Subject: Re: [PATCH 2/2] hijack: update task_alloc_security
Posted by Stephen Smalley on Tue, 27 Nov 2007 14:36:28 GMT
View Forum Message <> Reply to Message

On Tue, 2007-11-27 at 00:52 -0500, Joshua Brindle wrote:
> Mark Nelson wrote:
> > Subject: [PATCH 2/2] hijack: update task_alloc_security
> >
> > Update task_alloc_security() to take the hijacked task as a second
> > argument.
> >
> > For the selinux version, refuse permission if hijack_src!=current,
> > since we have no idea what the proper behavior is. Even if we
> > assume that the resulting child should be in the hijacked task's
> > domain, depending on the policy that may not be enough information
> > since init_t executing /bin/bash could result in a different domain
> > than login_t executing /bin/bash.
> >
> >
> This means its basically not possible to hijack tasks with SELinux
> right? It would be a shame if this weren't useful to people running SELinux.

I agree with this part - we don't want people to have to choose between
using containers and using selinux, so if hijack is going to be a
requirement for effective use of containers, then we need to make them
work together.

> It seems to me (I may be wrong, I'm sure someone will let me know if I
> am) that the right way to handle this with SELinux is to check to see if
> the current task (caller of sys_hijack) has permission to ptrace (or

I think this may already happen in the first patch, by virtue of calling
the existing ptrace checks including the security hook. Right?

> some other permission deemed suitable, perhaps a new one) and if so copy
> the security blob pointer from the hijacked task to the new one (we
> don't want tranquility problems).

