



Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permitted without express written permission.

An Introduction to SELinux For Administrators

**GCUX Practical
Version 3.0, Option 2**

**by Jeff Pike
15 April 2005**

Table of Contents

An Introduction to SELinux for Administrators	4
Abstract	4
Acknowledgments	4
Introduction	4
The Problem:	4
Traditional Solutions:	7
The SELinux Alternative:	7
A Brief History:	8
Fundamental SELinux Concepts	10
Security Models	10
Discretionary Access Control (DAC)	10
Mandatory Access Control (MAC)	10
Type Enforcement (TE)	11
Role Base Access Control (RBAC)	11
Domain Type Enforcement (DTE)	11
Flask Architecture	12
SELinux Architecture	12
Linux Security Modules (LSM)	13
Security Server	13
Security Policy	14
Fedora Core Targeted	15
Fedora Core Strict	15
Access Vectors	15
Security Contexts	16
Default Users	16
Default Roles	17
Some Important Default Types	17
Miscellaneous Elements	17
Classes	17
Type Attributes	17
Conditional/Policy Booleans	18
Initial SIDs	18
Macros	18
Fundamental SELinux Administration	19
Installation	19
Basic Administration	21
Operating Modes	22
Status	23
Changing Policies	23
Security Policy Administration	25
Logging	26
Policy Tweaking	27
Adding Users	31
Seaudit	35

<u>Miscellaneous Administration</u>	38
<u>Future Development and Conclusions</u>	40
<u>Future Development</u>	40
<u>Conclusions</u>	41
<u>Appendix A: Glossary</u>	42
<u>Appendix B: Quick Reference</u>	44
<u>Commands</u>	44
<u>Replaced Commands</u>	45
<u>RBAC/TE Policy</u>	46
<u>AV rules</u>	46
<u>FC Files</u>	47
<u>Generic Policy Notation</u>	47
<u>Miscellaneous Policy Keywords</u>	47
<u>Appendix C: List of References</u>	48

Table of Figures

<u>Figure 1: Layered Access Controls</u>	6
<u>Figure 2: Fedora Core Installation</u>	20
<u>Figure 3: system-config-securitylevel</u>	24
<u>Figure 4: Security Level Confirmation</u>	24
<u>Figure 5: Sepcut – mount.te</u>	29
<u>Figure 6: Sepcut – Test Policy</u>	30
<u>Figure 7: Seuserx – Add new user - Properties</u>	32
<u>Figure 8: Seuserx – Add new user – Advanced Options</u>	33
<u>Figure 9: Seuserx – Main Window</u>	34
<u>Figure 10: Seaudit</u>	36
<u>Figure 11: Seaudit – Query Policy</u>	37
<u>Figure 12: Apol – Domain Transition Analysis</u>	39

An Introduction to SELinux for Administrators

Abstract

This paper is an introduction to Security Enhanced Linux (SELinux) for Unix system administrators and analysts. It is assumed that the reader has a fundamental knowledge of Unix system administration and security issues. The first section of this paper will focus on the fundamental concepts of SELinux that make it a sound choice for secure hosting of applications. The second section will introduce basic SELinux administration topics and issues. Footnotes are used for references and other details beneficial to those seeking information not discussed in the text

Acknowledgments

The author would like to thank Adrienne Payette for the editorial review. Special thanks are owed to Spring Hellgrath for the painstaking technical review.

Introduction

About two years ago the author was asked to provide a vulnerability analysis of a system running SELinux. The lack of resources written for Unix administrators made the learning curve steep. The goal of this paper is to provide one such a resource for Unix administrators and security analysts. SELinux provides an excellent platform for secure hosting. It deserves consideration as a hosting option when increased security is required.

The Problem:

The problem is that operating systems do not provided protection from vulnerabilities constantly being discovered in software. Bruce Schneier says, "We wouldn't have to spend so much time, money, and effort on network security if we didn't have such bad software security."¹ The CERT archives and Bugtraq mailing lists are full of software vulnerabilities. Vulnerabilities exist in routing protocols, hash functions, office applications, web browsers, mail transfer agents, operating systems, etc. News of new software vulnerabilities is published daily. Vendors usually release patches for operating systems and applications a short time after vulnerabilities are publicly disclosed. Even so, all remain at the mercy of any attacker with knowledge of zero-day vulnerabilities hidden within their systems. Wouldn't it be better if operating systems actually offered some protection?

