

[PATCH 7/7] SECURITY: De-embed task security record from task and use refcounting

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2007-12/msg01513.html>

- *From:* David Howells <dhowells@xxxxxxxxxx>
 - *Date:* Wed, 05 Dec 2007 19:37:38 +0000
-

Remove the temporarily embedded task security record from `task_struct`. Instead it is made to dangle from the `task_struct::sec` and `task_struct::act_as` pointers with references counted for each.

`do_coredump()` is made to create a copy of the security record, modify it and then use that to override the main one for a task. `sys_faccessat()` is made to do the same.

The process and session keyrings are moved from `signal_struct` into a new `thread_group_security` struct. This is then refcounted, with pointers coming from the `task_security` struct instead of from `signal_struct`.

The keyring functions then take pointers to `task_security` structs rather than `task_structs` for their security contexts. This is so that `request_key()` can proceed asynchronously without having to worry about the initiator task's `act_as` pointer changing.

The LSM hooks for dealing with task security are modified to deal with the task security struct directly rather than going via the `task_struct` as appropriate.

This permits the subjective security context of a task to be overridden by changing its `act_as` pointer without altering its objective security pointer, and thus not breaking signalling, `ptrace`, etc. whilst the override is in force.

Signed-off-by: David Howells <dhowells@xxxxxxxxxx>

```
fs/exec.c | 15 +-
fs/open.c | 37 ++---
include/linux/init_task.h | 17 --
include/linux/key-ui.h | 10 +
include/linux/key.h | 31 +---
include/linux/sched.h | 40 ++++-
include/linux/security.h | 43 ++++--
kernel/Makefile | 2
kernel/cred.c | 139 ++++++
kernel/exit.c | 1
kernel/fork.c | 40 ++---
```

[PATCH 7/7] SECURITY: De-embed task security record from task and use refcounting

```
kernel/kmod.c | 10 +
kernel/sys.c | 16 +-
kernel/user.c | 2
net/rxrpc/ar-key.c | 4 -
security/dummy.c | 14 +-
security/keys/internal.h | 10 +
security/keys/key.c | 6 -
security/keys/keyctl.c | 6 -
security/keys/keyring.c | 14 +-
security/keys/permission.c | 5 -
security/keys/proc.c | 2
security/keys/process_keys.c | 290 ++++++-----
security/keys/request_key.c | 59 +++++-
security/keys/request_key_auth.c | 38 ++-
security/security.c | 20 +-
security/selinux/hooks.c | 46 +++++-
27 files changed, 526 insertions(+), 391 deletions(-)
```

```
diff --git a/fs/exec.c b/fs/exec.c
index da01655..f10e3fd 100644
--- a/fs/exec.c
+++ b/fs/exec.c
@@ -1674,13 +1674,13 @@ int get_dumpable(struct mm_struct *mm)
```

```
int do_coredump(long signr, int exit_code, struct pt_regs * regs)
{
+ struct task_security *sec, *old_act_as;
char corename[CORENAME_MAX_SIZE + 1];
struct mm_struct *mm = current->mm;
struct linux_binfmt * binfmt;
struct inode * inode;
struct file * file;
int retval = 0;
- int fsuid = current_fsuid();
int flag = 0;
int ispipe = 0;
unsigned long core_limit = current->signal->rlim[RLIMIT_CORE].rlim_cur;
@@ -1692,7 +1692,10 @@ int do_coredump(long signr, int exit_code, struct pt_regs * regs)
```

```
binfmt = current->binfmt;
if (!binfmt || !binfmt->core_dump)
- goto fail;
+ goto fail_nosubj;
+ sec = dup_task_security(current->sec);
+ if (!sec)
+ goto fail_nosubj;
down_write(&mm->mmap_sem);
/*
* If another thread got here first, or we are not dumpable, bail out.
@@ -1707,9 +1710,11 @@ int do_coredump(long signr, int exit_code, struct pt_regs * regs)
* process nor do we know its entire history. We only know it
```

[PATCH 7/7] SECURITY: De-embed task security record from task and use refcounting

```
* was tainted so we dump it as root in mode 2.
*/
+ old_act_as = current->act_as;
if (get_dumpable(mm) == 2) { /* Setuid core dump mode */
flag = O_EXCL; /* Stop rewrite attacks */
- current->act_as->fsuid = 0; /* Dump root private */
+ sec->fsuid = 0; /* Dump root private */
+ current->act_as = sec;
}

retval = coredump_wait(exit_code);
@@ -1805,8 +1810,10 @@ fail_unlock:
if (helper_argv)
argv_free(helper_argv);

- current->act_as->fsuid = fsuid;
+ current->act_as = old_act_as;
complete_all(&mm->core_done);
fail:
+ put_task_security(sec);
+fail_nosubj:
return retval;
}
diff --git a/fs/open.c b/fs/open.c
index 6d7a29c..710102a 100644
--- a/fs/open.c
+++ b/fs/open.c
@@ -420,34 +420,27 @@ out:
*/
asmlinkage long sys_faccessat(int dfd, const char __user *filename, int mode)
{
+ struct task_security *sec, *old_act_as;
struct nameidata nd;
- int old_fsuid, old_fsgid;
- kernel_cap_t old_cap;
int res;

if (mode & ~S_IRWXO) /* where's F_OK, X_OK, W_OK, R_OK? */
return -EINVAL;

- old_fsuid = current->act_as->fsuid;
- old_fsgid = current->act_as->fsgid;
- old_cap = current->act_as->cap_effective;
+ sec = dup_task_security(current->sec);
+ if (!sec)
+ return -ENOMEM;
+ sec->fsuid = current->sec->uid;
+ sec->fsgid = current->sec->gid;

- current->act_as->fsuid = current->act_as->uid;
- current->act_as->fsgid = current->act_as->gid;
```

[PATCH 7/7] SECURITY: De-embed task security record from task and use refcounting

```
-
- /*
- * Clear the capabilities if we switch to a non-root user
- *
- * FIXME: There is a race here against sys_capset. The
- * capabilities can change yet we will restore the old
- * value below. We should hold task_capabilities_lock,
- * but we cannot because user_path_walk can sleep.
- */
- if (current->act_as->uid)
- cap_clear(current->act_as->cap_effective);
+ /* Clear the capabilities if we switch to a non-root user */
+ if (current->sec->uid)
+ cap_clear(sec->cap_effective);
else
- current->act_as->cap_effective = current->act_as->cap_permitted;
+ sec->cap_effective = current->sec->cap_permitted;

+ old_act_as = current->act_as;
+ current->act_as = sec;
res = __user_walk_fd(dfd, filename, LOOKUP_FOLLOW|LOOKUP_ACCESS, &nd);
if (res)
goto out;
@@ -464,10 +457,8 @@ asmlinkage long sys_faccessat(int dfd, const char __user *filename, int mode)
out_path_release:
path_release(&nd);
out:
- current->act_as->fsuid = old_fsuid;
- current->act_as->fsgid = old_fsgid;
- current->act_as->cap_effective = old_cap;
-
+ current->act_as = old_act_as;
+ put_task_security(sec);
return res;
}
```

```
diff --git a/include/linux/init_task.h b/include/linux/init_task.h
index 6fa8413..380cc6a 100644
--- a/include/linux/init_task.h
+++ b/include/linux/init_task.h
@@ -116,18 +116,6 @@ extern struct group_info init_groups;
```

```
extern struct task_security init_task_security;
```

```
‑#define INIT_TASK_SECURITY(p) \
‑{ \
‑ .usage = ATOMIC_INIT(3), \
‑ .keep_capabilities = 0, \
‑ .cap_inheritable = CAP_INIT_INH_SET, \
‑ .cap_permitted = CAP_FULL_SET, \
‑ .cap_effective = CAP_INIT_EFF_SET, \
```

