Secure computing: SELinux

Michael Wikberg
Helsinki University of Technology
Michael.Wikberg@wikberg.fi

Abstract

Using mandatory access control greatly increases the security of an operating system. SELinux, which is an implementation of Linux Security Modules (LSM), implements several measures to prevent unauthorized system usage. The security architecture used is named Flask, and provides a clean separation of security policy and enforcement. This paper is an overview of the Flask architecture and the implementation in Linux.

KEYWORDS: SELinux, MAC, Security, Kernel, Linux, LSM, TE, RBAC, MLS

1 Introduction

Security is a very broad concept, and so is the security of a system. All too often, people believe that a system is way more secure that it in practice is, but the biggest problems is still the human factor of the users; the possibility of careless or malicious users are commonly overlooked.

The standard "Unix way" of providing authentication and authorization is not very well suited for more complicated and dynamic environments. The available Discretionary Access Control mechanisms gives the same rights to all users in a certain group, and all processes created by a user have exactly the same privileges. The acquired permissions can also be transferred to other subjects, and so a flaw in one software can lead to all the users' data being compromised. Some system wide permission restrictions can be enforced by using several user groups, limiting the use of certain software to users belonging to that group, but each user process still ahve all the permissions of all the groups that the owning user belongs to[4].

Government agencies, among other similar organizations, need a more advanced way of defining system security policies. That is why the National Security Agency (NSA) began developing, for internal use, their own set of patches to the Linux kernel. These patches became known as Security-Enhanced Linux (SELinux) and was later released under the GPL and included in the main Linux kernel tree.

Nowadays SELinux is a security module for the Linux Security Modules framework. This paper is an overview of the SELinux features and how it changes the security configuration aspect of Linux. The architecture that SELinux implements, the Flask architecture, and its components are described in section 3.1. A brief description of the SELinux LSM module is given in section 3.2. Section 4 describes some of the other projects that aim to increase Linux security.

This paper is intended as a brief overview of the technologies used, and thus does not contain very detailed information. There is plenty of in-depth documentation of the internals in the sources of this paper[3][10][6] and on the SELinux web page[5] among others.

2 Problem statement

Real security cannot be provided in user-space only. The need for security mechanisms in the operating system itself is evident as indicated in this section.

The access control mechanisms used in most operating systems (usually Discretionary Access Control) are not capable of providing strong enough system security. One of the obstacles for creating really secure systems, is that there is not a single security architecture that could satisfy nearly all the different security needs.

Malicious code, that manages to bypass the application level security, will usually be executed with the same permissions the current user has. This means that all the users' applications and data is compromised at once, and in case the user is an administrator, the whole system is compromised. It is not possible to limit the resources different programs can access, so a web browser can access the files belonging to the mail program, and also all other, possibly sensitive, files the user has access to.

Malicious or careless users might also leak sensitive data unless the rules for handling such data are not enforced by the system. Even very experienced and careful users might be using flawed programs that could leak information[4].

From the system point of view, there is really no point in making a distinction between malicious or flawed programs and hostile users. Proper security mechanisms would handle them in a similar manner anyway.

2.1 Sandboxes and signatures

One attempt to minimize the effects of malicious code is running code in a so called sandbox. The sandbox is a virtual environment set up by the code interpreter, such as the Java Virtual Machine, or Adobe Flash. This model relies entirely on the virtual machine implementation to be secure, and so a flaw in the virtual machine might grant the malicious code access to the host system with the privileges of the virtual machine; often administrative or even root.

Another way of trying to secure programs is to use code signing and allowing applications from trusted sources only. One problem with this solution is the cumbersomeness of

getting even the smallest applets signed by a trusted party. Another problem is that the signed code might be getting way more privileges than actually needed or desired. The signature verification mechanisms and key storage must also be protected against tampering for this method to have any effect.

For any of the solutions mentioned above to be effective, a secure operating system is needed to provide the basic primitives needed to secure them.

2.2 Data links

Data transmission between systems also needs to be secured. The current solutions such as IPSec and SSL only provide a partial solution, since they run in user space and therefore cannot provide a complete end-to-end secure channel. A compromise of the unsecured data on one of the endpoints renders the whole channel useless.

3 SELinux

SELinux started as a security research project at NSA, together with Secure Computing Corporation and the University of Utah, to demonstrate the benefits of mandatory access control over the user/group schema. Today SELinux is included in the mainstream Linux kernel as a security module in the LSM framework.

> SELinux implements a flexible mandatory access control (MAC) architecture in the major subsystems of the kernel and provides a mechanism to enforce the separation of information based on confidentiality and integrity requirements [5].