Hal Pomeranz describes UFS as the Untrustworthy File System.² This is quite true of most Unix operating systems. The problem lies in the fact that Unix was never designed to be a secure operating system. According to Ross Anderson:

¹ Viega and McGraw (2002); p. xix

² Pomeranz (2003); pp. 2-1 through 2-21

Unix...was originally designed as “single-user Multics” (hence the name). It then became an operating system used by a number of skilled and trustworthy people in a laboratory who were sharing a single machine. In this environment, the function of the security mechanism is primarily to contain mistakes, to prevent one user’s typing errors or program crashes from deleting or overwriting another user’s files. The original security mechanisms were quite adequate for this purpose.³

As time passed Unix was extended beyond this environment. Its security mechanisms were not. Its traditional discretionary access control (DAC) is effectively an all-or-nothing model. Users and processes can either access files or they can’t. The principle of least privilege, which states that no process should be given more privileges than necessary to perform its required function, is nowhere to be found here. The DAC model is not conducive to building secure systems in today’s environment. It lacks the granularity required to adequately enforce the principle of least privilege.

Access control determines what system resources a user or process can utilize in a system. Access controls as a whole can be thought of in terms of layers.⁴ Operating system access controls rely on the architecture of the processor. Databases, system- level services, and shared libraries rely on the access controls provided by the operating system. Applications rely on access controls implemented by any service below them which they use. There are even layers of applications. Mobile code relies upon the access controls and security mechanisms implemented in the web browser application. Access controls at each layer are dependent upon the correct function of those below them just like network protocol models. Peter Loscocco, *et al* describe the problem best:

The increased awareness of the need for security has resulted in an increase of efforts to add security to computing environments. However, these efforts suffer from the flawed assumption that security can adequately be provided in application space without certain security features in the operating system. In reality, operating systems security mechanisms play a critical role in supporting security at higher levels. This has been well understood for at least twenty five years and continues to be reaffirmed in literature. Yet today, debate in the research community as to what role operating systems should play in secure systems persists. The computer industry has not accepted the critical role of the operating system to security as evidenced by the inadequacies of the basic protection mechanisms provided by current mainstream operating systems.⁵

³ Anderson (2001); p. 69

⁴ Anderson (2001); p. 52

⁵ Loscocco, Smalley, Muckelbauer, Taylor, Turner, and Farrell (1998); p. 1

Mainstream commercial operating systems rarely support the principle of least privilege even in their discretionary access control architecture. Many operating systems only provide a distinction between a completely privileged security domain and a completely unprivileged security domain. Even in Microsoft Windows NT, the privilege mechanism fails to adequately protect against malicious programs because it does not limit the privileges that a program inherits from the invoking process based on the trustworthiness of the program.⁶

Figure 1 below depicts a common implementation of Ross Anderson's concept of layered access controls.⁷