[PATCH 7/7] SECURITY: De-embed task security record from task and use refcounting

```

- .user = INIT_USER, \
- .group_info = &init_groups, \
- .lock = __SPIN_LOCK_UNLOCKED(p.lock), \
-}
-
/*
* INIT_TASK is used to set up the first task table, touch at
* your own risk!. Base=0, limit=0x1ffff (=2MB)
@@ -157,9 +145,8 @@ extern struct task_security init_task_security;
.children = LIST_HEAD_INIT(tsk.children), \
.sibling = LIST_HEAD_INIT(tsk.sibling), \
.group_leader = &tsk, \
- __temp_sec = INIT_TASK_SECURITY(tsk.__temp_sec), \
- .sec = &tsk.__temp_sec, \
- .act_as = &tsk.__temp_sec, \
+ .sec = &init_task_security, \
+ .act_as = &init_task_security, \
.commm = "swapper", \
.thread = INIT_THREAD, \
.fs = &init_fs, \
diff --git a/include/linux/key-ui.h b/include/linux/key-ui.h
index e8b8a7a..f15ea9d 100644
--- a/include/linux/key-ui.h
+++ b/include/linux/key-ui.h
@@ -43,15 +43,13 @@ struct keyring_list {
* check to see whether permission is granted to use a key in the desired way
*/
extern int key_task_permission(const key_ref_t key_ref,
- struct task_struct *context,
+ struct task_security *sec,
key_perm_t perm);

-static inline int key_permission(const key_ref_t key_ref, key_perm_t perm)
-{
- return key_task_permission(key_ref, current, perm);
-}
+#define key_permission(key_ref, perm) \
+ key_task_permission((key_ref), current->act_as, (perm))

-extern key_ref_t lookup_user_key(struct task_struct *context,
+extern key_ref_t lookup_user_key(struct task_security *sec,
key_serial_t id, int create, int partial,
key_perm_t perm);

diff --git a/include/linux/key.h b/include/linux/key.h
index 4a6021a..f5edd89 100644
--- a/include/linux/key.h
+++ b/include/linux/key.h
@@ -70,6 +70,8 @@ struct key;
struct seq_file;
struct user_struct;

```

[PATCH 7/7] SECURITY: De-embed task security record from task and use refcounting

```
struct signal_struct;
+struct task_security;
+struct thread_group_security;

struct key_type;
struct key_owner;
@@ -178,7 +180,7 @@ struct key {
extern struct key *key_alloc(struct key_type *type,
const char *desc,
uid_t uid, gid_t gid,
- struct task_struct *ctx,
+ struct task_security *sec,
key_perm_t perm,
unsigned long flags);

@@ -245,7 +247,7 @@ extern int key_unlink(struct key *keyring,
struct key *key);

extern struct key *keyring_alloc(const char *description, uid_t uid, gid_t gid,
- struct task_struct *ctx,
+ struct task_security *sec,
unsigned long flags,
struct key *dest);

@@ -267,24 +269,16 @@ extern struct key *key_lookup(key_serial_t id);
*/
extern struct key root_user_keyring, root_session_keyring;
extern int alloc_uid_keyring(struct user_struct *user,
- struct task_struct *ctx);
+ struct task_security *sec);
extern void switch_uid_keyring(struct user_struct *new_user);
-extern int copy_keys(unsigned long clone_flags, struct task_struct *tsk);
-extern int copy_thread_group_keys(struct task_struct *tsk);
-extern void exit_keys(struct task_struct *tsk);
-extern void exit_thread_group_keys(struct signal_struct *tg);
+extern int copy_thread_group_keys(struct thread_group_security *tgsec);
extern int suid_keys(struct task_struct *tsk);
extern int exec_keys(struct task_struct *tsk);
-extern void key_fsuid_changed(struct task_struct *tsk);
-extern void key_fsgid_changed(struct task_struct *tsk);
+extern void key_fsuid_changed(struct task_security *sec);
+extern void key_fsgid_changed(struct task_security *sec);
extern void key_init(void);
-
-#define __install_session_keyring(tsk, keyring) \
-({ \
- struct key *old_session = tsk->signal->session_keyring; \
- tsk->signal->session_keyring = keyring; \
- old_session; \
-})
+extern void __install_session_keyring(struct task_struct *tsk,
```

[PATCH 7/7] SECURITY: De-embed task security record from task and use refcounting

```

+ struct key *keyring);

#else /* CONFIG_KEYS */

@@ -298,11 +292,8 @@ extern void key_init(void);
#define is_key_posessed(k) 0
#define alloc_uid_keyring(u,c) 0
#define switch_uid_keyring(u) do { } while(0)
-#define __install_session_keyring(t, k) ({ NULL; })
-#define copy_keys(f,t) 0
+#define __install_session_keyring(t, k) do { } while (0)
#define copy_thread_group_keys(t) 0
-#define exit_keys(t) do { } while(0)
-#define exit_thread_group_keys(tg) do { } while(0)
#define suid_keys(t) do { } while(0)
#define exec_keys(t) do { } while(0)
#define key_fsuid_changed(t) do { } while(0)
diff --git a/include/linux/sched.h b/include/linux/sched.h
index bcd785a..f6df115 100644
--- a/include/linux/sched.h
+++ b/include/linux/sched.h
@@ -491,12 +491,6 @@ struct signal_struct {

struct list_head cpu_timers[3];

- /* keep the process-shared keyrings here so that they do the right
- * thing in threads created with CLONE_THREAD */
-#ifndef CONFIG_KEYS
- struct key *session_keyring; /* keyring inherited over fork */
- struct key *process_keyring; /* keyring private to this process */
-#endif
#ifdef CONFIG_BSD_PROCESS_ACCT
struct pacct_struct pacct; /* per-process accounting information */
#endif
@@ -572,6 +566,20 @@ extern struct user_struct root_user;

/*
+ * The common security details for a thread group
+ * - shared by CLONE_THREAD
+ */
+#ifdef CONFIG_KEYS
+struct thread_group_security {
+ atomic_t usage;
+ pid_t tgid; /* thread group process ID */
+ spinlock_t lock;
+ struct key *session_keyring; /* keyring inherited over fork */
+ struct key *process_keyring; /* keyring private to this process */
+};
+#endif
+

```

[PATCH 7/7] SECURITY: De-embed task security record from task and use refcounting

```
+/*
* The security context of a task
*
* The parts of the context break down into two categories:
@@ -613,6 +621,7 @@ struct task_security {
* keys to */
struct key *thread_keyring; /* keyring private to this thread */
struct key *request_key_auth; /* assumed request_key authority */
+ struct thread_group_security *tgsec;
#endif
#ifdef CONFIG_SECURITY
void *security; /* subjective LSM security */
@@ -622,10 +631,28 @@ struct task_security {
spinlock_t lock; /* lock for pointer changes */
};

+extern struct task_security *dup_task_security(struct task_security *);
+extern int copy_task_security(struct task_struct *, unsigned long);
+extern void put_task_security(struct task_security *);
+
#define current_fsuid() (current->act_as->fsuid)
#define current_fsgid() (current->act_as->fsgid)
#define current_cap() (current->act_as->cap_effective)

+/**
+ * get_task_security - Get an extra reference on a task security record
+ * @sec: The security record to get the reference on
+ *
+ * Get an extra reference on a task security record. The caller must arrange
+ * for this to be released.
+ */
+static inline
+struct task_security *get_task_security(struct task_security *sec)
+{
+ atomic_inc(&sec->usage);
+ return sec;
+}
+

struct backing_dev_info;
struct reclaim_state;
@@ -1082,7 +1109,6 @@ struct task_struct {
struct list_head cpu_timers[3];

/* process credentials */
- struct task_security __temp_sec __deprecated; /* temporary security to be removed */
struct task_security *sec; /* actual/objective task security */
struct task_security *act_as; /* effective/subjective task security */

diff --git a/include/linux/security.h b/include/linux/security.h
index 8d9e946..b7ba073 100644
```