3.1 Basic architecture

NSA tried to get their SELinux patches included in the 2.5 development branch kernel back in 2001, but Linus Torvalds rejected the proposal since there were other similar ongoing projects at the same time. A more general solution was needed so that the kernel would be able to support as many security architectures and implementations as possible, without sticking too much to the ideas of any specific implementation.

3.1.1 Linux security modules

To support various security models, an interface "*Linux Security Module Interface*" was proposed[3] by Crispin Cowan. The Linux Security Modules framework[10] development got contributions from huge corporations, such as IBM and SGI, and naturally NSA.

In 2006, the only widely used LSM module included in the mainstream kernel was SELinux, but Torvalds still wanted to keep the door open for other implementations[1] and so SELinux was finally included in the mainstream 2.6 kernel as a security module in late 2003.

[1] See related work, section 4

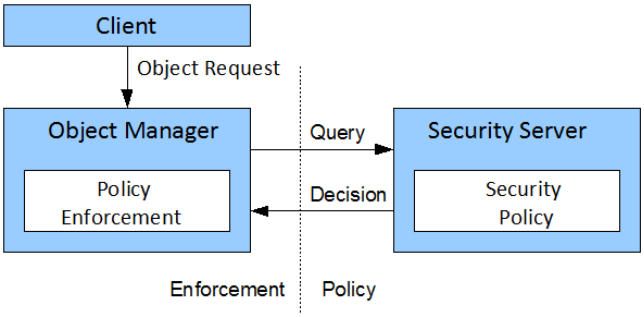

Figure 1: The Flask architecture

3.1.2 Flask architecture and concepts

The very flexible MAC architecture used in SELinux is Flask[8], which was derived from a micro-kernel based operating system named Fluke. Flask clearly separates the security policy from the enforcement mechanism (Fig. 1).

All *subjects* (processes) and *objects* (files, sockets, ...) have a set of security attributes, referred to as the *security context* of the object. The attributes depend on the specific security policy in use, but generally contain the user id, a role and type enforcement domain.

Instead of working with the security context all the time, the security server maintains a mapping between security attribute sets and security identifiers (SIDs). When the *object manager*, the enforcing component, request labeling or access decisions, it typically passes a pair of SIDs to the security server, which looks up the related security contexts and makes the decision based on the policy in use.

Polyinstantiation is used when a certain resource needs to be shared by many clients. Such a resource could be the /tmp directory or the TCP port space. Filenames or port numbers might disclose some information about the owning process, and shared directories are subject to race-condition attacks. With polyinstantiation, each user can only see his or her own version of the resource based on username and/or security context.

The security server exists to provide policy decisions, map security contexts to SIDs, provide new SIDs and manage the Access Vector Caches (AVC), which is presented in section 3.2.1. It usually also provides means for loading policies and it keeps track of which subjects can access its services.

3.1.3 Type Enforcement

The SELinux Type Enforcement (TE) model differs slightly from traditional models; by using the security class information provided by the Flask architecture and using a single type attribute for both processes and objects. This effectively means that a single matrix is used to specify the access to and interaction between different types, and objects of the same type can be treated differently if their associated security classes differ. Users are not directly bound to security types, but instead RBAC, which is presented in section 3.1.4, is used.

Process transition rules are based on the current process domain, while types created through object transition rules

are based on the creating process domain[2] (the security type of the process identifier), the object security class and the type of the related object (e.g. parent directory for files). A process cannot change its domain during execution.

Transition rules and access vectors have default policies for cases where a rule is not explicitly defined; Allowed transitions are not audited unless defined by a auditallow rule, transitions are allowed only if explicitly allow rule and denied transitions are audited. All kinds of access vectors can have rules, including allow, auditallow, dontaudit etc.

3.1.4 Role-Based Access Control

Role-based access control (RBAC) is used to define a set of roles that can be assigned to users. It is a more flexible model than standard DAC or MAC, and can simulate both of them using suitable rules. SELinux further extends the RBAC model to restrict roles to specified TE domains, and roles can be arranged in a priority hierarchy. Restricting roles to certain security domains allows most of the security decisions to be made through the TE configuration. The security context of a process contains a role attribute and also, while they are not actually applied, to objects.

Role transitions are usually limited to a few TE domains to limit transitions to defined programs and users that need the ability, thus reducing the impact of malicious code being executed.

3.1.5 MLS

While type enforcement is the most important provider of mandatory access control, there might sometimes be a need for traditional multilevel security (MLS). SELinux optionally provides MLS abilities, which allows defining a hierarchical "sensitivity" level and categories to objects and subjects (processes). Subjects and objects can have a range of security levels (e.g. directories might contain files with different security levels and some "trusted processes" might need to downgrade information) defined when needed, but usually only one level is used.

Any MLS defined constraints are enforced in addition to the TE policy, which means that checks must pass both of them for access to be granted.

