Linux Security Modules: General Security Hooks for Linux

Stephen Smalley, NAI Labs <ssmalley@nai.com>
Timothy Fraser, NAI Labs <tfraser@nai.com>
Chris Vance, NAI Labs <cvance@nai.com>

Table of Contents

Introduction 1
LSM Framework 1
LSM Capabilities Module 3

Introduction

In March 2001, the National Security Agency (NSA) gave a presentation about Security-Enhanced Linux (SELinux) at the 2.5 Linux Kernel Summit. SELinux is an implementation of flexible and fine-grained nondiscretionary access controls in the Linux kernel, originally implemented as its own particular kernel patch. Several other security projects (e.g. RSBAC, Medusa) have also developed flexible access control architectures for the Linux kernel, and various projects have developed particular access control models for Linux (e.g. LIDS, DTE, SubDomain). Each project has developed and maintained its own kernel patch to support its security needs.

In response to the NSA presentation, Linus Torvalds made a set of remarks that described a security framework he would be willing to consider for inclusion in the mainstream Linux kernel. He described a general framework that would provide a set of security hooks to control operations on kernel objects and a set of opaque security fields in kernel data structures for maintaining security attributes. This framework could then be used by loadable kernel modules to implement any desired model of security. Linus also suggested the possibility of migrating the Linux capabilities code into such a module.

The Linux Security Modules (LSM) project was started by WireX to develop such a framework. LSM is a joint development effort by several security projects, including Immunix, SELinux, SGI and Janus, and several individuals, including Greg Kroah-Hartman and James Morris, to develop a Linux kernel patch that implements this framework. The patch is currently tracking the 2.4 series and is targeted for integration into the 2.5 development series. This technical report provides an overview of the framework and the example capabilities security module provided by the LSM kernel patch.

LSM Framework

The LSM kernel patch provides a general kernel framework to support security modules. In particular, the LSM framework is primarily focused on supporting access control modules, although future development is likely to address other security needs such as auditing. By itself, the framework does not provide any additional security; it merely provides the infrastructure to support security modules. The LSM kernel patch also moves most of the capabilities logic into an optional security module, with the system defaulting to the traditional superuser logic. This capabilities module is discussed further in the section called “LSM Capabilities Module”.

The LSM kernel patch adds security fields to kernel data structures and inserts calls to hook functions at critical points in the kernel code to manage the security fields and to perform access control. It also adds functions for registering and unregistering security modules, and adds a general `security` system call to support new system calls for security-aware applications.

The LSM security fields are simply void* pointers. For process and program execution security information, security fields were added to struct task_struct and struct linux_binprm. For filesystem security information, a security field was added to struct super_block. For pipe, file, and socket security information, security fields were added to struct inode and struct file. For packet and network device security information, security fields were added to struct sk_buff and struct net_device. For System V IPC security information, security fields were added to struct kern_ipc_perm and struct msg_msg; additionally, the definitions for struct msg_msg, struct msg_queue, and struct shmid_kernel were moved to header files (`include/linux/msg.h` and `include/linux/shm.h` as appropriate) to allow the security modules to use these definitions.

Each LSM hook is a function pointer in a global table, security_ops. This table is a security_operations structure as defined by `include/linux/security.h`. Detailed documentation for each hook is included in this header file. At present, this structure consists of a collection of substructures that group related hooks based on the kernel object (e.g. task, inode, file, sk_buff, etc) as well as some top-level hook function pointers for system operations. This structure is likely to be flattened in the future for performance. The placement of the hook calls in the kernel code is described by the "called:" lines in the per-hook documentation in the header file. The hook calls can also be easily found in the kernel code by looking for the string "security_ops->".

Linus mentioned per-process security hooks in his original remarks as a possible alternative to global security hooks. However, if LSM were to start from the perspective of per-process hooks, then the base framework would have to deal with how to handle operations that involve multiple processes (e.g. kill), since each process might have its own hook for controlling the operation. This would require a general mechanism for composing hooks in the base framework. Additionally, LSM would still need global hooks for operations that have no process context (e.g. network input operations). Consequently, LSM provides global security hooks, but a security module is free to implement per-process hooks (where that makes sense) by storing a security_ops table in each process' security field and then invoking these per-process hooks from the global hooks. The problem of composition is thus deferred to the module.

The global security_ops table is initialized to a set of hook functions provided by a dummy security module that provides traditional superuser logic. A `register_security` function (in `security/security.c`) is provided to allow a security module to set security_ops to refer to its own hook functions, and an `unregister_security` function is provided to revert security_ops to the dummy module hooks. This mechanism is used to set the primary security module, which is responsible for making the final decision for each hook.

LSM also provides a simple mechanism for stacking additional security modules with the primary security module. It defines `register_security` and `unregister_security` hooks in the security_operations structure and provides `mod_reg_security` and `mod_unreg_security` functions that invoke these hooks after performing some sanity checking. A security module can call these functions in order to stack with other modules. However, the actual details of how this stacking is handled are deferred to the module, which can implement these hooks in any way it wishes (including always returning an error if it does not wish to support stacking). In this manner, LSM again defers the problem of composition to the module.

Although the LSM hooks are organized into substructures based on kernel object, all of the hooks can be viewed as falling into two major categories: hooks that are used to manage the security fields and hooks that are used to perform access control. Examples of the first category of hooks include the `alloc_security` and `free_security` hooks defined for each kernel data structure that has a security field. These hooks are used to allocate and free security structures for kernel objects. The first

category of hooks also includes hooks that set information in the security field after allocation, such as the `post_lookup` hook in struct inode_security_ops. This hook is used to set security information for inodes after successful lookup operations. An example of the second category of hooks is the `permission` hook in struct inode_security_ops. This hook checks permission when accessing an inode.

LSM Capabilities Module

The LSM kernel patch moves most of the existing POSIX.1e capabilities logic into an optional security module stored in the file `security/capability.c`. This change allows users who do not want to use capabilities to omit this code entirely from their kernel, instead using the dummy module for traditional superuser logic or any other module that they desire. This change also allows the developers of the capabilities logic to maintain and enhance their code more freely, without needing to integrate patches back into the base kernel.

In addition to moving the capabilities logic, the LSM kernel patch could move the capability-related fields from the kernel data structures into the new security fields managed by the security modules. However, at present, the LSM kernel patch leaves the capability fields in the kernel data structures. In his original remarks, Linus suggested that this might be preferable so that other security modules can be easily stacked with the capabilities module without needing to chain multiple security structures on the security field. It also avoids imposing extra overhead on the capabilities module to manage the security fields. However, the LSM framework could certainly support such a move if it is determined to be desirable, with only a few additional changes described below.

At present, the capabilities logic for computing process capabilities on `execve` and `set*uid`, checking capabilities for a particular process, saving and checking capabilities for netlink messages, and handling the `capget` and `capset` system calls have been moved into the capabilities module. There are still a few locations in the base kernel where capability-related fields are directly examined or modified, but the current version of the LSM patch does allow a security module to completely replace the assignment and testing of capabilities. These few locations would need to be changed if the capability-related fields were moved into the security field. The following is a list of known locations that still perform such direct examination or modification of capability-related fields:

- `fs/open.c:sys_access`
- `fs/lockd/host.c:nlm_bind_host`
- `fs/nfsd/auth.c:nfsd_setuser`
- `fs/proc/array.c:task_cap`