 15:03:56 GMT
View Forum Message <> Reply to Message

CAP_IPC_OWNER and CAP_IPC_LOCK can be checked against current_user_ns(),
because the resource comes from current's own ipc namespace.

setuid/setgid are to uids in own namespace, so again checks can be

against current_user_ns().

Changelog:
 Jan 11: Use task_ns_capable() in place of sched_capable().
 Jan 11: Use nsown_capable() as suggested by Bastian Blank.
 Jan 11: Clarify (hopefully) some logic in futex and sched.c
 Feb 15: use ns_capable for ipc, not nsown_capable

Signed-off-by: Serge E. Hallyn <serge.hallyn@canonical.com>
---
 ipc/shm.c | 2 +-
 ipc/util.c | 5 +++--
 kernel/futex.c | 11 ++++++++++-
 kernel/futex_compat.c | 11 ++++++++++-
 kernel/groups.c | 2 +-
 kernel/sched.c | 9 ++++++---
 kernel/uid16.c | 2 +-
 7 files changed, 32 insertions(+), 10 deletions(-)

diff --git a/ipc/shm.c b/ipc/shm.c
index 7d3bb22..e91e2e9 100644
--- a/ipc/shm.c
+++ b/ipc/shm.c
@@ -773,7 +773,7 @@ SYSCALL_DEFINE3(shmctl, int, shmid, int, cmd, struct shmid_ds __user
*, buf)

   audit_ipc_obj(&(shp->shm_perm));

- if (!capable(CAP_IPC_LOCK)) {
+ if (!ns_capable(ns->user_ns, CAP_IPC_LOCK)) {
    uid_t euid = current_euid();
    err = -EPERM;
    if (euid != shp->shm_perm.uid &&
diff --git a/ipc/util.c b/ipc/util.c
index 69a0cc1..8e7ec6a 100644
--- a/ipc/util.c
+++ b/ipc/util.c
@@ -627,7 +627,7 @@ int ipcperms (struct kern_ipc_perm *ipcp, short flag)
   granted_mode >>= 3;
  /* is there some bit set in requested_mode but not in granted_mode? */
  if ((requested_mode & ~granted_mode & 0007) &&
-     !capable(CAP_IPC_OWNER))
+   !ns_capable(current->nsproxy->ipc_ns->user_ns, CAP_IPC_OWNER))
   return -1;

  return security_ipc_permission(ipcp, flag);
@@ -800,7 +800,8 @@ struct kern_ipc_perm *ipcctl_pre_down(struct ipc_ids *ids, int id, int cmd,

```
  euid = current_euid();
  if (euid == ipcp->cuid ||
-     euid == ipcp->uid  || capable(CAP_SYS_ADMIN))
+     euid == ipcp->uid  ||
+     ns_capable(current->nsproxy->ipc_ns->user_ns, CAP_SYS_ADMIN))
    return ipcp;

  err = -EPERM;
diff --git a/kernel/futex.c b/kernel/futex.c
index b766d28..1e876f1 100644
--- a/kernel/futex.c
+++ b/kernel/futex.c
@@ -2421,10 +2421,19 @@ SYSCALL_DEFINE3(get_robust_list, int, pid,
    goto err_unlock;
   ret = -EPERM;
   pcred = __task_cred(p);
+  /* If victim is in different user_ns, then uids are not
+     comparable, so we must have CAP_SYS_PTRACE */
+  if (cred->user->user_ns != pcred->user->user_ns) {
+   if (!ns_capable(pcred->user->user_ns, CAP_SYS_PTRACE))
+    goto err_unlock;
+   goto ok;
+  }
+  /* If victim is in same user_ns, then uids are comparable */
   if (cred->euid != pcred->euid &&
       cred->euid != pcred->uid &&
-      !capable(CAP_SYS_PTRACE))
+      !ns_capable(pcred->user->user_ns, CAP_SYS_PTRACE))
    goto err_unlock;
+ok:
   head = p->robust_list;
   rcu_read_unlock();
  }
diff --git a/kernel/futex_compat.c b/kernel/futex_compat.c
index a7934ac..5f9e689 100644
--- a/kernel/futex_compat.c
+++ b/kernel/futex_compat.c
@@ -153,10 +153,19 @@ compat_sys_get_robust_list(int pid, compat_uptr_t __user *head_ptr,
    goto err_unlock;
   ret = -EPERM;
   pcred = __task_cred(p);
+  /* If victim is in different user_ns, then uids are not
+     comparable, so we must have CAP_SYS_PTRACE */
+  if (cred->user->user_ns != pcred->user->user_ns) {
+   if (!ns_capable(pcred->user->user_ns, CAP_SYS_PTRACE))
+    goto err_unlock;
+   goto ok;
+  }
```

```
+  /* If victim is in same user_ns, then uids are comparable */
   if (cred->euid != pcred->euid &&
       cred->euid != pcred->uid &&
-      !capable(CAP_SYS_PTRACE))
+      !ns_capable(pcred->user->user_ns, CAP_SYS_PTRACE))
    goto err_unlock;
+ok:
   head = p->compat_robust_list;
   rcu_read_unlock();
  }
diff --git a/kernel/groups.c b/kernel/groups.c
index 253dc0f..1cc476d 100644
--- a/kernel/groups.c
+++ b/kernel/groups.c
@@ -233,7 +233,7 @@ SYSCALL_DEFINE2(setgroups, int, gidsetsize, gid_t __user *, grouplist)
  struct group_info *group_info;
  int retval;

- if (!capable(CAP_SETGID))
+ if (!nsown_capable(CAP_SETGID))
   return -EPERM;
  if ((unsigned)gidsetsize > NGROUPS_MAX)
   return -EINVAL;
diff --git a/kernel/sched.c b/kernel/sched.c
index 18d38e4..dc12bc2 100644
--- a/kernel/sched.c
+++ b/kernel/sched.c
@@ -4761,8 +4761,11 @@ static bool check_same_owner(struct task_struct *p)

  rcu_read_lock();
  pcred = __task_cred(p);
- match = (cred->euid == pcred->euid ||
-   cred->euid == pcred->uid);
+ if (cred->user->user_ns == pcred->user->user_ns)
+  match = (cred->euid == pcred->euid ||
+    cred->euid == pcred->uid);
+ else
+  match = false;
  rcu_read_unlock();
  return match;
 }
@@ -5088,7 +5091,7 @@ long sched_setaffinity(pid_t pid, const struct cpumask *in_mask)
   goto out_free_cpus_allowed;
  }
  retval = -EPERM;
- if (!check_same_owner(p) && !capable(CAP_SYS_NICE))
+ if (!check_same_owner(p) && !task_ns_capable(p, CAP_SYS_NICE))
   goto out_unlock;
```

```
  retval = security_task_setscheduler(p);
diff --git a/kernel/uid16.c b/kernel/uid16.c
index 4192098..51c6e89 100644
--- a/kernel/uid16.c
+++ b/kernel/uid16.c
@@ -189,7 +189,7 @@ SYSCALL_DEFINE2(setgroups16, int, gidsetsize, old_gid_t __user *,
grouplist)
  struct group_info *group_info;
  int retval;

- if (!capable(CAP_SETGID))
+ if (!nsown_capable(CAP_SETGID))
   return -EPERM;
  if ((unsigned)gidsetsize > NGROUPS_MAX)
   return -EINVAL;
--
1.7.0.4
```

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

---

Subject: [PATCH 9/9] userns: check user namespace for task->file uid equivalence checks

Posted by serge on Thu, 17 Feb 2011 15:04:07 GMT
View Forum Message <> Reply to Message

Cheat for now and say all files belong to init_user_ns. Next
step will be to let superblocks belong to a user_ns, and derive
inode_userns(inode) from inode->i_sb->s_user_ns. Finally we'll
introduce more flexible arrangements.

Changelog:
Feb 15: make is_owner_or_cap take const struct inode

Signed-off-by: Serge E. Hallyn <serge.hallyn@canonical.com>

```
 fs/inode.c         |   17 ++++++++++++++++
 fs/namei.c         |   20 ++++++++++++++++-----
 include/linux/fs.h |    9 +++++++--
 3 files changed, 39 insertions(+), 7 deletions(-)

diff --git a/fs/inode.c b/fs/inode.c
index da85e56..1930b45 100644
--- a/fs/inode.c
```

```
+++ b/fs/inode.c
@@ -25,6 +25,7 @@
 #include <linux/async.h>
 #include <linux/posix_acl.h>
 #include <linux/ima.h>
+#include <linux/cred.h>

 /*
  * This is needed for the following functions:
@@ -1722,3 +1723,19 @@ void inode_init_owner(struct inode *inode, const struct inode *dir,
  inode->i_mode = mode;
 }
 EXPORT_SYMBOL(inode_init_owner);
+
+/*
+ * return 1 if current either has CAP_FOWNER to the
+ * file, or owns the file.
+ */
+int is_owner_or_cap(const struct inode *inode)
+{
+ struct user_namespace *ns = inode_userns(inode);
+
+ if (current_user_ns() == ns && current_fsuid() == inode->i_uid)
+  return 1;
+ if (ns_capable(ns, CAP_FOWNER))
+  return 1;
+ return 0;
+}
+EXPORT_SYMBOL(is_owner_or_cap);
diff --git a/fs/namei.c b/fs/namei.c
index 9e701e2..cfac5b4 100644
--- a/fs/namei.c
+++ b/fs/namei.c
@@ -176,6 +176,9 @@ static int acl_permission_check(struct inode *inode, int mask, unsigned
int flag

  mask &= MAY_READ | MAY_WRITE | MAY_EXEC;

+ if (current_user_ns() != inode_userns(inode))
+  goto other_perms;
+
  if (current_fsuid() == inode->i_uid)
   mode >>= 6;
  else {
@@ -189,6 +192,7 @@ static int acl_permission_check(struct inode *inode, int mask, unsigned
int flag
    mode >>= 3;
  }
```

```
+other_perms:
  /*
   * If the DACs are ok we don't need any capability check.
   */
@@ -230,7 +234,7 @@ int generic_permission(struct inode *inode, int mask, unsigned int flags,
   * Executable DACs are overridable if at least one exec bit is set.
   */
  if (!(mask & MAY_EXEC) || execute_ok(inode))
-  if (capable(CAP_DAC_OVERRIDE))
+  if (ns_capable(inode_userns(inode), CAP_DAC_OVERRIDE))
    return 0;

  /*
@@ -238,7 +242,7 @@ int generic_permission(struct inode *inode, int mask, unsigned int flags,
   */
  mask &= MAY_READ | MAY_WRITE | MAY_EXEC;
  if (mask == MAY_READ || (S_ISDIR(inode->i_mode) && !(mask & MAY_WRITE)))
-  if (capable(CAP_DAC_READ_SEARCH))
+  if (ns_capable(inode_userns(inode), CAP_DAC_READ_SEARCH))
    return 0;

  return -EACCES;
@@ -675,6 +679,7 @@ force_reval_path(struct path *path, struct nameidata *nd)
 static inline int exec_permission(struct inode *inode, unsigned int flags)
 {
  int ret;
+ struct user_namespace *ns = inode_userns(inode);

  if (inode->i_op->permission) {
   ret = inode->i_op->permission(inode, MAY_EXEC, flags);
@@ -687,7 +692,7 @@ static inline int exec_permission(struct inode *inode, unsigned int flags)
  if (ret == -ECHILD)
   return ret;

- if (capable(CAP_DAC_OVERRIDE) || capable(CAP_DAC_READ_SEARCH))
+ if (ns_capable(ns, CAP_DAC_OVERRIDE) || ns_capable(ns, CAP_DAC_READ_SEARCH))
   goto ok;

  return ret;
@@ -1940,11 +1945,15 @@ static inline int check_sticky(struct inode *dir, struct inode *inode)

  if (!(dir->i_mode & S_ISVTX))
   return 0;
+ if (current_user_ns() != inode_userns(inode))
+  goto other_userns;
  if (inode->i_uid == fsuid)
   return 0;
```

```
  if (dir->i_uid == fsuid)
   return 0;
- return !capable(CAP_FOWNER);
+
+other_userns:
+ return !ns_capable(inode_userns(inode), CAP_FOWNER);
 }

 /*
@@ -2635,7 +2644,8 @@ int vfs_mknod(struct inode *dir, struct dentry *dentry, int mode, dev_t
dev)
  if (error)
   return error;

- if ((S_ISCHR(mode) || S_ISBLK(mode)) && !capable(CAP_MKNOD))
+ if ((S_ISCHR(mode) || S_ISBLK(mode)) &&
+     !ns_capable(inode_userns(dir), CAP_MKNOD))
   return -EPERM;

  if (!dir->i_op->mknod)
diff --git a/include/linux/fs.h b/include/linux/fs.h
index bd32159..c84417a 100644
--- a/include/linux/fs.h
+++ b/include/linux/fs.h
@@ -1446,8 +1446,13 @@ enum {
 #define put_fs_excl() atomic_dec(&current->fs_excl)
 #define has_fs_excl() atomic_read(&current->fs_excl)

-#define is_owner_or_cap(inode) \
- ((current_fsuid() == (inode)->i_uid) || capable(CAP_FOWNER))
+/*
+ * until VFS tracks user namespaces for inodes, just make all files
+ * belong to init_user_ns
+ */
+extern struct user_namespace init_user_ns;
+#define inode_userns(inode) (&init_user_ns)
+extern int is_owner_or_cap(const struct inode *inode);

 /* not quite ready to be deprecated, but... */
 extern void lock_super(struct super_block *);
--
1.7.0.4
```

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

Subject: Re: userns: targeted capabilities v5
Posted by akpm on Fri, 18 Feb 2011 00:21:46 GMT
View Forum Message <> Reply to Message

On Thu, 17 Feb 2011 15:02:24 +0000
"Serge E. Hallyn" <serge@hallyn.com> wrote:

> Here is a repost of my previous user namespace patch, ported onto
> last night's git head.
>
> It fixes several things I was doing wrong in the last (v4)
> posting, in particular:
>
> 1. don't set uts_ns->user_ns to current's when !CLONE_NEWUTS
> 2. add a ipc_ns->user_ns which owns ipc_ns, and use that to
> decide CAP_IPC_OWNER
> 3. fix logic flaw caused by bad parantheses
> 4. allow do_prlimit to current
> 5. don't always give root full privs to init_user_ns
>
> The expected course of development for user namespaces is laid out
> at https://wiki.ubuntu.com/UserNamespace.

Seems like a nice feature to be developing.

I worry about the maturity of it all at this stage. How far along is
it *really*?

Is anyone else working with you on developing and reviewing this work?

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

Subject: Re: [PATCH 9/9] userns: check user namespace for task->file uid equivalence checks
Posted by ebiederm on Fri, 18 Feb 2011 01:29:32 GMT
View Forum Message <> Reply to Message

"Serge E. Hallyn" <serge@hallyn.com> writes:

> Cheat for now and say all files belong to init_user_ns. Next
> step will be to let superblocks belong to a user_ns, and derive
> inode_userns(inode) from inode->i_sb->s_user_ns. Finally we'll
> introduce more flexible arrangements.

This looks good. I am a little worried that a concept like
inode_user_ns will imply that there is only ever one.

However this looks like a good place to start and it will only
be strange filesystems that implement a notion of permissions
that is namespace aware so I don't expect the generic code
needs to handle that case other than allowing the permission checks
to be overridden.

Acked-by: "Eric W. Biederman" <ebiederm@xmission.com>

>
> Changelog:
> Feb 15: make is_owner_or_cap take const struct inode
>
> Signed-off-by: Serge E. Hallyn <serge.hallyn@canonical.com>
> ---
> fs/inode.c | 17 ++++++++++++++++
> fs/namei.c | 20 ++++++++++++++++++-----
> include/linux/fs.h | 9 +++++++--
> 3 files changed, 39 insertions(+), 7 deletions(-)
>
> diff --git a/fs/inode.c b/fs/inode.c
> index da85e56..1930b45 100644
> --- a/fs/inode.c
> +++ b/fs/inode.c
> @@ -25,6 +25,7 @@
> #include <linux/async.h>
> #include <linux/posix_acl.h>
> #include <linux/ima.h>
> +#include <linux/cred.h>
>
> /*
>   * This is needed for the following functions:
> @@ -1722,3 +1723,19 @@ void inode_init_owner(struct inode *inode, const struct inode *dir,
>   inode->i_mode = mode;
> }
> EXPORT_SYMBOL(inode_init_owner);
> +
> +/*
> + * return 1 if current either has CAP_FOWNER to the
> + * file, or owns the file.
> + */
> +int is_owner_or_cap(const struct inode *inode)
> +{
> + struct user_namespace *ns = inode_userns(inode);
> +
> + if (current_user_ns() == ns && current_fsuid() == inode->i_uid)

> + return 1;
> + if (ns_capable(ns, CAP_FOWNER))
> + return 1;
> + return 0;
> +}
> +EXPORT_SYMBOL(is_owner_or_cap);
> diff --git a/fs/namei.c b/fs/namei.c
> index 9e701e2..cfac5b4 100644
> --- a/fs/namei.c
> +++ b/fs/namei.c
> @@ -176,6 +176,9 @@ static int acl_permission_check(struct inode *inode, int mask, unsigned
int flag
>
>   mask &= MAY_READ | MAY_WRITE | MAY_EXEC;
>
> + if (current_user_ns() != inode_userns(inode))
> + goto other_perms;
> +
>   if (current_fsuid() == inode->i_uid)
>   mode >>= 6;
>   else {
> @@ -189,6 +192,7 @@ static int acl_permission_check(struct inode *inode, int mask, unsigned
int flag
>   mode >>= 3;
>   }
>
> +other_perms:
>   /*
>   * If the DACs are ok we don't need any capability check.
>   */
> @@ -230,7 +234,7 @@ int generic_permission(struct inode *inode, int mask, unsigned int
flags,
>   * Executable DACs are overridable if at least one exec bit is set.
>   */
>   if (!(mask & MAY_EXEC) || execute_ok(inode))
> - if (capable(CAP_DAC_OVERRIDE))
> + if (ns_capable(inode_userns(inode), CAP_DAC_OVERRIDE))
>   return 0;
>
>   /*
> @@ -238,7 +242,7 @@ int generic_permission(struct inode *inode, int mask, unsigned int
flags,
>   */
>   mask &= MAY_READ | MAY_WRITE | MAY_EXEC;
>   if (mask == MAY_READ || (S_ISDIR(inode->i_mode) && !(mask & MAY_WRITE)))
> - if (capable(CAP_DAC_READ_SEARCH))
> + if (ns_capable(inode_userns(inode), CAP_DAC_READ_SEARCH))
>   return 0;

>
> return -EACCES;
> @@ -675,6 +679,7 @@ force_reval_path(struct path *path, struct nameidata *nd)
> static inline int exec_permission(struct inode *inode, unsigned int flags)
> {
> int ret;
> + struct user_namespace *ns = inode_userns(inode);
>
> if (inode->i_op->permission) {
> ret = inode->i_op->permission(inode, MAY_EXEC, flags);
> @@ -687,7 +692,7 @@ static inline int exec_permission(struct inode *inode, unsigned int flags)
> if (ret == -ECHILD)
> return ret;
>
> - if (capable(CAP_DAC_OVERRIDE) || capable(CAP_DAC_READ_SEARCH))
> + if (ns_capable(ns, CAP_DAC_OVERRIDE) || ns_capable(ns, CAP_DAC_READ_SEARCH))
> goto ok;
>
> return ret;
> @@ -1940,11 +1945,15 @@ static inline int check_sticky(struct inode *dir, struct inode *inode)
>
> if (!(dir->i_mode & S_ISVTX))
> return 0;
> + if (current_user_ns() != inode_userns(inode))
> + goto other_userns;
> if (inode->i_uid == fsuid)
> return 0;
> if (dir->i_uid == fsuid)
> return 0;
> - return !capable(CAP_FOWNER);
> +
> +other_userns:
> + return !ns_capable(inode_userns(inode), CAP_FOWNER);
> }
>
> /*
> @@ -2635,7 +2644,8 @@ int vfs_mknod(struct inode *dir, struct dentry *dentry, int mode, dev_t
dev)
> if (error)
> return error;
>
> - if ((S_ISCHR(mode) || S_ISBLK(mode)) && !capable(CAP_MKNOD))
> + if ((S_ISCHR(mode) || S_ISBLK(mode)) &&
> +   !ns_capable(inode_userns(dir), CAP_MKNOD))
> return -EPERM;
>
> if (!dir->i_op->mknod)
> diff --git a/include/linux/fs.h b/include/linux/fs.h

> index bd32159..c84417a 100644
> --- a/include/linux/fs.h
> +++ b/include/linux/fs.h
> @@ -1446,8 +1446,13 @@ enum {
> #define put_fs_excl() atomic_dec(&current->fs_excl)
> #define has_fs_excl() atomic_read(&current->fs_excl)
>
> -#define is_owner_or_cap(inode) \
> - ((current_fsuid() == (inode)->i_uid) || capable(CAP_FOWNER))
> +/*
> + * until VFS tracks user namespaces for inodes, just make all files
> + * belong to init_user_ns
> + */
> +extern struct user_namespace init_user_ns;
> +#define inode_userns(inode) (&init_user_ns)
> +extern int is_owner_or_cap(const struct inode *inode);
>
> /* not quite ready to be deprecated, but... */
> extern void lock_super(struct super_block *);

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containe rs

---

Subject: Re: [PATCH 8/9] user namespaces: convert several capable() calls
Posted by ebiederm on Fri, 18 Feb 2011 01:51:25 GMT
View Forum Message <> Reply to Message

"Serge E. Hallyn" <serge@hallyn.com> writes:

> CAP_IPC_OWNER and CAP_IPC_LOCK can be checked against current_user_ns(),
> because the resource comes from current's own ipc namespace.
>
> setuid/setgid are to uids in own namespace, so again checks can be
> against current_user_ns().

Some nits below. But this generally looks good if a little bit all over
the map for a single patch.