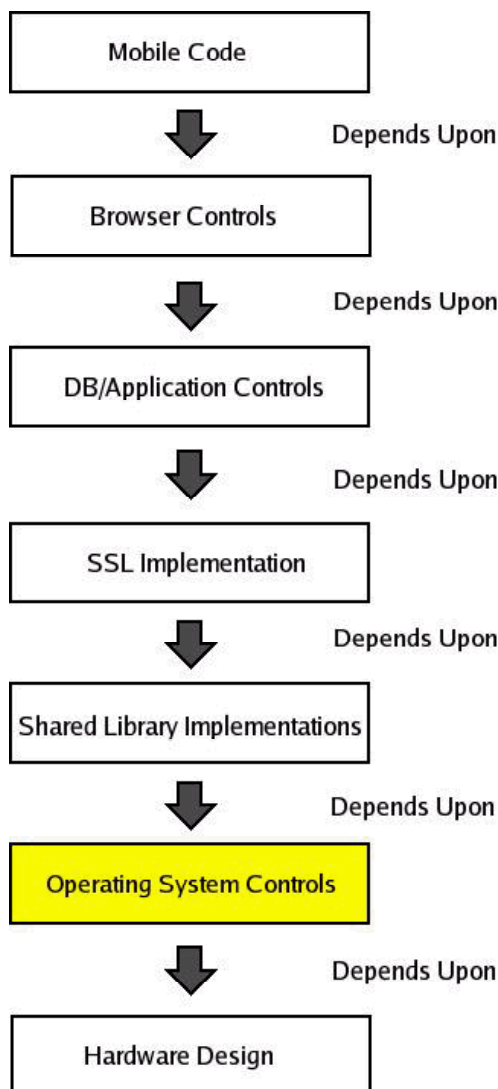


Figure 1: Layered Access Controls

⁶ Loscocco, Smalley, Muckelbauer, Taylor, Turner, and Farrell (1998); p. 4

⁷ Anderson (2001); p.52

For many years industry has made up for poor operating system access controls by attempting to implement them at higher levels. Most modern processor architectures, such as the i386, support two modes of execution: kernel and user.⁸ The privileged kernel mode is used to protect key operating system components from less privileged user mode processes. Memory is protected by virtual memory schemes. Above that level are the familiar operating system discretionary access controls. Any processes, such as databases, services, and applications rely upon the OS and hardware design as a foundation. Unfortunately, the discretionary access controls implemented by most modern operating systems offer an insufficient foundation upon which to build.

Processes run with the privileges of some system or user account. Unfortunately, most software processes run with privileges far in excess of what is needed to perform their function. Many run with root privileges. Because this entire house of cards is built on a foundation that does not enforce least privilege, it is subject to collapse. An attacker exploiting a vulnerable process will have access to all its system resources. In some cases that means root. Because of weakness in the DAC model, the protection offered by other layers falls through to the OS layer once any process is exploited.

Traditional Solutions:

The principle of least privilege offers sufficient protection when properly applied. In a nutshell, the principle states that any user, process, application, or service should have no more privilege than is necessary to complete any required function. If the principle of least privilege is actually applied to a system hosting a vulnerable application, the worst that can happen is compromise of that application rather than the entire system.

Some traditional methods of enforcing least privilege on Unix hosts include minimizing services, fix-modes, sudo, and chroot(). The principle of least privilege can also be applied to networks in the form of segregation, packet filtering, and firewalling. There are less common methods that attempt to address the heart of the problem in Unix environments including grsecurity, User Mode Linux, Trusted BSD, Trusted Solaris, and SELinux. Many of these potential solutions are gaining momentum. The remainder of this paper will focus on the leading open source solution, SELinux.

The SELinux Alternative:

As delineated by Loscocco, *et al*, "If security practitioners were to more openly acknowledge their security solution's operating system dependencies as requirements for future operating systems, then the increased demand for secure operating systems would lead to new research and development in the area and ultimately to commercially viable secure systems."⁹ An organizations

⁸ Or privileged and unprivileged.

security staff and system administrators cannot correct vulnerabilities in software that their organization merely licenses from another party. However, there are operating systems that offer some protection in the form of stronger access controls. One leading alternative is SELinux.

In traditional DAC, the owner of an object (or file) is able to grant permissions to other users. Mandatory Access Control (MAC), by contrast, is a system-enforced mechanism that cannot be overridden by the owner. Role Based Access Control (RBAC) controls access to resources through the use of profiles and roles. SELinux augments the traditional DAC model in Linux with MAC and RBAC. This provides a powerful yet flexible mechanism for controlling access to the system.

SELinux can be used anywhere increased access controls are required. Increased granularity in access controls gives administrators and architects the ability to enforce least privilege with increased vigor. SELinux is a sound choice for bastion hosts in a DMZ such as web and DNS servers. SELinux also makes a good host operating system for security appliances such as firewalls and intrusion detection systems. When vulnerabilities are discovered against applications running on SELinux, the worst likely result is denial of service against that application. SELinux provides the MAC high security environments. SELinux comes installed and enabled by default on Fedora Core 3 (FC3).