[PATCH 7/7] SECURITY: De-embed task security record from task and use refcounting

```
--- a/include/linux/security.h
+++ b/include/linux/security.h
@@ -550,8 +550,13 @@ struct request_sock;
* allocated.
* Return 0 if operation was successful.
* @task_free_security:
- * @p contains the task_struct for process.
+ * @p points to the task_security struct to be freed.
* Deallocate and clear the p->security field.
+ * @task_dup_security:
+ * @p points to the task_security struct to be copied
+ * Duplicate and attach the security structure currently attached to the
+ * p->security field.
+ * Return 0 if operation was successful.
* @task_setuid:
* Check permission before setting one or more of the user identity
* attributes of the current process. The @flags parameter indicates
@@ -944,6 +949,7 @@ struct request_sock;
* Permit allocation of a key and assign security data. Note that key does
* not have a serial number assigned at this point.
* @key points to the key.
+ * @sec points to the task security record to use.
* @flags is the allocation flags
* Return 0 if permission is granted, -ve error otherwise.
* @key_free:
@@ -954,8 +960,8 @@ struct request_sock;
* See whether a specific operational right is granted to a process on a
* key.
* @key_ref refers to the key (key pointer + possession attribute bit).
- * @context points to the process to provide the context against which to
- * evaluate the security data on the key.
+ * @sec points to the process's security record to provide the context
+ * against which to evaluate the security data on the key.
* @perm describes the combination of permissions required of this key.
* Return 1 if permission granted, 0 if permission denied and -ve if the
* normal permissions model should be effected.
@@ -1312,8 +1318,9 @@ struct security_operations {
int (*dentry_open) (struct file *file);

int (*task_create) (unsigned long clone_flags);
- int (*task_alloc_security) (struct task_struct * p);
- void (*task_free_security) (struct task_struct * p);
+ int (*task_alloc_security) (struct task_struct *p);
+ void (*task_free_security) (struct task_security *p);
+ int (*task_dup_security) (struct task_security *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 */,
uid_t old_euid, uid_t old_suid, int flags);
@@ -1443,10 +1450,11 @@ struct security_operations {

/* key management security hooks */
```

[PATCH 7/7] SECURITY: De-embed task security record from task and use refcounting

```
#ifdef CONFIG_KEYS
- int (*key_alloc)(struct key *key, struct task_struct *tsk, unsigned long flags);
+ int (*key_alloc)(struct key *key, struct task_security *context,
+ unsigned long flags);
void (*key_free)(struct key *key);
int (*key_permission)(key_ref_t key_ref,
- struct task_struct *context,
+ struct task_security *context,
key_perm_t perm);
int (*key_getsecurity)(struct key *key, char **_buffer);
#endif /* CONFIG_KEYS */
@@ -1561,7 +1569,8 @@ 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);
-void security_task_free(struct task_struct *p);
+void security_task_free(struct task_security *p);
+int security_task_dup(struct task_security *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,
uid_t old_suid, int flags);
@@ -2032,14 +2041,19 @@ static inline int security_task_create (unsigned long clone_flags)
return 0;
}

-static inline int security_task_alloc (struct task_struct *p)
+static inline int security_task_alloc(struct task_struct *p)
{
return 0;
}

-static inline void security_task_free (struct task_struct *p)
+static inline void security_task_free(struct task_security *p)
{ }

+static inline int security_task_dup(struct task_security *p)
+{
+ return 0;
+}
+
static inline int security_task_setuid (uid_t id0, uid_t id1, uid_t id2,
int flags)
{
@@ -2574,16 +2588,17 @@ static inline void security_skb_classify_flow(struct sk_buff *skb, struct flowi
#ifdef CONFIG_KEYS
#ifdef CONFIG_SECURITY
- int security_key_alloc(struct key *key, struct task_struct *tsk, unsigned long flags);
+ int security_key_alloc(struct key *key, struct task_security *sec,
+ unsigned long flags);
void security_key_free(struct key *key);
```

[PATCH 7/7] SECURITY: De-embed task security record from task and use refcounting

```
int security_key_permission(key_ref_t key_ref,
- struct task_struct *context, key_perm_t perm);
+ struct task_security *sec, key_perm_t perm);
int security_key_getsecurity(struct key *key, char **_buffer);

#else

static inline int security_key_alloc(struct key *key,
- struct task_struct *tsk,
+ struct task_security *sec,
unsigned long flags)
{
return 0;
@@ -2594,7 +2609,7 @@ static inline void security_key_free(struct key *key)
}

static inline int security_key_permission(key_ref_t key_ref,
- struct task_struct *context,
+ struct task_security *sec,
key_perm_t perm)
{
return 0;
diff --git a/kernel/Makefile b/kernel/Makefile
index dfa9695..eb2af05 100644
--- a/kernel/Makefile
+++ b/kernel/Makefile
@@ -9,7 +9,7 @@ obj-y = sched.o fork.o exec_domain.o panic.o printk.o profile.o \
rcupdate.o extable.o params.o posix-timers.o \
kthread.o wait.o kfifo.o sys_ni.o posix-cpu-timers.o mutex.o \
hrtimer.o rwsem.o latency.o nsproxy.o srcu.o \
- utsname.o notifier.o
+ utsname.o notifier.o cred.o

obj-$(CONFIG_SYSCTL) += sysctl_check.o
obj-$(CONFIG_STACKTRACE) += stacktrace.o
diff --git a/kernel/cred.c b/kernel/cred.c
new file mode 100644
index 0000000..ddf98c6
--- /dev/null
+++ b/kernel/cred.c
@@ -0,0 +1,139 @@
+/* Tasks security and credentials management
+ *
+ * Copyright (C) 2007 Red Hat, Inc. All Rights Reserved.
+ * Written by David Howells (dhowells@xxxxxxxxxxx)
+ *
+ * This program is free software; you can redistribute it and/or
+ * modify it under the terms of the GNU General Public Licence
+ * as published by the Free Software Foundation; either version
+ * 2 of the Licence, or (at your option) any later version.
+ */
```

[PATCH 7/7] SECURITY: De-embed task security record from task and use refcounting

```
+#include <linux/module.h>
+#include <linux/sched.h>
+#include <linux/key.h>
+#include <linux/init_task.h>
+#include <linux/security.h>
+
+#ifdef CONFIG_KEYS
+static struct thread_group_security init_thread_group_security = {
+ .usage = ATOMIC_INIT(2),
+ .lock = __SPIN_LOCK_UNLOCKED(init_thread_group_security.lock),
+};
+#endif
+
+struct task_security init_task_security = {
+ .usage = ATOMIC_INIT(3),
+ .keep_capabilities = 0,
+ .cap_inheritable = CAP_INIT_INH_SET,
+ .cap_permitted = CAP_FULL_SET,
+ .cap_effective = CAP_INIT_EFF_SET,
+ .user = INIT_USER,
+#ifdef CONFIG_KEYS
+ .tgsec = &init_thread_group_security,
+#endif
+ .group_info = &init_groups,
+ .lock = __SPIN_LOCK_UNLOCKED(init_task_security.lock),
+};
+
+/**
+ * dup_task_security - Duplicate task security record
+ * @sec: The record to duplicate
+ *
+ * Returns a duplicate of a task security record or NULL if out of memory.
+ */
+struct task_security *dup_task_security(struct task_security *_sec)
+{
+ struct task_security *sec;
+
+ sec = kmemdup(_sec, sizeof(*sec), GFP_KERNEL);
+ if (sec) {
+ atomic_set(&sec->usage, 1);
+ get_uid(sec->user);
+ get_group_info(sec->group_info);
+ key_get(sec->thread_keyring);
+ key_get(sec->request_key_auth);
+ security_task_dup(sec);
+ }
+ return sec;
+}
+EXPORT_SYMBOL(dup_task_security);
+
+/**
```

[PATCH 7/7] SECURITY: De-embed task security record from task and use refcounting

```
+ * copy_task_security – Copy the task security records for fork
+ * @p: The new task
+ * @clone_flags: The details of what the new process shares of the old
+ *
+ * Copy the task security records on a task so that it can affect objects
+ * in the same way as its parent. Returns 0 if successful or –ENOMEM if out of
+ * memory.
+ */
+int copy_task_security(struct task_struct *p, unsigned long clone_flags)
+{
+ struct task_security *sec;
+
+ sec = kmemdup(p->sec, sizeof(*sec), GFP_KERNEL);
+ if (!sec)
+ return –ENOMEM;
+
+ atomic_set(&sec->usage, 2);
+ spin_lock_init(&sec->lock);
+ get_group_info(sec->group_info);
+ get_uid(p->sec->user);
+
+#ifdef CONFIG_KEYS
+ if (clone_flags & CLONE_THREAD) {
+ atomic_inc(&sec->tgsec->usage);
+ } else {
+ struct thread_group_security *tgsec;
+
+ tgsec = kcalloc(sizeof(*tgsec), GFP_KERNEL);
+ if (!tgsec) {
+ kfree(sec);
+ return –ENOMEM;
+ }
+ atomic_set(&tgsec->usage, 1);
+ spin_lock_init(&tgsec->lock);
+ tgsec->tgid = p->tgid;
+ copy_thread_group_keys(tgsec);
+ sec->tgsec = tgsec;
+ }
+ key_get(sec->request_key_auth);
+ sec->thread_keyring = NULL;
+#endif
+
+#ifdef CONFIG_SECURITY
+ sec->security = NULL;
+#endif
+
+ p->act_as = p->sec = sec;
+ return 0;
+}
+EXPORT_SYMBOL(copy_task_security);
+
```

