ISSN:0975-9646

SELinux Integration into Linux Kernel.

Vinay Karajagi ,Meenakshi Garg
VES Institute of Technology (VESIT), Chembur Mumbai:-74, Maharahstra, India

Abstract: The protection mechanisms of many mainstream operating systems were inadequate to support confidentiality and integrity requirements for end systems. To address this problem, the National Security Agency (NSA) worked with Secure Computing Corporation (SCC) to develop a strong, flexible mandatory access control architecture based on Type Enforcement. The NSA has integrated the SELinux architecture into the Linux operating system to transfer the technology to a larger developer and user community. This paper presents the design and implementation for integrating the security mechanisms of the SELinux architecture into the Linux kernel.

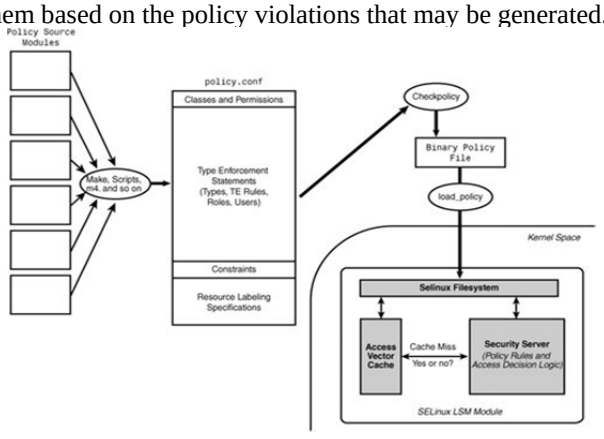
1. INTRODUCTION:

The Linux kernel is a Unix-like computer operating system kernel. The Linux kernel is a widely used operating system kernel world-wide, the Linux operating system is based on it and deployed on both traditional computer systems, usually in the form of Linux distributions, and on embedded devices such as routers. The Android operating system for tablet computers and smartphones is also based atop the Linux kernel. The Linux kernel was initially conceived and created in 1991 by Finnish computer science student Linus Torvalds, for his personal computer and with no cross-platform intentions, but has since expanded to support a huge array of computer architectures, many more than other operating systems or kernels. Linux rapidly attracted developers and users who adapted code from other free software projects for use with the new operating system. The Linux kernel has received contributions from nearly 12,000 programmers from more than 1,200 companies, including some of the largest software and hardware vendors.

Security-Enhanced Linux (SELinux) is a Linux kernel security module that provides a mechanism for supporting access control security policies, including United States Department of Defense style mandatory access controls (MAC). SELinux is a set of kernel modifications and user-space tools that have been added to various Linux distributions. Its architecture strives to separate enforcement of security decisions from the security policy itself and streamlines the volume of software charged with security policy enforcement. The key concepts underlying SELinux can be traced to several earlier projects by the United States National Security Agency.

We can further progress with learning the integration strategy for SELinux. The integration style greatly depends on the end system core structure. End systems must be able to enforce the separation of information based on confidentiality and integrity requirements to provide system security. Operating system security mechanisms are the foundation for ensuring such separation. Unfortunately, existing mainstream operating systems lack the critical security feature required for enforcing separation: mandatory access control. As a consequence, application security mechanisms are vulnerable to tampering and bypass, and malicious or flawed applications can easily cause failures in system security. To address this problem, the National Security Agency (NSA) worked with Secure Computing Corporation (SCC) to research a strong, flexible mandatory access control architecture based on Type Enforcement, a mechanism first developed for the LOCK system . The NSA and SCC developed two Mach-based prototypes of the architecture: DTMach and DTOS. The NSA and SCC then worked with the University of Utah's Flux research group to transfer the architecture to the Fluke research operating system. During the transfer, the architecture was enhanced to provide better support for dynamic security policies. This enhanced architecture was named SELinux. The NSA is now integrating the SELinux architecture into the Linux operating system to transfer the technology to a larger developer and user community. Researchers in the NSA's Information Assurance Research Office have implemented the architecture in the major subsystems of the Linux kernel, including mandatory access controls for operations on processes, files, and sockets. The Secure Execution Environments (SEE) group at NAI Labs is working with the NSA in further developing and configuring this security-enhanced Linux system. SCC and MITRE are assisting the NSA in developing application security policies and enhanced utility programs. This paper describes work by the NSA and NAI Labs in integrating the security mechanisms of the SELinux architecture into the Linux kernel.