Acked-by: "Eric W. Biederman" <ebiederm@xmission.com>

> Changelog:
> Jan 11: Use task_ns_capable() in place of sched_capable().
> Jan 11: Use nsown_capable() as suggested by Bastian Blank.
> Jan 11: Clarify (hopefully) some logic in futex and sched.c
> Feb 15: use ns_capable for ipc, not nsown_capable

>
> Signed-off-by: Serge E. Hallyn <serge.hallyn@canonical.com>
> ---
> ipc/shm.c | 2 +-
> ipc/util.c | 5 +++--
> kernel/futex.c | 11 ++++++++++-
> kernel/futex_compat.c | 11 ++++++++++-
> kernel/groups.c | 2 +-
> kernel/sched.c | 9 ++++++---
> kernel/uid16.c | 2 +-
> 7 files changed, 32 insertions(+), 10 deletions(-)
>
> diff --git a/ipc/shm.c b/ipc/shm.c
> index 7d3bb22..e91e2e9 100644
> --- a/ipc/shm.c
> +++ b/ipc/shm.c
> @@ -773,7 +773,7 @@ SYSCALL_DEFINE3(shmctl, int, shmid, int, cmd, struct shmid_ds
__user *, buf)
>
>   audit_ipc_obj(&(shp->shm_perm));
>
> - if (!capable(CAP_IPC_LOCK)) {
> + if (!ns_capable(ns->user_ns, CAP_IPC_LOCK)) {
>   uid_t euid = current_euid();
>   err = -EPERM;
>   if (euid != shp->shm_perm.uid &&
> diff --git a/ipc/util.c b/ipc/util.c
> index 69a0cc1..8e7ec6a 100644
> --- a/ipc/util.c
> +++ b/ipc/util.c
> @@ -627,7 +627,7 @@ int ipcperms (struct kern_ipc_perm *ipcp, short flag)
>   granted_mode >>= 3;
>   /* is there some bit set in requested_mode but not in granted_mode? */
>   if ((requested_mode & ~granted_mode & 0007) &&
> -   !capable(CAP_IPC_OWNER))
> +   !ns_capable(current->nsproxy->ipc_ns->user_ns, CAP_IPC_OWNER))
>   return -1;

Serge can we please modify the code to pass the ns down from
ipcget_public to ipcperms. It is passed in and dropping the value and
going back to current to get it just feels wrong.

Strictly speaking this code is correct but it requires an audit of all
of the callers to know that, which is unfortunate.

>   return security_ipc_permission(ipcp, flag);
> @@ -800,7 +800,8 @@ struct kern_ipc_perm *ipcctl_pre_down(struct ipc_ids *ids, int id, int
cmd,

>
>   euid = current_euid();
>   if (euid == ipcp->cuid ||
> -   euid == ipcp->uid || capable(CAP_SYS_ADMIN))
> +   euid == ipcp->uid ||
> +   ns_capable(current->nsproxy->ipc_ns->user_ns, CAP_SYS_ADMIN))
>   return ipcp;

Like the other ipc call can we please pass the ipc_ns into ipcctl_pre_down.

The code as constructed appears correct but because we are passing the
namespace into the caller of this function always using current to get
the ipc_ns seems confusing and unnecessary.

>   err = -EPERM;
> diff --git a/kernel/futex.c b/kernel/futex.c
> index b766d28..1e876f1 100644
> --- a/kernel/futex.c
> +++ b/kernel/futex.c
> @@ -2421,10 +2421,19 @@ SYSCALL_DEFINE3(get_robust_list, int, pid,
>   goto err_unlock;
>   ret = -EPERM;
>   pcred = __task_cred(p);
> + /* If victim is in different user_ns, then uids are not
> +   comparable, so we must have CAP_SYS_PTRACE */
> + if (cred->user->user_ns != pcred->user->user_ns) {
> +   if (!ns_capable(pcred->user->user_ns, CAP_SYS_PTRACE))
> +   goto err_unlock;
> +   goto ok;
> + }
> + /* If victim is in same user_ns, then uids are comparable */
>   if (cred->euid != pcred->euid &&
>   cred->euid != pcred->uid &&
> -   !capable(CAP_SYS_PTRACE))
> +   !ns_capable(pcred->user->user_ns, CAP_SYS_PTRACE))
>   goto err_unlock;
> +ok:
>   head = p->robust_list;
>   rcu_read_unlock();
>   }
> diff --git a/kernel/futex_compat.c b/kernel/futex_compat.c
> index a7934ac..5f9e689 100644
> --- a/kernel/futex_compat.c
> +++ b/kernel/futex_compat.c
> @@ -153,10 +153,19 @@ compat_sys_get_robust_list(int pid, compat_uptr_t __user
*head_ptr,
>   goto err_unlock;
>   ret = -EPERM;

```
>    pcred = __task_cred(p);
> +  /* If victim is in different user_ns, then uids are not
> +     comparable, so we must have CAP_SYS_PTRACE */
> +  if (cred->user->user_ns != pcred->user->user_ns) {
> +   if (!ns_capable(pcred->user->user_ns, CAP_SYS_PTRACE))
> +    goto err_unlock;
> +   goto ok;
> +  }
> +  /* If victim is in same user_ns, then uids are comparable */
>    if (cred->euid != pcred->euid &&
>        cred->euid != pcred->uid &&
> -      !capable(CAP_SYS_PTRACE))
> +      !ns_capable(pcred->user->user_ns, CAP_SYS_PTRACE))
>     goto err_unlock;
> +ok:
>    head = p->compat_robust_list;
>    rcu_read_unlock();
>   }
> diff --git a/kernel/groups.c b/kernel/groups.c
> index 253dc0f..1cc476d 100644
> --- a/kernel/groups.c
> +++ b/kernel/groups.c
> @@ -233,7 +233,7 @@ SYSCALL_DEFINE2(setgroups, int, gidsetsize, gid_t __user *,
grouplist)
>   struct group_info *group_info;
>   int retval;
>
> - if (!capable(CAP_SETGID))
> + if (!nsown_capable(CAP_SETGID))
>    return -EPERM;
>   if ((unsigned)gidsetsize > NGROUPS_MAX)
>    return -EINVAL;
> diff --git a/kernel/sched.c b/kernel/sched.c
> index 18d38e4..dc12bc2 100644
> --- a/kernel/sched.c
> +++ b/kernel/sched.c
> @@ -4761,8 +4761,11 @@ static bool check_same_owner(struct task_struct *p)
>
>   rcu_read_lock();
>   pcred = __task_cred(p);
> - match = (cred->euid == pcred->euid ||
> -   cred->euid == pcred->uid);
> + if (cred->user->user_ns == pcred->user->user_ns)
> +  match = (cred->euid == pcred->euid ||
> +    cred->euid == pcred->uid);
> + else
> +  match = false;
>   rcu_read_unlock();
```

> return match;
> }
> @@ -5088,7 +5091,7 @@ long sched_setaffinity(pid_t pid, const struct cpumask *in_mask)
>   goto out_free_cpus_allowed;
> }
> retval = -EPERM;
> - if (!check_same_owner(p) && !capable(CAP_SYS_NICE))
> + if (!check_same_owner(p) && !task_ns_capable(p, CAP_SYS_NICE))
>   goto out_unlock;
>
> retval = security_task_setscheduler(p);
> diff --git a/kernel/uid16.c b/kernel/uid16.c
> index 4192098..51c6e89 100644
> --- a/kernel/uid16.c
> +++ b/kernel/uid16.c
> @@ -189,7 +189,7 @@ SYSCALL_DEFINE2(setgroups16, int, gidsetsize, old_gid_t __user *,
grouplist)
> struct group_info *group_info;
> int retval;
>
> - if (!capable(CAP_SETGID))
> + if (!nsown_capable(CAP_SETGID))
>   return -EPERM;
> if ((unsigned)gidsetsize > NGROUPS_MAX)
>   return -EINVAL;

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

---

Subject: Re: [PATCH 6/9] user namespaces: convert all capable checks in kernel/sys.c

Posted by ebiederm on Fri, 18 Feb 2011 01:57:15 GMT

View Forum Message <> Reply to Message

"Serge E. Hallyn" <serge@hallyn.com> writes:

> This allows setuid/setgid in containers. It also fixes some
> corner cases where kernel logic foregoes capability checks when
> uids are equivalent. The latter will need to be done throughout
> the whole kernel.

Except for the extra printk in sethostname this looks good.

Acked-by: "Eric W. Biederman" <ebiederm@xmission.com>

>

```
> Changelog:
>  Jan 11: Use nsown_capable() as suggested by Bastian Blank.
>  Jan 11: Fix logic errors in uid checks pointed out by Bastian.
>  Feb 15: allow prlimit to current (was regression in previous version)
>
> Signed-off-by: Serge E. Hallyn <serge.hallyn@canonical.com>
> ---
>  kernel/sys.c |   74 ++++++
```

```
>   if (rgid != (gid_t) -1) {
>    if (old->gid == rgid ||
>        old->egid == rgid ||
> -      capable(CAP_SETGID))
> +      nsown_capable(CAP_SETGID))
>     new->gid = rgid;
>    else
>     goto error;
> @@ -511,7 +523,7 @@ SYSCALL_DEFINE2(setregid, gid_t, rgid, gid_t, egid)
>    if (old->gid == egid ||
>        old->egid == egid ||
>        old->sgid == egid ||
> -      capable(CAP_SETGID))
> +      nsown_capable(CAP_SETGID))
>     new->egid = egid;
>    else
>     goto error;
> @@ -546,7 +558,7 @@ SYSCALL_DEFINE1(setgid, gid_t, gid)
>   old = current_cred();
>
>   retval = -EPERM;
> - if (capable(CAP_SETGID))
> + if (nsown_capable(CAP_SETGID))
>    new->gid = new->egid = new->sgid = new->fsgid = gid;
>   else if (gid == old->gid || gid == old->sgid)
>    new->egid = new->fsgid = gid;
> @@ -613,7 +625,7 @@ SYSCALL_DEFINE2(setreuid, uid_t, ruid, uid_t, euid)
>    new->uid = ruid;
>    if (old->uid != ruid &&
>        old->euid != ruid &&
> -      !capable(CAP_SETUID))
> +      !nsown_capable(CAP_SETUID))
>     goto error;
>   }
>
> @@ -622,7 +634,7 @@ SYSCALL_DEFINE2(setreuid, uid_t, ruid, uid_t, euid)
>    if (old->uid != euid &&
>        old->euid != euid &&
>        old->suid != euid &&
> -      !capable(CAP_SETUID))
> +      !nsown_capable(CAP_SETUID))
>     goto error;
>   }
>
> @@ -670,7 +682,7 @@ SYSCALL_DEFINE1(setuid, uid_t, uid)
>   old = current_cred();
>
>   retval = -EPERM;
```

```
> - if (capable(CAP_SETUID)) {
> + if (nsown_capable(CAP_SETUID)) {
>    new->suid = new->uid = uid;
>    if (uid != old->uid) {
>     retval = set_user(new);
> @@ -712,7 +724,7 @@ SYSCALL_DEFINE3(setresuid, uid_t, ruid, uid_t, euid, uid_t, suid)
>   old = current_cred();
>
>   retval = -EPERM;
> - if (!capable(CAP_SETUID)) {
> + if (!nsown_capable(CAP_SETUID)) {
>    if (ruid != (uid_t) -1 && ruid != old->uid &&
>        ruid != old->euid  && ruid != old->suid)
>     goto error;
> @@ -776,7 +788,7 @@ SYSCALL_DEFINE3(setresgid, gid_t, rgid, gid_t, egid, gid_t, sgid)
>   old = current_cred();
>
>   retval = -EPERM;
> - if (!capable(CAP_SETGID)) {
> + if (!nsown_capable(CAP_SETGID)) {
>    if (rgid != (gid_t) -1 && rgid != old->gid &&
>        rgid != old->egid  && rgid != old->sgid)
>     goto error;
> @@ -836,7 +848,7 @@ SYSCALL_DEFINE1(setfsuid, uid_t, uid)
>
>   if (uid == old->uid  || uid == old->euid  ||
>       uid == old->suid || uid == old->fsuid ||
> -     capable(CAP_SETUID)) {
> +     nsown_capable(CAP_SETUID)) {
>    if (uid != old_fsuid) {
>     new->fsuid = uid;
>     if (security_task_fix_setuid(new, old, LSM_SETID_FS) == 0)
> @@ -869,7 +881,7 @@ SYSCALL_DEFINE1(setfsgid, gid_t, gid)
>
>   if (gid == old->gid  || gid == old->egid  ||
>       gid == old->sgid || gid == old->fsgid ||
> -     capable(CAP_SETGID)) {
> +     nsown_capable(CAP_SETGID)) {
>    if (gid != old_fsgid) {
>     new->fsgid = gid;
>     goto change_okay;
> @@ -1177,8 +1189,11 @@ SYSCALL_DEFINE2(sethostname, char __user *, name, int, len)
>   int errno;
>   char tmp[__NEW_UTS_LEN];
>
> - if (!ns_capable(current->nsproxy->uts_ns->user_ns, CAP_SYS_ADMIN))
> + if (!ns_capable(current->nsproxy->uts_ns->user_ns, CAP_SYS_ADMIN)) {
> +  printk(KERN_NOTICE "%s: did not have CAP_SYS_ADMIN\n", __func__);
```

```
>    return -EPERM;
> + }
> + printk(KERN_NOTICE "%s: did have CAP_SYS_ADMIN\n", __func__);
```

Ouch! This new print statement could be really annoying if an
unprivileged user calls sethostname. Could you remove it?

```
>   if (len < 0 || len > __NEW_UTS_LEN)
>    return -EINVAL;
>   down_write(&uts_sem);
> @@ -1226,7 +1241,7 @@ SYSCALL_DEFINE2(setdomainname, char __user *, name, int, len)
>   int errno;
>   char tmp[__NEW_UTS_LEN];
>
> - if (!capable(CAP_SYS_ADMIN))
> + if (!ns_capable(current->nsproxy->uts_ns->user_ns, CAP_SYS_ADMIN))
>    return -EPERM;
>   if (len < 0 || len > __NEW_UTS_LEN)
>    return -EINVAL;
> @@ -1341,6 +1356,8 @@ int do_prlimit(struct task_struct *tsk, unsigned int resource,
>   rlim = tsk->signal->rlim + resource;
>   task_lock(tsk->group_leader);
>   if (new_rlim) {
> +  /* Keep the capable check against init_user_ns until
> +     cgroups can contain all limits */
>    if (new_rlim->rlim_max > rlim->rlim_max &&
>      !capable(CAP_SYS_RESOURCE))
>     retval = -EPERM;
> @@ -1384,19 +1401,22 @@ static int check_prlimit_permission(struct task_struct *task)
>  {
>   const struct cred *cred = current_cred(), *tcred;
>
> - tcred = __task_cred(task);
> - if (current != task &&
> -     (cred->uid != tcred->euid ||
> -      cred->uid != tcred->suid ||
> -      cred->uid != tcred->uid  ||
> -      cred->gid != tcred->egid ||
> -      cred->gid != tcred->sgid ||
> -      cred->gid != tcred->gid) &&
> -      !capable(CAP_SYS_RESOURCE)) {
> -  return -EPERM;
> - }
> + if (current == task)
> +  return 0;
>
> - return 0;
> + tcred = __task_cred(task);
```

> + if (cred->user->user_ns == tcred->user->user_ns &&
> +   (cred->uid == tcred->euid &&
> +   cred->uid == tcred->suid &&
> +   cred->uid == tcred->uid &&
> +   cred->gid == tcred->egid &&
> +   cred->gid == tcred->sgid &&
> +   cred->gid == tcred->gid))
> + return 0;
> + if (ns_capable(tcred->user->user_ns, CAP_SYS_RESOURCE))
> + return 0;
> +
> + return -EPERM;
> }
>
> SYSCALL_DEFINE4(prlimit64, pid_t, pid, unsigned int, resource,

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containe rs

---

Subject: Re: [PATCH 5/9] Allow ptrace from non-init user namespaces
Posted by ebiederm on Fri, 18 Feb 2011 02:59:57 GMT
View Forum Message <> Reply to Message

"Serge E. Hallyn" <serge@hallyn.com> writes:

> ptrace is allowed to tasks in the same user namespace according to
> the usual rules (i.e. the same rules as for two tasks in the init
> user namespace). ptrace is also allowed to a user namespace to
> which the current task the has CAP_SYS_PTRACE capability.

I don't see how it can go wrong at the moment but
same_or_ancestore_user_ns is too permissive and potentially inefficient.
Can you please replace it with a simple user namespace equality check.

Eric

> Changelog:
> Dec 31: Address feedback by Eric:
>   . Correct ptrace uid check
>   . Rename may_ptrace_ns to ptrace_capable
>   . Also fix the cap_ptrace checks.
> Jan 1: Use const cred struct
> Jan 11: use task_ns_capable() in place of ptrace_capable().
>

> Signed-off-by: Serge E. Hallyn <serge.hallyn@canonical.com>
> ---
> include/linux/capability.h | 2 +
> include/linux/user_namespace.h | 9 +++++++
> kernel/ptrace.c | 27 ++++++++++++----------
> kernel/user_namespace.c | 16 +++++++++++++
> security/commoncap.c | 48 ++++++++++++++++++++++++++++++++++------
> 5 files changed, 82 insertions(+), 20 deletions(-)
>
> diff --git a/include/linux/capability.h b/include/linux/capability.h
> index cb3d2d9..bc0f262 100644
> --- a/include/linux/capability.h
> +++ b/include/linux/capability.h
> @@ -546,6 +546,8 @@ extern const kernel_cap_t __cap_init_eff_set;
> */
> #define has_capability(t, cap) (security_real_capable((t), &init_user_ns, (cap)) == 0)
>
> +#define has_ns_capability(t, ns, cap) (security_real_capable((t), (ns), (cap)) == 0)
> +
> /**
> * has_capability_noaudit - Determine if a task has a superior capability available (unaudited)
> * @t: The task in question
> diff --git a/include/linux/user_namespace.h b/include/linux/user_namespace.h
> index faf4679..862fc59 100644
> --- a/include/linux/user_namespace.h
> +++ b/include/linux/user_namespace.h
> @@ -39,6 +39,9 @@ static inline void put_user_ns(struct user_namespace *ns)
> uid_t user_ns_map_uid(struct user_namespace *to, const struct cred *cred, uid_t uid);
> gid_t user_ns_map_gid(struct user_namespace *to, const struct cred *cred, gid_t gid);
>
> +int same_or_ancestor_user_ns(struct task_struct *task,
> + struct task_struct *victim);
> +
> #else
>
> static inline struct user_namespace *get_user_ns(struct user_namespace *ns)
> @@ -66,6 +69,12 @@ static inline gid_t user_ns_map_gid(struct user_namespace *to,
> return gid;
> }
>
> +static inline int same_or_ancestor_user_ns(struct task_struct *task,
> + struct task_struct *victim)
> +{
> + return 1;
> +}
> +
> #endif
>

```
>  #endif /* _LINUX_USER_H */
> diff --git a/kernel/ptrace.c b/kernel/ptrace.c
> index 1708b1e..cde4655 100644
> --- a/kernel/ptrace.c
> +++ b/kernel/ptrace.c
> @@ -134,21 +134,24 @@ int __ptrace_may_access(struct task_struct *task, unsigned int
mode)
>    return 0;
>   rcu_read_lock();
>   tcred = __task_cred(task);
> - if ((cred->uid != tcred->euid ||
> -      cred->uid != tcred->suid ||
> -      cred->uid != tcred->uid  ||
> -      cred->gid != tcred->egid ||
> -      cred->gid != tcred->sgid ||
> -      cred->gid != tcred->gid) &&
> -     !capable(CAP_SYS_PTRACE)) {
> -  rcu_read_unlock();
> -  return -EPERM;
> - }
> + if (cred->user->user_ns == tcred->user->user_ns &&
> +     (cred->uid == tcred->euid &&
> +      cred->uid == tcred->suid &&
> +      cred->uid == tcred->uid  &&
> +      cred->gid == tcred->egid &&
> +      cred->gid == tcred->sgid &&
> +      cred->gid == tcred->gid))
> +  goto ok;
> + if (ns_capable(tcred->user->user_ns, CAP_SYS_PTRACE))
> +  goto ok;
> + rcu_read_unlock();
> + return -EPERM;
> +ok:
>   rcu_read_unlock();
>   smp_rmb();
>   if (task->mm)
>    dumpable = get_dumpable(task->mm);
> - if (!dumpable && !capable(CAP_SYS_PTRACE))
> + if (!dumpable && !task_ns_capable(task, CAP_SYS_PTRACE))
>    return -EPERM;
>
>   return security_ptrace_access_check(task, mode);
> @@ -198,7 +201,7 @@ int ptrace_attach(struct task_struct *task)
>    goto unlock_tasklist;
>
>   task->ptrace = PT_PTRACED;
> - if (capable(CAP_SYS_PTRACE))
> + if (task_ns_capable(task, CAP_SYS_PTRACE))
```

>   task->ptrace |= PT_PTRACE_CAP;
>
>   __ptrace_link(task, current);
> diff --git a/kernel/user_namespace.c b/kernel/user_namespace.c
> index 9da289c..0ef2258 100644
> --- a/kernel/user_namespace.c
> +++ b/kernel/user_namespace.c
> @@ -129,6 +129,22 @@ gid_t user_ns_map_gid(struct user_namespace *to, const struct cred
*cred, gid_t
>   return overflowgid;
> }
>
> +int same_or_ancestor_user_ns(struct task_struct *task,
> + struct task_struct *victim)
> +{
> + struct user_namespace *u1 = task_cred_xxx(task, user)->user_ns;
> + struct user_namespace *u2 = task_cred_xxx(victim, user)->user_ns;
> + for (;;) {
> + if (u1 == u2)
> +   return 1;
> + if (u1 == &init_user_ns)
> +   return 0;
> + u1 = u1->creator->user_ns;
> + }
> + /* We never get here */
> + return 0;
> +}
> +
> static __init int user_namespaces_init(void)
> {
>   user_ns_cachep = KMEM_CACHE(user_namespace, SLAB_PANIC);
> diff --git a/security/commoncap.c b/security/commoncap.c
> index 51fa9ec..12ff65c 100644
> --- a/security/commoncap.c
> +++ b/security/commoncap.c
> @@ -130,18 +130,34 @@ int cap_settime(struct timespec *ts, struct timezone *tz)
>   * @child: The process to be accessed
>   * @mode: The mode of attachment.
>   *
> + * If we are in the same or an ancestor user_ns and have all the target
> + * task's capabilities, then ptrace access is allowed.
> + * If we have the ptrace capability to the target user_ns, then ptrace
> + * access is allowed.
> + * Else denied.
> + *
>   * Determine whether a process may access another, returning 0 if permission
>   * granted, -ve if denied.
>   */

> int cap_ptrace_access_check(struct task_struct *child, unsigned int mode)
> {
>   int ret = 0;
> + const struct cred *cred, *tcred;
>
>   rcu_read_lock();
> - if (!cap_issubset(__task_cred(child)->cap_permitted,
> -   current_cred()->cap_permitted) &&
> -   !capable(CAP_SYS_PTRACE))
> - ret = -EPERM;
> + cred = current_cred();
> + tcred = __task_cred(child);
> + /*
> + * The ancestor user_ns check may be gratuitous, as I think
> + * we've already guaranteed that in kernel/ptrace.c.
> + */
> + if (same_or_ancestor_user_ns(current, child) &&
> +   cap_issubset(tcred->cap_permitted, cred->cap_permitted))
> + goto out;

I have commented on this before but I took a good hard look this time,
and can comment more intelligently.