[PATCH 7/7] SECURITY: De-embed task security record from task and use refcounting

```
+/**
+ * put_task_security – Release a ref on a task security record
+ * @sec: The record to release
+ *
+ * Release a reference to a task security record and destroy it when
+ * there are no references remaining.
+ */
+void put_task_security(struct task_security *sec)
+{
+ if (sec && atomic_dec_and_test(&sec->usage)) {
+ security_task_free(sec);
+ key_put(sec->thread_keyring);
+ key_put(sec->request_key_auth);
+ put_group_info(sec->group_info);
+ free_uid(sec->user);
+
+ #ifdef CONFIG_KEYS
+ if (atomic_dec_and_test(&sec->tgsec->usage)) {
+ key_put(sec->tgsec->session_keyring);
+ key_put(sec->tgsec->process_keyring);
+ }
+ #endif
+
+ kfree(sec);
+ }
+}
+EXPORT_SYMBOL(put_task_security);
diff --git a/kernel/exit.c b/kernel/exit.c
index d793e22..507b54b 100644
--- a/kernel/exit.c
+++ b/kernel/exit.c
@@ -1000,7 +1000,6 @@ fastcall NORET_TYPE void do_exit(long code)
check_stack_usage();
exit_thread();
cgroup_exit(tsk, 1);
- exit_keys(tsk);

if (group_dead && tsk->signal->leader)
disassociate_ctty(1);
diff --git a/kernel/fork.c b/kernel/fork.c
index 2f267bd..e4572b0 100644
--- a/kernel/fork.c
+++ b/kernel/fork.c
@@ -121,9 +121,8 @@ void __put_task_struct(struct task_struct *tsk)
WARN_ON(atomic_read(&tsk->usage));
WARN_ON(tsk == current);

- security_task_free(tsk);
- free_uid(tsk->__temp_sec.user);
- put_group_info(tsk->__temp_sec.group_info);
+ put_task_security(tsk->sec);
```

[PATCH 7/7] SECURITY: De-embed task security record from task and use refcounting

```
+ put_task_security(tsk->act_as);
delayacct_tsk_free(tsk);

if (!profile_handoff_task(tsk))
@@ -845,7 +844,6 @@ void __cleanup_sighand(struct sighand_struct *sighand)
static int copy_signal(unsigned long clone_flags, struct task_struct *tsk)
{
struct signal_struct *sig;
- int ret;

if (clone_flags & CLONE_THREAD) {
atomic_inc(&current->signal->count);
@@ -857,12 +855,6 @@ static int copy_signal(unsigned long clone_flags, struct task_struct *tsk)
if (!sig)
return -ENOMEM;

- ret = copy_thread_group_keys(tsk);
- if (ret < 0) {
- kmem_cache_free(signal_cachep, sig);
- return ret;
- }
-
atomic_set(&sig->count, 1);
atomic_set(&sig->live, 1);
init_waitqueue_head(&sig->wait_chldexit);
@@ -920,7 +912,6 @@ static int copy_signal(unsigned long clone_flags, struct task_struct *tsk)

void __cleanup_signal(struct signal_struct *sig)
{
- exit_thread_group_keys(sig);
kmem_cache_free(signal_cachep, sig);
}

@@ -1014,18 +1005,19 @@ static struct task_struct *copy_process(unsigned long clone_flags,
DEBUG_LOCKS_WARN_ON(!p->hardirqs_enabled);
DEBUG_LOCKS_WARN_ON(!p->softirqs_enabled);
#endif
- p->act_as = p->sec = &p->__temp_sec;
+ retval = copy_task_security(p, clone_flags);
+ if (retval < 0)
+ goto bad_fork_free;
+
retval = -EAGAIN;
if (atomic_read(&p->sec->user->processes) >=
p->signal->rlim[RLIMIT_NPROC].rlim_cur) {
if (!capable(CAP_SYS_ADMIN) && !capable(CAP_SYS_RESOURCE) &&
p->sec->user != current->nsproxy->user_ns->root_user)
- goto bad_fork_free;
+ goto bad_fork_cleanup_put_task_sec;
}
}
```

[PATCH 7/7] SECURITY: De-embed task security record from task and use refcounting

```
- atomic_inc(&p->sec->user->__count);
atomic_inc(&p->sec->user->processes);
- get_group_info(p->sec->group_info);

/*
 * If multiple threads are within copy_process(), then this check
@@ -1080,9 +1072,6 @@ static struct task_struct *copy_process(unsigned long clone_flags,
do_posix_clock_monotonic_gettime(&p->start_time);
p->real_start_time = p->start_time;
monotonic_to_bootbased(&p->real_start_time);
-#ifdef CONFIG_SECURITY
- p->sec->security = NULL;
-#endif
p->io_context = NULL;
p->audit_context = NULL;
cgroup_fork(p);
@@ -1130,7 +1119,7 @@ static struct task_struct *copy_process(unsigned long clone_flags,
if ((retval = security_task_alloc(p)))
goto bad_fork_cleanup_policy;
if ((retval = audit_alloc(p))
- goto bad_fork_cleanup_security;
+ goto bad_fork_cleanup_policy;
/* copy all the process information */
if ((retval = copy_semundo(clone_flags, p)))
goto bad_fork_cleanup_audit;
@@ -1144,10 +1133,8 @@ static struct task_struct *copy_process(unsigned long clone_flags,
goto bad_fork_cleanup_sighand;
if ((retval = copy_mm(clone_flags, p))
goto bad_fork_cleanup_signal;
- if ((retval = copy_keys(clone_flags, p))
- goto bad_fork_cleanup_mm;
if ((retval = copy_namespaces(clone_flags, p))
- goto bad_fork_cleanup_keys;
+ goto bad_fork_cleanup_mm;
retval = copy_thread(0, clone_flags, stack_start, stack_size, p, regs);
if (retval)
goto bad_fork_cleanup_namespaces;
@@ -1329,8 +1316,6 @@ bad_fork_free_pid:
free_pid(pid);
bad_fork_cleanup_namespaces:
exit_task_namespaces(p);
-bad_fork_cleanup_keys:
- exit_keys(p);
bad_fork_cleanup_mm:
if (p->mm)
mmapput(p->mm);
@@ -1346,8 +1331,6 @@ bad_fork_cleanup_semundo:
exit_sem(p);
bad_fork_cleanup_audit:
audit_free(p);
-bad_fork_cleanup_security:
```

[PATCH 7/7] SECURITY: De-embed task security record from task and use refcounting

```
- security_task_free(p);
bad_fork_cleanup_policy:
#ifdef CONFIG_NUMA
mpol_free(p->mempolicy);
@@ -1360,9 +1343,10 @@ bad_fork_cleanup_cgroup:
bad_fork_cleanup_put_domain:
module_put(task_thread_info(p)->exec_domain->module);
bad_fork_cleanup_count:
- put_group_info(p->sec->group_info);
atomic_dec(&p->sec->user->processes);
- free_uid(p->sec->user);
+bad_fork_cleanup_put_task_sec:
+ put_task_security(p->act_as);
+ put_task_security(p->sec);
bad_fork_free:
free_task(p);
fork_out:
diff --git a/kernel/kmod.c b/kernel/kmod.c
index c6a4f8a..7e34d46 100644
--- a/kernel/kmod.c
+++ b/kernel/kmod.c
@@ -133,20 +133,18 @@ struct subprocess_info {
static int ____call_usermodehelper(void *data)
{
struct subprocess_info *sub_info = data;
- struct key *new_session, *old_session;
int retval;

- /* Unblock all signals and set the session keyring. */
- new_session = key_get(sub_info->ring);
+ /* Set the session keyring. */
+ __install_session_keyring(current, sub_info->ring);
+
+ /* Unblock all signals. */
spin_lock_irq(&current->siglock);
- old_session = __install_session_keyring(current, new_session);
flush_signal_handlers(current, 1);
sigemptyset(&current->blocked);
recalc_sigpending();
spin_unlock_irq(&current->siglock);

- key_put(old_session);
-
/* Install input pipe when needed */
if (sub_info->stdin) {
struct files_struct *f = current->files;
diff --git a/kernel/sys.c b/kernel/sys.c
index 14acc1b..e331b6f 100644
--- a/kernel/sys.c
+++ b/kernel/sys.c
@@ -521,7 +521,7 @@ asmlinkage long sys_setregid(gid_t rgid, gid_t egid)
```