Just to clarify, we wouldn't be copying the pointer; here we are
allocating and populating a new task's security structure. We can
either continue to inherit the SIDs from current in all cases, or we
could set tsec1 = hijack_src->security; in selinux_task_alloc_security()
if we wanted to inherit from the hijacked task instead. The latter
would be similar to what you do in copy_hijackable_taskinfo() for uids
and capabilities IIUC. However, which behavior is right needs more
discussion I think, as the new task is a mixture of the caller's state
and the hijacked task's state. Which largely seems a recipe for
disaster.

```
>  From your paragraph above it seems like you were thinking there should
> be a transition at hijack time but we don't automatically transition
> anywhere except exec.
>
> Anyway, I just don't think you should completely disable this for
> SELinux users.
>
> > Signed-off-by: Serge Hallyn <serue@us.ibm.com>
> > Signed-off-by: Mark Nelson <markn@au1.ibm.com>
> > ---
> >  include/linux/security.h |   12 +++++++++---
> >  kernel/fork.c            |    2 +-
> >  security/dummy.c         |    3 ++-
> >  security/security.c      |    4 ++--
> >  security/selinux/hooks.c |    6 +++++-
> >  5 files changed, 19 insertions(+), 8 deletions(-)
> >
> > Index: upstream/include/linux/security.h
> > ===================================================================
> > --- upstream.orig/include/linux/security.h
> > +++ upstream/include/linux/security.h
> > @@ -545,9 +545,13 @@ struct request_sock;
> >   * Return 0 if permission is granted.
> >   * @task_alloc_security:
> >   * @p contains the task_struct for child process.
> > + * @task contains the task_struct for process to be hijacked
> >   * Allocate and attach a security structure to the p->security field. The
> >   * security field is initialized to NULL when the task structure is
> >   * allocated.
> > + * @task will usually be current.  If it is not equal to current, then
> > + * a sys_hijack system call is going on, and current is asking for a
> > + * child to be created in the context of the hijack src, @task.
> >   * Return 0 if operation was successful.
> >   * @task_free_security:
> >   * @p contains the task_struct for process.
> > @@ -1301,7 +1305,8 @@ struct security_operations {
> >   int (*dentry_open)  (struct file *file);
> >
> >   int (*task_create) (unsigned long clone_flags);
> > - int (*task_alloc_security) (struct task_struct * p);
> > + int (*task_alloc_security) (struct task_struct *p,
> > +        struct task_struct *task);
> >   void (*task_free_security) (struct task_struct * p);
> >   int (*task_setuid) (uid_t id0, uid_t id1, uid_t id2, int flags);
> >   int (*task_post_setuid) (uid_t old_ruid /* or fsuid */ ,
> > @@ -1549,7 +1554,7 @@ int security_file_send_sigiotask(struct
> >  int security_file_receive(struct file *file);
> >  int security_dentry_open(struct file *file);
> >  int security_task_create(unsigned long clone_flags);
> > -int security_task_alloc(struct task_struct *p);
> > +int security_task_alloc(struct task_struct *p, struct task_struct *task);
> >  void security_task_free(struct task_struct *p);
> >  int security_task_setuid(uid_t id0, uid_t id1, uid_t id2, int flags);
> >  int security_task_post_setuid(uid_t old_ruid, uid_t old_euid,
> > @@ -2021,7 +2026,8 @@ static inline int security_task_create (
> >   return 0;
> >  }
> >
> > -static inline int security_task_alloc (struct task_struct *p)
> > +static inline int security_task_alloc(struct task_struct *p,
> > +           struct task_struct *task)
> >  {
> >   return 0;
> >  }
> > Index: upstream/kernel/fork.c
> > ===================================================================
> > --- upstream.orig/kernel/fork.c
> > +++ upstream/kernel/fork.c
> > @@ -1177,7 +1177,7 @@ static struct task_struct *copy_process(
> >   /* Perform scheduler related setup. Assign this task to a CPU. */
> >   sched_fork(p, clone_flags);
> >
> > - if ((retval = security_task_alloc(p)))
> > + if ((retval = security_task_alloc(p, task)))
> >    goto bad_fork_cleanup_policy;
> >   if ((retval = audit_alloc(p)))
> >    goto bad_fork_cleanup_security;
> > Index: upstream/security/dummy.c
> > ===================================================================
> > --- upstream.orig/security/dummy.c
> > +++ upstream/security/dummy.c
> > @@ -475,7 +475,8 @@ static int dummy_task_create (unsigned l
> >   return 0;
> >  }
> >
> > -static int dummy_task_alloc_security (struct task_struct *p)
> > +static int dummy_task_alloc_security(struct task_struct *p,
> > +          struct task_struct *task)
> >  {
> >   return 0;
> >  }
> > Index: upstream/security/security.c
> > ===================================================================
> > --- upstream.orig/security/security.c
> > +++ upstream/security/security.c
> > @@ -568,9 +568,9 @@ int security_task_create(unsigned long c
> >   return security_ops->task_create(clone_flags);
> >  }
> >
> > -int security_task_alloc(struct task_struct *p)
> > +int security_task_alloc(struct task_struct *p, struct task_struct *task)
> >  {
> > - return security_ops->task_alloc_security(p);
> > + return security_ops->task_alloc_security(p, task);
> >  }
> >
> >  void security_task_free(struct task_struct *p)
> > Index: upstream/security/selinux/hooks.c
> > ===================================================================
> > --- upstream.orig/security/selinux/hooks.c
> > +++ upstream/security/selinux/hooks.c
> > @@ -2788,11 +2788,15 @@ static int selinux_task_create(unsigned
> >   return task_has_perm(current, current, PROCESS__FORK);
> >  }
> >
> > -static int selinux_task_alloc_security(struct task_struct *tsk)
> > +static int selinux_task_alloc_security(struct task_struct *tsk,
> > +          struct task_struct *hijack_src)
> >  {
> >   struct task_security_struct *tsec1, *tsec2;
> >   int rc;
> >
> > + if (hijack_src != current)
> > +  return -EPERM;
> > +
> >   tsec1 = current->security;
> >
> >   rc = task_alloc_security(tsk);
> > -
```
> > To unsubscribe from this list: send the line "unsubscribe linux-security-module" in
> > the body of a message to majordomo@vger.kernel.org
> > More majordomo info at http://vger.kernel.org/majordomo-info.html
> >
> >
>
>
> -
> To unsubscribe from this list: send the line "unsubscribe linux-security-module" in
> the body of a message to majordomo@vger.kernel.org
> More majordomo info at http://vger.kernel.org/majordomo-info.html
--
Stephen Smalley
National Security Agency