The cap_issubset check is for the case where we don't use the
CAP_SYS_PTRACE capability, as such is only valid in the same user
namespace. Furthermore capabilities really are not comparable between
different user namespaces. So can you please replace the
same_or_ancestor_user_ns with a simple namespace equality check.
Having the wrong logic in here will just be confusing in the future.

Also could you name tcred child_cread I think that would be clearer in
the test below.

> + if (ns_capable(tcred->user->user_ns, CAP_SYS_PTRACE))
> + goto out;
> + ret = -EPERM;
> +out:
>   rcu_read_unlock();
>   return ret;

I also find it strange that we allow CAP_SYS_PTRACE to allow us to
ptrace processes with more capabilities than ourselves. But that is an
entirely different issue.

> }
> @@ -150,18 +166,34 @@ int cap_ptrace_access_check(struct task_struct *child, unsigned int
mode)
>   * cap_ptrace_traceme - Determine whether another process may trace the current

```
>   * @parent: The task proposed to be the tracer
>   *
> + * If parent is in the same or an ancestor user_ns and has all current's
> + * capabilities, then ptrace access is allowed.
> + * If parent has the ptrace capability to current's user_ns, then ptrace
> + * access is allowed.
> + * Else denied.
> + *
>   * Determine whether the nominated task is permitted to trace the current
>   * process, returning 0 if permission is granted, -ve if denied.
>   */
>  int cap_ptrace_traceme(struct task_struct *parent)
>  {
>   int ret = 0;
> + const struct cred *cred, *tcred;
>
>   rcu_read_lock();
> - if (!cap_issubset(current_cred()->cap_permitted,
> -     __task_cred(parent)->cap_permitted) &&
> -     !has_capability(parent, CAP_SYS_PTRACE))
> -  ret = -EPERM;
> + cred = __task_cred(parent);
> + tcred = current_cred();
> + /*
> +  * The ancestor user_ns check may be gratuitous, as I think
> +  * we've already guaranteed that in kernel/ptrace.c.
> +  */
> + if (same_or_ancestor_user_ns(parent, current) &&
> +     cap_issubset(tcred->cap_permitted, cred->cap_permitted))
> +  goto out;
> + if (has_ns_capability(parent, tcred->user->user_ns, CAP_SYS_PTRACE))
> +  goto out;
> + ret = -EPERM;
> +out:
>   rcu_read_unlock();
>   return ret;
>  }
```

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containe rs

---

Subject: Re: [PATCH 4/9] allow killing tasks in your own or child userns
Posted by ebiederm on Fri, 18 Feb 2011 03:00:44 GMT
View Forum Message <> Reply to Message

"Serge E. Hallyn" <serge@hallyn.com> writes:

> Changelog:
> Dec 8: Fixed bug in my check_kill_permission pointed out by
>   Eric Biederman.
> Dec 13: Apply Eric's suggestion to pass target task into kill_ok_by_cred()
>   for clarity
> Dec 31: address comment by Eric Biederman:
>   don't need cred/tcred in check_kill_permission.
> Jan 1: use const cred struct.
> Jan 11: Per Bastian Blank's advice, clean up kill_ok_by_cred().
> Feb 16: kill_ok_by_cred: fix bad parentheses

Acked-by: "Eric W. Biederman" <ebiederm@xmission.com>

>
> Signed-off-by: Serge E. Hallyn <serge.hallyn@canonical.com>
> ---
> kernel/signal.c | 30 ++++++++++++++++++++++--------
> 1 files changed, 22 insertions(+), 8 deletions(-)
>
> diff --git a/kernel/signal.c b/kernel/signal.c
> index 4e3cff1..ffe4bdf 100644
> --- a/kernel/signal.c
> +++ b/kernel/signal.c
> @@ -636,13 +636,33 @@ static inline bool si_fromuser(const struct siginfo *info)
> }
>
> /*
> + * called with RCU read lock from check_kill_permission()
> + */
> +static inline int kill_ok_by_cred(struct task_struct *t)
> +{
> + const struct cred *cred = current_cred();
> + const struct cred *tcred = __task_cred(t);
> +
> + if (cred->user->user_ns == tcred->user->user_ns &&
> +   (cred->euid == tcred->suid ||
> +   cred->euid == tcred->uid ||
> +   cred->uid == tcred->suid ||
> +   cred->uid == tcred->uid))
> + return 1;
> +
> + if (ns_capable(tcred->user->user_ns, CAP_KILL))
> + return 1;
> +
> + return 0;
> +}
> +

> +/*
> * Bad permissions for sending the signal
> * - the caller must hold the RCU read lock
> */
> static int check_kill_permission(int sig, struct siginfo *info,
>   struct task_struct *t)
> {
> - const struct cred *cred, *tcred;
> struct pid *sid;
> int error;
>
> @@ -656,14 +676,8 @@ static int check_kill_permission(int sig, struct siginfo *info,
> if (error)
>   return error;
>
> - cred = current_cred();
> - tcred = __task_cred(t);
> if (!same_thread_group(current, t) &&
> -   (cred->euid ^ tcred->suid) &&
> -   (cred->euid ^ tcred->uid) &&
> -   (cred->uid ^ tcred->suid) &&
> -   (cred->uid ^ tcred->uid) &&
> -   !capable(CAP_KILL)) {
> +   !kill_ok_by_cred(t)) {
>   switch (sig) {
>   case SIGCONT:
>   sid = task_session(t);

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containe rs

---

Subject: Re: [PATCH 3/9] allow sethostname in a container
Posted by ebiederm on Fri, 18 Feb 2011 03:05:21 GMT
View Forum Message <> Reply to Message

"Serge E. Hallyn" <serge@hallyn.com> writes:

Acked-by: "Eric W. Biederman" <ebiederm@xmission.com>

> Signed-off-by: Serge E. Hallyn <serge.hallyn@canonical.com>
> ---
> kernel/sys.c | 2 +-
> 1 files changed, 1 insertions(+), 1 deletions(-)
>
> diff --git a/kernel/sys.c b/kernel/sys.c

> index 18da702..7a1bbad 100644
> --- a/kernel/sys.c
> +++ b/kernel/sys.c
> @@ -1177,7 +1177,7 @@ SYSCALL_DEFINE2(sethostname, char __user *, name, int, len)
> int errno;
> char tmp[__NEW_UTS_LEN];
>
> - if (!capable(CAP_SYS_ADMIN))
> + if (!ns_capable(current->nsproxy->uts_ns->user_ns, CAP_SYS_ADMIN))
> return -EPERM;
> if (len < 0 || len > __NEW_UTS_LEN)
> return -EINVAL;

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

---

Subject: Re: [PATCH 7/9] add a user namespace owner of ipc ns
Posted by ebiederm on Fri, 18 Feb 2011 03:19:58 GMT
View Forum Message <> Reply to Message

"Serge E. Hallyn" <serge@hallyn.com> writes:

> Changelog:
> Feb 15: Don't set new ipc->user_ns if we didn't create a new
> ipc_ns.

Acked-by: "Eric W. Biederman" <ebiederm@xmission.com>

>
> Signed-off-by: Serge E. Hallyn <serge.hallyn@canonical.com>
> ---
> include/linux/ipc_namespace.h | 3 +++
> ipc/msgutil.c | 3 +++
> ipc/namespace.c | 9 +++++++--
> kernel/nsproxy.c | 5 +++++
> 4 files changed, 18 insertions(+), 2 deletions(-)
>
> diff --git a/include/linux/ipc_namespace.h b/include/linux/ipc_namespace.h
> index 5195298..46d2eb4 100644
> --- a/include/linux/ipc_namespace.h
> +++ b/include/linux/ipc_namespace.h
> @@ -24,6 +24,7 @@ struct ipc_ids {
> struct idr ipcs_idr;
> };
>
> +struct user_namespace;

> struct ipc_namespace {
>   atomic_t count;
>   struct ipc_ids ids[3];
> @@ -56,6 +57,8 @@ struct ipc_namespace {
>   unsigned int   mq_msg_max;   /* initialized to DFLT_MSGMAX */
>   unsigned int   mq_msgsize_max; /* initialized to DFLT_MSGSIZEMAX */
>
> + /* user_ns which owns the ipc ns */
> + struct user_namespace *user_ns;
> };
>
> extern struct ipc_namespace init_ipc_ns;
> diff --git a/ipc/msgutil.c b/ipc/msgutil.c
> index f095ee2..d91ff4b 100644
> --- a/ipc/msgutil.c
> +++ b/ipc/msgutil.c
> @@ -20,6 +20,8 @@
>
> DEFINE_SPINLOCK(mq_lock);
>
> +extern struct user_namespace init_user_ns;
> +
> /*
>   * The next 2 defines are here bc this is the only file
>   * compiled when either CONFIG_SYSVIPC and CONFIG_POSIX_MQUEUE
> @@ -32,6 +34,7 @@ struct ipc_namespace init_ipc_ns = {
>   .mq_msg_max   = DFLT_MSGMAX,
>   .mq_msgsize_max = DFLT_MSGSIZEMAX,
> #endif
> + .user_ns = &init_user_ns,
> };
>
> atomic_t nr_ipc_ns = ATOMIC_INIT(1);
> diff --git a/ipc/namespace.c b/ipc/namespace.c
> index a1094ff..aa18899 100644
> --- a/ipc/namespace.c
> +++ b/ipc/namespace.c
> @@ -11,10 +11,11 @@
> #include <linux/slab.h>
> #include <linux/fs.h>
> #include <linux/mount.h>
> +#include <linux/user_namespace.h>
>
> #include "util.h"
>
> -static struct ipc_namespace *create_ipc_ns(void)
> +static struct ipc_namespace *create_ipc_ns(struct ipc_namespace *old_ns)
> {

```
>   struct ipc_namespace *ns;
>   int err;
> @@ -43,6 +44,9 @@ static struct ipc_namespace *create_ipc_ns(void)
>   ipcns_notify(IPCNS_CREATED);
>   register_ipcns_notifier(ns);
>
> + ns->user_ns = old_ns->user_ns;
> + get_user_ns(ns->user_ns);
> +
>   return ns;
>  }
>
> @@ -50,7 +54,7 @@ struct ipc_namespace *copy_ipcs(unsigned long flags, struct
ipc_namespace *ns)
>  {
>   if (!(flags & CLONE_NEWIPC))
>    return get_ipc_ns(ns);
> - return create_ipc_ns();
> + return create_ipc_ns(ns);
>  }
>
>  /*
> @@ -105,6 +109,7 @@ static void free_ipc_ns(struct ipc_namespace *ns)
>    * order to have a correct value when recomputing msgmni.
>    */
>   ipcns_notify(IPCNS_REMOVED);
> + put_user_ns(ns->user_ns);
>  }
>
>  /*
> diff --git a/kernel/nsproxy.c b/kernel/nsproxy.c
> index 034dc2e..b6dbff2 100644
> --- a/kernel/nsproxy.c
> +++ b/kernel/nsproxy.c
> @@ -85,6 +85,11 @@ static struct nsproxy *create_new_namespaces(unsigned long flags,
>    err = PTR_ERR(new_nsp->ipc_ns);
>    goto out_ipc;
>   }
> + if (new_nsp->ipc_ns != tsk->nsproxy->ipc_ns) {
> +  put_user_ns(new_nsp->ipc_ns->user_ns);
> +  new_nsp->ipc_ns->user_ns = task_cred_xxx(tsk, user)->user_ns;
> +  get_user_ns(new_nsp->ipc_ns->user_ns);
> + }
>
>   new_nsp->pid_ns = copy_pid_ns(flags, task_active_pid_ns(tsk));
>   if (IS_ERR(new_nsp->pid_ns)) {
```

_____

Containers mailing list

Subject: Re: [PATCH 1/9] Add a user_namespace as creator/owner of uts_namespace
Posted by ebiederm on Fri, 18 Feb 2011 03:31:04 GMT
View Forum Message <> Reply to Message

"Serge E. Hallyn" <serge@hallyn.com> writes:

> copy_process() handles CLONE_NEWUSER before the rest of the
> namespaces. So in the case of clone(CLONE_NEWUSER|CLONE_NEWUTS)
> the new uts namespace will have the new user namespace as its
> owner. That is what we want, since we want root in that new
> userns to be able to have privilege over it.

Acked-by: "Eric W. Biederman" <ebiederm@xmission.com>

>
> Changelog:
> Feb 15: don't set uts_ns->user_ns if we didn't create
> a new uts_ns.
>
> Signed-off-by: Serge E. Hallyn <serge.hallyn@canonical.com>
> ---
> include/linux/utsname.h | 3 +++
> init/version.c | 2 ++
> kernel/nsproxy.c | 5 +++++
> kernel/user.c | 8 ++++++--
> kernel/utsname.c | 4 ++++
> 5 files changed, 20 insertions(+), 2 deletions(-)
>
> diff --git a/include/linux/utsname.h b/include/linux/utsname.h
> index 69f3997..85171be 100644
> --- a/include/linux/utsname.h
> +++ b/include/linux/utsname.h
> @@ -37,9 +37,12 @@ struct new_utsname {
> #include <linux/nsproxy.h>
> #include <linux/err.h>
>
> +struct user_namespace;
> +
> struct uts_namespace {
> struct kref kref;
> struct new_utsname name;
> + struct user_namespace *user_ns;
> };

> extern struct uts_namespace init_uts_ns;
>
> diff --git a/init/version.c b/init/version.c
> index adff586..97bb86f 100644
> --- a/init/version.c
> +++ b/init/version.c
> @@ -21,6 +21,7 @@ extern int version_string(LINUX_VERSION_CODE);
> int version_string(LINUX_VERSION_CODE);
> #endif
>
> +extern struct user_namespace init_user_ns;
> struct uts_namespace init_uts_ns = {
>   .kref = {
>   .refcount = ATOMIC_INIT(2),
> @@ -33,6 +34,7 @@ struct uts_namespace init_uts_ns = {
>   .machine = UTS_MACHINE,
>   .domainname = UTS_DOMAINNAME,
>   },
> + .user_ns = &init_user_ns,
> };
> EXPORT_SYMBOL_GPL(init_uts_ns);
>
> diff --git a/kernel/nsproxy.c b/kernel/nsproxy.c
> index f74e6c0..034dc2e 100644
> --- a/kernel/nsproxy.c
> +++ b/kernel/nsproxy.c
> @@ -74,6 +74,11 @@ static struct nsproxy *create_new_namespaces(unsigned long flags,
>   err = PTR_ERR(new_nsp->uts_ns);
>   goto out_uts;
>   }
> + if (new_nsp->uts_ns != tsk->nsproxy->uts_ns) {
> + put_user_ns(new_nsp->uts_ns->user_ns);
> + new_nsp->uts_ns->user_ns = task_cred_xxx(tsk, user)->user_ns;
> + get_user_ns(new_nsp->uts_ns->user_ns);
> + }
>
>   new_nsp->ipc_ns = copy_ipcs(flags, tsk->nsproxy->ipc_ns);
>   if (IS_ERR(new_nsp->ipc_ns)) {
> diff --git a/kernel/user.c b/kernel/user.c
> index 5c598ca..9e03e9c 100644
> --- a/kernel/user.c
> +++ b/kernel/user.c
> @@ -17,9 +17,13 @@
> #include <linux/module.h>
> #include <linux/user_namespace.h>
>
> +/*
> + * userns count is 1 for root user, 1 for init_uts_ns,

```
> + * and 1 for... ?
> + */
>  struct user_namespace init_user_ns = {
>   .kref = {
> -  .refcount = ATOMIC_INIT(2),
> +  .refcount = ATOMIC_INIT(3),
>   },
>   .creator = &root_user,
>  };
> @@ -47,7 +51,7 @@ static struct kmem_cache *uid_cachep;
>   */
>  static DEFINE_SPINLOCK(uidhash_lock);
>
> -/* root_user.__count is 2, 1 for init task cred, 1 for init_user_ns->creator */
> +/* root_user.__count is 2, 1 for init task cred, 1 for init_user_ns->user_ns */
>  struct user_struct root_user = {
>   .__count = ATOMIC_INIT(2),
>   .processes = ATOMIC_INIT(1),
> diff --git a/kernel/utsname.c b/kernel/utsname.c
> index 8a82b4b..a7b3a8d 100644
> --- a/kernel/utsname.c
> +++ b/kernel/utsname.c
> @@ -14,6 +14,7 @@
>  #include <linux/utsname.h>
>  #include <linux/err.h>
>  #include <linux/slab.h>
> +#include <linux/user_namespace.h>
>
>  static struct uts_namespace *create_uts_ns(void)
>  {
> @@ -40,6 +41,8 @@ static struct uts_namespace *clone_uts_ns(struct uts_namespace
*old_ns)
>
>   down_read(&uts_sem);
>   memcpy(&ns->name, &old_ns->name, sizeof(ns->name));
> + ns->user_ns = old_ns->user_ns;
> + get_user_ns(ns->user_ns);
>   up_read(&uts_sem);
>   return ns;
>  }
> @@ -71,5 +74,6 @@ void free_uts_ns(struct kref *kref)
>   struct uts_namespace *ns;
>
>   ns = container_of(kref, struct uts_namespace, kref);
> + put_user_ns(ns->user_ns);
>   kfree(ns);
>  }
```

Subject: Re: [PATCH 2/9] security: Make capabilities relative to the user namespace.
Posted by ebiederm on Fri, 18 Feb 2011 03:46:09 GMT
View Forum Message <> Reply to Message

"Serge E. Hallyn" <serge@hallyn.com> writes:

> - Introduce ns_capable to test for a capability in a non-default
> user namespace.
> - Teach cap_capable to handle capabilities in a non-default
> user namespace.
>
> The motivation is to get to the unprivileged creation of new
> namespaces. It looks like this gets us 90% of the way there, with
> only potential uid confusion issues left.
>
> I still need to handle getting all caps after creation but otherwise I
> think I have a good starter patch that achieves all of your goals.