3.1.6 User Identity

The "Unix" way of representing user identities using UIDs and GIDs is insufficient for SELinux, since changing a user role (e.g. su) involves changing the UID, which means that the actions following are actually performed as the other user and not just as the same user in another role. This makes auditing and accounting very difficult. SELinux user identity attribute is persistent in the security context, which is independent of the current UID. This means that SELinux policies can be enforced without affecting compatibility with Linux DAC permissions.

Only a limited number of programs, like login, sshd and cron, need the ability to change the User Identity, so it is usually restricted to their respective TE domains. Depending on security configuration, the programs may or may not be able to change the user identity more than once. Allowing programs started from cron to change their user identity, for example, impacts accountability.

[2]The security server does not internally distinguish domains from types. Traditional TE models use domains for processes and types for objects but SELinux does not. Hence two names for the same thing.

3.2 SELinux LSM Module

SELinux uses the LSM framework to accomplish its mission. The framework adds security fields to kernel data structures and calls to hook functions in critical points (kernel calls), to manage the security fields and perform access control. The most commonly used filesystems have been updated to support the file security attributes[3]. The hook calls are initialized to a dummy module that emulates traditional "Unix" superuser logic, and each security module, at load-time, registers the hooks it uses.

3.2.1 Internal Architecture

The SELinux module has six major components; the security server, the access vector cache, the network interface table, the netlink event notification code, the selinuxfs pseudo filesystem and the hook function implementations.

The default security server implements a combination of the Flask architecture components (TE, RBAC and optionally MLS), but it can be changed or replaced without affecting the rest of the SELinux module.

The AVC provides caching of the decisions provided by the security server to minimize overhead in hook function calls and also provides an interface to the security server for managing the cache, like propagating policy updates.

One of the drawbacks of the LSM framework is that it does not provide a security field for network devices, therefore a separate mapping to security contexts, the network interface table, is needed. Network interfaces are added and removed automatically, and there are callback functions defined for device configuration or policy changes.

The netlink event notification code is used to keep the userspace AVC in sync with the kernel one, which enables the use of user-space enforcers, like security-enhanced X.

The selinuxfs pseudo filesystem provides the security server API to processes and provides the lowlevel support for policy manipulation calls.

The hook functions are responsible for retrieving security information from the security server and AVC, and for enforcing the policies. They also maintain the security context of files.

The SELinux module provides only rudimentary support for stacking with other LSM modules, but there is ongoing work to improve the stacking support.

3.3 What is new

SELinux is constantly evolving and expanding. Some of the latest key extensions and feature improvements[7] are presented here.

[3]Not to be confused with ext2/ext3 extended attributes (.e.g. Immutable,Undelete etc)

- SELinux Loadable policy modules: An attempt to ease policy maintenance, reduce resource requirements by removing the need for a policy development environment and allow for loosely coupled policies. Local customizations can be made to more generic or even 3rd party policies, and the policy source need not be available.
- The Reference Policy[6] aims to be a baseline security policy, on which custom policies are easy to build. The policy is based on a single source and is clean and modular. It was originally based on the NSA example policy, which is somewhat difficult to comprehend without a good understanding of the underlying SELinux technologies, but has since evolved quite a bit.
- Policy Management Interface: In addition to the Policy Modules, a new policy server is being developed. It facilitates fine-grained control over who can change the policy and how (currently any process that can change the policy, can change everything), User-space Object Managers to extend TE to user-space (e.g. the Security Enhanced X-Window system). Network wide deployment and management of security policies will also be possible through the use of security modules. A standard library for policy management (libsemanage) is introduced for policy tools to use.
- Enhanced Audit Support: Syscall audit records have been extended with security contexts, audit records can be filtered based on security context and auditing of several SELinux specific events has been added.
- Enhanced Multi Level Security Support (labeled networking, application integration), end-to-end network data labeling and traffic control (including policy-based packet filtering). Separation of network data labeling from enforcement; iptables is used for data labeling and SELinux for enforcing the policies.
- Securing the desktop: The X Access Control Extension (XACE) framework is integrated into the Xorg server as of version 1.2. It is a general framework for X, much like the LSM kernel framework but for user-space. A Flask policy module, XSELinux, that will provide flexible MAC is also under development.
- Troubleshooting and reporting has been improved, and several tools for security policy generation and management have been added or updated.

4 Related work

SELinux is not the only security project going on. This chapter aims to give a short introduction to related projects.

4.1 TrustedBSD

TrustedBSD consists of several branches; Access Control Lists, Event Auditing and OpenBSM, Extended Attributes and UFS2, Fine-Grained Capabilities, GEOM, Mandatory Access Control and OpenPAM[9].

The MAC Framework provides discretionary access control for all subjects and objects. Security-Enhanced BSD (SEBSD) is a port of the FLASK/TE implementation from SELinux, that works as a module in the TrustedBSD MAC framework, but is not yet ready for production use due to lacking support in vital userspace applications.