A Brief History:

Much research was conducted by the government on security models in the 1970's. The government has always had an interest in developing secure operating systems for military applications. The National Security Agency (NSA) has long been involved in operating systems security research as part of its information assurance mission.¹⁰ SELinux incorporates the results of several previous research projects including DTMach, DTOS,¹¹ and Flux. Each of these projects borrowed from their ancestors and the resulting architecture was Flask.¹² SELinux is an implementation of the Flask architecture.

SELinux was developed by the NSA, Secure Computing Corporation (SCC), and the University of Utah's Flux research group. It was first released to the public on December 22nd, 2000 based on the Linux kernel version 2.2.12 and RedHat 6.1 utilities. It was designed to provide a MAC architecture in an effort to address confidentiality and integrity requirements not met by existing mainstream operating systems.

There have been four major releases of SELinux:¹³

⁹ Loscocco, Smalley, Muckelbauer, Taylor, Turner, and Farrell (1998); p. 10

¹⁰ National Security Agency, "Security-Enhanced Linux"

¹¹ University of Utah, (2000); "The Distributed Trusted Operating System (DTOS) Home Page"

¹² University of Utah, (2000); "Flask: Flux Advanced Security Kernel"

¹³ National Security Agency, "Historical Versions of SELinux"

Original Implementation - Pre-LSM kernel patch:

Before Linux Security Module (LSM) framework was integrated into the Linux kernel, SELinux was installed as a kernel patch. This version supported Linux 2.2 and Linux 2.4

Old LSM SELinux:

The LSM hooks were made use of in this release. Extended attributes were used for security labeling. This release was the first step in preparing SELinux for integration into mainline Linux. SELinux was still installed as a patch against the 2.4.21 or 2.5.66 kernels.

2.4-based SELinux (2.6 version back-ported):

This version is no longer supported or maintained. It was based on 2.6 SELinux and back ported for those still using 2.4 based kernels. It still had to be installed as a kernel patch, and the last one was the 2.4.26 kernel. It contains all the features of current 2.6 SELinux, but it does not have any recent improvements.

Current 2.6-based SELinux:

SELinux is included in the mainline 2.6 kernel. NSA's patch against the kernel could be applied to get the latest enhancements providing the required kernel sources are present. For most users of Fedora Core, RedHat Enterprise Linux, or other 2.6 based distributions there will be no compelling reason to keep up with the latest.

Fundamental SELinux Concepts

This first of two main sections will discuss the security features and underlying mechanisms of SELinux that make it a more secure OS. SELinux and relevant security models have a language all their own. “Appendix A: Glossary” of this document makes a handy reference while reading this section.

Security Models

To understand how SELinux works, it is necessary to have a basic concept of several security models which play a part in its overall architecture. These models are made up of abstract ideas and virtual components.

Discretionary Access Control (DAC)

DAC is the traditional all-or-nothing security model used in Unix and Windows systems. Each user is able to assign permissions to objects they own. Because of this, system-wide security policies cannot be enforced. Essentially, there are two levels of permissions with this model: root and non-root. Software and processes are often given permissions far in excess of what they require to perform their functions. DAC does a poor job of implementing the concept of least privilege.

Within SELinux the standard DAC architecture implemented by Linux is still used. Standard Linux permissions are checked for all actions first. If actions are permitted by DAC, then control is passed to the SELinux security server.

Mandatory Access Control (MAC)

Mandatory access control has its roots in the Bell-Lapadula model and multi-level security (MLS) systems used within the DOD. The Bell-Lapadula model itself was introduced in 1973. Within Bell-Lapadula, users and processes are considered subjects. Processes and files are considered objects.¹⁴ Subjects are said to be granted permission to act on objects as pre-defined by the security policy. Any action not explicitly permitted by the pre-defined security policy is denied.

There are reasons MAC has not been widely adopted outside the government. The traditional multi-level security implementation of the government is too limiting for most environments. Additionally, every organization would have to implement a different MAC policy depending upon their environment and applications to accurately apply the principle of least privilege.

Although SELinux is said to implement a form of MAC, it overcomes some of the limitations of vintage MAC by implementing the policy-flexible Flask architecture.

¹⁴ It is important to note that process can be both subjects and objects in SELinux.

