LINUX SECURITY PRIMER: SELINUX AND SMACK FRAMEWORKS

KATHY TUFTO, PRODUCT MANAGER

WHITEPAPER

Mentor®
A Siemens Business

EMBEDDED SYSTEMS

www.mentor.com

INTRODUCTION

With the proliferation of smart devices and the Internet of Things (IoT), securing Internet-connected devices has never been more important. Developers from small and large companies are increasingly concerned with protecting their devices from malicious attacks. The research firm, Gartner, has predicted that over 8.4 billion Internet-connected "things" will be in use worldwide in 2017, up 31 percent from last year. By 2020, Gartner expects 20.8 billion connected devices.

Further, high profile attacks on major corporate and government computer systems have heightened the public awareness of computer system security. Many of these highly publicized cyber attacks have compromised the personal information of millions of consumers, resulting in the re-issue of millions of credit cards whose numbers had been stolen by the cyber criminals behind these attacks. Software vulnerabilities with names such as Rex, Mirai, Heartbleed, and Shellshock have become familiar terms even outside of computer circles. For these and other reasons, building secure Internet-connected devices has never been more important.

Building a secure system involves many components and layers of security. From physical security to protection from cloud-based threats, every aspect of a system must be evaluated and protected. Linux® offers several frameworks for protecting the operating system and associated components. Here, we'll examine two popular Linux frameworks, SELinux and SMACK.

Security-Enhanced Linux (SELinux) was first integrated into the open source Linux kernel (Release 2.6) in 2003. Simple Mandatory Access Control Kernel (SMACK) is the newcomer to Linux security frameworks, and has found traction in embedded devices because it is more compact, and easier to administer and configure. Both SELinux and SMACK are mechanisms to protect operating system resources from unauthorized access using a mechanism called Mandatory Access Control (MAC).

DISCRETIONARY ACCESS CONTROL

To understand the difference between the two modes of access control, let's consider first how Discretionary Access Control (DAC) behaves. Standard Linux systems use a set of access attributes that are part of every file system resource. These attributes govern the access permissions for a given file system resource. These permissions include "Read", "Write", and "eXecute", or (RWX).

Attributes exist in three categories of system user: the User, Groups, and Other. The *User* category refers to a single individual login account on a system. Linux and UNIX systems contain many *Groups* to help partition access to system resources. For example, user Judy might belong to two groups "Marketing" and "Administrators." The *Other* category refers to any other users on the system beyond the current user or any defined groups.

DAC grants the owner of the resource the authority to decide who gets access to those resources. It is suitable for protection from accidental access violations. DAC policy centers on users. DAC security policy answers these questions:

1) What file system resources can this user read, write, or execute? (User)

2) What file system resources can this group of users read, write, or execute? (Group)

3) What file system resources on the system can everyone else read, write, or execute? (Other)

4) Who can execute programs (also file system resources) on this system? (Execute permissions)

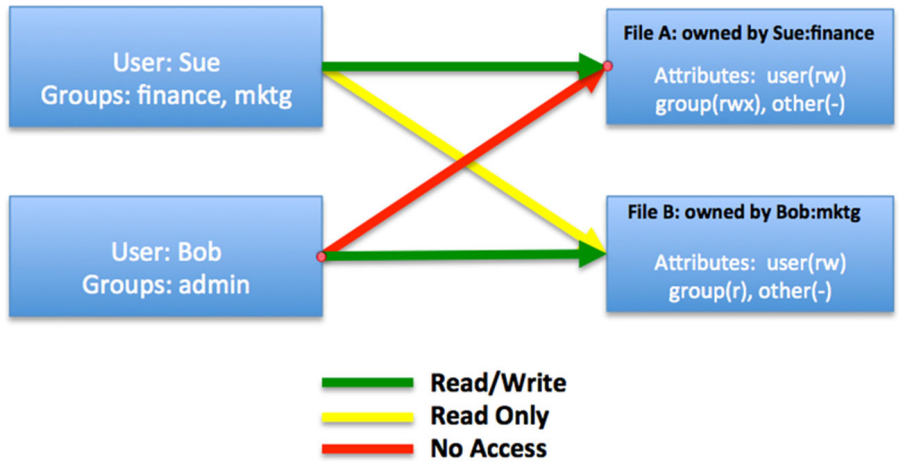

Figure 1: *A simple scenario using DAC.*

Figure 1 illustrates two users on the system, Sue and Bob, who have varying access permissions to a pair of file system resources. File A is owned by Sue and belongs to group "Finance" and has Read (R) and Write (W) attributes for user Sue. Therefore, Sue has Read/Write access to this file. Because File A has no access (-) for "Other," and because user Bob is not in the "Finance" group, he has no access to this file.

