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