Type Enforcement (TE)

Type Enforcement is an access control mechanism originally developed in 1985. Like MAC it deals with subjects and objects. Subjects such as users and processes are associated with domains. Objects such as files and directories are associated with types. The Domain Definition Table (DDT) controls access between domains and types. The original type enforcement model was considered difficult to implement in practice. Challenges included the difficulty in labeling objects with type attributes and the inherent complexity of large tables.

Type Enforcement® technology is actually a registered trademark of Secure Computing Corporation¹⁵. Note that Secure Computing Corporation's highly regarded Sidewinder firewall appliance is built on a type enforcement model. It is said that Sidewinder has never been vulnerable.

Role Base Access Control (RBAC)

RBAC was developed¹⁶ in 1992 as a means to address the reality that most organizations have unique security requirements that cannot be met through DAC and MAC alone. Access to system resources is determined by a user's assignment to a role in an organization or membership in a group. RBAC is actually a type of MAC not strictly based on MLS requirements of the Bell-LaPadula model.

This is the secondary access control model in SELinux. It is used to add flexibility to SELinux's domain type enforcement, introduced below.

Domain Type Enforcement (DTE)

Domain and Type Enforcement is an extended version of type enforcement that was added¹⁷ in 1995. DTE introduced two improvements to overcome the challenges of previous TE concepts. First, it introduced the Domain and Type Enforcement Language (DTEL) as high level language for defining security policy. This is a user-friendly language that allows administrators to customize the security policy to meet their requirements. Second, DTE security labels are not stored one-to-one with files on the hard disk, but are instead maintained implicitly based on directory hierarchy.¹⁸

SELinux implements a form of domain type enforcement¹⁹ as its primary access control mechanism. The policy configuration language for SELinux is somewhat different than the original DTEL mentioned here,²⁰ but it is equally intuitive.

¹⁵ Secure Computing Corporation, "Sidewinder G2 Firewall Type Enforcement Technology"

¹⁶ Ferraiolo and Kuhn (1992)

¹⁷ Badger, Sterne, Sherman, Walker, Haghighat (1995)

¹⁸ Badger, Sterne, Sherman, Walker, Haghighat (1995)

¹⁹ Walker, Sterne, badger, Petkac, Sherman, Oosendorp (1996)

²⁰ Loscocco and Smalley (2001); "Integrating Flexible Support for Security Policies into the Linux

Flask Architecture

The Flask security architecture grew out of several other projects and was integrated with Linux to put the 'SE' in SELinux. Unlike traditional MAC, the Flask architecture provided MAC can support many different security policies.²¹ The security policy is encapsulated in the security server (kernel) which makes all security policy decisions. The security policy may be modified and reloaded into the security server, which provides flexibility. Enforcing components called object managers include other kernel subsystems. These are the components that actually enforce the security policy after the security decision has been made. Examples include IPC, sockets, and file management. The security server itself is a kernel subsystem.²²

Flask also includes the access vector cache (AVC), which caches security decisions. This caching of security decisions allows the security policy to be enforced with minimal impact on performance. According to Loscocco and Smalley, most permission checks do not even incur the cost of an extra function call.²³

The Flask architecture supports labeling using security contexts and SIDs. The security context is an actual representation of the security label. A security identifier (SID) is a pointer to the security context. Individual object managers (kernel subsystems) handle the labeling of their associated objects.

The Flask architecture should not be confused with its SELinux implementation. As described by Smalley and Loscocco:

The Flask architecture merely specifies the interfaces provided by the security server to the object managers. The implementation of the security server, including any policy language it may support, are not specified by the architecture.²⁴

SELinux Architecture

SELinux is a specific implementation of the Flask architecture and more. SELinux supports a variety of security models. It more-or-less incorporates elements from all of the previously mentioned models. The sections that follow provide an overview of the current implementation of SELinux.