[PATCH 7/7] SECURITY: De-embed task security record from task and use refcounting

```
sec->fsgid = new_egid;
sec->egid = new_egid;
sec->gid = new_rgid;
- key_fsgid_changed(current);
+ key_fsgid_changed(sec);
proc_id_connector(current, PROC_EVENT_GID);
return 0;
}
@@ -557,7 +557,7 @@ asmlinkage long sys_setgid(gid_t gid)
else
return -EPERM;

- key_fsgid_changed(current);
+ key_fsgid_changed(sec);
proc_id_connector(current, PROC_EVENT_GID);
return 0;
}
@@ -646,7 +646,7 @@ asmlinkage long sys_setreuid(uid_t ruid, uid_t euid)
sec->suid = sec->euid;
sec->fsuid = sec->euid;

- key_fsuid_changed(current);
+ key_fsuid_changed(sec);
proc_id_connector(current, PROC_EVENT_UID);

return security_task_post_setuid(old_ruid, old_euid, old_suid, LSM_SETID_RE);
@@ -694,7 +694,7 @@ asmlinkage long sys_setuid(uid_t uid)
sec->fsuid = sec->euid = uid;
sec->suid = new_suid;

- key_fsuid_changed(current);
+ key_fsuid_changed(sec);
proc_id_connector(current, PROC_EVENT_UID);

return security_task_post_setuid(old_ruid, old_euid, old_suid, LSM_SETID_ID);
@@ -744,7 +744,7 @@ asmlinkage long sys_setresuid(uid_t ruid, uid_t euid, uid_t suid)
if (suid != (uid_t) -1)
sec->suid = suid;

- key_fsuid_changed(current);
+ key_fsuid_changed(sec);
proc_id_connector(current, PROC_EVENT_UID);

return security_task_post_setuid(old_ruid, old_euid, old_suid, LSM_SETID_RES);
@@ -798,7 +798,7 @@ asmlinkage long sys_setresgid(gid_t rgid, gid_t egid, gid_t sgid)
if (sgid != (gid_t) -1)
sec->sgid = sgid;

- key_fsgid_changed(current);
+ key_fsgid_changed(sec);
proc_id_connector(current, PROC_EVENT_GID);
```

[PATCH 7/7] SECURITY: De-embed task security record from task and use refcounting

```
return 0;
}
@@ -841,7 +841,7 @@ asmlinkage long sys_setfsuid(uid_t uid)
sec->fsuid = uid;
}

- key_fsuid_changed(current);
+ key_fsuid_changed(sec);
proc_id_connector(current, PROC_EVENT_UID);

security_task_post_setuid(old_fsuid, (uid_t)-1, (uid_t)-1, LSM_SETID_FS);
@@ -869,7 +869,7 @@ asmlinkage long sys_setfsgid(gid_t gid)
smp_wmb();
}
sec->fsgid = gid;
- key_fsgid_changed(current);
+ key_fsgid_changed(sec);
proc_id_connector(current, PROC_EVENT_GID);
}
return old_fsgid;
diff --git a/kernel/user.c b/kernel/user.c
index d9a84b1..0555576 100644
--- a/kernel/user.c
+++ b/kernel/user.c
@@ -356,7 +356,7 @@ struct user_struct * alloc_uid(struct user_namespace *ns, uid_t uid)
#endif
new->locked_shm = 0;

- if (alloc_uid_keyring(new, current) < 0) {
+ if (alloc_uid_keyring(new, current->sec) < 0) {
kmem_cache_free(uid_cachep, new);
uids_mutex_unlock();
return NULL;
diff --git a/net/rxrpc/ar-key.c b/net/rxrpc/ar-key.c
index 9a8ff68..14979a5 100644
--- a/net/rxrpc/ar-key.c
+++ b/net/rxrpc/ar-key.c
@@ -297,7 +297,7 @@ int rxrpc_get_server_data_key(struct rxrpc_connection *conn,
_enter("");

- key = key_alloc(&key_type_rxrpc, "x", 0, 0, current, 0,
+ key = key_alloc(&key_type_rxrpc, "x", 0, 0, current->act_as, 0,
KEY_ALLOC_NOT_IN_QUOTA);
if (IS_ERR(key)) {
_leave(" = -ENOMEM [alloc %ld]", PTR_ERR(key));
@@ -343,7 +343,7 @@ struct key *rxrpc_get_null_key(const char *keyname)
struct key *key;
int ret;

- key = key_alloc(&key_type_rxrpc, keyname, 0, 0, current,
```

[PATCH 7/7] SECURITY: De-embed task security record from task and use refcounting

```
+ key = key_alloc(&key_type_rxrpc, keyname, 0, 0, current->act_as,
KEY_POS_SEARCH, KEY_ALLOC_NOT_IN_QUOTA);
if (IS_ERR(key))
return key;
diff --git a/security/dummy.c b/security/dummy.c
index 287f3ec..526549d 100644
--- a/security/dummy.c
+++ b/security/dummy.c
@@ -480,16 +480,21 @@ static int dummy_task_create (unsigned long clone_flags)
return 0;
}

-static int dummy_task_alloc_security (struct task_struct *p)
+static int dummy_task_alloc_security(struct task_struct *p)
{
return 0;
}

-static void dummy_task_free_security (struct task_struct *p)
+static void dummy_task_free_security(struct task_security *sec)
{
return;
}

+static int dummy_task_dup_security(struct task_security *p)
+{
+ return 0;
+}
+
+static int dummy_task_setuid (uid_t id0, uid_t id1, uid_t id2, int flags)
{
return 0;
@@ -943,7 +948,7 @@ static void dummy_release_secctx(char *secddata, u32 seclen)
}

#ifdef CONFIG_KEYS
-static inline int dummy_key_alloc(struct key *key, struct task_struct *ctx,
+static inline int dummy_key_alloc(struct key *key, struct task_security *sec,
unsigned long flags)
{
return 0;
@@ -954,7 +959,7 @@ static inline void dummy_key_free(struct key *key)
}

static inline int dummy_key_permission(key_ref_t key_ref,
- struct task_struct *context,
+ struct task_security *sec,
key_perm_t perm)
{
return 0;
@@ -1057,6 +1062,7 @@ void security_fixup_ops (struct security_operations *ops)
```

[PATCH 7/7] SECURITY: De-embed task security record from task and use refcounting

```
set_to_dummy_if_null(ops, task_create);
set_to_dummy_if_null(ops, task_alloc_security);
set_to_dummy_if_null(ops, task_free_security);
+ set_to_dummy_if_null(ops, task_dup_security);
set_to_dummy_if_null(ops, task_setuid);
set_to_dummy_if_null(ops, task_post_setuid);
set_to_dummy_if_null(ops, task_setgid);
diff --git a/security/keys/internal.h b/security/keys/internal.h
index f004835..a439889 100644
--- a/security/keys/internal.h
+++ b/security/keys/internal.h
@@ -92,7 +92,7 @@ extern struct key *keyring_search_instkey(struct key *keyring,
typedef int (*key_match_func_t)(const struct key *, const void *);

extern key_ref_t keyring_search_aux(key_ref_t keyring_ref,
- struct task_struct *tsk,
+ struct task_security *sec,
struct key_type *type,
const void *description,
key_match_func_t match);
@@ -100,12 +100,12 @@ extern key_ref_t keyring_search_aux(key_ref_t keyring_ref,
extern key_ref_t search_process_keyrings(struct key_type *type,
const void *description,
key_match_func_t match,
- struct task_struct *tsk);
+ struct task_security *sec);

extern struct key *find_keyring_by_name(const char *name, key_serial_t bound);

-extern int install_thread_keyring(struct task_struct *tsk);
-extern int install_process_keyring(struct task_struct *tsk);
+extern int install_thread_keyring(struct task_security *sec);
+extern int install_process_keyring(struct task_security *sec);

extern struct key *request_key_and_link(struct key_type *type,
const char *description,
@@ -120,7 +120,7 @@ extern struct key *request_key_and_link(struct key_type *type,
*/
struct request_key_auth {
struct key *target_key;
- struct task_struct *context;
+ struct task_security *sec;
void *callout_info;
size_t callout_len;
pid_t pid;
diff --git a/security/keys/key.c b/security/keys/key.c
index 1692988..68483d7 100644
--- a/security/keys/key.c
+++ b/security/keys/key.c
@@ -243,7 +243,7 @@ serial_exists:
* instantiate the key or discard it before returning
```