Acked-by: "Eric W. Biederman" <ebiederm@xmission.com>

>
> Changelog:
> 11/05/2010: [serge] add apparmor
> 12/14/2010: [serge] fix capabilities to created user namespaces
> Without this, if user serge creates a user_ns, he won't have
> capabilities to the user_ns he created. THis is because we
> were first checking whether his effective caps had the caps
> he needed and returning -EPERM if not, and THEN checking whether
> he was the creator. Reverse those checks.
> 12/16/2010: [serge] security_real_capable needs ns argument in !security case
> 01/11/2011: [serge] add task_ns_capable helper
> 01/11/2011: [serge] add nsown_capable() helper per Bastian Blank suggestion
> 02/16/2011: [serge] fix a logic bug: the root user is always creator of
> init_user_ns, but should not always have capabilities to
> it! Fix the check in cap_capable().
>
> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
> Signed-off-by: Serge E. Hallyn <serge.hallyn@canonical.com>
> ---
> include/linux/capability.h | 10 ++++++++--
> include/linux/security.h | 25 +++++++++++++++++----------
> kernel/capability.c | 32 ++++++++++++++++++++++++++++++++--

> security/apparmor/lsm.c | 5 +++--
> security/commoncap.c | 40 ++++++++++++++++++++++++++++++++++++++-------
> security/security.c | 16 ++++++++++------
> security/selinux/hooks.c | 14 +++++++++-----
> 7 files changed, 107 insertions(+), 35 deletions(-)
>
> diff --git a/include/linux/capability.h b/include/linux/capability.h
> index fb16a36..cb3d2d9 100644
> --- a/include/linux/capability.h
> +++ b/include/linux/capability.h
> @@ -544,7 +544,7 @@ extern const kernel_cap_t __cap_init_eff_set;
>   *
>   * Note that this does not set PF_SUPERPRIV on the task.
>   */
> -#define has_capability(t, cap) (security_real_capable((t), (cap)) == 0)
> +#define has_capability(t, cap) (security_real_capable((t), &init_user_ns, (cap)) == 0)
>
> /**
>   * has_capability_noaudit - Determine if a task has a superior capability available (unaudited)
> @@ -558,9 +558,15 @@ extern const kernel_cap_t __cap_init_eff_set;
>   * Note that this does not set PF_SUPERPRIV on the task.
>   */
> #define has_capability_noaudit(t, cap) \
> - (security_real_capable_noaudit((t), (cap)) == 0)
> + (security_real_capable_noaudit((t), &init_user_ns, (cap)) == 0)
>
> +struct user_namespace;
> +extern struct user_namespace init_user_ns;
> extern int capable(int cap);
> +extern int ns_capable(struct user_namespace *ns, int cap);
> +extern int task_ns_capable(struct task_struct *t, int cap);
> +
> +#define nsown_capable(cap) (ns_capable(current_user_ns(), (cap)))
>
> /* audit system wants to get cap info from files as well */
> struct dentry;
> diff --git a/include/linux/security.h b/include/linux/security.h
> index b2b7f97..6bbee08 100644
> --- a/include/linux/security.h
> +++ b/include/linux/security.h
> @@ -46,13 +46,14 @@
>
> struct ctl_table;
> struct audit_krule;
> +struct user_namespace;
>
> /*
>   * These functions are in security/capability.c and are used

>   * as the default capabilities functions
>   */
> extern int cap_capable(struct task_struct *tsk, const struct cred *cred,
> -   int cap, int audit);
> +   struct user_namespace *ns, int cap, int audit);
> extern int cap_settime(struct timespec *ts, struct timezone *tz);
> extern int cap_ptrace_access_check(struct task_struct *child, unsigned int mode);
> extern int cap_ptrace_traceme(struct task_struct *parent);
> @@ -1254,6 +1255,7 @@ static inline void security_free_mnt_opts(struct security_mnt_opts
*opts)
>   * credentials.
>   * @tsk contains the task_struct for the process.
>   * @cred contains the credentials to use.
> + *   @ns contains the user namespace we want the capability in
>   * @cap contains the capability <include/linux/capability.h>.
>   * @audit: Whether to write an audit message or not
>   * Return 0 if the capability is granted for @tsk.
> @@ -1382,7 +1384,7 @@ struct security_operations {
>   const kernel_cap_t *inheritable,
>   const kernel_cap_t *permitted);
>   int (*capable) (struct task_struct *tsk, const struct cred *cred,
> -   int cap, int audit);
> +   struct user_namespace *ns, int cap, int audit);
>   int (*sysctl) (struct ctl_table *table, int op);
>   int (*quotactl) (int cmds, int type, int id, struct super_block *sb);
>   int (*quota_on) (struct dentry *dentry);
> @@ -1662,9 +1664,9 @@ int security_capset(struct cred *new, const struct cred *old,
>   const kernel_cap_t *effective,
>   const kernel_cap_t *inheritable,
>   const kernel_cap_t *permitted);
> -int security_capable(const struct cred *cred, int cap);
> -int security_real_capable(struct task_struct *tsk, int cap);
> -int security_real_capable_noaudit(struct task_struct *tsk, int cap);
> +int security_capable(struct user_namespace *ns, const struct cred *cred, int cap);
> +int security_real_capable(struct task_struct *tsk, struct user_namespace *ns, int cap);
> +int security_real_capable_noaudit(struct task_struct *tsk, struct user_namespace *ns, int cap);
> int security_sysctl(struct ctl_table *table, int op);
> int security_quotactl(int cmds, int type, int id, struct super_block *sb);
> int security_quota_on(struct dentry *dentry);
> @@ -1856,28 +1858,29 @@ static inline int security_capset(struct cred *new,
>   return cap_capset(new, old, effective, inheritable, permitted);
> }
>
> -static inline int security_capable(const struct cred *cred, int cap)
> +static inline int security_capable(struct user_namespace *ns,
> +   const struct cred *cred, int cap)
> {
> - return cap_capable(current, cred, cap, SECURITY_CAP_AUDIT);

> + return cap_capable(current, cred, ns, cap, SECURITY_CAP_AUDIT);
> }
>
> -static inline int security_real_capable(struct task_struct *tsk, int cap)
> +static inline int security_real_capable(struct task_struct *tsk, struct user_namespace *ns, int
cap)
> {
>   int ret;
>
>   rcu_read_lock();
> - ret = cap_capable(tsk, __task_cred(tsk), cap, SECURITY_CAP_AUDIT);
> + ret = cap_capable(tsk, __task_cred(tsk), ns, cap, SECURITY_CAP_AUDIT);
>   rcu_read_unlock();
>   return ret;
> }
>
> static inline
> -int security_real_capable_noaudit(struct task_struct *tsk, int cap)
> +int security_real_capable_noaudit(struct task_struct *tsk, struct user_namespace *ns, int cap)
> {
>   int ret;
>
>   rcu_read_lock();
> - ret = cap_capable(tsk, __task_cred(tsk), cap,
> + ret = cap_capable(tsk, __task_cred(tsk), ns, cap,
>   SECURITY_CAP_NOAUDIT);
>   rcu_read_unlock();
>   return ret;
> diff --git a/kernel/capability.c b/kernel/capability.c
> index 9e9385f..916658c 100644
> --- a/kernel/capability.c
> +++ b/kernel/capability.c
> @@ -14,6 +14,7 @@
> #include <linux/security.h>
> #include <linux/syscalls.h>
> #include <linux/pid_namespace.h>
> +#include <linux/user_namespace.h>
> #include <asm/uaccess.h>
>
> /*
> @@ -301,15 +302,42 @@ error:
>   */
> int capable(int cap)
> {
> + return ns_capable(&init_user_ns, cap);
> +}
> +EXPORT_SYMBOL(capable);
> +

```
> +/**
> + * ns_capable - Determine if the current task has a superior capability in effect
> + * @ns:  The usernamespace we want the capability in
> + * @cap: The capability to be tested for
> + *
> + * Return true if the current task has the given superior capability currently
> + * available for use, false if not.
> + *
> + * This sets PF_SUPERPRIV on the task if the capability is available on the
> + * assumption that it's about to be used.
> + */
> +int ns_capable(struct user_namespace *ns, int cap)
> +{
>   if (unlikely(!cap_valid(cap))) {
>    printk(KERN_CRIT "capable() called with invalid cap=%u\n", cap);
>    BUG();
>   }
>
> - if (security_capable(current_cred(), cap) == 0) {
> + if (security_capable(ns, current_cred(), cap) == 0) {
>    current->flags |= PF_SUPERPRIV;
>    return 1;
>   }
>   return 0;
>  }
> -EXPORT_SYMBOL(capable);
> +EXPORT_SYMBOL(ns_capable);
> +
> +/*
> + * does current have capability 'cap' to the user namespace of task
> + * 't'.  Return true if it does, false otherwise.
> + */
> +int task_ns_capable(struct task_struct *t, int cap)
> +{
> + return ns_capable(task_cred_xxx(t, user)->user_ns, cap);
> +}
> +EXPORT_SYMBOL(task_ns_capable);
> diff --git a/security/apparmor/lsm.c b/security/apparmor/lsm.c
> index b7106f1..b37c2cd 100644
> --- a/security/apparmor/lsm.c
> +++ b/security/apparmor/lsm.c
> @@ -22,6 +22,7 @@
>  #include <linux/ctype.h>
>  #include <linux/sysctl.h>
>  #include <linux/audit.h>
> +#include <linux/user_namespace.h>
>  #include <net/sock.h>
>
```

> #include "include/apparmor.h"
> @@ -136,11 +137,11 @@ static int apparmor_capget(struct task_struct *target, kernel_cap_t
*effective,
> }
>
> static int apparmor_capable(struct task_struct *task, const struct cred *cred,
> -   int cap, int audit)
> +   struct user_namespace *ns, int cap, int audit)
> {
>   struct aa_profile *profile;
>   /* cap_capable returns 0 on success, else -EPERM */
> - int error = cap_capable(task, cred, cap, audit);
> + int error = cap_capable(task, cred, ns, cap, audit);
>   if (!error) {
>   profile = aa_cred_profile(cred);
>   if (!unconfined(profile))
> diff --git a/security/commoncap.c b/security/commoncap.c
> index 64c2ed9..51fa9ec 100644
> --- a/security/commoncap.c
> +++ b/security/commoncap.c
> @@ -27,6 +27,7 @@
> #include <linux/sched.h>
> #include <linux/prctl.h>
> #include <linux/securebits.h>
> +#include <linux/user_namespace.h>
>
> /*
>   * If a non-root user executes a setuid-root binary in
> @@ -68,6 +69,7 @@ EXPORT_SYMBOL(cap_netlink_recv);
>   * cap_capable - Determine whether a task has a particular effective capability
>   * @tsk: The task to query
>   * @cred: The credentials to use
> + * @ns: The user namespace in which we need the capability
>   * @cap: The capability to check for
>   * @audit: Whether to write an audit message or not
>   *
> @@ -79,10 +81,32 @@ EXPORT_SYMBOL(cap_netlink_recv);
>   * cap_has_capability() returns 0 when a task has a capability, but the
>   * kernel's capable() and has_capability() returns 1 for this case.
>   */
> -int cap_capable(struct task_struct *tsk, const struct cred *cred, int cap,
> - int audit)
> +int cap_capable(struct task_struct *tsk, const struct cred *cred,
> + struct user_namespace *targ_ns, int cap, int audit)
> {
> - return cap_raised(cred->cap_effective, cap) ? 0 : -EPERM;
> + for (;;) {
> + /* The creator of the user namespace has all caps. */

```
> +  if (targ_ns != &init_user_ns && targ_ns->creator == cred->user)
> +   return 0;
> +
> +  /* Do we have the necessary capabilities? */
> +  if (targ_ns == cred->user->user_ns)
> +   return cap_raised(cred->cap_effective, cap) ? 0 : -EPERM;
> +
> +  /* Have we tried all of the parent namespaces? */
> +  if (targ_ns == &init_user_ns)
> +   return -EPERM;
> +
> +  /* If you have the capability in a parent user ns you have it
> +   * in the over all children user namespaces as well, so see
> +   * if this process has the capability in the parent user
> +   * namespace.
> +   */
> +  targ_ns = targ_ns->creator->user_ns;
> + }
> +
> + /* We never get here */
> + return -EPERM;
>  }
>
>  /**
> @@ -177,7 +201,8 @@ static inline int cap_inh_is_capped(void)
>   /* they are so limited unless the current task has the CAP_SETPCAP
>    * capability
>    */
> - if (cap_capable(current, current_cred(), CAP_SETPCAP,
> + if (cap_capable(current, current_cred(),
> +   current_cred()->user->user_ns, CAP_SETPCAP,
>     SECURITY_CAP_AUDIT) == 0)
>    return 0;
>   return 1;
> @@ -829,7 +854,8 @@ int cap_task_prctl(int option, unsigned long arg2, unsigned long arg3,
>         & (new->securebits ^ arg2))   /*[1]*/
>        || ((new->securebits & SECURE_ALL_LOCKS & ~arg2)) /*[2]*/
>        || (arg2 & ~(SECURE_ALL_LOCKS | SECURE_ALL_BITS)) /*[3]*/
> -      || (cap_capable(current, current_cred(), CAP_SETPCAP,
> +      || (cap_capable(current, current_cred(),
> +        current_cred()->user->user_ns, CAP_SETPCAP,
>          SECURITY_CAP_AUDIT) != 0)  /*[4]*/
>     /*
>      * [1] no changing of bits that are locked
> @@ -894,7 +920,7 @@ int cap_vm_enough_memory(struct mm_struct *mm, long pages)
>  {
>   int cap_sys_admin = 0;
>
```

> - if (cap_capable(current, current_cred(), CAP_SYS_ADMIN,
> + if (cap_capable(current, current_cred(), &init_user_ns, CAP_SYS_ADMIN,
>   SECURITY_CAP_NOAUDIT) == 0)
>   cap_sys_admin = 1;
>   return __vm_enough_memory(mm, pages, cap_sys_admin);
> @@ -921,7 +947,7 @@ int cap_file_mmap(struct file *file, unsigned long reqprot,
>   int ret = 0;
>
>   if (addr < dac_mmap_min_addr) {
> - ret = cap_capable(current, current_cred(), CAP_SYS_RAWIO,
> + ret = cap_capable(current, current_cred(), &init_user_ns, CAP_SYS_RAWIO,
>   SECURITY_CAP_AUDIT);
>   /* set PF_SUPERPRIV if it turns out we allow the low mmap */
>   if (ret == 0)
> diff --git a/security/security.c b/security/security.c
> index 7b7308a..7a6a0d0 100644
> --- a/security/security.c
> +++ b/security/security.c
> @@ -154,29 +154,33 @@ int security_capset(struct cred *new, const struct cred *old,
>   effective, inheritable, permitted);
> }
>
> -int security_capable(const struct cred *cred, int cap)
> +int security_capable(struct user_namespace *ns, const struct cred *cred,
> +   int cap)
> {
> - return security_ops->capable(current, cred, cap, SECURITY_CAP_AUDIT);
> + return security_ops->capable(current, cred, ns, cap,
> +   SECURITY_CAP_AUDIT);
> }
>
> -int security_real_capable(struct task_struct *tsk, int cap)
> +int security_real_capable(struct task_struct *tsk, struct user_namespace *ns,
> +   int cap)
> {
>   const struct cred *cred;
>   int ret;
>
>   cred = get_task_cred(tsk);
> - ret = security_ops->capable(tsk, cred, cap, SECURITY_CAP_AUDIT);
> + ret = security_ops->capable(tsk, cred, ns, cap, SECURITY_CAP_AUDIT);
>   put_cred(cred);
>   return ret;
> }
>
> -int security_real_capable_noaudit(struct task_struct *tsk, int cap)
> +int security_real_capable_noaudit(struct task_struct *tsk,
> +   struct user_namespace *ns, int cap)

```
>  {
>   const struct cred *cred;
>   int ret;
>
>   cred = get_task_cred(tsk);
> - ret = security_ops->capable(tsk, cred, cap, SECURITY_CAP_NOAUDIT);
> + ret = security_ops->capable(tsk, cred, ns, cap, SECURITY_CAP_NOAUDIT);
>   put_cred(cred);
>   return ret;
>  }
> diff --git a/security/selinux/hooks.c b/security/selinux/hooks.c
> index c8d6992..6dcda48 100644
> --- a/security/selinux/hooks.c
> +++ b/security/selinux/hooks.c
> @@ -77,6 +77,7 @@
>  #include <linux/mutex.h>
>  #include <linux/posix-timers.h>
>  #include <linux/syslog.h>
> +#include <linux/user_namespace.h>
>
>  #include "avc.h"
>  #include "objsec.h"
> @@ -1423,6 +1424,7 @@ static int current_has_perm(const struct task_struct *tsk,
>  /* Check whether a task is allowed to use a capability. */
>  static int task_has_capability(struct task_struct *tsk,
>             const struct cred *cred,
> +            struct user_namespace *ns,
>             int cap, int audit)
>  {
>   struct common_audit_data ad;
> @@ -1851,15 +1853,15 @@ static int selinux_capset(struct cred *new, const struct cred *old,
>   */
>
>  static int selinux_capable(struct task_struct *tsk, const struct cred *cred,
> -       int cap, int audit)
> +       struct user_namespace *ns, int cap, int audit)
>  {
>   int rc;
>
> - rc = cap_capable(tsk, cred, cap, audit);
> + rc = cap_capable(tsk, cred, ns, cap, audit);
>   if (rc)
>    return rc;
>
> - return task_has_capability(tsk, cred, cap, audit);
> + return task_has_capability(tsk, cred, ns, cap, audit);
>  }
>
```

> static int selinux_sysctl_get_sid(ctl_table *table, u16 tclass, u32 *sid)
> @@ -2012,7 +2014,8 @@ static int selinux_vm_enough_memory(struct mm_struct *mm, long pages)
> {
>   int rc, cap_sys_admin = 0;
>
> - rc = selinux_capable(current, current_cred(), CAP_SYS_ADMIN,
> + rc = selinux_capable(current, current_cred(),
> +   &init_user_ns, CAP_SYS_ADMIN,
>   SECURITY_CAP_NOAUDIT);
>   if (rc == 0)
>   cap_sys_admin = 1;
> @@ -2829,7 +2832,8 @@ static int selinux_inode_getsecurity(const struct inode *inode, const char *name
>   * and lack of permission just means that we fall back to the
>   * in-core context value, not a denial.
>   */
> - error = selinux_capable(current, current_cred(), CAP_MAC_ADMIN,
> + error = selinux_capable(current, current_cred(),
> +   &init_user_ns, CAP_MAC_ADMIN,
>   SECURITY_CAP_NOAUDIT);
>   if (!error)
>   error = security_sid_to_context_force(isec->sid, &context,

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

---

Subject: Re: userns: targeted capabilities v5
Posted by ebiederm on Fri, 18 Feb 2011 03:53:37 GMT
View Forum Message <> Reply to Message

Andrew Morton <akpm@linux-foundation.org> writes:

> On Thu, 17 Feb 2011 15:02:24 +0000
> "Serge E. Hallyn" <serge@hallyn.com> wrote:
>
>> Here is a repost of my previous user namespace patch, ported onto
>> last night's git head.
>>
>> It fixes several things I was doing wrong in the last (v4)
>> posting, in particular:
>>
>> 1. don't set uts_ns->user_ns to current's when !CLONE_NEWUTS
>> 2. add a ipc_ns->user_ns which owns ipc_ns, and use that to
>>   decide CAP_IPC_OWNER
>> 3. fix logic flaw caused by bad parantheses

>> 4. allow do_prlimit to current
>> 5. don't always give root full privs to init_user_ns
>>
>> The expected course of development for user namespaces is laid out
>> at https://wiki.ubuntu.com/UserNamespace.
>
> Seems like a nice feature to be developing.
>
> I worry about the maturity of it all at this stage. How far along is
> it *really*?
>
> Is anyone else working with you on developing and reviewing this work?

I did a lot of the initial design and I have been reviewing as I have
time.

Andrew at a practical level we have to merge this incrementally.
Anything much bigger than Serge's current patchset will be too big to
review. The first really bit step is making capabilities local to the
user namespace, and that is what this patchset does along with using
that localness in some good places.

Eric

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

---

Subject: Re: userns: targeted capabilities v5
Posted by serge on Fri, 18 Feb 2011 04:28:15 GMT
View Forum Message <> Reply to Message

Quoting Andrew Morton (akpm@linux-foundation.org):
> On Thu, 17 Feb 2011 15:02:24 +0000
> "Serge E. Hallyn" <serge@hallyn.com> wrote:
>
> > Here is a repost of my previous user namespace patch, ported onto
> > last night's git head.
> >
> > It fixes several things I was doing wrong in the last (v4)
> > posting, in particular:
> >
> > 1. don't set uts_ns->user_ns to current's when !CLONE_NEWUTS
> > 2. add a ipc_ns->user_ns which owns ipc_ns, and use that to
> >   decide CAP_IPC_OWNER
> > 3. fix logic flaw caused by bad parantheses

> > 4. allow do_prlimit to current
> > 5. don't always give root full privs to init_user_ns
> >
> > The expected course of development for user namespaces is laid out
> > at https://wiki.ubuntu.com/UserNamespace.
>
> Seems like a nice feature to be developing.
>
> I worry about the maturity of it all at this stage. How far along is
> it *really*?
>
> Is anyone else working with you on developing and reviewing this work?

Thanks, Andrew. I'm not sure what definition of 'maturity' you were looking for here. If you meant completeness of the feature, it's definately not there. Of the goals for user namespaces sandboxing will be the quickest to mature. Completing that will largely be an exercise of running the breadth of the kernel looking for simple uid/gid comparisons and making them namespace aware.

The design has been meshed around (publicly) on and off for many years by eric and I. This particular patchset has gotten some great reviews by Eric Biederman and Bastian Blank (to who, unfortunately, to this day I cannot send a direct email - they're always bounced).

As Eric said, this feature will have to go in incrementally. Furthermore, each piece touches scary code so it's likely to go pretty slowly. My hope is less than a year for sandboxing, and two years for containers. It might go way faster, but experience tells me that's unlikely :)

thanks,
-serge

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

---

Subject: Re: [PATCH 5/9] Allow ptrace from non-init user namespaces
Posted by serge on Fri, 18 Feb 2011 04:36:02 GMT

View Forum Message <> Reply to Message

Quoting Eric W. Biederman (ebiederm@xmission.com):
> "Serge E. Hallyn" <serge@hallyn.com> writes:
>
> > ptrace is allowed to tasks in the same user namespace according to
> > the usual rules (i.e. the same rules as for two tasks in the init

> > user namespace). ptrace is also allowed to a user namespace to
> > which the current task the has CAP_SYS_PTRACE capability.
>
>
> I don't see how it can go wrong at the moment but
> same_or_ancestore_user_ns is too permissive and potentially inefficient.
> Can you please replace it with a simple user namespace equality check.
>
> Eric
>
>
> > Changelog:
> > Dec 31: Address feedback by Eric:
> >   . Correct ptrace uid check
> >   . Rename may_ptrace_ns to ptrace_capable
> >   . Also fix the cap_ptrace checks.
> > Jan 1: Use const cred struct
> > Jan 11: use task_ns_capable() in place of ptrace_capable().
> >
> > Signed-off-by: Serge E. Hallyn <serge.hallyn@canonical.com>
> > ---
> > include/linux/capability.h | 2 +
> > include/linux/user_namespace.h | 9 +++++++
> > kernel/ptrace.c | 27 ++++++++++++++----------
> > kernel/user_namespace.c | 16 ++++++++++++
> > security/commoncap.c | 48 ++++++++++++++++++++++++++++++++++++------
> > 5 files changed, 82 insertions(+), 20 deletions(-)
> >
> > diff --git a/include/linux/capability.h b/include/linux/capability.h
> > index cb3d2d9..bc0f262 100644
> > --- a/include/linux/capability.h
> > +++ b/include/linux/capability.h
> > @@ -546,6 +546,8 @@ extern const kernel_cap_t __cap_init_eff_set;
> >   */
> > #define has_capability(t, cap) (security_real_capable((t), &init_user_ns, (cap)) == 0)
> >
> > +#define has_ns_capability(t, ns, cap) (security_real_capable((t), (ns), (cap)) == 0)
> > +
> > /**
> >   * has_capability_noaudit - Determine if a task has a superior capability available (unaudited)
> >   * @t: The task in question
> > diff --git a/include/linux/user_namespace.h b/include/linux/user_namespace.h
> > index faf4679..862fc59 100644
> > --- a/include/linux/user_namespace.h
> > +++ b/include/linux/user_namespace.h
> > @@ -39,6 +39,9 @@ static inline void put_user_ns(struct user_namespace *ns)
> > uid_t user_ns_map_uid(struct user_namespace *to, const struct cred *cred, uid_t uid);
> > gid_t user_ns_map_gid(struct user_namespace *to, const struct cred *cred, gid_t gid);