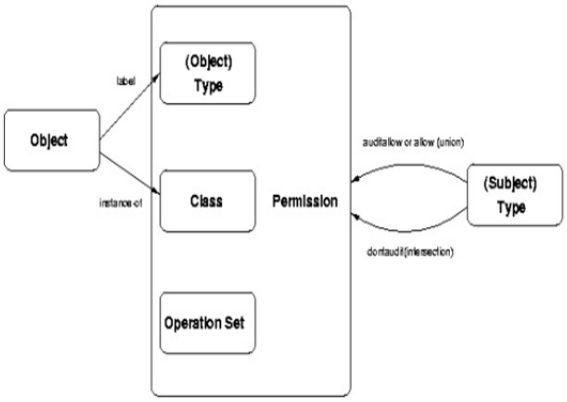
The paper begins by providing an overview of the SELinux policy model and its Linux security approach.While the use of a simpler access control model might make it easier to ensure that security goals are met, we believe that this would result in applications failing to run conveniently, and ultimately, the circumvention of these security goals. The comprehensive nature of the SELinux policy model enables flexible trade-offs between application and security goals. For example, the SELinux example policy itself is developed by proposing application policies and refining them based on the policy violations that may be generated.



2. SELINUX POLICY MODEL:

While SELinux supports a variety of access control policy models , the main focus of SELinux policy development has been an extended Type Enforcement (TE) model . In this section, we provide a brief overview of the SELinux policy model concepts, focusing only on the concepts that are relevant to the analysis that we perform. A number of other concepts are represented in the SELinux extended TE model, such as roles and identity descriptors, that we do not cover here. The traditional TE model has subject types (e.g., processes) and object types (e.g., files, sockets, etc.), and access control is represented by the permissions of the subject types to the object types. In SELinux, the distinction between subject and object types has been dropped, so there is only one set of types that are object types and may also act as subject types. All objects are labelled with a type. All objects are an instance of a particular class (i.e., data type) which has its own set of operations. Permission associates a type, a class, and an operation set (a subset of the class's operations). Thus, permissions associated with SELinux types can be applied independently to different classes. For example, different rights can be granted to a user's files than to their directories. In fact, since the objects are of different classes, they have different operations. Should the administrator want to give different access rights to two objects of the same class, then these objects must belong to different types. Permission for a (subject) type to perform operations on a (object) type are granted by the allow statement. Any element of the permission relationship can be expressed using this statement, so the expression of least privilege rights is possible. The dontaudit statement provides a variation on the basic permission assignment. A combination of allow statements result in a union of the rights specified, whereas a combination of dontaudit statements on the same type pair and class are intersected. In addition, the extended TE model also has type at tributes that represent a set of types (i.e., all the types with that attribute assigned). Type attributes enable assignment to multiple types at a time. For example, permission can be assigned to each subject type with that attribute or a subject can be assigned permission to each object type with that attribute.

Containment is enforced by limiting the permissions accessible to a subject type (as described above), limiting the relabelling of object types, and limiting the domain transitions that can be made by a subject type. Relabel rights are controlled in SELinux by limiting access to relabel-from and relabel-to operations. As the names indicate, relabel-to enables objects to be relabelled to that type and relabel-from enables objects of a particular type to be relabelled. Domain transitions can occur when a subject type executes a new program. Again, SELinux defines an operation, called transition, to perform these transitions. A subject type must have transition permission for the resultant subject type in order to affect a domain transition. The SELinux model also has statements for type transition and type change. Type transition statements are used by SELinux to automatically compute transitions, but are not necessary for control (i.e., transition permissions are always necessary). Type change statements alter the type of an object upon access by the specified subject type. Such statements are useful when a system administrator logins in using a user's teletype. Type change statements transition the object type of the teletype to prevent users from altering input. In order to simplify the task of expressing policies, the SELinux extended TE model also includes a large number of macros for expressing sets of policy statements that commonly occur together.