[PATCH 7/7] SECURITY: De-embed task security record from task and use refcounting

```
*/
struct key *key_alloc(struct key_type *type, const char *desc,
- uid_t uid, gid_t gid, struct task_struct *ctx,
+ uid_t uid, gid_t gid, struct task_security *sec,
key_perm_t perm, unsigned long flags)
{
struct key_user *user = NULL;
@@ -314,7 +314,7 @@ struct key *key_alloc(struct key_type *type, const char *desc,
#endif

/* let the security module know about the key */
- ret = security_key_alloc(key, ctx, flags);
+ ret = security_key_alloc(key, sec, flags);
if (ret < 0)
goto security_error;

@@ -818,7 +818,7 @@ key_ref_t key_create_or_update(key_ref_t keyring_ref,

/* allocate a new key */
key = key_alloc(ktype, description, current_fsuid(), current_fsgid(),
- current, perm, flags);
+ current->act_as, perm, flags);
if (IS_ERR(key)) {
key_ref = ERR_PTR(PTR_ERR(key));
goto error_3;
diff --git a/security/keys/keyctl.c b/security/keys/keyctl.c
index 4051948..2900451 100644
--- a/security/keys/keyctl.c
+++ b/security/keys/keyctl.c
@@ -878,7 +878,7 @@ long keyctl_instantiate_key(key_serial_t id,
* requesting task */
keyring_ref = NULL;
if (ringid) {
- keyring_ref = lookup_user_key(rka->context, ringid, 1, 0,
+ keyring_ref = lookup_user_key(rka->sec, ringid, 1, 0,
KEY_WRITE);
if (IS_ERR(keyring_ref)) {
ret = PTR_ERR(keyring_ref);
@@ -973,13 +973,13 @@ long keyctl_set_reqkey_keyring(int reqkey_defl)

switch (reqkey_defl) {
case KEY_REQKEY_DEFL_THREAD_KEYRING:
- ret = install_thread_keyring(current);
+ ret = install_thread_keyring(sec);
if (ret < 0)
return ret;
goto set;

case KEY_REQKEY_DEFL_PROCESS_KEYRING:
- ret = install_process_keyring(current);
+ ret = install_process_keyring(sec);
```

[PATCH 7/7] SECURITY: De-embed task security record from task and use refcounting

```
if (ret < 0)
return ret;
```

```
diff --git a/security/keys/keyring.c b/security/keys/keyring.c
index 76b89b2..6ccd8f8 100644
--- a/security/keys/keyring.c
+++ b/security/keys/keyring.c
@@ -244,14 +244,14 @@ static long keyring_read(const struct key *keyring,
 * allocate a keyring and link into the destination keyring
 */
struct key *keyring_alloc(const char *description, uid_t uid, gid_t gid,
- struct task_struct *ctx, unsigned long flags,
+ struct task_security *sec, unsigned long flags,
struct key *dest)
{
struct key *keyring;
int ret;
```

```
keyring = key_alloc(&key_type_keyring, description,
- uid, gid, ctx,
+ uid, gid, sec,
(KEY_POS_ALL & ~KEY_POS_SETATTR) | KEY_USR_ALL,
flags);
```

```
@@ -280,7 +280,7 @@ struct key *keyring_alloc(const char *description, uid_t uid, gid_t gid,
 * - we propagate the possession attribute from the keyring ref to the key ref
 */
key_ref_t keyring_search_aux(key_ref_t keyring_ref,
- struct task_struct *context,
+ struct task_security *sec,
struct key_type *type,
const void *description,
key_match_func_t match)
@@ -303,7 +303,7 @@ key_ref_t keyring_search_aux(key_ref_t keyring_ref,
key_check(keyring);
```

```
/* top keyring must have search permission to begin the search */
- err = key_task_permission(keyring_ref, context, KEY_SEARCH);
+ err = key_task_permission(keyring_ref, sec, KEY_SEARCH);
if (err < 0) {
key_ref = ERR_PTR(err);
goto error;
@@ -376,7 +376,7 @@ descend:
```

```
/* key must have search permissions */
if (key_task_permission(make_key_ref(key, possessed),
- context, KEY_SEARCH) < 0)
+ sec, KEY_SEARCH) < 0)
continue;
```

```
/* we set a different error code if we pass a negative key */
```

[PATCH 7/7] SECURITY: De-embed task security record from task and use refcounting

```
@@ -403,7 +403,7 @@ ascend:
continue;

if (key_task_permission(make_key_ref(key, possessed),
- context, KEY_SEARCH) < 0)
+ sec, KEY_SEARCH) < 0)
continue;

/* stack the current position */
@@ -458,7 +458,7 @@ key_ref_t keyring_search(key_ref_t keyring,
if (!type->match)
return ERR_PTR(-ENOKEY);

- return keyring_search_aux(keyring, current,
+ return keyring_search_aux(keyring, current->sec,
type, description, type->match);

} /* end keyring_search() */
diff --git a/security/keys/permission.c b/security/keys/permission.c
index 07898bd..eff3e29 100644
--- a/security/keys/permission.c
+++ b/security/keys/permission.c
@@ -19,10 +19,9 @@
* but permit the security modules to override
*/
int key_task_permission(const key_ref_t key_ref,
- struct task_struct *context,
+ struct task_security *sec,
key_perm_t perm)
{
- struct task_security *sec = context->act_as;
struct key *key;
key_perm_t kperm;
int ret;
@@ -69,7 +68,7 @@ use_these_perms:
return -EACCES;

/* let LSM be the final arbiter */
- return security_key_permission(key_ref, context, perm);
+ return security_key_permission(key_ref, sec, perm);

} /* end key_task_permission() */

diff --git a/security/keys/proc.c b/security/keys/proc.c
index 3e0d0a6..91e6df2 100644
--- a/security/keys/proc.c
+++ b/security/keys/proc.c
@@ -141,7 +141,7 @@ static int proc_keys_show(struct seq_file *m, void *v)

/* check whether the current task is allowed to view the key (assuming
* non-possession) */
```

[PATCH 7/7] SECURITY: De-embed task security record from task and use refcounting

```
- rc = key_task_permission(make_key_ref(key, 0), current, KEY_VIEW);
+ rc = key_task_permission(make_key_ref(key, 0), current->sec, KEY_VIEW);
if (rc < 0)
return 0;
```

```
diff --git a/security/keys/process_keys.c b/security/keys/process_keys.c
index 98854b2..f51a08c 100644
```

```
--- a/security/keys/process_keys.c
```

```
+++ b/security/keys/process_keys.c
```

```
@@ -68,7 +68,7 @@ struct key root_session_keyring = {
```

```
* allocate the keyrings to be associated with a UID
```

```
*/
```

```
int alloc_uid_keyring(struct user_struct *user,
```

```
- struct task_struct *ctx)
```

```
+ struct task_security *sec)
```

```
{
```

```
struct key *uid_keyring, *session_keyring;
```

```
char buf[20];
```

```
@@ -77,7 +77,7 @@ int alloc_uid_keyring(struct user_struct *user,
```

```
/* concoct a default session keyring */
```

```
sprintf(buf, "_uid_ses.%u", user->uid);
```

```
- session_keyring = keyring_alloc(buf, user->uid, (gid_t) -1, ctx,
```

```
+ session_keyring = keyring_alloc(buf, user->uid, (gid_t) -1, sec,
```

```
KEY_ALLOC_IN_QUOTA, NULL);
```

```
if (IS_ERR(session_keyring)) {
```

```
ret = PTR_ERR(session_keyring);
```

```
@@ -88,7 +88,7 @@ int alloc_uid_keyring(struct user_struct *user,
```

```
* keyring */
```

```
sprintf(buf, "_uid.%u", user->uid);
```

```
- uid_keyring = keyring_alloc(buf, user->uid, (gid_t) -1, ctx,
```

```
+ uid_keyring = keyring_alloc(buf, user->uid, (gid_t) -1, sec,
```

```
KEY_ALLOC_IN_QUOTA, session_keyring);
```

```
if (IS_ERR(uid_keyring)) {
```

```
key_put(session_keyring);
```

```
@@ -135,33 +135,29 @@ void switch_uid_keyring(struct user_struct *new_user)
```

```
/*
```

```
*/
```

```
- * install a fresh thread keyring, discarding the old one
```

```
+ * make sure a thread keyring is installed
```

```
*/
```

```
-int install_thread_keyring(struct task_struct *task)
```

```
+int install_thread_keyring(struct task_security *sec)
```

```
{
```

```
- struct key *keyring, *old;
```

```
+ struct key *keyring;
```

```
char buf[20];
```

```
- int ret;
```