The features included in the base FreeBSD distribution are Access Control Lists (basically an implementation of the POSIX.1eD17 draft specification), Security Event Auditing (based on Sun's Basic Security Module API), UFS2 (default file system which supports file tagging), GEOM (provides data transformation services between the kernel and I/O devices), OpenPAM (similar to Linux PAM, but more compatible with the Solaris PAM implementation).

4.2 AppArmor

AppArmor is a Linux Security Module, which relies on application profiles for decision making[1]. It provides Mandatory Access Control for file paths, instead of by inode. AppArmor was mostly maintained by Novell from 2005 to September 2007 as part of their SUSE distributions, but is now available under the GPL in other distributions also.

4.3 LIDS

The LIDS (Linux Intrusion Detection System) kernel patch version 1 was designed for Linux kernel version 2.2 and later 2.4. Version 2 will be turned into a module for the LSM framework, but is currently work in progress. LIDS includes several measures to minimize damage in case of a system compromise. Files can be marked as immutable, and changes to vital system configurations (like network setting and loaded modules) can be prevented. Even the root user cannot circumvent the protection without first stopping the LIDS system, which requires a password which is set at compile time.

4.4 Trusted Solaris

Solaris 10 has a component called Trusted Solaris Extensions, which is the successor of the former Trusted Solaris release. Trusted Solaris includes accounting, auditing, RBAC and Mandatory Access Control Labeling. MAC is enforced in every aspect of the OS by adding sensitivity labels to objects, allowing only explicitly authorized users or applications to access the objects.

4.5 GRSecurity

One major component of GRSecurity is PaX[2]. Memory page access is guarded by least privilege protection, the program memory is randomly arranged, and pages can be flagged as non-executable or non-writable. PaX reduces the impact of buffer overruns and other possible arbitrary code execution attacks to a mere Denial of Service attack, causing only the affected program to crash.

GRSecurity also provides a full Role-Based Access Control system, advanced auditing of specified user groups, and chroot jail hardening. Access to certain programs like dmesg

and netstat is usually limited to root only, and Trusted Path Execution can optionally prohibit users from executing non-root owned binaries, effectively eliminating trojans and other malicious code from being run.

5 Conclusions

SELinux provides a much more fine-grained control over the security of a Linux system compared to the "Unix" standard. The baseline security configuration is almost usable for most environments, but some configuration is needed in most cases. The difficulty of configuration has maybe been the reason why most people have not taken SELinux in use, but the policy management tools are getting better.

Some people claim[4] that the security framework provided by LSM is not extensive enough, that several critical security hooks are missing and that SELinux security relies on the kernel being bug free. These claims are probably at least partially true, but the latest development in SELinux tries to address the remaining security issues (e.g. minimize the processed that can change the policy etc).

Complete system security is an utopia, but SELinux is one step in that direction. It is being included in most major Linux distributions, even though it might not be enabled by default. Installing or activating SELinux is pretty straightforward, and no enforcement is being done until the user has checked the logfiles for possible problems and decides that the configuration is good enough.

References

[1] AppArmor webpage. `http://en.opensuse.org/AppArmor`.

[2] The PaX team, PaX webpage. `http://pax.grsecurity.net/`.

[3] C. Cowan. Linux Security Module Interface, 2001. Linux Kernel Mailing List `http://marc.info/?l=linux-kernel&m=98695004126478&w=2`.

[4] P. A. Loscocco, S. D. Smalley, P. A. Muckelbauer, R. C. Taylor, S. J. Turner, and J. F. Farrell. The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments. In *21st National Information Systems Security Conference*, pages 303–314. NSA, 1998.

[5] NSA. Nsa selinux webpage. `http://www.nsa.gov/selinux/`.

[6] C. J. PeBenito, F. Mayer, and K. MacMillan. Reference Policy for Security Enhanced Linux. In *SELinux Symposium*, 2006.

[7] S. D. Smalley. What's New With SELinux presentation at SELinux Symposium `http://www.nsa.gov/selinux/papers/whatsnew07-abs.cfm`, 2007.

[8] R. Spencer, S. Smalley, P. Loscocco, M. Hibler, D. Andersen, and J. Lepreau. The Flask Security Architecture: System Support for Diverse Security Policies. In *The Eighth USENIX Security Symposium*, pages 123–139, August 1999.

[9] R. Watson. TrustedBSD webpage. `http://www.trustedbsd.org/`.

[10] C. Wright, C. Cowan, J. Morris, S. Smalley, and G. Kroah-Hartman. Linux Security Modules: General security support for the Linux kernel. In *USENIX Security*, 2002.

[4] GRSecurity authors and also others