Similarly for File B owned by Bob, Sue can Read (R), but not Write (W) to File B, because Sue belongs to "Marketing" and the group attribute for File B is set to Read (R). Of course, each can Read (R) and Write (W) to their own respective files because user attributes are set to Read/Write in each case.

The discussion around Figure 1 is a classic example of DAC and is used in virtually all Linux systems. DAC policy permits users to modify system policy – it's where the term "discretionary" comes from.

MANDATORY ACCESS CONTROL

Policy rules for Linux systems based on Mandatory Access Control (MAC) are centrally controlled by the operating system, and cannot be modified by ordinary system users. A policy developer controls which programs or processes can perform specific operations on system resources. Users cannot modify access permissions either accidentally or intentionally. MAC policy centers on programs rather than users, answering the question "What can this program do or not do?" Users execute programs on their behalf on behalf of the system.

By contrast, DAC systems govern access primarily to file system resources, by attaching permission attributes to a file, such as Read, Write and Execute. SELinux has a much finer-grained control governing access rights to resources beyond the file system to include memory, I/O, sockets, IPC elements, and more.

SECURITY-ENHANCED LINUX (SELINUX)

Security-Enhanced Linux (SELinux) is a framework and set of tools originally developed by the United States National Security Agency (NSA) used to harden Linux systems against potential threats. These threats can include deliberate attacks, misuse, or software vulnerabilities including viruses and malware. SELinux was originally integrated into the mainline Linux kernel over a decade ago, in the early days of the Linux 2.6 kernel. While no framework can protect against certain software bugs, SELinux has the potential to make a system

much more robust and far less vulnerable to external threats including viruses and malware. SELinux is an important tool in the arsenal of security analysts, and is used as a key component of an overall system security strategy.

SELinux by default allows no access. Rules must be created and loaded into the operating system to specify allowable access rights. Using SELinux, all accesses must be explicitly granted. The collection of rules is called the SELinux Policy.

Unlike DAC permission attributes, SELinux uses access control attributes attached to each file system called a security context. The most commonly used context fields include user, role, and type, usually written as user:role:type. Every object (processes, files, etc.) has a security context, especially and most notably processes and files. Policy rules allow a process in one context to perform operations on an object in another context.

SELinux contexts for programs might look like these:

```
/sbin/dhclient -> system _ u:object _ r:dhcpc _ exec _ t:s0

/sbin/fdisk    -> system _ u:object _ r:bin _ t:s0

/etc/passwd    -> system _ u:object _ r:etc _ t:s0
```

The `s0` is called a level and is used for multi-level and other types of security policies, and often ignored in policy rules. All three of these binary programs are labeled as *user:system_u* and *role:object_r*. Each has its own *type* based on its use in the system so that rules can be constructed separately for each program.

Policy rules must be written to allow access to every object on the system. *Allow* rules specify a subject and an object and the permissions granted for a specific access. *Allow* rules have four elements: source_type(s), target_types(s), object_class(es), and permission(s). A policy *Allow* rule might look like this:

```
allow user _ t bin _ t : file {read execute getattr};
```

This rule specifies that a process with domain type *user_t* has read, execute, and "get attributes" (stat) permission on a file object with type *bin_t*. (*Linux has a getattr()* system call that allows a process to get a file's metadata without actually reading it.)

SELinux comes with utilities that help manage its configuration. For example, an audit tool logs every system access that is denied. Another tool analyzes the audit log and creates an access rule based on the log entry of the denial. Yet another utility can append that rule into the policy to make it permanent. This makes it easy to detect and fix access issues in SELinux systems. Furthermore, a developer need not create rules from scratch. Several reference policies exist which range from minimal to comprehensive that form a good basis from which to build your own custom SELinux policy.

The power of SELinux comes from its granularity. SELinux controls many kernel resources, beyond the DAC model of only controlling file permissions. Because of this granularity, SELinux policies for even simple systems contain hundreds or even thousands of rules. It is well understood that SELinux has both a learning curve and significant administrative burden.

SIMPLIFIED MAC KERNEL (SMACK)

The complexities of SELinux gave birth to Simplified MAC Kernel (SMACK.) SMACK uses the same underlying kernel infrastructure as SELinux, but reduces the granularity to make system development, configuration, and administration easier. It's still a MAC system, meaning it is governed by a central policy and not by system users. According to its designers, simplicity was its primary design goal.