[PATCH 7/7] SECURITY: De-embed task security record from task and use refcounting

```
- sprintf(buf, "_tid.%u", tsk->pid);
+ sprintf(buf, "_tid.%u", current->pid);

- keyring = keyring_alloc(buf, tsk->sec->uid, tsk->sec->gid, tsk,
+ keyring = keyring_alloc(buf, sec->uid, sec->gid, sec,
KEY_ALLOC_QUOTA_OVERRUN, NULL);
- if (IS_ERR(keyring)) {
- ret = PTR_ERR(keyring);
- goto error;
- }
-
- task_lock(tsk);
- old = tsk->sec->thread_keyring;
- tsk->sec->thread_keyring = keyring;
- task_unlock(tsk);
+ if (IS_ERR(keyring))
+ return PTR_ERR(keyring);

- ret = 0;
+ spin_lock(&sec->lock);
+ if (!sec->thread_keyring) {
+ sec->thread_keyring = keyring;
+ keyring = NULL;
+ }
+ spin_unlock(&sec->lock);

- key_put(old);
-error:
- return ret;
+ key_put(keyring);
+ return 0;

} /* end install_thread_keyring() */

@@ -169,38 +165,36 @@ error:
/*
* make sure a process keyring is installed
*/
-int install_process_keyring(struct task_struct *tsk)
+int install_process_keyring(struct task_security *sec)
{
+ struct thread_group_security *tgsec;
struct key *keyring;
char buf[20];
- int ret;

might_sleep();
+ sec = current->sec;
+ tgsec = sec->tgsec;

- if (!tsk->signal->process_keyring) {
```

[PATCH 7/7] SECURITY: De-embed task security record from task and use refcounting

```
- sprintf(buf, "_pid.%u", tsk->tgid);
+ if (!tgsec->process_keyring) {
+ sprintf(buf, "_pid.%u", tgsec->tgid);

- keyring = keyring_alloc(buf, tsk->sec->uid, tsk->sec->gid, tsk,
+ keyring = keyring_alloc(buf, sec->uid, sec->gid, sec,
KEY_ALLOC_QUOTA_OVERRUN, NULL);
- if (IS_ERR(keyring)) {
- ret = PTR_ERR(keyring);
- goto error;
- }
+ if (IS_ERR(keyring))
+ return PTR_ERR(keyring);

/* attach keyring */
- spin_lock_irq(&tsk->sigand->siglock);
- if (!tsk->signal->process_keyring) {
- tsk->signal->process_keyring = keyring;
+ spin_lock(&tgsec->lock);
+ if (!tgsec->process_keyring) {
+ tgsec->process_keyring = keyring;
keyring = NULL;
}
- spin_unlock_irq(&tsk->sigand->siglock);
+ spin_unlock(&tgsec->lock);

key_put(keyring);
}

- ret = 0;
-error:
- return ret;
+ return 0;

} /* end install_process_keyring() */

@@ -209,37 +203,38 @@ error:
* install a session keyring, discarding the old one
* - if a keyring is not supplied, an empty one is invented
*/
-static int install_session_keyring(struct task_struct *tsk,
+static int install_session_keyring(struct task_security *sec,
struct key *keyring)
{
+ struct thread_group_security *tgsec;
unsigned long flags;
struct key *old;
char buf[20];

might_sleep();
+ tgsec = sec->tgsec;
```

[PATCH 7/7] SECURITY: De-embed task security record from task and use refcounting

```
/* create an empty session keyring */
if (!keyring) {
- sprintf(buf, "_ses.%u", tsk->tgid);
+ sprintf(buf, "_ses.%u", tgsec->tgid);

flags = KEY_ALLOC_QUOTA_OVERRUN;
- if (tsk->signal->session_keyring)
+ if (tgsec->session_keyring)
flags = KEY_ALLOC_IN_QUOTA;

- keyring = keyring_alloc(buf, tsk->sec->uid, tsk->sec->gid, tsk,
+ keyring = keyring_alloc(buf, sec->uid, sec->gid, sec,
flags, NULL);
if (IS_ERR(keyring))
return PTR_ERR(keyring);
- }
- else {
+ } else {
atomic_inc(&keyring->usage);
}

/* install the keyring */
- spin_lock_irq(&tsk->sigband->siglock);
- old = tsk->signal->session_keyring;
- rcu_assign_pointer(tsk->signal->session_keyring, keyring);
- spin_unlock_irq(&tsk->sigband->siglock);
+ spin_lock_irq(&tgsec->lock);
+ old = tgsec->session_keyring;
+ rcu_assign_pointer(tgsec->session_keyring, keyring);
+ spin_unlock_irq(&tgsec->lock);

/* we're using RCU on the pointer, but there's no point synchronising
* on it if it didn't previously point to anything */
@@ -252,68 +247,49 @@ static int install_session_keyring(struct task_struct *tsk,

} /* end install_session_keyring() */

-/**
- * copy the keys in a thread group for fork without CLONE_THREAD
- * install a session keyring for kmod
- */
-int copy_thread_group_keys(struct task_struct *tsk)
+void __install_session_keyring(struct task_struct *tsk, struct key *keyring)
{
- key_check(current->thread_group->session_keyring);
- key_check(current->thread_group->process_keyring);
-
- /* no process keyring yet */
- tsk->signal->process_keyring = NULL;
```

[PATCH 7/7] SECURITY: De-embed task security record from task and use refcounting

```

+ struct thread_group_security *tgsec = tsk->sec->tgsec;
+ struct key *old;

- /* same session keyring */
- rcu_read_lock();
- tsk->signal->session_keyring =
- key_get(rcu_dereference(current->signal->session_keyring));
- rcu_read_unlock();
+ key_get(keyring);

- return 0;
+ spin_lock(&tgsec->lock);
+ old = tgsec->session_keyring;
+ rcu_assign_pointer(tgsec->session_keyring, keyring);
+ spin_unlock(&tgsec->lock);

-} /* end copy_thread_group_keys() */
+ /* we're using RCU on the pointer, but there's no point synchronising
+ * on it if it didn't previously point to anything */
+ if (old) {
+ synchronize_rcu();
+ key_put(old);
+ }
+ }

/******
/*
- * copy the keys for fork
+ * copy the keys in a thread group for fork without CLONE_THREAD
*/
-int copy_keys(unsigned long clone_flags, struct task_struct *tsk)
+int copy_thread_group_keys(struct thread_group_security *tgsec)
{
- key_check(tsk->sec->thread_keyring);
- key_check(tsk->sec->request_key_auth);
+ key_check(tgsec->session_keyring);
+ key_check(tgsec->process_keyring);

- /* no thread keyring yet */
- tsk->sec->thread_keyring = NULL;
+ /* no process keyring yet */
+ tgsec->process_keyring = NULL;

- /* copy the request_key() authorisation for this thread */
- key_get(tsk->sec->request_key_auth);
+ /* same session keyring */
+ rcu_read_lock();
+ tgsec->session_keyring =
+ key_get(rcu_dereference(current->sec->tgsec->session_keyring));
+ rcu_read_unlock();