2.1 SELinux Example Policy:

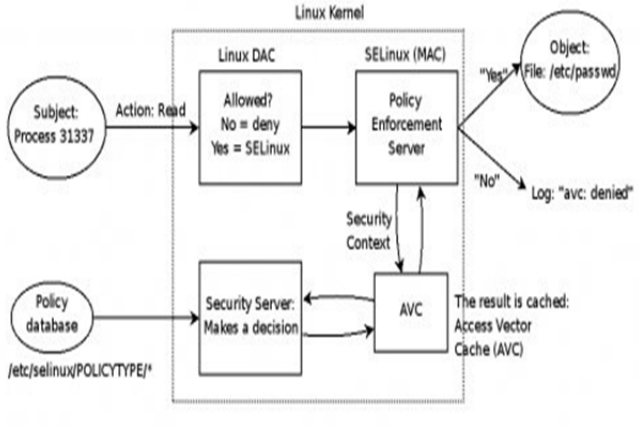
The SELinux community is working jointly on the development of UNIX application policies whose composition is called the SELinux example policy. The SELinux example policy does not define a secure system, but is intended as input to the development of a custom policy for each site's security goals, commonly called a security target. Unfortunately, customization is not simply composition of the policies for the applications of interest. The application policies themselves are somewhat specialized to the environment in which they were developed, and interactions between the policies of multiple applications may lead to vulnerabilities. In general, the composition of policies that are proven secure may not result in a secure system. The task of customization is further complicated by the size of the example policy and the complexity of the extended TE model. The SELinux example policy for Linux 2.4.19 consists of over 50,000 policy statements (i.e., the processed macro statements in policy.conf). Accordingly, this specification represents over 700 subject types and 100,000 permission assignments. We believe that size and complexity of the SELinux example policy make it impractical to expect that typical administrators can customize it to ensure protection of their trusted computing base (TCB) and to satisfy their site's security goals on this TCB. SELinux example policy is valuable to building secure systems, for following two reasons primarily:

(1) It provides a flexible enough representation to capture the permissions necessary for UNIX applications to execute conveniently.

(2) It provides a comprehensive definition of a reference monitor for UNIX.

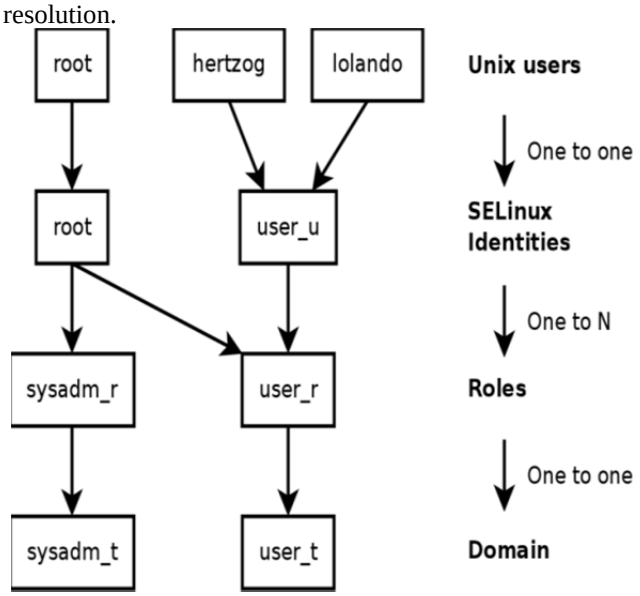
First, the SELinux example policy is developed per application in a manner that identifies a superset of the permissions required to run an application conveniently

while possibly meeting a particular security target. What typically happens is that a proposal is made for an application policy, then this policy is tested by the community when they use the application. Since SELinux reports authorization failures (i.e., the lack of permission requested), it is much easier to determine that insufficient permissions were assigned than whether security vulnerability is created. Thus, a verified proposal for least privilege permissions for each application is represented by the SELinux policy. What we need is a better way to test whether our security goals are satisfied, such that conflicts can be identified and addressed. Second, the SELinux example policy is a comprehensive representation of UNIX access control. The SELinux model aims to comprehensively control access to all classes (i.e., kernel data types) that may be operated upon by a user-level Linux process. There are 29 classes defined in the SELinux example policy. Each class has its own set of operations that are intended to capture all the relevant subtleties in accessing and modifying a class. Given the scope of the SELinux example policy at this granularity, the SELinux example policy provides as precise and comprehensive a repository of UNIX application access control information as exists today. We need to leverage this repository in the development and refinement of security goals, but provide such leverage through higher-level concepts that enable effective management.