> >
> > +int same_or_ancestor_user_ns(struct task_struct *task,
> > + struct task_struct *victim);
> > +
> > #else
> >
> > static inline struct user_namespace *get_user_ns(struct user_namespace *ns)
> > @@ -66,6 +69,12 @@ static inline gid_t user_ns_map_gid(struct user_namespace *to,
> > return gid;
> > }
> >
> > +static inline int same_or_ancestor_user_ns(struct task_struct *task,
> > + struct task_struct *victim)
> > +{
> > + return 1;
> > +}
> > +
> > #endif
> >
> > #endif /* _LINUX_USER_H */
> > diff --git a/kernel/ptrace.c b/kernel/ptrace.c
> > index 1708b1e..cde4655 100644
> > --- a/kernel/ptrace.c
> > +++ b/kernel/ptrace.c
> > @@ -134,21 +134,24 @@ int __ptrace_may_access(struct task_struct *task, unsigned int mode)
> > return 0;
> > rcu_read_lock();
> > tcred = __task_cred(task);
> > - if ((cred->uid != tcred->euid ||
> > - cred->uid != tcred->suid ||
> > - cred->uid != tcred->uid ||
> > - cred->gid != tcred->egid ||
> > - cred->gid != tcred->sgid ||
> > - cred->gid != tcred->gid) &&
> > - !capable(CAP_SYS_PTRACE)) {
> > - rcu_read_unlock();
> > - return -EPERM;
> > - }
> > + if (cred->user->user_ns == tcred->user->user_ns &&
> > + (cred->uid == tcred->euid &&
> > + cred->uid == tcred->suid &&
> > + cred->uid == tcred->uid &&
> > + cred->gid == tcred->egid &&
> > + cred->gid == tcred->sgid &&
> > + cred->gid == tcred->gid))
> > + goto ok;
> > + if (ns_capable(tcred->user->user_ns, CAP_SYS_PTRACE))

```
> > +  goto ok;
> > + rcu_read_unlock();
> > + return -EPERM;
> > +ok:
> >   rcu_read_unlock();
> >   smp_rmb();
> >   if (task->mm)
> >    dumpable = get_dumpable(task->mm);
> > - if (!dumpable && !capable(CAP_SYS_PTRACE))
> > + if (!dumpable && !task_ns_capable(task, CAP_SYS_PTRACE))
> >    return -EPERM;
> >
> >   return security_ptrace_access_check(task, mode);
> > @@ -198,7 +201,7 @@ int ptrace_attach(struct task_struct *task)
> >    goto unlock_tasklist;
> >
> >   task->ptrace = PT_PTRACED;
> > - if (capable(CAP_SYS_PTRACE))
> > + if (task_ns_capable(task, CAP_SYS_PTRACE))
> >    task->ptrace |= PT_PTRACE_CAP;
> >
> >   __ptrace_link(task, current);
> > diff --git a/kernel/user_namespace.c b/kernel/user_namespace.c
> > index 9da289c..0ef2258 100644
> > --- a/kernel/user_namespace.c
> > +++ b/kernel/user_namespace.c
> > @@ -129,6 +129,22 @@ gid_t user_ns_map_gid(struct user_namespace *to, const struct
cred *cred, gid_t
> >   return overflowgid;
> >  }
> >
> > +int same_or_ancestor_user_ns(struct task_struct *task,
> > + struct task_struct *victim)
> > +{
> > + struct user_namespace *u1 = task_cred_xxx(task, user)->user_ns;
> > + struct user_namespace *u2 = task_cred_xxx(victim, user)->user_ns;
> > + for (;;) {
> > +  if (u1 == u2)
> > +   return 1;
> > +  if (u1 == &init_user_ns)
> > +   return 0;
> > +  u1 = u1->creator->user_ns;
> > + }
> > + /* We never get here */
> > + return 0;
> > +}
> > +
> >  static __init int user_namespaces_init(void)
```

> > {
> > user_ns_cachep = KMEM_CACHE(user_namespace, SLAB_PANIC);
> > diff --git a/security/commoncap.c b/security/commoncap.c
> > index 51fa9ec..12ff65c 100644
> > --- a/security/commoncap.c
> > +++ b/security/commoncap.c
> > @@ -130,18 +130,34 @@ int cap_settime(struct timespec *ts, struct timezone *tz)
> > * @child: The process to be accessed
> > * @mode: The mode of attachment.
> > *
> > + * If we are in the same or an ancestor user_ns and have all the target
> > + * task's capabilities, then ptrace access is allowed.
> > + * If we have the ptrace capability to the target user_ns, then ptrace
> > + * access is allowed.
> > + * Else denied.
> > + *
> > * Determine whether a process may access another, returning 0 if permission
> > * granted, -ve if denied.
> > */
> > int cap_ptrace_access_check(struct task_struct *child, unsigned int mode)
> > {
> > int ret = 0;
> > + const struct cred *cred, *tcred;
> >
> > rcu_read_lock();
> > - if (!cap_issubset(__task_cred(child)->cap_permitted,
> > - current_cred()->cap_permitted) &&
> > - !capable(CAP_SYS_PTRACE))
> > - ret = -EPERM;
> > + cred = current_cred();
> > + tcred = __task_cred(child);
> > + /*
> > + * The ancestor user_ns check may be gratuitous, as I think
> > + * we've already guaranteed that in kernel/ptrace.c.
> > + */
> > + if (same_or_ancestor_user_ns(current, child) &&
> > + cap_issubset(tcred->cap_permitted, cred->cap_permitted))
> > + goto out;
>
> I have commented on this before but I took a good hard look this time,
> and can comment more intelligently.

Thanks, Eric.

> The cap_issubset check is for the case where we don't use the
> CAP_SYS_PTRACE capability, as such is only valid in the same user
> namespace. Furthermore capabilities really are not comparable between
> different user namespaces. So can you please replace the

> same_or_ancestor_user_ns with a simple namespace equality check.
> Having the wrong logic in here will just be confusing in the future.

I see. You're right, what's there is conceptually not quite right.
So I think we can just do:

```
 if (current_user_ns() == tcred->user->userns) &&
     cap_issubset(child, current)
  goto out;
 if ns_capable(tcred->user->user_ns, CAP_SYS_PTRACE)
  goto out;
```

This suffices since
 root in an ancestor ns will have CAP_SYS_PTRACE
 the user who created the user_ns will have CAP_SYS_PTRACE

Any user in an ancestor ns who does not have CAP_SYS_PTRACE will
be denied.

> Also could you name tcred child_cread I think that would be clearer in
> the test below.

Yes, will do, that'll be clearer. Thanks.

> > + if (ns_capable(tcred->user->user_ns, CAP_SYS_PTRACE))
> > + goto out;
> > + ret = -EPERM;
> > +out:
> >   rcu_read_unlock();
> >   return ret;
>
> I also find it strange that we allow CAP_SYS_PTRACE to allow us to
> ptrace processes with more capabilities than ourselves. But that is an
> entirely different issue.

I concur. I'm pretty sure it must stay, but it does feel icky.

I'll send out a new version

thanks
-serge

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

Subject: Re: [PATCH 1/9] Add a user_namespace as creator/owner of uts_namespace
Posted by Daniel Lezcano on Fri, 18 Feb 2011 16:57:41 GMT
View Forum Message <> Reply to Message

On 02/17/2011 04:02 PM, Serge E. Hallyn wrote:
> copy_process() handles CLONE_NEWUSER before the rest of the
> namespaces. So in the case of clone(CLONE_NEWUSER|CLONE_NEWUTS)
> the new uts namespace will have the new user namespace as its
> owner. That is what we want, since we want root in that new
> userns to be able to have privilege over it.
>
> Changelog:
> Feb 15: don't set uts_ns->user_ns if we didn't create
> a new uts_ns.
>
> Signed-off-by: Serge E. Hallyn<serge.hallyn@canonical.com>

Acked-by: Daniel Lezcano <daniel.lezcano@free.fr>

A couple of comments.

> ---
> include/linux/utsname.h | 3 +++
> init/version.c | 2 ++
> kernel/nsproxy.c | 5 +++++
> kernel/user.c | 8 ++++++--
> kernel/utsname.c | 4 ++++
> 5 files changed, 20 insertions(+), 2 deletions(-)
>
> diff --git a/include/linux/utsname.h b/include/linux/utsname.h
> index 69f3997..85171be 100644
> --- a/include/linux/utsname.h
> +++ b/include/linux/utsname.h
> @@ -37,9 +37,12 @@ struct new_utsname {
> #include<linux/nsproxy.h>
> #include<linux/err.h>
>
> +struct user_namespace;
> +
> struct uts_namespace {
> struct kref kref;
> struct new_utsname name;
> + struct user_namespace *user_ns;
> };
> extern struct uts_namespace init_uts_ns;
>
> diff --git a/init/version.c b/init/version.c
> index adff586..97bb86f 100644

> --- a/init/version.c
> +++ b/init/version.c
> @@ -21,6 +21,7 @@ extern int version_string(LINUX_VERSION_CODE);
> int version_string(LINUX_VERSION_CODE);
> #endif
>
> +extern struct user_namespace init_user_ns;
> struct uts_namespace init_uts_ns = {
>   .kref = {
>   .refcount = ATOMIC_INIT(2),
> @@ -33,6 +34,7 @@ struct uts_namespace init_uts_ns = {
>   .machine = UTS_MACHINE,
>   .domainname = UTS_DOMAINNAME,
>   },
> + .user_ns =&init_user_ns,
> };
> EXPORT_SYMBOL_GPL(init_uts_ns);
>
> diff --git a/kernel/nsproxy.c b/kernel/nsproxy.c
> index f74e6c0..034dc2e 100644
> --- a/kernel/nsproxy.c
> +++ b/kernel/nsproxy.c
> @@ -74,6 +74,11 @@ static struct nsproxy *create_new_namespaces(unsigned long flags,
>   err = PTR_ERR(new_nsp->uts_ns);
>   goto out_uts;
>   }
> + if (new_nsp->uts_ns != tsk->nsproxy->uts_ns) {
> + put_user_ns(new_nsp->uts_ns->user_ns);
> + new_nsp->uts_ns->user_ns = task_cred_xxx(tsk, user)->user_ns;
> + get_user_ns(new_nsp->uts_ns->user_ns);
> + }

IMO you should add a comment telling this code assume create_user_ns was
called before (via copy_cred).

>
>   new_nsp->ipc_ns = copy_ipcs(flags, tsk->nsproxy->ipc_ns);
>   if (IS_ERR(new_nsp->ipc_ns)) {

[...]

> static struct uts_namespace *create_uts_ns(void)
> {
> @@ -40,6 +41,8 @@ static struct uts_namespace *clone_uts_ns(struct uts_namespace
*old_ns)
>
>   down_read(&uts_sem);
>   memcpy(&ns->name,&old_ns->name, sizeof(ns->name));

> + ns->user_ns = old_ns->user_ns;
> + get_user_ns(ns->user_ns);

ns->user_ns = get_user_ns(old_ns->user_ns);

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

---

Subject: Re: [PATCH 2/9] security: Make capabilities relative to the user namespace.
Posted by Daniel Lezcano on Fri, 18 Feb 2011 23:44:50 GMT
View Forum Message <> Reply to Message

On 02/17/2011 04:03 PM, Serge E. Hallyn wrote:
> - Introduce ns_capable to test for a capability in a non-default
> user namespace.
> - Teach cap_capable to handle capabilities in a non-default
> user namespace.
>
> The motivation is to get to the unprivileged creation of new
> namespaces. It looks like this gets us 90% of the way there, with
> only potential uid confusion issues left.
>
> I still need to handle getting all caps after creation but otherwise I
> think I have a good starter patch that achieves all of your goals.
>
> Changelog:
> 11/05/2010: [serge] add apparmor
> 12/14/2010: [serge] fix capabilities to created user namespaces
> Without this, if user serge creates a user_ns, he won't have
> capabilities to the user_ns he created. THis is because we
> were first checking whether his effective caps had the caps
> he needed and returning -EPERM if not, and THEN checking whether
> he was the creator. Reverse those checks.
> 12/16/2010: [serge] security_real_capable needs ns argument in !security case
> 01/11/2011: [serge] add task_ns_capable helper
> 01/11/2011: [serge] add nsown_capable() helper per Bastian Blank suggestion
> 02/16/2011: [serge] fix a logic bug: the root user is always creator of
> init_user_ns, but should not always have capabilities to
> it! Fix the check in cap_capable().
>
> Signed-off-by: Eric W. Biederman<ebiederm@xmission.com>
> Signed-off-by: Serge E. Hallyn<serge.hallyn@canonical.com>
> ---

Acked-by: Daniel Lezcano <daniel.lezcano@free.fr>

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

---

Subject: Re: [PATCH 3/9] allow sethostname in a container
Posted by Daniel Lezcano on Fri, 18 Feb 2011 23:46:15 GMT
View Forum Message <> Reply to Message

On 02/17/2011 04:03 PM, Serge E. Hallyn wrote:
> Signed-off-by: Serge E. Hallyn<serge.hallyn@canonical.com>
> ---
Acked-by: Daniel Lezcano <daniel.lezcano@free.fr>

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

---

Subject: Re: [PATCH 1/9] Add a user_namespace as creator/owner of uts_namespace
Posted by akpm on Fri, 18 Feb 2011 23:59:12 GMT
View Forum Message <> Reply to Message

On Thu, 17 Feb 2011 15:02:57 +0000
"Serge E. Hallyn" <serge@hallyn.com> wrote:

> +/*
> + * userns count is 1 for root user, 1 for init_uts_ns,
> + * and 1 for... ?
> + */

?

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

---

Subject: Re: [PATCH 2/9] security: Make capabilities relative to the user namespace.
Posted by akpm on Fri, 18 Feb 2011 23:59:18 GMT
View Forum Message <> Reply to Message

On Thu, 17 Feb 2011 15:03:06 +0000
"Serge E. Hallyn" <serge@hallyn.com> wrote:

> - Introduce ns_capable to test for a capability in a non-default
>   user namespace.
> - Teach cap_capable to handle capabilities in a non-default
>   user namespace.
>
> The motivation is to get to the unprivileged creation of new
> namespaces. It looks like this gets us 90% of the way there, with
> only potential uid confusion issues left.
>
> I still need to handle getting all caps after creation but otherwise I
> think I have a good starter patch that achieves all of your goals.
>
>
> ...
>
> --- a/include/linux/capability.h
> +++ b/include/linux/capability.h
> @@ -544,7 +544,7 @@ extern const kernel_cap_t __cap_init_eff_set;
>   *
>   * Note that this does not set PF_SUPERPRIV on the task.
>   */
> -#define has_capability(t, cap) (security_real_capable((t), (cap)) == 0)
> +#define has_capability(t, cap) (security_real_capable((t), &init_user_ns, (cap)) == 0)
>
> /**
>   * has_capability_noaudit - Determine if a task has a superior capability available (unaudited)
> @@ -558,9 +558,15 @@ extern const kernel_cap_t __cap_init_eff_set;
>   * Note that this does not set PF_SUPERPRIV on the task.
>   */
> #define has_capability_noaudit(t, cap) \
> - (security_real_capable_noaudit((t), (cap)) == 0)
> + (security_real_capable_noaudit((t), &init_user_ns, (cap)) == 0)
>
> +struct user_namespace;
> +extern struct user_namespace init_user_ns;

Two icky-should-be-written-in-C macros which reference init_user_ns,
followed by the declaration of init_user_ns and its type. Declarations
which duplicate those in other header files. It's ripe for some
upcleaning, methinks?

Also, please ensure that the forward struct declarations are all at
top-of-file (as in include/linux/security.h). Otherwise we can end up
accumulating multiple forward declarations of the same thing in the one
file.

> extern int capable(int cap);
> +extern int ns_capable(struct user_namespace *ns, int cap);
> +extern int task_ns_capable(struct task_struct *t, int cap);
> +
> +#define nsown_capable(cap) (ns_capable(current_user_ns(), (cap)))

macroitis!

> @@ -301,15 +302,42 @@ error:
>   */
> int capable(int cap)
> {
> + return ns_capable(&init_user_ns, cap);
> +}
> +EXPORT_SYMBOL(capable);
> +
> +/**
> + * ns_capable - Determine if the current task has a superior capability in effect
> + * @ns: The usernamespace we want the capability in
> + * @cap: The capability to be tested for
> + *
> + * Return true if the current task has the given superior capability currently
> + * available for use, false if not.

Actually it doesn't return true or false - it returns 1 or 0. Using a
`bool' return type would fix the comment :)

> + * This sets PF_SUPERPRIV on the task if the capability is available on the
> + * assumption that it's about to be used.
> + */
> +int ns_capable(struct user_namespace *ns, int cap)
> +{
>   if (unlikely(!cap_valid(cap))) {
>   printk(KERN_CRIT "capable() called with invalid cap=%u\n", cap);
>   BUG();
>   }
>
> - if (security_capable(current_cred(), cap) == 0) {
> + if (security_capable(ns, current_cred(), cap) == 0) {
>   current->flags |= PF_SUPERPRIV;
>   return 1;
>   }
>   return 0;
> }
> -EXPORT_SYMBOL(capable);
> +EXPORT_SYMBOL(ns_capable);
> +

> +/*
> + * does current have capability 'cap' to the user namespace of task
> + * 't'. Return true if it does, false otherwise.
> + */

Other comments were kerneldocified.

> +int task_ns_capable(struct task_struct *t, int cap)
> +{
> + return ns_capable(task_cred_xxx(t, user)->user_ns, cap);
> +}
> +EXPORT_SYMBOL(task_ns_capable);

Could return bool.

>
> ...
>
> +int cap_capable(struct task_struct *tsk, const struct cred *cred,
> + struct user_namespace *targ_ns, int cap, int audit)
> {
> - return cap_raised(cred->cap_effective, cap) ? 0 : -EPERM;
> + for (;;) {
> + /* The creator of the user namespace has all caps. */
> + if (targ_ns != &init_user_ns && targ_ns->creator == cred->user)
> +   return 0;
> +
> + /* Do we have the necessary capabilities? */
> + if (targ_ns == cred->user->user_ns)
> +   return cap_raised(cred->cap_effective, cap) ? 0 : -EPERM;
> +
> + /* Have we tried all of the parent namespaces? */
> + if (targ_ns == &init_user_ns)
> +   return -EPERM;
> +
> + /* If you have the capability in a parent user ns you have it
> +   * in the over all children user namespaces as well, so see
> +   * if this process has the capability in the parent user
> +   * namespace.
> +   */
> + targ_ns = targ_ns->creator->user_ns;
> + }
> +
> + /* We never get here */
> + return -EPERM;

So delete the code? Or does the compiler warn? If so, it's pretty busted.

> }
>
> /**
>
> ...
>

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

---

Subject: Re: [PATCH 4/9] allow killing tasks in your own or child userns
Posted by akpm on Fri, 18 Feb 2011 23:59:21 GMT
View Forum Message <> Reply to Message

On Thu, 17 Feb 2011 15:03:25 +0000
"Serge E. Hallyn" <serge@hallyn.com> wrote:

> /*
> + * called with RCU read lock from check_kill_permission()
> + */
> +static inline int kill_ok_by_cred(struct task_struct *t)
> +{
> + const struct cred *cred = current_cred();
> + const struct cred *tcred = __task_cred(t);
> +
> + if (cred->user->user_ns == tcred->user->user_ns &&
> + (cred->euid == tcred->suid ||
> + cred->euid == tcred->uid ||
> + cred->uid == tcred->suid ||
> + cred->uid == tcred->uid))
> + return 1;
> +
> + if (ns_capable(tcred->user->user_ns, CAP_KILL))
> + return 1;
> +
> + return 0;
> +}

The compiler will inline this for us.

> +/*
> * Bad permissions for sending the signal
> * - the caller must hold the RCU read lock
> */
> static int check_kill_permission(int sig, struct siginfo *info,

```
>       struct task_struct *t)
>  {
> - const struct cred *cred, *tcred;
>   struct pid *sid;
>   int error;
>
> @@ -656,14 +676,8 @@ static int check_kill_permission(int sig, struct siginfo *info,
>   if (error)
>    return error;
>
> - cred = current_cred();
> - tcred = __task_cred(t);
>   if (!same_thread_group(current, t) &&
> -     (cred->euid ^ tcred->suid) &&
> -     (cred->euid ^ tcred->uid) &&
> -     (cred->uid  ^ tcred->suid) &&
> -     (cred->uid  ^ tcred->uid) &&
> -     !capable(CAP_KILL)) {
> +     !kill_ok_by_cred(t)) {
>    switch (sig) {
>    case SIGCONT:
>     sid = task_session(t);
```

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

Subject: Re: [PATCH 5/9] Allow ptrace from non-init user namespaces
Posted by akpm on Fri, 18 Feb 2011 23:59:25 GMT
View Forum Message <> Reply to Message

On Thu, 17 Feb 2011 15:03:33 +0000
"Serge E. Hallyn" <serge@hallyn.com> wrote:

```
> ptrace is allowed to tasks in the same user namespace according to
> the usual rules (i.e. the same rules as for two tasks in the init
> user namespace).  ptrace is also allowed to a user namespace to
> which the current task the has CAP_SYS_PTRACE capability.
>
>
> ...
>
> --- a/include/linux/capability.h
> +++ b/include/linux/capability.h
> @@ -546,6 +546,8 @@ extern const kernel_cap_t __cap_init_eff_set;
>   */
```

> #define has_capability(t, cap) (security_real_capable((t), &init_user_ns, (cap)) == 0)
>
> +#define has_ns_capability(t, ns, cap) (security_real_capable((t), (ns), (cap)) == 0)

macroitis.

> /**
>   * has_capability_noaudit - Determine if a task has a superior capability available (unaudited)
>   * @t: The task in question
> diff --git a/include/linux/user_namespace.h b/include/linux/user_namespace.h
> index faf4679..862fc59 100644
> --- a/include/linux/user_namespace.h
> +++ b/include/linux/user_namespace.h
> @@ -39,6 +39,9 @@ static inline void put_user_ns(struct user_namespace *ns)
> uid_t user_ns_map_uid(struct user_namespace *to, const struct cred *cred, uid_t uid);
> gid_t user_ns_map_gid(struct user_namespace *to, const struct cred *cred, gid_t gid);
>
> +int same_or_ancestor_user_ns(struct task_struct *task,
> + struct task_struct *victim);

bool.