```

[PATCH 7/7] SECURITY: De-embed task security record from task and use refcounting

```

return 0;
-
-} /* end copy_keys() */
-
-/*
-/*
- * dispose of thread group keys upon thread group destruction
- */
-void exit_thread_group_keys(struct signal_struct *tg)
- {
- key_put(tg->session_keyring);
- key_put(tg->process_keyring);
-
-} /* end exit_thread_group_keys() */
-
-/*
- * dispose of per-thread keys upon thread exit
- */
-void exit_keys(struct task_struct *tsk)
- {
- key_put(tsk->sec->thread_keyring);
- key_put(tsk->sec->request_key_auth);
-
-} /* end exit_keys() */
+}

/*
@@ -321,21 +297,23 @@ void exit_keys(struct task_struct *tsk)
*/
int exec_keys(struct task_struct *tsk)
{
+ struct thread_group_security *tgsec = tsk->sec->tgsec;
+ struct task_security *sec = tsk->sec;
struct key *old;

/* newly exec'd tasks don't get a thread keyring */
- task_lock(tsk);
- old = tsk->sec->thread_keyring;
- tsk->sec->thread_keyring = NULL;
- task_unlock(tsk);
+ spin_lock(&sec->lock);
+ old = sec->thread_keyring;
+ sec->thread_keyring = NULL;
+ spin_unlock(&sec->lock);

key_put(old);

/* discard the process keyring from a newly exec'd task */
- spin_lock_irq(&tsk->sigband->siglock);

```

[PATCH 7/7] SECURITY: De-embed task security record from task and use refcounting

```
- old = tsk->signal->process_keyring;
- tsk->signal->process_keyring = NULL;
- spin_unlock_irq(&tsk->sigband->siglock);
+ spin_lock(&tgsec->lock);
+ old = tgsec->process_keyring;
+ tgsec->process_keyring = NULL;
+ spin_unlock(&tgsec->lock);
```

```
key_put(old);
```

```
@@ -358,14 +336,13 @@ int suid_keys(struct task_struct *tsk)
```

```
/*
```

```
* the filesystem user ID changed
```

```
*/
```

```
-void key_fsuid_changed(struct task_struct *tsk)
+void key_fsuid_changed(struct task_security *sec)
{
/* update the ownership of the thread keyring */
- BUG_ON(!tsk->sec);
- if (tsk->sec->thread_keyring) {
- down_write(&tsk->sec->thread_keyring->sem);
- tsk->sec->thread_keyring->uid = tsk->sec->fsuid;
- up_write(&tsk->sec->thread_keyring->sem);
+ if (sec->thread_keyring) {
+ down_write(&sec->thread_keyring->sem);
+ sec->thread_keyring->uid = sec->fsuid;
+ up_write(&sec->thread_keyring->sem);
}
}
```

```
 } /* end key_fsuid_changed() */
```

```
@@ -374,14 +351,13 @@ void key_fsuid_changed(struct task_struct *tsk)
```

```
/*
```

```
* the filesystem group ID changed
```

```
*/
```

```
-void key_fsgid_changed(struct task_struct *tsk)
+void key_fsgid_changed(struct task_security *sec)
{
/* update the ownership of the thread keyring */
- BUG_ON(!tsk->sec);
- if (tsk->sec->thread_keyring) {
- down_write(&tsk->sec->thread_keyring->sem);
- tsk->sec->thread_keyring->gid = tsk->sec->fsgid;
- up_write(&tsk->sec->thread_keyring->sem);
+ if (sec->thread_keyring) {
+ down_write(&sec->thread_keyring->sem);
+ sec->thread_keyring->gid = sec->fsgid;
+ up_write(&sec->thread_keyring->sem);
}
}
```

```
 } /* end key_fsgid_changed() */
```

```
@@ -397,7 +373,7 @@ void key_fsgid_changed(struct task_struct *tsk)
```

[PATCH 7/7] SECURITY: De-embed task security record from task and use refcounting

```
key_ref_t search_process_keyrings(struct key_type *type,
const void *description,
key_match_func_t match,
- struct task_struct *context)
+ struct task_security *sec)
{
struct request_key_auth *rka;
key_ref_t key_ref, ret, err;
@@ -416,10 +392,10 @@ key_ref_t search_process_keyrings(struct key_type *type,
err = ERR_PTR(-EAGAIN);

/* search the thread keyring first */
- if (context->sec->thread_keyring) {
+ if (sec->thread_keyring) {
key_ref = keyring_search_aux(
- make_key_ref(context->sec->thread_keyring, 1),
- context, type, description, match);
+ make_key_ref(sec->thread_keyring, 1),
+ sec, type, description, match);
if (!IS_ERR(key_ref))
goto found;

@@ -437,10 +413,10 @@ key_ref_t search_process_keyrings(struct key_type *type,
}

/* search the process keyring second */
- if (context->signal->process_keyring) {
+ if (sec->tgsec->process_keyring) {
key_ref = keyring_search_aux(
- make_key_ref(context->signal->process_keyring, 1),
- context, type, description, match);
+ make_key_ref(sec->tgsec->process_keyring, 1),
+ sec, type, description, match);
if (!IS_ERR(key_ref))
goto found;

@@ -458,13 +434,13 @@ key_ref_t search_process_keyrings(struct key_type *type,
}

/* search the session keyring */
- if (context->signal->session_keyring) {
+ if (sec->tgsec->session_keyring) {
rcu_read_lock();
key_ref = keyring_search_aux(
make_key_ref(rcu_dereference(
- context->signal->session_keyring),
+ sec->tgsec->session_keyring),
1),
- context, type, description, match);
+ sec, type, description, match);
rcu_read_unlock();
```

[PATCH 7/7] SECURITY: De-embed task security record from task and use refcounting

```
if (!IS_ERR(key_ref))
@@ -485,8 +461,8 @@ key_ref_t search_process_keyrings(struct key_type *type,
/* or search the user-session keyring */
else {
key_ref = keyring_search_aux(
- make_key_ref(context->sec->user->session_keyring, 1),
- context, type, description, match);
+ make_key_ref(sec->user->session_keyring, 1),
+ sec, type, description, match);
if (!IS_ERR(key_ref))
goto found;

@@ -507,20 +483,20 @@ key_ref_t search_process_keyrings(struct key_type *type,
* search the keyrings of the process mentioned there
* - we don't permit access to request_key auth keys via this method
*/
- if (context->sec->request_key_auth &&
- context == current &&
+ if (sec->request_key_auth &&
+ sec == current->sec &&
type != &key_type_request_key_auth
) {
/* defend against the auth key being revoked */
- down_read(&context->sec->request_key_auth->sem);
+ down_read(&sec->request_key_auth->sem);

- if (key_validate(context->sec->request_key_auth) == 0) {
- rka = context->sec->request_key_auth->payload.data;
+ if (key_validate(sec->request_key_auth) == 0) {
+ rka = sec->request_key_auth->payload.data;

key_ref = search_process_keyrings(type, description,
- match, rka->context);
+ match, rka->sec);

- up_read(&context->sec->request_key_auth->sem);
+ up_read(&sec->request_key_auth->sem);

if (!IS_ERR(key_ref))
goto found;
@@ -537,7 +513,7 @@ key_ref_t search_process_keyrings(struct key_type *type,
break;
}
} else {
- up_read(&context->sec->request_key_auth->sem);
+ up_read(&sec->request_key_auth->sem);
}
}

@@ -565,78 +541,78 @@ static int lookup_user_key_posessed(const struct key *key, const void *target)
```

[PATCH 7/7] SECURITY: De-embed task security record from task and use refcounting

```
* – don't create special keyrings unless so requested
* – partially constructed keys aren't found unless requested
*/
–key_ref_t lookup_user_key(struct task_struct *context, key_serial_t id,
+key_ref_t lookup_user_key(struct task_security *sec, key_serial_t id,
int create, int partial, key_perm_t perm)
{
key_ref_t key_ref, skey_ref;
struct key *key;
int ret;

– if (!context)
– context = current;
+ if (!sec)
+ sec = current->act_as;

key_ref = ERR_PTR(-ENOKEY);

switch (id) {
case KEY_SPEC_THREAD_KEYRING:
– if (!context->sec->thread_keyring) {
+ if (!sec->thread_keyring) {
if (!create)
goto error;

– ret = install_thread_keyring(context);
+ ret = install_thread_keyring(sec);
if (ret < 0) {
key = ERR_PTR(ret);
goto error;
}
}

– key = context->sec-g
```