Operating System"; p. 10

²¹ or MAC policies

²² Loscocco and Smalley (2001), "Integrating Flexible support for Security Policies into the Linux Operating System"; pp. 2-3

²³ Loscocco and Smalley (2001), "Integrating Flexible support for Security Policies into the Linux Operating System"; p. 3

²⁴ Loscocco and Smalley (2001), "Integrating Flexible support for Security Policies into the Linux Operating System"; p. 3

Linux Security Modules (LSM)

LSM allow support for SELinux to be built into the Linux 2.6 mainline kernel. This is the framework upon which the current version of SELinux is built. In March of 2001, the NSA gave a presentation about SELinux at the 2.5 Linux Kernel Summit. Smalley, Vance, and Salmon explain the results:

In response to the NSA presentation, Linus Torvalds made a set of remarks that described a security framework we 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 the kernel data structures for maintaining security attributes. This framework could then be used by loadable kernel modules to implement any desired model of security. The Linux Security Modules (LSM) project was started by Immunix to develop such a framework.²⁵

Because of LSM, SELinux is now integrated into the mainline kernel, so no patching is necessary. LSM improved the SELinux implementation in interesting and well documented ways²⁶ from the original SELinux kernel patch.

SELinux makes use of the extended attributes feature of the 2.6 kernel for labeling files with name/value pairs.²⁷ The `setxattr`, `getxattr`, `listxattr`, and `removexattr` system calls can be used by programs to manipulate extended attributes.

Security Server

In traditional security architecture, domains generally refer to subjects such as users and processes, while types generally refer to objects like files and devices. Within SELinux the terms *type* and *domain* are often used interchangeably. This is particularly noticeable when defining and administering the security policy. Stephen Smalley describes the relationship best: "The SELinux TE model differs from the traditional TE model in that it uses a single attribute in the security context for both processes and objects. A domain is simply a type that can be associated with a process."²⁸

Optional type attributes can be used to associate multiple types with the properties they share.²⁹ Their names are self-descriptive and include such examples as *domain*, *privuser*, *auth*, *device_type*, and *unrestricted*. There are about 80 type attributes in the FC3 implementation of SELinux.

²⁵ Smalley, Vance, and Salmon (2004); pp 6-7

²⁶ Smalley, Vance, and Salmon (2004), pp. 9-17

²⁷ Love, (2003), p. 197

²⁸ Smalley (2005); p. 5

²⁹ Smalley, Frasier (2001); p. 5

Security Policy

Security policy is the heart of SELinux. It is compiled from numerous source files and loaded into the security server (or kernel), which compares it to appropriately labeled objects for access decisions. It can be customized as needed to suit the needs of varied organizations. Several example policies are available as a starting point. Security policies are made up of the following elements:³⁰

- **Flask definitions:** define security classes, initial SIDS, and access vector permissions. These are built into SELinux. Most people will have no need to change these
- **TE statements:** define the fine-grained type enforcement rules for subjects and objects. These make up the largest component of SELinux policy. They include: type declarations, type transition rules, type changes rules, access vector rules, assertions
- **RBAC statements:** define roles, hierarchical relationships between roles, and authorized transitions. Statement types consist of role declarations, role dominance definitions, and allow rules
- **user declarations:** define the SELinux user identities and associate the users with various roles. Note that these are different than Unix users. All users must be declared to be recognized within a security context
- **constraint definitions:** define restrictions on access vector rules based on a broad combination of information that can be evaluated in the form of a boolean expression. Most people will have little reason to change these
- **security context specifications:** provide security context information for general objects such as initial SIDS, persistent and non-persistent filesystems, and network objects

Each business has unique needs, and these needs can be reflected in its SELinux security policy. For example, if SELinux is chosen as a hosting platform for a device such as a security appliance, a strict, well-tested policy could ensure that each service has no more privileges than the minimum required to perform its function. On the other hand, an everyday user might only desire a few server processes to be constrained by SELinux. Perhaps a bastion host providing name services might require a strict policy with some customization to allow for regular upgrades of BIND. NSA provides an example policy, and the Fedora Core distribution comes with two diverse policies that can be used as a starting point

³⁰Smalley (2005); pp 7-24

Fedora Core Targeted

Because of the difficulties in trying to apply the somewhat strict NSA example policy to the wide range of Fedora users, the *Targeted* policy was created. This policy focuses on locking down specific daemons including dhcpd, httpd, named, ncsd, ntpd, portmap, snmpd, squid, and syslog. Each of these secured daemons runs in its own domain. The rest of the system is allowed to run transparently under the standard Linux security model. Unsecured processes run under the *unconfined_t* domain, which is not constrained by the SELinux policy. As addressed in the Fedora Core 3 SELinux FAQ:

Specific network daemons have policy written for them, and the *unconfined_t* policy transitions to those policies when the application starts. For example, on system boot, init runs under the *unconfined_t* policy, but when named starts it is transitioned to the *named_t* domain and is locked down by the appropriate policy.³¹

Fedora Core Strict

The Fedora Core Strict is an implementation of the NSA example policy, which is available at the NSA website.³² As the name suggests, the strict policy applies access controls to all objects rather than a few specific processes. It is based closely on the original example policy developed by NSA for SELinux,³³ and it will need to be customized for each environment in which it is installed in. The *strict* policy demonstrates the full power of the access control mechanisms provided by SELinux.

Access Vectors

There are about 192 different permissions defined in *~/flask/access_vectors* grouped into about 53 object classes in *~/flask/security_classes*.³⁴ In SELinux the type enforcement policy is managed using the higher level concept of roles. It is important to remember that domains and types are interchangeable in SELinux.

Access vector may be thought of as one of four actions the kernel can take with a request. The syntax is *domain type:class operation*. Possibilities are:

- 1) *allow* – allow the operation and don't log it
- 2) *auditallow* – allow the operation and log it
- 3) *auditdeny* – deny the operation and log it
- 4) *dontaudit* – deny the operation and don't log it.
- 5) *neverallow* – not an access vector per se; defines policy constraints

³¹ Fedora Core 3 SELinux FAQ

³² The version is policy-1.22 at the time of this writing

³³ Hally (2004)

³⁴ Where ~ represents the root of the policy source tree such as */etc/selinux/strict/src/policy*

Security Contexts

Security contexts are the primary elements of security policy. Each user, process, file, and device has a security context that determines what access can be granted. The subject/requestor of an action has the source context. The object/type acted on has the target context. Security contexts are made up of three elements:

- 1) identity (sometimes called user)
- 2) role
- 3) type (sometimes called domain)

An example of a security context can be seen below. In this case the user pike has an identity of *user_u*, a role of *user_r*, and a type of *user_t*.

```
[pike@localhost ~]$ id -Z35  
user_u:user_r:user_t
```

As a separate example, the user pike lists the contexts of the metasploit subdirectory under his home directory. The objects have an identity of *system_u*, a role of *object_r*, and a type of *user_home_t*.³⁶

```
[pike@localhost metasploit]$ ls -aslZ metasploit  
total 1320  
drwxrwxr-x pike pike system_u:object_r:user_home_t .  
drwx----- pike pike system_u:object_r:user_home_dir_t ..  
drwxr-xr-x pike 408 system_u:object_r:user_home_t  
framework-2.2  
-rw-rw-r-- pike pike system_u:object_r:user_home_t  
framework-2.2-snapshot.tar.gz
```

The listings that follow describe the default elements that make up SELinux security contexts.

Default Users

- cyrus: set aside for the Cyrus IMAP Daemon (Fedora specific)
- root: the system administrator
- system_u: user account for system processes
- user_u: unprivileged user account

³⁵ The id, ps, and ls commands include a -Z option that displays SELinux security context information.

³⁶ Note that the metasploit tarball is labeled with a type according to the directory it is located in as defined by the security policy. For example, if it were copied to the root directory it would have the *default_t* type. Files copied to the /root directory are labeled with the *staff_home_t* type. Objects in the /var/log directory receive the *var_log_t* type.

Default Roles

- `cyrus_r`: role for the Cyrus IMAP daemon (Fedora specific)
- `object_r`: default object role in strict policy
- `staff_r`: default role for sysadmins; authorized to enter `sysadm_r`
- `sysadm_r`: role used to perform administrative tasks
- `system_r`: used by processes
- `user_r`: unprivileged user role

Some Important Default Types³⁷

- `default_t`: default domain for objects in strict policy
- `staff_home_t`: default type for objects in user home directories
- `sysadm_t`: domain for administrative tasks
- `staff_t`: default domain for system administrators; authorized to transition to `sysadm_t`
- `unconfined_t`: a domain without enforcement for unrestricted apps in the target policy
- `user_t`: domain for `user_r` role

Miscellaneous Elements

Although the security context of a subject or object is the primary consideration when modifying security policy, there are some other elements that round out the SELinux architecture.