> #else
>
> static inline struct user_namespace *get_user_ns(struct user_namespace *ns)
>
> ...
>
> --- a/kernel/user_namespace.c
> +++ b/kernel/user_namespace.c
> @@ -129,6 +129,22 @@ gid_t user_ns_map_gid(struct user_namespace *to, const struct cred
*cred, gid_t
>   return overflowgid;
> }
>
> +int same_or_ancestor_user_ns(struct task_struct *task,
> + struct task_struct *victim)
> +{
> + struct user_namespace *u1 = task_cred_xxx(task, user)->user_ns;
> + struct user_namespace *u2 = task_cred_xxx(victim, user)->user_ns;
> + for (;;) {
> + if (u1 == u2)
> +   return 1;
> + if (u1 == &init_user_ns)
> +   return 0;
> + u1 = u1->creator->user_ns;
> + }
> + /* We never get here */

> + return 0;

Remove?

> +}
> +
> static __init int user_namespaces_init(void)
> {
>   user_ns_cachep = KMEM_CACHE(user_namespace, SLAB_PANIC);
>
> ...
>
> int cap_ptrace_access_check(struct task_struct *child, unsigned int mode)
> {
>   int ret = 0;
> + const struct cred *cred, *tcred;
>
>   rcu_read_lock();
> - if (!cap_issubset(__task_cred(child)->cap_permitted,
> -   current_cred()->cap_permitted) &&
> -   !capable(CAP_SYS_PTRACE))
> - ret = -EPERM;
> + cred = current_cred();
> + tcred = __task_cred(child);
> + /*
> + * The ancestor user_ns check may be gratuitous, as I think
> + * we've already guaranteed that in kernel/ptrace.c.
> + */

?

> + if (same_or_ancestor_user_ns(current, child) &&
> +   cap_issubset(tcred->cap_permitted, cred->cap_permitted))
> + goto out;
> + if (ns_capable(tcred->user->user_ns, CAP_SYS_PTRACE))
> + goto out;
> + ret = -EPERM;
> +out:
>   rcu_read_unlock();
>   return ret;
> }
>
> ...
>

_____
Containers mailing list
Containers@lists.linux-foundation.org

Subject: Re: [PATCH 6/9] user namespaces: convert all capable checks in kernel/sys.c
Posted by akpm on Fri, 18 Feb 2011 23:59:28 GMT
View Forum Message <> Reply to Message

On Thu, 17 Feb 2011 15:03:42 +0000
"Serge E. Hallyn" <serge@hallyn.com> wrote:

> This allows setuid/setgid in containers. It also fixes some
> corner cases where kernel logic foregoes capability checks when
> uids are equivalent. The latter will need to be done throughout
> the whole kernel.
>
>
> ...
>
> --- a/kernel/sys.c
> +++ b/kernel/sys.c
> @@ -118,17 +118,29 @@ EXPORT_SYMBOL(cad_pid);
>
> void (*pm_power_off_prepare)(void);
>
> +/* called with rcu_read_lock, creds are safe */
> +static inline int set_one_prio_perm(struct task_struct *p)
> +{
> + const struct cred *cred = current_cred(), *pcred = __task_cred(p);
> +
> + if (pcred->user->user_ns == cred->user->user_ns &&
> +   (pcred->uid == cred->euid ||
> +   pcred->euid == cred->euid))
> + return 1;
> + if (ns_capable(pcred->user->user_ns, CAP_SYS_NICE))
> + return 1;
> + return 0;
> +}

uninline. Document return value?

>
> ...
>

_____
Containers mailing list
Containers@lists.linux-foundation.org

Subject: Re: [PATCH 7/9] add a user namespace owner of ipc ns
Posted by akpm on Fri, 18 Feb 2011 23:59:31 GMT
View Forum Message <> Reply to Message

On Thu, 17 Feb 2011 15:03:49 +0000
"Serge E. Hallyn" <serge@hallyn.com> wrote:

>
> ...
>
> --- a/include/linux/ipc_namespace.h
> +++ b/include/linux/ipc_namespace.h
> @@ -24,6 +24,7 @@ struct ipc_ids {
> struct idr ipcs_idr;
> };
>
> +struct user_namespace;

Move to top of file.

> struct ipc_namespace {
> atomic_t count;
> struct ipc_ids ids[3];
> @@ -56,6 +57,8 @@ struct ipc_namespace {
> unsigned int mq_msg_max; /* initialized to DFLT_MSGMAX */
> unsigned int mq_msgsize_max; /* initialized to DFLT_MSGSIZEMAX */
>
> + /* user_ns which owns the ipc ns */
> + struct user_namespace *user_ns;
> };
>
> extern struct ipc_namespace init_ipc_ns;
> diff --git a/ipc/msgutil.c b/ipc/msgutil.c
> index f095ee2..d91ff4b 100644
> --- a/ipc/msgutil.c
> +++ b/ipc/msgutil.c
> @@ -20,6 +20,8 @@
>
> DEFINE_SPINLOCK(mq_lock);
>
> +extern struct user_namespace init_user_ns;

Should be declared in .h, not in .c.

> /*

> * The next 2 defines are here bc this is the only file
> * compiled when either CONFIG_SYSVIPC and CONFIG_POSIX_MQUEUE
>
> ...
>

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

---

Subject: Re: [PATCH 9/9] userns: check user namespace for task->file uid equivalence checks
Posted by akpm on Fri, 18 Feb 2011 23:59:35 GMT
View Forum Message <> Reply to Message

On Thu, 17 Feb 2011 15:04:07 +0000
"Serge E. Hallyn" <serge@hallyn.com> wrote:

> Cheat for now and say all files belong to init_user_ns. Next
> step will be to let superblocks belong to a user_ns, and derive
> inode_userns(inode) from inode->i_sb->s_user_ns. Finally we'll
> introduce more flexible arrangements.
>
>
> ...
>
> +
> +/*
> + * return 1 if current either has CAP_FOWNER to the
> + * file, or owns the file.
> + */
> +int is_owner_or_cap(const struct inode *inode)
> +{
> + struct user_namespace *ns = inode_userns(inode);
> +
> + if (current_user_ns() == ns && current_fsuid() == inode->i_uid)
> + return 1;
> + if (ns_capable(ns, CAP_FOWNER))
> + return 1;
> + return 0;
> +}

bool?

> +EXPORT_SYMBOL(is_owner_or_cap);

There's a fairly well adhered to convention that global symbols (and
often static symbols) have a prefix which identifies the subsystem to
which they belong. This patchset rather scorns that convention.

Most of these identifiers are pretty obviously from the capability
subsystem, but still...

>
> ...
>

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

---

Subject: Re: [PATCH 6/9] user namespaces: convert all capable checks in kernel/sys.c
Posted by akpm on Sat, 19 Feb 2011 00:01:24 GMT
View Forum Message <> Reply to Message

On Thu, 17 Feb 2011 15:03:42 +0000
"Serge E. Hallyn" <serge@hallyn.com> wrote:

> @@ -1177,8 +1189,11 @@ SYSCALL_DEFINE2(sethostname, char __user *, name, int, len)
> int errno;
> char tmp[__NEW_UTS_LEN];
>
> - if (!ns_capable(current->nsproxy->uts_ns->user_ns, CAP_SYS_ADMIN))
> + if (!ns_capable(current->nsproxy->uts_ns->user_ns, CAP_SYS_ADMIN)) {
> + printk(KERN_NOTICE "%s: did not have CAP_SYS_ADMIN\n", __func__);
> return -EPERM;
> + }
> + printk(KERN_NOTICE "%s: did have CAP_SYS_ADMIN\n", __func__);
> if (len < 0 || len > __NEW_UTS_LEN)
> return -EINVAL;
> down_write(&uts_sem);

Left over debugging printks?

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

---

Subject: Re: [PATCH 4/9] allow killing tasks in your own or child userns

Posted by Daniel Lezcano on Sat, 19 Feb 2011 10:55:01 GMT
View Forum Message <> Reply to Message

On 02/17/2011 04:03 PM, Serge E. Hallyn wrote:
> Changelog:
> Dec 8: Fixed bug in my check_kill_permission pointed out by
> Eric Biederman.
> Dec 13: Apply Eric's suggestion to pass target task into kill_ok_by_cred()
> for clarity
> Dec 31: address comment by Eric Biederman:
> don't need cred/tcred in check_kill_permission.
> Jan 1: use const cred struct.
> Jan 11: Per Bastian Blank's advice, clean up kill_ok_by_cred().
> Feb 16: kill_ok_by_cred: fix bad parentheses
>
> Signed-off-by: Serge E. Hallyn<serge.hallyn@canonical.com>

Acked-by: Daniel Lezcano <daniel.lezcano@free.fr>

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

Subject: Re: [PATCH 5/9] Allow ptrace from non-init user namespaces
Posted by Daniel Lezcano on Sat, 19 Feb 2011 17:49:47 GMT
View Forum Message <> Reply to Message

On 02/17/2011 04:03 PM, Serge E. Hallyn wrote:
> ptrace is allowed to tasks in the same user namespace according to
> the usual rules (i.e. the same rules as for two tasks in the init
> user namespace). ptrace is also allowed to a user namespace to
> which the current task the has CAP_SYS_PTRACE capability.
>
> Changelog:
> Dec 31: Address feedback by Eric:
> . Correct ptrace uid check
> . Rename may_ptrace_ns to ptrace_capable
> . Also fix the cap_ptrace checks.
> Jan 1: Use const cred struct
> Jan 11: use task_ns_capable() in place of ptrace_capable().
>
> Signed-off-by: Serge E. Hallyn<serge.hallyn@canonical.com>
Acked-by: Daniel Lezcano <daniel.lezcano@free.fr>

_____
Containers mailing list
Containers@lists.linux-foundation.org

Subject: Re: [PATCH 6/9] user namespaces: convert all capable checks in kernel/sys.c
Posted by Daniel Lezcano on Sat, 19 Feb 2011 17:52:46 GMT
View Forum Message <> Reply to Message

On 02/17/2011 04:03 PM, Serge E. Hallyn wrote:
> This allows setuid/setgid in containers. It also fixes some
> corner cases where kernel logic foregoes capability checks when
> uids are equivalent. The latter will need to be done throughout
> the whole kernel.
>
> Changelog:
> Jan 11: Use nsown_capable() as suggested by Bastian Blank.
> Jan 11: Fix logic errors in uid checks pointed out by Bastian.
> Feb 15: allow prlimit to current (was regression in previous version)
>
> Signed-off-by: Serge E. Hallyn<serge.hallyn@canonical.com>

Acked-by: Daniel Lezcano <daniel.lezcano@free.fr>

>
> - if (!ns_capable(current->nsproxy->uts_ns->user_ns, CAP_SYS_ADMIN))
> + if (!ns_capable(current->nsproxy->uts_ns->user_ns, CAP_SYS_ADMIN)) {
> + printk(KERN_NOTICE "%s: did not have CAP_SYS_ADMIN\n", __func__);
>   return -EPERM;
> + }
> + printk(KERN_NOTICE "%s: did have CAP_SYS_ADMIN\n", __func__);

A couple of printk left here.

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

Subject: Re: [PATCH 7/9] add a user namespace owner of ipc ns
Posted by Daniel Lezcano on Sat, 19 Feb 2011 17:57:25 GMT
View Forum Message <> Reply to Message

On 02/17/2011 04:03 PM, Serge E. Hallyn wrote:
> Changelog:
> Feb 15: Don't set new ipc->user_ns if we didn't create a new

> ipc_ns.
>
> Signed-off-by: Serge E. Hallyn<serge.hallyn@canonical.com>
> ---

Acked-by: Daniel Lezcano <daniel.lezcano@free.fr>

[...]

> + ns->user_ns = old_ns->user_ns;
> + get_user_ns(ns->user_ns);

A mindless change.

ns->user_ns = get_user_ns(old_ns->user_ns);

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

---

Subject: Re: [PATCH 8/9] user namespaces: convert several capable() calls
Posted by Daniel Lezcano on Sat, 19 Feb 2011 19:07:56 GMT
View Forum Message <> Reply to Message

On 02/17/2011 04:03 PM, Serge E. Hallyn wrote:
> CAP_IPC_OWNER and CAP_IPC_LOCK can be checked against current_user_ns(),
> because the resource comes from current's own ipc namespace.
>
> setuid/setgid are to uids in own namespace, so again checks can be
> against current_user_ns().
>
> Changelog:
> Jan 11: Use task_ns_capable() in place of sched_capable().
> Jan 11: Use nsown_capable() as suggested by Bastian Blank.
> Jan 11: Clarify (hopefully) some logic in futex and sched.c
> Feb 15: use ns_capable for ipc, not nsown_capable
>
> Signed-off-by: Serge E. Hallyn<serge.hallyn@canonical.com>
> ---

Acked-by: Daniel Lezcano <daniel.lezcano@free.fr>

_____
Containers mailing list
Containers@lists.linux-foundation.org

Subject: Re: [PATCH 9/9] userns: check user namespace for task->file uid equivalence checks
Posted by Daniel Lezcano on Sat, 19 Feb 2011 19:22:39 GMT
View Forum Message <> Reply to Message

On 02/17/2011 04:04 PM, Serge E. Hallyn wrote:
> Cheat for now and say all files belong to init_user_ns. Next
> step will be to let superblocks belong to a user_ns, and derive
> inode_userns(inode) from inode->i_sb->s_user_ns. Finally we'll
> introduce more flexible arrangements.
>
> Changelog:
> Feb 15: make is_owner_or_cap take const struct inode
>
> Signed-off-by: Serge E. Hallyn<serge.hallyn@canonical.com>
> ---

Acked-by: Daniel Lezcano <daniel.lezcano@free.fr>

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

Subject: Re: [PATCH 2/9] security: Make capabilities relative to the user namespace.
Posted by David Howells on Wed, 23 Feb 2011 11:40:09 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn <serge@hallyn.com> wrote:

> int (*capable) (struct task_struct *tsk, const struct cred *cred,
> - int cap, int audit);
> + struct user_namespace *ns, int cap, int audit);

Hmmm... A chunk of the contents of the cred struct are user-namespaced.
Could you add the user_namespace pointer to the cred struct and thus avoid
passing it as an argument to other things.

In fact, I think you probably have to do this so that cachefiles, for example,
can override the user namespace when it operates on behalf of a process that's
in another namespace.

In fact, looking later on in your patch, I see:

> - || (cap_capable(current, current_cred(), CAP_SETPCAP,
> + || (cap_capable(current, current_cred(),
> + current_cred()->user->user_ns, CAP_SETPCAP,

so the user_ns _is_ already available through the creds. So is there really a need to pass it as an argument to anything that already takes a cred?

> * @cred contains the credentials to use.
> + * @ns contains the user namespace we want the capability in
> * @cap contains the capability <include/linux/capability.h>.

That should be tabbed to match the lines either side.

David

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

---

Subject: Re: [PATCH 2/9] security: Make capabilities relative to the user namespace.
Posted by David Howells on Wed, 23 Feb 2011 12:01:57 GMT
View Forum Message <> Reply to Message

David Howells <dhowells@redhat.com> wrote:

> > int (*capable) (struct task_struct *tsk, const struct cred *cred,
> > - int cap, int audit);
> > + struct user_namespace *ns, int cap, int audit);
>
> Hmmm... A chunk of the contents of the cred struct are user-namespaced.
> Could you add the user_namespace pointer to the cred struct and thus avoid
> passing it as an argument to other things.

Ah, no... Ignore that, I think I see that you do need it.

> +int cap_capable(struct task_struct *tsk, const struct cred *cred,
> + struct user_namespace *targ_ns, int cap, int audit)
> {
> - return cap_raised(cred->cap_effective, cap) ? 0 : -EPERM;
> + for (;;) {
> + /* The creator of the user namespace has all caps. */
> + if (targ_ns != &init_user_ns && targ_ns->creator == cred->user)
> + return 0;

Why is that last comment so? Why should the creating namespace sport all

possible capabilities? Do you have to have all capabilities available to you to be permitted create a new user namespace?

Also, would it be worth having a separate cap_ns_capable()? Wouldn't most calls to cap_capable() only be checking the caps granted in the current user namespace?

David

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

---

Subject: User namespaces and keys
Posted by David Howells on Wed, 23 Feb 2011 12:05:29 GMT
View Forum Message <> Reply to Message

I guess we need to look at how to mix keys and namespaces again.

Possibly the trickiest problem with keys is how to upcall key construction to /sbin/request-key when the keys may be of a different user namespace.

David

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

---

Subject: Re: [PATCH 2/9] security: Make capabilities relative to the user namespace.
Posted by serge on Wed, 23 Feb 2011 13:43:52 GMT
View Forum Message <> Reply to Message

Quoting David Howells (dhowells@redhat.com):
> David Howells <dhowells@redhat.com> wrote:
>
> > > int (*capable) (struct task_struct *tsk, const struct cred *cred,
> > > - int cap, int audit);
> > > + struct user_namespace *ns, int cap, int audit);
> >
> > Hmmm... A chunk of the contents of the cred struct are user-namespaced.
> > Could you add the user_namespace pointer to the cred struct and thus avoid
> > passing it as an argument to other things.
>
> Ah, no... Ignore that, I think I see that you do need it.

Thanks for reviewing, David.

> > +int cap_capable(struct task_struct *tsk, const struct cred *cred,
> > + struct user_namespace *targ_ns, int cap, int audit)
> > {
> > - return cap_raised(cred->cap_effective, cap) ? 0 : -EPERM;
> > + for (;;) {
> > + /* The creator of the user namespace has all caps. */
> > + if (targ_ns != &init_user_ns && targ_ns->creator == cred->user)
> > +   return 0;
>
> Why is that last comment so? Why should the creating namespace sport all
> possible capabilities? Do you have to have all capabilities available to you
> to be permitted create a new user namespace?

It's not the creating namespace, but the creating user, which has all caps.

So for instance, if uid 500 in init_user_ns creates a namespace, then:
  a. uid 500 in init_user_ns has all caps to the child user_ns, so it can
     kill the tasks in the userns, clean up, etc.
  b. uid X in any other child user_ns has no caps to the child user_ns.
  c. root in init_user_ns has whatever capabilities are in his pE to the
     child user_ns. Again, this is so that the admin in any user_ns can
     clean up any messes made by users in his user_ns.

One of the goals of the user namespaces it to make it safe to allow
unprivileged users to create child user namespaces in which they have
targeted privilege. Anything which happens in that child user namespace
should be:

  a. constrained to resources which the user can control anyway
  b. able to be cleaned up by the user
  c. (especially) able to be cleaned up by the privileged user in the
     parent user_ns.

> Also, would it be worth having a separate cap_ns_capable()? Wouldn't most
> calls to cap_capable() only be checking the caps granted in the current user
> namespace?

Hm. There is nsown_capable() which is targeted to current_userns(), but
that still needs to enable the caps for the privileged ancestors as
described above.

thanks,
-serge

_____
Containers mailing list

Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containe rs

Subject: Re: User namespaces and keys
Posted by serge on Wed, 23 Feb 2011 13:58:14 GMT
View Forum Message <> Reply to Message

Quoting David Howells (dhowells@redhat.com):
>
> I guess we need to look at how to mix keys and namespaces again.

>From strictly kernel pov, at the moment, keys are strictly usable only
by the user in your own user namespace.

We may want to look at this again, but for now I think that would be a
safe enough default. Later, we'll probably want the user creating a
child_user_ns to allow his keys to be inherited by the child user_ns.
Though, as I type that, it seems to me that that'll just become a
maintenance pain, and it's just plain safer to have the user re-enter
his keys, sharing them over a file if needed.

I'm going to not consider the TPM at the moment :)

> Possibly the trickiest problem with keys is how to upcall key construction to
> /sbin/request-key when the keys may be of a different user namespace.

Hm, jinkeys, yes.

-serge

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

Subject: Re: User namespaces and keys
Posted by ebiederm on Wed, 23 Feb 2011 14:46:00 GMT
View Forum Message <> Reply to Message

"Serge E. Hallyn" <serge@hallyn.com> writes:

> Quoting David Howells (dhowells@redhat.com):
>>
>> I guess we need to look at how to mix keys and namespaces again.
>
>>From strictly kernel pov, at the moment, keys are strictly usable only

> by the user in your own user namespace.
>
> We may want to look at this again, but for now I think that would be a
> safe enough default. Later, we'll probably want the user creating a
> child_user_ns to allow his keys to be inherited by the child user_ns.
> Though, as I type that, it seems to me that that'll just become a
> maintenance pain, and it's just plain safer to have the user re-enter
> his keys, sharing them over a file if needed.
>
> I'm going to not consider the TPM at the moment :)
>
>> Possibly the trickiest problem with keys is how to upcall key construction to
>> /sbin/request-key when the keys may be of a different user namespace.
>
> Hm, jinkeys, yes.

Serge short term this is where I think we need to add a check or two so
that keys only work in the init_user_ns. When the rest of the details
are sorted out we can open up the use of keys some more.

As the first stage of converting the network stack we added patches
that turned off everything in non init namespaces. Those patches
were trivial and easy to review, and it made the conversion process
a lot easier. I suspect for keys and possibly security modules and
anything else that does is related we want to turn off by default.

Then once the core of user namespaces is safe to unshare without
privilege we can come back and get the more difficult bits.

Eric

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

---

Subject: Re: User namespaces and keys
Posted by David Howells on Wed, 23 Feb 2011 15:06:14 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn <serge@hallyn.com> wrote:

> > I guess we need to look at how to mix keys and namespaces again.
>
> From strictly kernel pov, at the moment, keys are strictly usable only
> by the user in your own user namespace.

I'm not sure that's currently completely true. Key quota maintenance is

namespaced, and the key's owner UID/GID belong to that namespace, so that's okay, but:

(*) key_task_permission() does not distinguish UIDs and GIDs from different namespaces.

(*) A key can be referred to by its serial number, no matter whose namespace it is in, and will yield up its given UID/GID, even if these aren't actually meaningful in your namespace.

This means request_key() can successfully upcall at the moment.

I wonder if I should make the following changes:

(1) If the key and the accessor are in different user namespaces, then skip the UID and GID comparisons in key_task_permission(). That means that to be able to access the key you'd have to possess the key and the key would have to grant you Possessor access, or the key would have to grant you Other access.

(2) If the key and someone viewing the key description are in different namespaces, then indicate that the UID and the GID are -1, irrespective of the actual values.

(3) When an upcall is attempting to instantiate a key, it is allowed to access the keys of requestor using the requestor's credentials (UID, GID, groups, security label). Ensure that this will be done in the requestor's user namespace.