SMACK consists of three components: a SMACK-enabled kernel, SMACK utilities, and the configuration data policy. SMACK is based on labels attached to objects. The only operation ever done on a label is to test for equality. Every task on a SMACK-based system is assigned a label. Special labels are assigned to system tasks such as *init*. Several special labels have specific meaning, such as the asterisk as a wild card character.

SMACK uses traditional UNIX/Linux access permissions such as read, write, and execute. The SMACK rules are very straight-forward:

- Any access requested by a task labeled "*" is denied
- A read or execute access requested by a task labeled "^" is permitted
- A read or execute access requested on an object labeled "_" is permitted
- Any access requested on an object labeled "*" is permitted
- Any access requested by a task on an object with the same label is permitted
- Any access requested that is explicitly defined in the loaded rule set is permitted
- Any other access is denied

SMACK rules have the format:

```
[subject-label]   [object-label]   [access]
```

The access string uses characters familiar to Linux sysadmins: r, w, x, for read, write, execute, etc.

Subject	Object	Access
TopSecret	Secret	rx
Secret	Unclass	r
Manager	Game	x
User	HR	rw
New	Old	r
Close	Off	-

Figure 2: *Rules on a typical SMACK system.*

Examining Figure 2, we see that any subject labeled "TopSecret" wishing to perform an operation on an object labeled "Secret" can do so with read and execute permissions. Similarly, a subject labeled "Manager" can execute his favorite object, so long as that object has label "Game." Subjects labeled "Close" have no access to objects labeled "Off." SMACK is much easier to configure, but unlike the more powerful SELinux, can only moderate access to file system objects.

SUMMARY

SELinux is complex, and designed for enterprise-level security. Even a simple system requires thousands of individual rules to operate. SELinux permissions are much more granular, and therefore can protect system resources such as memory, I/O, sockets, and more. SELinux is mature and has good tools for monitoring and control, troubleshooting, policy examination, and maintenance.

SMACK was designed with embedded systems in mind and for securing Internet-connected devices. As a result, it is easier to configure and maintain. Please note that SMACK tools are not as rich as those available for use with SELinux. Permissions are not as granular.

Whatever framework you chose, remember that access control is only one component of a secure system. Security for embedded Linux devices must be considered at design time, and can be challenging to get right. One thing is certain: The importance of securing your devices from malicious attacks can no longer be an afterthought.

The SMACK capability discussed in this paper is offered as part of the Mentor® Embedded Linux® Security add-on.

Please visit the Mentor Embedded Linux website for more information on security or for more general information on Mentor's Linux offering.

Author's biography

Kathy Tufto is the product manager at Mentor's Embedded Systems Division responsible for Mentor Embedded Linux and Mentor Embedded Sourcery CodeBench. Before joining Mentor, Kathy worked at The MathWorks as a senior training engineer and senior course developer where she taught and developed courses in the area of multi-domain simulation, model-based design, and embedded code generation for dynamic and embedded systems. Kathy holds an EE degree from Boston University.

Corporate Headquarters
Mentor Graphics Corporation
8005 SW Boeckman Road
Wilsonville, OR 97070-7777
Phone: 503.685.7000
Fax: 503.685.1204

Sales and Product Information
Phone: 800.547.3000
sales_info@mentor.com

Silicon Valley
Mentor Graphics Corporation
46871 Bayside Parkway
Fremont, CA 94538 USA
Phone: 510.354.7400
Fax: 510.354.7467

North American Support Center
Phone: 800.547.4303

Europe
Mentor Graphics
Deutschland GmbH
Arnulfstrasse 201
80634 Munich
Germany
Phone: +49.89.57096.0
Fax: +49.89.57096.400

Pacific Rim
Mentor Graphics (Taiwan)
11F, No. 120, Section 2,
Gongdao 5th Road
HsinChu City 300,
Taiwan, ROC
Phone: 886.3.513.1000
Fax: 886.3.573.4734

Japan
Mentor Graphics Japan Co., Ltd.
Gotenyama Trust Tower
7-35, Kita-Shinagawa 4-chome
Shinagawa-Ku, Tokyo 140-0001
Japan
Phone: +81.3.5488.3033
Fax: +81.3.5488.3004

Mentor® A Siemens Business

MGC VM17 TECH15490-W