
Subject: Re: [PATCH 2/2] hijack: update task_alloc_security

Posted by [serue](#) on Tue, 27 Nov 2007 15:50:13 GMT

[View Forum Message](#) <> [Reply to Message](#)

Quoting Rodrigo Rubira Branco (BSDaemon) (rodrigo@kernelhacking.com):

> It will give another easy way to locate selinux security structures inside
> the kernel, will not?

By locate, you mean actually finding the structures in kernel memory to
overwrite them?

> Again, if you have a kernel vulnerability and this feature, someone will
> easily disable selinux for the process, or just change the security concerns
> for it ;).

Maybe hijack should go under a kernel config for security reasons.

thanks,
-serge

> cya,

>

>

> Rodrigo (BSDaemon).

>

> --

> <http://www.kernelhacking.com/rodrigo>

>

> Kernel Hacking: If i really know, i can hack

>

> GPG KeyID: 1FCEDEA1

>

>

> ----- Mensagem Original -----

> De: Joshua Brindle <method@manicmethod.com>

> Para: Mark Nelson <markn@au1.ibm.com>

> C?pia: containers@lists.linux-foundation.org,

> linux-security-module@vger.kernel.org, selinux@tycho.nsa.gov,

> menage@google.com, Stephen Smalley <sds@tycho.nsa.gov>, James Morris

> <jmorris@namei.org>, Serge E. Hallyn <serue@us.ibm.com>

> Assunto: Re: [PATCH 2/2] hijack: update task_alloc_security

> Data: 27/11/07 02:38

>

> >

> > Mark Nelson wrote:

> > > Subject: [PATCH 2/2] hijack: update task_alloc_security

> > >

> > > Update task_alloc_security() to take the hijacked task as a second

```

> > &gt; argument.
> > &gt;
> > &gt; For the selinux version, refuse permission if hijack_src!=current,
> > &gt; since we have no idea what the proper behavior is. Even if we
> > &gt; assume that the resulting child should be in the hijacked task's
> > &gt; domain, depending on the policy that may not be enough information
> > &gt; since init_t executing /bin/bash could result in a different domain
> > &gt; than login_t executing /bin/bash.
> > &gt;
> > &gt;
> > 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.
> >
> > 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
> > 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).
> >
> > 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.
> >
> > &gt; Signed-off-by: Serge Hallyn <serue@us.ibm.com>
> > &gt; Signed-off-by: Mark Nelson <markn@au1.ibm.com>
> > &gt; ---
> > &gt; include/linux/security.h | 12 ++++++---
> > &gt; kernel/fork.c           |  2 +-
> > &gt; security/dummy.c        |  3 ++-
> > &gt; security/security.c     |  4 ++--
> > &gt; security/selinux/hooks.c |  6 +++++-
> > &gt; 5 files changed, 19 insertions(+), 8 deletions(-)
> > &gt;
> > &gt; Index: upstream/include/linux/security.h
> > &gt; =====
> > &gt; --- upstream.orig/include/linux/security.h
> > &gt; +++ upstream/include/linux/security.h
> > &gt; @@ -545,9 +545,13 @@ struct request_sock;
> > &gt; * Return 0 if permission is granted.
> > &gt; * @task_alloc_security:
> > &gt; * @p contains the task_struct for child process.
> > &gt; + * @task contains the task_struct for process to be hijacked
> > &gt; * Allocate and attach a security structure to the p->security