Nothing should need to be done here, since search_process_keyrings() switches to the requestor's creds.

Oh, and are security labels user-namespaced?

> We may want to look at this again, but for now I think that would be a
> safe enough default. Later, we'll probably want the user creating a
> child_user_ns to allow his keys to be inherited by the child user_ns.

That depends what you mean by 'allow his keys to be inherited'. Do you mean copying all the creator's keys en mass? You may find all you need to do is to provide the new intended user with a new session keyring with a link back to the creator's session keyring.

> Though, as I type that, it seems to me that that'll just become a
> maintenance pain, and it's just plain safer to have the user re-enter
> his keys,

That would certainly be simpler.

> sharing them over a file if needed.

I'm not sure what you mean by that. Do you mean some sort of cred passing
similar to that that can be done over AF_UNIX sockets with fds?

> I'm going to not consider the TPM at the moment :)

I'm not sure the TPM is that much of a problem, assuming you're just referring
to its keystore capability...

David

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

---

Subject: Re: User namespaces and keys
Posted by ebiederm on Wed, 23 Feb 2011 15:45:49 GMT
View Forum Message <> Reply to Message

David Howells <dhowells@redhat.com> writes:

> Serge E. Hallyn <serge@hallyn.com> wrote:
>
>> > I guess we need to look at how to mix keys and namespaces again.
>>
>> From strictly kernel pov, at the moment, keys are strictly usable only
>> by the user in your own user namespace.
>
> I'm not sure that's currently completely true. Key quota maintenance is
> namespaced, and the key's owner UID/GID belong to that namespace, so that's
> okay, but:
>
> (*) key_task_permission() does not distinguish UIDs and GIDs from different
>   namespaces.
>
> (*) A key can be referred to by its serial number, no matter whose namespace
>   it is in, and will yield up its given UID/GID, even if these aren't
>   actually meaningful in your namespace.
>
>   This means request_key() can successfully upcall at the moment.
>
> I wonder if I should make the following changes:
>
> (1) If the key and the accessor are in different user namespaces, then skip
>   the UID and GID comparisons in key_task_permission(). That means that to

> be able to access the key you'd have to possess the key and the key would
> have to grant you Possessor access, or the key would have to grant you
> Other access.
>
> (2) If the key and someone viewing the key description are in different
> namespaces, then indicate that the UID and the GID are -1, irrespective of
> the actual values.
>
> (3) When an upcall is attempting to instantiate a key, it is allowed to access
> the keys of requestor using the requestor's credentials (UID, GID, groups,
> security label). Ensure that this will be done in the requestor's user
> namespace.
>
> Nothing should need to be done here, since search_process_keyrings()
> switches to the requestor's creds.
>
> Oh, and are security labels user-namespaced?

Not at this time. The user namespace as currently merged is little more
than a place holder for a proper implementation. Serge is busily
fleshing out that proper implementation.

Until we reach the point where all checks that have historically been
"if (uid1 == uid2)" become "if ((uidns1 == uidns2) && (uid1 == uid2))"
there will be problems.

The security labels and probably lsm's in general need to be per user
namespace but we simply have not gotten that far. For the short term I
will be happy when we get a minimally usable user namespace.

Eric

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

---

Subject: Re: User namespaces and keys
Posted by Serge E. Hallyn on Wed, 23 Feb 2011 15:53:29 GMT
View Forum Message <> Reply to Message

Quoting Eric W. Biederman (ebiederm@xmission.com):
> David Howells <dhowells@redhat.com> writes:
>
> > Serge E. Hallyn <serge@hallyn.com> wrote:
> >
> >> > I guess we need to look at how to mix keys and namespaces again.
> >>

> >> From strictly kernel pov, at the moment, keys are strictly usable only
> >> by the user in your own user namespace.
> >
> > I'm not sure that's currently completely true. Key quota maintenance is
> > namespaced, and the key's owner UID/GID belong to that namespace, so that's
> > okay, but:
> >
> > (*) key_task_permission() does not distinguish UIDs and GIDs from different
> >   namespaces.
> >
> > (*) A key can be referred to by its serial number, no matter whose namespace
> >   it is in, and will yield up its given UID/GID, even if these aren't
> >   actually meaningful in your namespace.
> >
> >   This means request_key() can successfully upcall at the moment.
> >
> > I wonder if I should make the following changes:
> >
> > (1) If the key and the accessor are in different user namespaces, then skip
> >   the UID and GID comparisons in key_task_permission(). That means that to
> >   be able to access the key you'd have to possess the key and the key would
> >   have to grant you Possessor access, or the key would have to grant you
> >   Other access.
> >
> > (2) If the key and someone viewing the key description are in different
> >   namespaces, then indicate that the UID and the GID are -1, irrespective of
> >   the actual values.
> >
> > (3) When an upcall is attempting to instantiate a key, it is allowed to access
> >   the keys of requestor using the requestor's credentials (UID, GID, groups,
> >   security label). Ensure that this will be done in the requestor's user
> >   namespace.
> >
> >   Nothing should need to be done here, since search_process_keyrings()
> >   switches to the requestor's creds.
> >
> > Oh, and are security labels user-namespaced?
>
> Not at this time. The user namespace as currently merged is little more
> than a place holder for a proper implementation. Serge is busily
> fleshing out that proper implementation.
>
> Until we reach the point where all checks that have historically been
> "if (uid1 == uid2)" become "if ((uidns1 == uidns2) && (uid1 == uid2))"
> there will be problems.
>
> The security labels and probably lsm's in general need to be per user
> namespace but we simply have not gotten that far. For the short term I

> will be happy when we get a minimally usable user namespace.

Note also that when Eric brought this up at the LSM mini-conf two or three
years ago, there was pretty general, strong objection to the idea.

Like Eric says, I think that'll have to wait. In the meantime, isolating
user namespace sandboxes (or containers) using simple LSM configurations
is a very good idea.

-serge

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

---

Subject: Re: [PATCH 2/9] security: Make capabilities relative to the user namespace.
Posted by David Howells on Wed, 23 Feb 2011 16:59:19 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn <serge@hallyn.com> wrote:

> + /* If you have the capability in a parent user ns you have it
> +   * in the over all children user namespaces as well, so see
> +   * if this process has the capability in the parent user
> +   * namespace.
> +   */

"... in the over all children user namespaces ..." I think need a couple of
words dropping.

David

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

---

Subject: Re: [PATCH 5/9] Allow ptrace from non-init user namespaces
Posted by David Howells on Wed, 23 Feb 2011 17:05:41 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn <serge@hallyn.com> wrote:

> ptrace is allowed to tasks in the same user namespace according to

There's a verb missing after 'allowed to' - presumably 'trace'.

David

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

---

Subject: Re: [PATCH 5/9] Allow ptrace from non-init user namespaces
Posted by David Howells on Wed, 23 Feb 2011 17:11:01 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn <serge@hallyn.com> wrote:

> +int same_or_ancestor_user_ns(struct task_struct *task,
> + struct task_struct *victim)
> +{
> + struct user_namespace *u1 = task_cred_xxx(task, user)->user_ns;
> + struct user_namespace *u2 = task_cred_xxx(victim, user)->user_ns;

Hmmm. task_cred_xxx() uses task->real_cred, which is correct for victim (the object), but normally you'd use task->cred for task (the subject). However, in this case, I think it's probably okay.

David

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

---

Subject: Re: [PATCH 1/9] Add a user_namespace as creator/owner of uts_namespace
Posted by David Howells on Wed, 23 Feb 2011 17:16:33 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn <serge@hallyn.com> wrote:

> struct uts_namespace {
> struct kref kref;
> struct new_utsname name;
> + struct user_namespace *user_ns;
> };

If a uts_namespace belongs to a user_namespace, should CLONE_NEWUSER then imply CLONE_NEWUTS?

Or is uts_namespace::user_ns more an implication that the set of users in user_namespace are the only ones authorised to alter the uts data.

I presume that the uts_namespace of a process must be owned by one of the user_namespaces in the alternating inheritance chain of namespaces and their creators leading from current_user_ns() to init_user_ns.

With that in mind, looking at patch 3:

```
- if (!capable(CAP_SYS_ADMIN))
+ if (!ns_capable(current->nsproxy->uts_ns->user_ns, CAP_SYS_ADMIN))
```

what is it you're actually asking? I presume it's 'does this user have CAP_SYS_ADMIN capability over objects belonging to the uts_namespace's user_namespace?'

So, to look at the important bit of patch 2:

```
-int cap_capable(struct task_struct *tsk, const struct cred *cred, int cap,
-  int audit)
+int cap_capable(struct task_struct *tsk, const struct cred *cred,
+  struct user_namespace *targ_ns, int cap, int audit)
 {
- return cap_raised(cred->cap_effective, cap) ? 0 : -EPERM;
+ for (;;) {
+  /* The creator of the user namespace has all caps. */
+  if (targ_ns != &init_user_ns && targ_ns->creator == cred->user)
+   return 0;
+
+  /* Do we have the necessary capabilities? */
+  if (targ_ns == cred->user->user_ns)
+   return cap_raised(cred->cap_effective, cap) ? 0 : -EPERM;
+
+  /* Have we tried all of the parent namespaces? */
+  if (targ_ns == &init_user_ns)
+   return -EPERM;
+
+  /* If you have the capability in a parent user ns you have it
+   * in the over all children user namespaces as well, so see
+   * if this process has the capability in the parent user
+   * namespace.
+   */
+  targ_ns = targ_ns->creator->user_ns;
+ }
+
+ /* We never get here */
+ return -EPERM;
```

```
}
```

On entry, as we're called from ns_capable(), cred->user is the user that the current process is running as, and, as such, may be in a separate namespace from uts_namespace - which may itself be in a separate namespace from init_user_ns.

So, assume for the sake of argument that there are three user_namespaces along the chain from the calling process to the root, and that the uts_namespace belongs to the middle one.

```
 if (targ_ns != &init_user_ns && targ_ns->creator == cred->user)
  return 0;
```

Can never match because targ_ns->creator cannot be cred->user; even if the uts_namespace belongs to our namespace, given that the creator lies outside our namespace.

```
 if (targ_ns == cred->user->user_ns)
  return cap_raised(cred->cap_effective, cap) ? 0 : -EPERM;
```

Can only match if we are in the target user_namespace (ie. the one to which uts_namespace belongs), whether or not we have CAP_SYS_ADMIN.

Which means that unless the uts_namespace belongs to our user_namespace, we cannot change it. Is that correct?

So ns_capable() restricts you to only doing interesting things to objects that belong to a user_namespace if they are in your own user_namespace. Is that correct?

If that is so, is the loop required for ns_capable()?

Looking further at patch 2:

```
#define nsown_capable(cap) (ns_capable(current_user_ns(), (cap)))
```

Given what I've said above, I presume the loop isn't necessary here either.

I think you're using ns_capable() in two different ways:

(1) You're using it to see if a process has power over its descendents in a user_namespace that can be traced back to a clone() that it did with CLONE_NEWUSER.

    For example, automatically granting a process permission to kill

descendents in a namespace it created.

(2) You're using it to see if a process can access objects that might be
outside its own user_namespace.

For example, setting the hostname.

Is it worth giving two different interfaces to make this clearer (even if they
actually do the same thing)?

Sorry if this seems rambly, but I'm trying to get my head round your code.

David

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

---

Subject: Re: User namespaces and keys
Posted by Casey Schaufler on Wed, 23 Feb 2011 19:24:20 GMT
View Forum Message <> Reply to Message

On 2/23/2011 7:53 AM, Serge E. Hallyn wrote:
> Quoting Eric W. Biederman (ebiederm@xmission.com):
>> David Howells <dhowells@redhat.com> writes:
>>
>>> Serge E. Hallyn <serge@hallyn.com> wrote:
>>>
>>>> I guess we need to look at how to mix keys and namespaces again.
>>>> From strictly kernel pov, at the moment, keys are strictly usable only
>>>> by the user in your own user namespace.
>>> I'm not sure that's currently completely true. Key quota maintenance is
>>> namespaced, and the key's owner UID/GID belong to that namespace, so that's
>>> okay, but:
>>>
>>> (*) key_task_permission() does not distinguish UIDs and GIDs from different
>>>   namespaces.
>>>
>>> (*) A key can be referred to by its serial number, no matter whose namespace
>>>   it is in, and will yield up its given UID/GID, even if these aren't
>>>   actually meaningful in your namespace.
>>>
>>>   This means request_key() can successfully upcall at the moment.
>>>
>>> I wonder if I should make the following changes:

>>>
>>> (1) If the key and the accessor are in different user namespaces, then skip
>>>   the UID and GID comparisons in key_task_permission(). That means that to
>>>   be able to access the key you'd have to possess the key and the key would
>>>   have to grant you Possessor access, or the key would have to grant you
>>>   Other access.
>>>
>>> (2) If the key and someone viewing the key description are in different
>>>   namespaces, then indicate that the UID and the GID are -1, irrespective of
>>>   the actual values.
>>>
>>> (3) When an upcall is attempting to instantiate a key, it is allowed to access
>>>   the keys of requestor using the requestor's credentials (UID, GID, groups,
>>>   security label). Ensure that this will be done in the requestor's user
>>>   namespace.
>>>
>>>   Nothing should need to be done here, since search_process_keyrings()
>>>   switches to the requestor's creds.
>>>
>>> Oh, and are security labels user-namespaced?
>> Not at this time. The user namespace as currently merged is little more
>> than a place holder for a proper implementation. Serge is busily
>> fleshing out that proper implementation.
>>
>> Until we reach the point where all checks that have historically been
>> "if (uid1 == uid2)" become "if ((uidns1 == uidns2) && (uid1 == uid2))"
>> there will be problems.
>>
>> The security labels and probably lsm's in general need to be per user
>> namespace but we simply have not gotten that far. For the short term I
>> will be happy when we get a minimally usable user namespace.
> Note also that when Eric brought this up at the LSM mini-conf two or three
> years ago, there was pretty general, strong objection to the idea.
>
> Like Eric says, I think that'll have to wait. In the meantime, isolating
> user namespace sandboxes (or containers) using simple LSM configurations
> is a very good idea.

I confess that I remain less well educated on namespaces than
I probably should be, but with what I do know it seems that the
relationships between user namespaces and LSMs are bound to be
strained from the beginning. Some LSMs (SELinux and Smack) are
providing similar sandbox capabilities to what you get from user
namespaces, but from different directions and with different
use cases.

> -serge
> --

> To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
> the body of a message to majordomo@vger.kernel.org
> More majordomo info at http://vger.kernel.org/majordomo-info.html
> Please read the FAQ at http://www.tux.org/lkml/
>
>

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

---

Subject: Re: User namespaces and keys
Posted by ebiederm on Wed, 23 Feb 2011 20:55:01 GMT
View Forum Message <> Reply to Message

Casey Schaufler <casey@schaufler-ca.com> writes:

> I confess that I remain less well educated on namespaces than
> I probably should be, but with what I do know it seems that the
> relationships between user namespaces and LSMs are bound to be
> strained from the beginning. Some LSMs (SELinux and Smack) are
> providing similar sandbox capabilities to what you get from user
> namespaces, but from different directions and with different
> use cases.

Casey I won't argue about the possibility of things being strained, but
I think if we focus on the semantics and not on the end goal of exactly
how the pieces are to be used there can be some reasonable dialog.

>From 10,000 feet an user namespace represents one administrative domain.
uids, gids and other external tokens are all controlled by a single
group with a single security policy. In that single administrative
domain things like nfs are expected to work without translating uids and
gids. Before the implementation of user namespaces a single kernel
could not even express the ability of dealing with multiple user
namespaces simultaneously (nfs has usually punted and said despite being
multiple machines you still have to be in the same administrative domain
so the same user identifiers can be used throughout).

>From that principle we have the concept that use assigned/visible
identifiers uid, gid, capabilities, security keys?, security labels?
logically belong to the user namespace.

Which means in implementing there are two pieces of work in implementing
the user namespace.

- Find all of the interesting comparisons and change them from
  "if (id == id)" to "if (userns == userns) && (id == id)".

- Potentially define and handle what happens when you mix user
  namespaces.

I think for the first round of this we simply want to define lsms
and the user namespace as not mixing or the lsm interfaces only being
visible if you are in the initial user namespace. Thus reserving the
definition of what happens when you have lsms and multiple user
namespaces as something we can define later. I expect the proper
definition is something that would allow nested lsm policy.

Regardless. The namespaces are all about making the identifiers that
processes deal with local to the namespace, while the underlying object
manipulations should not care.

The big advantage of the user namespace is that it is the only way I can
see to get us out of the bind we are in with suid root executables,
where we can not enable new features to untrusted applications that can
change the environment for a suid root executable, because it might
confuse those suid root executables to misusing their power. Once
inside a user namespace nothing has dangerous powers because even a
root owned process with a full set of capabilities is less powerful than
a guest user outside of the namespace. No process having dangerous
powers allows us to enable unsharing of other namespaces and potentially
other things that today are restricted with capabilities only because
they can be used to confuse a privileged executable to do something
malicious.

Eric

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

---

Subject: Re: [PATCH 1/9] Add a user_namespace as creator/owner of uts_namespace
Posted by ebiederm on Wed, 23 Feb 2011 21:21:28 GMT
View Forum Message <> Reply to Message

David Howells <dhowells@redhat.com> writes:

> Serge E. Hallyn <serge@hallyn.com> wrote:
>
>> struct uts_namespace {

>>   struct kref kref;
>>   struct new_utsname name;
>> + struct user_namespace *user_ns;
>> };
>
> If a uts_namespace belongs to a user_namespace, should CLONE_NEWUSER then
> imply CLONE_NEWUTS?
>
> Or is uts_namespace::user_ns more an implication that the set of users in
> user_namespace are the only ones authorised to alter the uts data.

The later.

> I presume that the uts_namespace of a process must be owned by one of the
> user_namespaces in the alternating inheritance chain of namespaces and their
> creators leading from current_user_ns() to init_user_ns.
>
> With that in mind, looking at patch 3:
>
> - if (!capable(CAP_SYS_ADMIN))
> + if (!ns_capable(current->nsproxy->uts_ns->user_ns, CAP_SYS_ADMIN))
>
> what is it you're actually asking? I presume it's 'does this user have
> CAP_SYS_ADMIN capability over objects belonging to the uts_namespace's
> user_namespace?'

Yes.

> So, to look at the important bit of patch 2:
>
> -int cap_capable(struct task_struct *tsk, const struct cred *cred, int cap,
> - int audit)
> +int cap_capable(struct task_struct *tsk, const struct cred *cred,
> + struct user_namespace *targ_ns, int cap, int audit)
> {
> - return cap_raised(cred->cap_effective, cap) ? 0 : -EPERM;
> + for (;;) {
> + /* The creator of the user namespace has all caps. */
> + if (targ_ns != &init_user_ns && targ_ns->creator == cred->user)
> +   return 0;
> +
> + /* Do we have the necessary capabilities? */
> + if (targ_ns == cred->user->user_ns)
> +   return cap_raised(cred->cap_effective, cap) ? 0 : -EPERM;
> +
> + /* Have we tried all of the parent namespaces? */
> + if (targ_ns == &init_user_ns)
> +   return -EPERM;

> +
> + /* If you have the capability in a parent user ns you have it
> +   * in the over all children user namespaces as well, so see
> +   * if this process has the capability in the parent user
> +   * namespace.
> +   */
> + targ_ns = targ_ns->creator->user_ns;
> + }
> +
> + /* We never get here */
> + return -EPERM;
> }
>
> On entry, as we're called from ns_capable(), cred->user is the user that the
> current process is running as, and, as such, may be in a separate namespace
> from uts_namespace - which may itself be in a separate namespace from
> init_user_ns.
>
> So, assume for the sake of argument that there are three user_namespaces along
> the chain from the calling process to the root, and that the uts_namespace
> belongs to the middle one.

So we have the nested stack of:
user_ns3->creator->user_ns == user_ns2
user_ns2->creator->user_ns == &init_user_ns
uts_ns2->user_ns == user_ns2

>
>   if (targ_ns != &init_user_ns && targ_ns->creator == cred->user)
>   return 0;
>
> Can never match because targ_ns->creator cannot be cred->user; even if the
> uts_namespace belongs to our namespace, given that the creator lies outside
> our namespace.

Initially we come in with targ_ns == user_ns2 and cred->user->user_ns in
one of (user_ns3, user_ns2, or &init_user_ns).

targ_ns takes on values user_ns2 and &init_user_ns.

So when targ_ns becomes &init_user_ns. If the user in question is
uts_ns2->user_ns->creator. This check will indeed match.

>   if (targ_ns == cred->user->user_ns)
>   return cap_raised(cred->cap_effective, cap) ? 0 : -EPERM;
>
> Can only match if we are in the target user_namespace (ie. the one to which
> uts_namespace belongs), whether or not we have CAP_SYS_ADMIN.

As before targ_ns takes on values of user_ns2 and &init_user_ns.

Which means this check will match if we have CAP_SYS_ADMIN in
&init_user_ns or in user_ns2.

> Which means that unless the uts_namespace belongs to our user_namespace, we
> cannot change it. Is that correct?

No. If you are root in a parent namespace you can also change it.

> So ns_capable() restricts you to only doing interesting things to objects that
> belong to a user_namespace if they are in your own user_namespace. Is that
> correct?

No. Root outside your user namespace is also allowed to do interesting
things.

> If that is so, is the loop required for ns_capable()?

Yes.

> Looking further at patch 2:
>
> #define nsown_capable(cap) (ns_capable(current_user_ns(), (cap)))
>
> Given what I've said above, I presume the loop isn't necessary here either.
>
>
> I think you're using ns_capable() in two different ways:
>
> (1) You're using it to see if a process has power over its descendents in a
>   user_namespace that can be traced back to a clone() that it did with
>   CLONE_NEWUSER.
>
>   For example, automatically granting a process permission to kill
>   descendents in a namespace it created.
>
> (2) You're using it to see if a process can access objects that might be
>   outside its own user_namespace.
>
>   For example, setting the hostname.
>
> Is it worth giving two different interfaces to make this clearer (even if they
> actually do the same thing)?
>
>

> Sorry if this seems rambly, but I'm trying to get my head round your
> code.

I am all for making that loop a little clearer, because it is something
you have to pause and think about to understand but so far I don't think
the loop is wrong, and it is simple.

