Linux Security Modules: General Security Hooks for Linux

Author: Stephen Smalley
Author: Timothy Fraser
Author: Chris Vance

> **Note**
>
> The APIs described in this book are outdated.

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 was 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 work was incorporated in the mainstream in December of 2003. This technical report provides an overview of the framework and the capabilities security module.

LSM Framework

The LSM framework 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 sandboxing. By itself, the framework does not provide any additional security; it merely provides the infrastructure to support security modules. The LSM framework is optional, requiring *CONFIG_SECURITY* to be enabled. The capabilities logic is implemented as a security module. This capabilities module is discussed further in LSM Capabilities Module.

The LSM framework includes security fields in kernel data structures and 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 security modules. An interface */sys/kernel/security/lsm* reports a comma separated list of security modules that are active on the system.

The LSM security fields are simply `void*` pointers. The data is referred to as a blob, which may be managed by the framework or by the individual security modules that use it. Security blobs that are used by more than one security module are typically managed by the framework. For process and program execution security information, security fields are included in :c:type:`struct task_struct <task_struct>` and :c:type:`struct cred <cred>`. For filesystem security information, a security field is included in :c:type:`struct super_block <super_block>`. For pipe, file, and socket security information, security fields are included in :c:type:`struct inode <inode>` and :c:type:`struct file <file>`. For System V IPC security information, security fields were added to :c:type:`struct kern_ipc_perm <kern_ipc_perm>` and :c:type:`struct msg_msg <msg_msg>`; additionally, the definitions for :c:type:`struct msg_msg <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.

> **System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\security\[linux-master][Documentation][security]lsm.rst`, line 67); *backlink***
>
> Unknown interpreted text role "c:type".

> **System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\security\[linux-master][Documentation][security]lsm.rst`, line 67); *backlink***
>
> Unknown interpreted text role "c:type".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\security\[linux-master][Documentation][security]lsm.rst`, line 67); *backlink*

Unknown interpreted text role "c:type".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\security\[linux-master][Documentation][security]lsm.rst`, line 67); *backlink*

Unknown interpreted text role "c:type".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\security\[linux-master][Documentation][security]lsm.rst`, line 67); *backlink*

Unknown interpreted text role "c:type".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\security\[linux-master][Documentation][security]lsm.rst`, line 67); *backlink*

Unknown interpreted text role "c:type".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\security\[linux-master][Documentation][security]lsm.rst`, line 67); *backlink*

Unknown interpreted text role "c:type".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\security\[linux-master][Documentation][security]lsm.rst`, line 67); *backlink*

Unknown interpreted text role "c:type".

For packet and network device security information, security fields were added to :c:type:`struct sk_buff <sk_buff>` and :c:type:`struct scm_cookie <scm_cookie>`. Unlike the other security module data, the data used here is a 32-bit integer. The security modules are required to map or otherwise associate these values with real security attributes.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\security\[linux-master][Documentation][security]lsm.rst`, line 90); *backlink*

Unknown interpreted text role "c:type".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\security\[linux-master][Documentation][security]lsm.rst`, line 90); *backlink*

Unknown interpreted text role "c:type".

LSM hooks are maintained in lists. A list is maintained for each hook, and the hooks are called in the order specified by CONFIG_LSM. Detailed documentation for each hook is included in the *include/linux/lsm_hooks.h* header file.

The LSM framework provides for a close approximation of general security module stacking. It defines security_add_hooks() to which each security module passes a :c:type:`struct security_hooks_list <security_hooks_list>`, which are added to the lists. The LSM framework does not provide a mechanism for removing hooks that have been registered. The SELinux security module has implemented a way to remove itself, however the feature has been deprecated.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\security\[linux-master][Documentation][security]lsm.rst`, line 103); *backlink*

Unknown interpreted text role "c:type".

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 security_inode_alloc() and security_inode_free() These hooks are used to allocate and free security structures for inode objects. An example of the second category of hooks is the security_inode_permission() hook. This hook checks permission when accessing an inode.

LSM Capabilities Module

The POSIX.1e capabilities logic is maintained as a security module stored in the file `security/commoncap.c`. The capabilities module uses the order field of the :c:type:`lsm_info` description to identify it as the first security module to be registered. The capabilities security module does not use the general security blobs, unlike other modules. The reasons are historical and are based on overhead, complexity and performance concerns.

> **System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\security\[linux-master][Documentation][security]lsm.rst`, line 125); *backlink***
>
> Unknown interpreted text role "c:type".