```

```

> field. The
> > &gt; * security field is initialized to NULL when the task structure is
> > &gt; * allocated.
> > &gt; + * @task will usually be current. If it is not equal to current,
> then
> > &gt; + * a sys_hijack system call is going on, and current is asking for a
> > &gt; + * child to be created in the context of the hijack src, @task.
> > &gt; * Return 0 if operation was successful.
> > &gt; * @task_free_security:
> > &gt; * @p contains the task_struct for process.
> > &gt; @@ -1301,7 +1305,8 @@ struct security_operations {
> > &gt; int (*dentry_open) (struct file *file);
> > &gt;
> > &gt; int (*task_create) (unsigned long clone_flags);
> > &gt; - int (*task_alloc_security) (struct task_struct * p);
> > &gt; + int (*task_alloc_security) (struct task_struct *p,
> > &gt; + struct task_struct *task);
> > &gt; void (*task_free_security) (struct task_struct * p);
> > &gt; int (*task_setuid) (uid_t id0, uid_t id1, uid_t id2, int flags);
> > &gt; int (*task_post_setuid) (uid_t old_ruid /* or fsuid */ ,
> > &gt; @@ -1549,7 +1554,7 @@ int security_file_send_sigiotask(struct
> > &gt; int security_file_receive(struct file *file);
> > &gt; int security_dentry_open(struct file *file);
> > &gt; int security_task_create(unsigned long clone_flags);
> > &gt; -int security_task_alloc(struct task_struct *p);
> > &gt; +int security_task_alloc(struct task_struct *p, struct task_struct
> *task);
> > &gt; void security_task_free(struct task_struct *p);
> > &gt; int security_task_setuid(uid_t id0, uid_t id1, uid_t id2, int
> flags);
> > &gt; int security_task_post_setuid(uid_t old_ruid, uid_t old_euid,
> > &gt; @@ -2021,7 +2026,8 @@ static inline int security_task_create (
> > &gt; return 0;
> > &gt; }
> > &gt;
> > &gt; -static inline int security_task_alloc (struct task_struct *p)
> > &gt; +static inline int security_task_alloc(struct task_struct *p,
> > &gt; + struct task_struct *task)
> > &gt; {
> > &gt; return 0;
> > &gt; }
> > &gt; Index: upstream/kernel/fork.c
> > &gt; =====
> > &gt; --- upstream.orig/kernel/fork.c
> > &gt; +++ upstream/kernel/fork.c
> > &gt; @@ -1177,7 +1177,7 @@ static struct task_struct *copy_process(
> > &gt; /* Perform scheduler related setup. Assign this task to a CPU. */
> > &gt; sched_fork(p, clone_flags);

```

```

> > &gt;
> > &gt; - if ((retval = security_task_alloc(p)))
> > &gt; + if ((retval = security_task_alloc(p, task)))
> > &gt; goto bad_fork_cleanup_policy;
> > &gt; if ((retval = audit_alloc(p)))
> > &gt; goto bad_fork_cleanup_security;
> > &gt; Index: upstream/security/dummy.c
> > &gt; =====
> > &gt; --- upstream.orig/security/dummy.c
> > &gt; +++ upstream/security/dummy.c
> > &gt; @@ -475,7 +475,8 @@ static int dummy_task_create (unsigned l
> > &gt; return 0;
> > &gt; }
> > &gt;
> > &gt; -static int dummy_task_alloc_security (struct task_struct *p)
> > &gt; +static int dummy_task_alloc_security(struct task_struct *p,
> > &gt; + struct task_struct *task)
> > &gt; {
> > &gt; return 0;
> > &gt; }
> > &gt; Index: upstream/security/security.c
> > &gt; =====
> > &gt; --- upstream.orig/security/security.c
> > &gt; +++ upstream/security/security.c
> > &gt; @@ -568,9 +568,9 @@ int security_task_create(unsigned long c
> > &gt; return security_ops-&gt;task_create(clone_flags);
> > &gt; }
> > &gt;
> > &gt; -int security_task_alloc(struct task_struct *p)
> > &gt; +int security_task_alloc(struct task_struct *p, struct task_struct
> > &gt; *task)
> > &gt; {
> > &gt; - return security_ops-&gt;task_alloc_security(p);
> > &gt; + return security_ops-&gt;task_alloc_security(p, task);
> > &gt; }
> > &gt;
> > &gt; void security_task_free(struct task_struct *p)
> > &gt; Index: upstream/security/selinux/hooks.c
> > &gt; =====
> > &gt; --- upstream.orig/security/selinux/hooks.c
> > &gt; +++ upstream/security/selinux/hooks.c
> > &gt; @@ -2788,11 +2788,15 @@ static int selinux_task_create(unsigned
> > &gt; return task_has_perm(current, current, PROCESS__FORK);
> > &gt; }
> > &gt;
> > &gt;
> > &gt; -static int selinux_task_alloc_security(struct task_struct *tsk)
> > &gt; +static int selinux_task_alloc_security(struct task_struct *tsk,
> > &gt; + struct task_struct *hijack_src)

```