Eric

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

---

Subject: Re: User namespaces and keys
Posted by Casey Schaufler on Wed, 23 Feb 2011 21:37:16 GMT
View Forum Message <> Reply to Message

On 2/23/2011 12:55 PM, Eric W. Biederman wrote:
> Casey Schaufler <casey@schaufler-ca.com> writes:
>
>> I confess that I remain less well educated on namespaces than
>> I probably should be, but with what I do know it seems that the
>> relationships between user namespaces and LSMs are bound to be
>> strained from the beginning. Some LSMs (SELinux and Smack) are
>> providing similar sandbox capabilities to what you get from user
>> namespaces, but from different directions and with different
>> use cases.
> Casey I won't argue about the possibility of things being strained, but
> I think if we focus on the semantics and not on the end goal of exactly
> how the pieces are to be used there can be some reasonable dialog.

I'm sure that there will be cases where they will work together
like horses in a troika. Making sensible semantics for the interactions
is key, and it is entirely possible that in some cases a comparison
of semantics and behaviors will lead an end user to chose either an
LSM or namespaces over the combination. Just like I expect that even
when we allow multiple LSMs the SELinux and Smack combination will be
rare among the sane.

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

---

Subject: Re: [PATCH 1/9] Add a user_namespace as creator/owner of

uts_namespace
Posted by David Howells on Wed, 23 Feb 2011 23:19:08 GMT
View Forum Message <> Reply to Message

Eric W. Biederman <ebiederm@xmission.com> wrote:

> > Which means that unless the uts_namespace belongs to our user_namespace, we
> > cannot change it. Is that correct?
>
> No. If you are root in a parent namespace you can also change it.

But surely, by definition, if you're a user in this namespace, you can't also
be root in a parent namespace...

For the case I worked through current_user() is a member of current_user_ns()
and can't also be a member of its parent, grandparent, etc. - or can it?

David

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

Subject: Re: [PATCH 1/9] Add a user_namespace as creator/owner of uts_namespace
Posted by ebiederm on Wed, 23 Feb 2011 23:54:12 GMT
View Forum Message <> Reply to Message

David Howells <dhowells@redhat.com> writes:

> Eric W. Biederman <ebiederm@xmission.com> wrote:
>
>> > Which means that unless the uts_namespace belongs to our user_namespace, we
>> > cannot change it. Is that correct?
>>
>> No. If you are root in a parent namespace you can also change it.
>
> But surely, by definition, if you're a user in this namespace, you can't also
> be root in a parent namespace...

To be clear the case you looked at was:

> - if (!capable(CAP_SYS_ADMIN))
> + if (!ns_capable(current->nsproxy->uts_ns->user_ns, CAP_SYS_ADMIN))
>
> what is it you're actually asking? I presume it's 'does this user have
> CAP_SYS_ADMIN capability over objects belonging to the uts_namespace's

> user_namespace?'

Here "current->nsproxy->uts_ns->user_ns" (the target_ns value) is only
refers to the uts_ns we are talking about.

The user itself comes from current_user().

> For the case I worked through current_user() is a member of current_user_ns()
> and can't also be a member of its parent, grandparent, etc. - or can
> it?

Right now if you are looking at current_user() because of limitations in
the creation ordering I think you are correct.

However in the near term pile of changes to merge, are the syscalls for
joining an existing namespace. At which point there is no reason in
general to suppose the current limitations of creation apply.

Although it is conceivable that unshare of namespaces can also get you
to someplace similar to joining prexisting namespaces.

Eric

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

---

Subject: Re: [PATCH 5/9] Allow ptrace from non-init user namespaces
Posted by serge on Thu, 24 Feb 2011 00:43:37 GMT

View Forum Message <> Reply to Message

Quoting Andrew Morton (akpm@linux-foundation.org):
> On Thu, 17 Feb 2011 15:03:33 +0000
> "Serge E. Hallyn" <serge@hallyn.com> wrote:
>
> > ptrace is allowed to tasks in the same user namespace according to
> > the usual rules (i.e. the same rules as for two tasks in the init
> > user namespace). ptrace is also allowed to a user namespace to
> > which the current task the has CAP_SYS_PTRACE capability.
> >
> >
> > ...
> >
> > --- a/include/linux/capability.h
> > +++ b/include/linux/capability.h
> > @@ -546,6 +546,8 @@ extern const kernel_cap_t __cap_init_eff_set;
> >   */

> > #define has_capability(t, cap) (security_real_capable((t), &init_user_ns, (cap)) == 0)
> >
> > +#define has_ns_capability(t, ns, cap) (security_real_capable((t), (ns), (cap)) == 0)
>
> macroitis.

Thanks for the review, Andrew. Unfortunately this one is hard to turn
into a function beecause it uses security_real_capable(), which is
sometimes defined in security/security.c as a real function, and
other times as a static inline in include/linux/security.h. So
I'd have to #include security.h in capability.h, but security.h
already #includes capability.h.

All the other comments affect same_or_ancestor_user_ns(), which
following Eric's feedback is going away.

> > /**
> >   * has_capability_noaudit - Determine if a task has a superior capability available (unaudited)
> >   * @t: The task in question
> > diff --git a/include/linux/user_namespace.h b/include/linux/user_namespace.h
> > index faf4679..862fc59 100644
> > --- a/include/linux/user_namespace.h
> > +++ b/include/linux/user_namespace.h
> > @@ -39,6 +39,9 @@ static inline void put_user_ns(struct user_namespace *ns)
> > uid_t user_ns_map_uid(struct user_namespace *to, const struct cred *cred, uid_t uid);
> > gid_t user_ns_map_gid(struct user_namespace *to, const struct cred *cred, gid_t gid);
> >
> > +int same_or_ancestor_user_ns(struct task_struct *task,
> > + struct task_struct *victim);
>
> bool.
>
> > #else
> >
> > static inline struct user_namespace *get_user_ns(struct user_namespace *ns)
> >
> > ...
> >
> > --- a/kernel/user_namespace.c
> > +++ b/kernel/user_namespace.c
> > @@ -129,6 +129,22 @@ gid_t user_ns_map_gid(struct user_namespace *to, const struct
cred *cred, gid_t
> >   return overflowgid;
> > }
> >
> > +int same_or_ancestor_user_ns(struct task_struct *task,
> > + struct task_struct *victim)
> > +{

> > + struct user_namespace *u1 = task_cred_xxx(task, user)->user_ns;
> > + struct user_namespace *u2 = task_cred_xxx(victim, user)->user_ns;
> > + for (;;) {
> > + if (u1 == u2)
> > +   return 1;
> > + if (u1 == &init_user_ns)
> > +   return 0;
> > + u1 = u1->creator->user_ns;
> > + }
> > + /* We never get here */
> > + return 0;
>
> Remove?
>
> > +}
> > +
> > static __init int user_namespaces_init(void)
> > {
> >   user_ns_cachep = KMEM_CACHE(user_namespace, SLAB_PANIC);
> >
> > ...
> >
> > int cap_ptrace_access_check(struct task_struct *child, unsigned int mode)
> > {
> >   int ret = 0;
> > + const struct cred *cred, *tcred;
> >
> >   rcu_read_lock();
> > - if (!cap_issubset(__task_cred(child)->cap_permitted,
> > -   current_cred()->cap_permitted) &&
> > -   !capable(CAP_SYS_PTRACE))
> > - ret = -EPERM;
> > + cred = current_cred();
> > + tcred = __task_cred(child);
> > + /*
> > + * The ancestor user_ns check may be gratuitous, as I think
> > + * we've already guaranteed that in kernel/ptrace.c.
> > + */
>
> ?
>
> > + if (same_or_ancestor_user_ns(current, child) &&
> > +   cap_issubset(tcred->cap_permitted, cred->cap_permitted))
> > + goto out;
> > + if (ns_capable(tcred->user->user_ns, CAP_SYS_PTRACE))
> > + goto out;
> > + ret = -EPERM;
> > +out:

> > rcu_read_unlock();
> > return ret;
> > }
> >
> > ...
> >
>

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

---

Subject: Re: [PATCH 4/9] allow killing tasks in your own or child userns
Posted by serge on Thu, 24 Feb 2011 00:48:18 GMT
View Forum Message <> Reply to Message

Quoting Andrew Morton (akpm@linux-foundation.org):
> On Thu, 17 Feb 2011 15:03:25 +0000
> "Serge E. Hallyn" <serge@hallyn.com> wrote:
>
> > /*
> > + * called with RCU read lock from check_kill_permission()
> > + */
> > +static inline int kill_ok_by_cred(struct task_struct *t)
> > +{
> > + const struct cred *cred = current_cred();
> > + const struct cred *tcred = __task_cred(t);
> > +
> > + if (cred->user->user_ns == tcred->user->user_ns &&
> > + (cred->euid == tcred->suid ||
> > + cred->euid == tcred->uid ||
> > + cred->uid == tcred->suid ||
> > + cred->uid == tcred->uid))
> > + return 1;
> > +
> > + if (ns_capable(tcred->user->user_ns, CAP_KILL))
> > + return 1;
> > +
> > + return 0;
> > +}
>
> The compiler will inline this for us.

Is that simply true with everything (worth inlining) nowadays, or is
there a particular implicit hint to the compiler that'll make that
happen?

Not that I guess it's even particularly important in this case.

From: Serge E. Hallyn <serge.hallyn@canonical.com>
Date: Thu, 24 Feb 2011 00:26:02 +0000
Subject: [PATCH 1/2] userns: let compiler inline kill_ok_by_cred (per akpm)

Signed-off-by: Serge E. Hallyn <serge.hallyn@canonical.com>
---
 kernel/signal.c | 2 +-
 1 files changed, 1 insertions(+), 1 deletions(-)

diff --git a/kernel/signal.c b/kernel/signal.c
index ffe4bdf..12702b4 100644
--- a/kernel/signal.c
+++ b/kernel/signal.c
@@ -638,7 +638,7 @@ static inline bool si_fromuser(const struct siginfo *info)
 /*
  * called with RCU read lock from check_kill_permission()
  */
-static inline int kill_ok_by_cred(struct task_struct *t)
+static int kill_ok_by_cred(struct task_struct *t)
 {
  const struct cred *cred = current_cred();
  const struct cred *tcred = __task_cred(t);
--
1.7.0.4

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

---

Subject: [PATCH] userns: ptrace: incorporate feedback from Eric
Posted by serge on Thu, 24 Feb 2011 00:49:01 GMT

View Forum Message <> Reply to Message

same_or_ancestore_user_ns() was not an appropriate check to
constrain cap_issubset. Rather, cap_issubset() only is
meaningful when both capsets are in the same user_ns.

Signed-off-by: Serge E. Hallyn <serge.hallyn@canonical.com>
---
 include/linux/user_namespace.h | 9 ---------
 kernel/user_namespace.c | 16 ----------------
 security/commoncap.c | 28 ++++++++++++++++------------
 3 files changed, 10 insertions(+), 43 deletions(-)

```
diff --git a/include/linux/user_namespace.h b/include/linux/user_namespace.h
index 862fc59..faf4679 100644
--- a/include/linux/user_namespace.h
+++ b/include/linux/user_namespace.h
@@ -39,9 +39,6 @@ static inline void put_user_ns(struct user_namespace *ns)
 uid_t user_ns_map_uid(struct user_namespace *to, const struct cred *cred, uid_t uid);
 gid_t user_ns_map_gid(struct user_namespace *to, const struct cred *cred, gid_t gid);

-int same_or_ancestor_user_ns(struct task_struct *task,
- struct task_struct *victim);
-
 #else

 static inline struct user_namespace *get_user_ns(struct user_namespace *ns)
@@ -69,12 +66,6 @@ static inline gid_t user_ns_map_gid(struct user_namespace *to,
  return gid;
 }

-static inline int same_or_ancestor_user_ns(struct task_struct *task,
- struct task_struct *victim)
-{
- return 1;
-}
-
 #endif

 #endif /* _LINUX_USER_H */
diff --git a/kernel/user_namespace.c b/kernel/user_namespace.c
index 0ef2258..9da289c 100644
--- a/kernel/user_namespace.c
+++ b/kernel/user_namespace.c
@@ -129,22 +129,6 @@ gid_t user_ns_map_gid(struct user_namespace *to, const struct cred
*cred, gid_t
  return overflowgid;
 }

-int same_or_ancestor_user_ns(struct task_struct *task,
- struct task_struct *victim)
-{
- struct user_namespace *u1 = task_cred_xxx(task, user)->user_ns;
- struct user_namespace *u2 = task_cred_xxx(victim, user)->user_ns;
- for (;;) {
-  if (u1 == u2)
-   return 1;
-  if (u1 == &init_user_ns)
-   return 0;
-  u1 = u1->creator->user_ns;
- }
```

```
- /* We never get here */
- return 0;
-}
-
 static __init int user_namespaces_init(void)
 {
  user_ns_cachep = KMEM_CACHE(user_namespace, SLAB_PANIC);
diff --git a/security/commoncap.c b/security/commoncap.c
index 12ff65c..526106f 100644
--- a/security/commoncap.c
+++ b/security/commoncap.c
@@ -142,19 +142,15 @@ int cap_settime(struct timespec *ts, struct timezone *tz)
 int cap_ptrace_access_check(struct task_struct *child, unsigned int mode)
 {
  int ret = 0;
- const struct cred *cred, *tcred;
+ const struct cred *cred, *child_cred;

  rcu_read_lock();
  cred = current_cred();
- tcred = __task_cred(child);
- /*
-  * The ancestor user_ns check may be gratuitous, as I think
-  * we've already guaranteed that in kernel/ptrace.c.
-  */
- if (same_or_ancestor_user_ns(current, child) &&
-     cap_issubset(tcred->cap_permitted, cred->cap_permitted))
+ child_cred = __task_cred(child);
+ if (cred->user->user_ns == child_cred->user->user_ns &&
+     cap_issubset(child_cred->cap_permitted, cred->cap_permitted))
   goto out;
- if (ns_capable(tcred->user->user_ns, CAP_SYS_PTRACE))
+ if (ns_capable(child_cred->user->user_ns, CAP_SYS_PTRACE))
   goto out;
  ret = -EPERM;
 out:
@@ -178,19 +174,15 @@ out:
 int cap_ptrace_traceme(struct task_struct *parent)
 {
  int ret = 0;
- const struct cred *cred, *tcred;
+ const struct cred *cred, *child_cred;

  rcu_read_lock();
  cred = __task_cred(parent);
- tcred = current_cred();
- /*
-  * The ancestor user_ns check may be gratuitous, as I think
```

```
-  * we've already guaranteed that in kernel/ptrace.c.
-  */
- if (same_or_ancestor_user_ns(parent, current) &&
-     cap_issubset(tcred->cap_permitted, cred->cap_permitted))
+ child_cred = current_cred();
+ if (cred->user->user_ns == child_cred->user->user_ns &&
+     cap_issubset(child_cred->cap_permitted, cred->cap_permitted))
    goto out;
- if (has_ns_capability(parent, tcred->user->user_ns, CAP_SYS_PTRACE))
+ if (has_ns_capability(parent, child_cred->user->user_ns, CAP_SYS_PTRACE))
    goto out;
  ret = -EPERM;
 out:
--
1.7.0.4
```

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

---

Subject: Re: [PATCH 4/9] allow killing tasks in your own or child userns
Posted by akpm on Thu, 24 Feb 2011 00:54:30 GMT
View Forum Message <> Reply to Message

On Thu, 24 Feb 2011 00:48:18 +0000
"Serge E. Hallyn" <serge@hallyn.com> wrote:

> Quoting Andrew Morton (akpm@linux-foundation.org):
> > On Thu, 17 Feb 2011 15:03:25 +0000
> > "Serge E. Hallyn" <serge@hallyn.com> wrote:
> >
> > > /*
> > > + * called with RCU read lock from check_kill_permission()
> > > + */
> > > +static inline int kill_ok_by_cred(struct task_struct *t)
> > > +{
> > > + const struct cred *cred = current_cred();
> > > + const struct cred *tcred = __task_cred(t);
> > > +
> > > + if (cred->user->user_ns == tcred->user->user_ns &&
> > > +   (cred->euid == tcred->suid ||
> > > +   cred->euid == tcred->uid ||
> > > +   cred->uid == tcred->suid ||
> > > +   cred->uid == tcred->uid))
> > > + return 1;
> > > +

> > > + if (ns_capable(tcred->user->user_ns, CAP_KILL))
> > > + return 1;
> > > +
> > > + return 0;
> > > +}
> >
> > The compiler will inline this for us.
>
> Is that simply true with everything (worth inlining) nowadays, or is
> there a particular implicit hint to the compiler that'll make that
> happen?

We've basically stopped inlining things nowadays. gcc inlines
aggressively and sometimes we have to use noinline to stop it. Also,
modern gcc's like to ignore the inline directive anwyay, so we have to
resort to __always_inline when we disagree.

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

---

Subject: Re: [PATCH] userns: ptrace: incorporate feedback from Eric
Posted by akpm on Thu, 24 Feb 2011 00:56:51 GMT
View Forum Message <> Reply to Message

On Thu, 24 Feb 2011 00:49:01 +0000
"Serge E. Hallyn" <serge@hallyn.com> wrote:

> same_or_ancestore_user_ns() was not an appropriate check to
> constrain cap_issubset. Rather, cap_issubset() only is
> meaningful when both capsets are in the same user_ns.

I queued this as a fix against
userns-allow-ptrace-from-non-init-user-namespaces.patch, but I get the
feeling that it would be better to just drop everything and start
again?

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

---

Subject: Re: [PATCH] userns: ptrace: incorporate feedback from Eric

Posted by serge on Thu, 24 Feb 2011 03:15:18 GMT
View Forum Message <> Reply to Message

Quoting Andrew Morton (akpm@linux-foundation.org):
> On Thu, 24 Feb 2011 00:49:01 +0000
> "Serge E. Hallyn" <serge@hallyn.com> wrote:
>
> > same_or_ancestore_user_ns() was not an appropriate check to
> > constrain cap_issubset. Rather, cap_issubset() only is
> > meaningful when both capsets are in the same user_ns.
>
> I queued this as a fix against
> userns-allow-ptrace-from-non-init-user-namespaces.patch, but I get the
> feeling that it would be better to just drop everything and start
> again?

Sure, I'll rebase and resend. I wonder if I should trim the Cc list
for the next round.

thanks,
-serge

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

Subject: Re: [PATCH 9/9] userns: check user namespace for task->file uid equivalence checks
Posted by serge on Thu, 24 Feb 2011 03:24:16 GMT
View Forum Message <> Reply to Message

Quoting Andrew Morton (akpm@linux-foundation.org):
> On Thu, 17 Feb 2011 15:04:07 +0000
> "Serge E. Hallyn" <serge@hallyn.com> wrote:
>
> > Cheat for now and say all files belong to init_user_ns. Next
> > step will be to let superblocks belong to a user_ns, and derive
> > inode_userns(inode) from inode->i_sb->s_user_ns. Finally we'll
> > introduce more flexible arrangements.
> >
> >
> > ...
> >
> > +
> > +/*
> > + * return 1 if current either has CAP_FOWNER to the
> > + * file, or owns the file.

> > + */
> > +int is_owner_or_cap(const struct inode *inode)
> > +{
> > + struct user_namespace *ns = inode_userns(inode);
> > +
> > + if (current_user_ns() == ns && current_fsuid() == inode->i_uid)
> > + return 1;
> > + if (ns_capable(ns, CAP_FOWNER))
> > + return 1;
> > + return 0;
> > +}
>
> bool?
>
> > +EXPORT_SYMBOL(is_owner_or_cap);
>
> There's a fairly well adhered to convention that global symbols (and
> often static symbols) have a prefix which identifies the subsystem to
> which they belong. This patchset rather scorns that convention.
>
> Most of these identifiers are pretty obviously from the capability
> subsystem, but still...

Would 'inode_owner_or_capable' be better and and make sense?

-serge

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

---

Subject: Re: [PATCH 9/9] userns: check user namespace for task->file uid equivalence checks
Posted by akpm on Thu, 24 Feb 2011 05:08:08 GMT
View Forum Message <> Reply to Message

On Thu, 24 Feb 2011 03:24:16 +0000 "Serge E. Hallyn" <serge@hallyn.com> wrote:

> Quoting Andrew Morton (akpm@linux-foundation.org):
> > On Thu, 17 Feb 2011 15:04:07 +0000
> > "Serge E. Hallyn" <serge@hallyn.com> wrote:
> >
> > There's a fairly well adhered to convention that global symbols (and
> > often static symbols) have a prefix which identifies the subsystem to
> > which they belong. This patchset rather scorns that convention.
> >
> > Most of these identifiers are pretty obviously from the capability

> > subsystem, but still...
>
> Would 'inode_owner_or_capable' be better and and make sense?
>

I suppose so. We've totally screwed that pooch in the VFS (grep EXPORT fs/inode.c). But it's never to late to start.

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

---

Subject: Re: User namespaces and keys
Posted by ebiederm on Thu, 24 Feb 2011 06:56:01 GMT
View Forum Message <> Reply to Message

Casey Schaufler <casey@schaufler-ca.com> writes:

> On 2/23/2011 12:55 PM, Eric W. Biederman wrote:
>> Casey Schaufler <casey@schaufler-ca.com> writes:
>>
>>> I confess that I remain less well educated on namespaces than
>>> I probably should be, but with what I do know it seems that the
>>> relationships between user namespaces and LSMs are bound to be
>>> strained from the beginning. Some LSMs (SELinux and Smack) are
>>> providing similar sandbox capabilities to what you get from user
>>> namespaces, but from different directions and with different
>>> use cases.
>> Casey I won't argue about the possibility of things being strained, but
>> I think if we focus on the semantics and not on the end goal of exactly
>> how the pieces are to be used there can be some reasonable dialog.
>
> I'm sure that there will be cases where they will work together
> like horses in a troika. Making sensible semantics for the interactions
> is key, and it is entirely possible that in some cases a comparison
> of semantics and behaviors will lead an end user to chose either an
> LSM or namespaces over the combination. Just like I expect that even
> when we allow multiple LSMs the SELinux and Smack combination will be
> rare among the sane.

That sounds about right.

Eric

_____
Containers mailing list