2.2 SELinux Security:

Unlike early MAC models like Bell-LaPadula and Biba, a TE model does not explicitly indicate the security goals of the policy. Thus, the policy implies the security goals of the system. For a TE system, more like an access matrix, we only learn that certain subjects can only perform certain operations on certain objects. The security goals of the policy are not represented at a higher-level than this. The SELinux model provides an approach by which secrecy and integrity properties may be achieved with least privilege permissions and containment of services. The system administrators create a policy that is restrictive with respect to granting rights that violate secrecy and integrity properties and we use the notions of least privilege and containment to minimize the damage due to compromises where these occur. From our perspective, the integrity of the TCB is the basis of security, so that is the focus of our analysis. In general, it is preferable to have a "minimal" TCB. The smaller the TCB, the easier it is to verify the components. However, if the minimal TCB subjects are dependent on other subjects, then these other subjects must be added to the TCB or dependencies must be removed. In this paper, we will identify dependencies and determine how to resolve them to keep our TCB as small as is feasible. Since we are striving for a minimal TCB, we do not assume a two-level integrity system (system and user), but rather we start with the most fundamental system services and try to determine how the integrity of these can be enforced. In this paper, we only explicitly examine the TCB and non-TCB boundary. Further, we note that the benefits of least privilege permissions and containment are not relevant to the protection of the TCB. Since the TCB subject types can legitimately transition to any other subject type, containment is not possible for the TCB subjects. Therefore, the focus is on the integrity of these services. kernel_t is the primordial subject type in the SELinux system. It transitions to init_t which then can start a variety of services. Key to our analysis are the administrative (e.g., sysadm_t, load_policy, setfiles_t, etc.) and authentication subject types (e.g sshd_t, local_login_t, etc.) that determine the basis for security decisions in SELinux. We also include initrc_t and inetd_t because these services initiate many of the services in a UNIX system. Of course, there are lots of other services upon which the correct execution of applications is necessary, but we chose this proposal for a minimal TCB based primarily on the early appearance of these services in the type transition hierarchy. Both of these features indicate that vulnerabilities in that subject type will be difficult to contain. While this TCB represents a small number of subject types, the complexity of their interactions with the rest of the system in the SELinux policy makes manual verification impractical. First, each subject type is included in around 500 to over 1000 policy statements in policy.conf. Manual examination of this many statements alone are impractical, but these statements must be compared to the other thousands to determine whether a significant conflict exists. Automated tools are necessary to represent the security goals, identify conflicts, and provide as much support as possible to conflict resolution.



3. CONCLUSION:

In this paper, we have presented an approach for analyzing integrity protection of the SELinux example policy. The SELinux module supports the recent Linux Security Modules (LSM) framework for implementing mandatory access control on the Linux kernel. The SELinux example policy is undergoing active development and is being applied in several installations. The aim is for administrators to take the SELinux example policy and customize it to their site's security goals. This is quite difficult because the SELinux policy model is quite complex and the SELinux example policy is large. Our aim is to provide an access control model to express site security goals and resolve them against the SELinux policy. In particular, we want to identify a minimal system TCB for the SELinux example policy that satisfies Clark-Wilson integrity restrictions relative to the rest of the system. UNIX systems are not designed to meet Biba integrity, but the Clark-Wilson integrity policy enables a description where key data can be identified (those data used by TCB subject types), and sanitization of low integrity data is possible. Understanding this, we can further represent the state of the integrity resolution which could be used by the access control module to make authorization, audit, and intrusion detection decisions.

REFERENCES:

1. C. Wright, C. Cowan, S. Smalley, J. Morris, and G. Kroah-Hartman. Linux Security Modules: Generalv security support for the Linux kernel. Proceedings of the Eleventh USENIX Security Symposium, August 2002.
2. C. J. PeBenito, F. Mayer, and K. MacMillan. Reference Policy for Security Enhanced Linux. In SELinux Symposium, 2006.
3. Michael Wikberg, Helsinki University of Technology, Secure computing: SELinux.
4. 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.