Classes³⁸

Classes are logical groupings of objects. For the targeted policy in FC3 they are defined in `/etc/selinux/targeted/src/policy/flask/security_classes`. Examples of classes include:

- `processes`: a class for process (recall that processes can be objects)
- `tcp_socket`: all tcp sockets
- `netif`: a class for network interfaces

Type Attributes³⁹

Type attributes are names bound to one or more types that are used to define a set of types sharing some property.⁴⁰ For the targeted policy⁴¹ in FC3 they are defined in `/etc/selinux/targeted/src/policy/attrib.te`. Examples of type attributes include:

- `privrole`: identifies domains that can change roles
- `admin`: identifies administrator types/domains
- `proc_fs`: identifies types/domains that may be assigned to files under

³⁷ There are 1287 default types on the author's system when using the strict policy.

³⁸ There are 53 default object classes on the author's system when using the strict policy.

³⁹ There are 80 type attributes on the author's system when using the strict policy.

⁴⁰ McCarty (2004); p. 134

⁴¹ See the sections on Security Policy and Fedora Core Targeted for further clarification.

/proc

© SANS Institute 2000 - 2005, Author retains full rights.

Conditional/Policy Booleans⁴²

Booleans are true/false conditional values that can be used to tune a policy. Their values can be retrieved with the **getsebool** command and set with the **setsebool** command. They can be found in the `/selinux/booleans` directory.

Examples of booleans include:

- `httpd_enable_cgi`: allows httpd cgi support
- `httpd_ssi_exe`: allows httpd to run SSI executable in the same domain as system CGI scripts.
- `portmap_disable_trans`: disables SELinux protection for portmap daemon.

Initial SIDs⁴³

Initial SIDs are predefined values used to define security contexts during system initialization. For the FC3 strict policy, these values are contained in the `/etc/selinux/strict/src/policy/initial_sid_contexts` file.⁴⁴ Example of initial SIDs include:

- `fs`: `system_u:object_r:fs_t`
- `sysctl`: `system_u:object_r:fs_t`
- `policy`: `system_u:object_r:unlabeled_t`

Macros⁴⁵

There are several types of macros including global, administrative, user, policy, and program macros. They are used to ease the burden of defining users, types, domains, transitions, and access vector rules. These M4⁴⁶ macros make administering the security policy more manageable. For the strict policy on FC3 they are mostly defined under the `/etc/selinux/strict/policy/macros` directory.

Examples of macros include:

- `use_games`: a policy macro that, if defined, will allow users to run games
- `domain_auto_trans`: a global macro that specifies and authorizes a transition related to the execution of a program defined as a domain entry point.⁴⁷
- `in_user_role`: A user macro used to permit roles to access a domain

⁴² There are 27 conditional Booleans defined on the author's system by default using the strict policy.

⁴³ There are 27 initial SIDS defined on the author's system by default using the strict policy.

⁴⁴ The initial SIDs are the same for both the *targeted* and *strict* policies.

⁴⁵ The exact number of macros is unknown. On the author's system 171 were counted under the `/etc/selinux/strict/src/policy/macros` and `/etc/selinux/strict/src/policy/macros/programs` using the following kludged command string: `grep define * |grep -v undefine |cut -f1 -d' ' |sort -u |wc -l`

⁴⁶ See Dunne (2000) for a good overview of M4.

⁴⁷ McCarty (2004); p. 140

Fundamental SELinux Administration

This section will introduce fundamental SELinux installation, configuration, and administration issues. The goal is to provide enough information to allow the reader a running start into further exploration of SELinux. “Appendix B: Quick Reference” may be useful to readers of this section.

SELinux is known to work on many distributions⁴⁸ including: Debian,⁴⁹ Gentoo,⁵⁰ SUSE,⁵¹ Slackware,⁵² and RedHat Enterprise Linux.⁵³ However, SELinux on Fedora⁵⁴ Core 3 will be the focus of this paper for the following reasons:

- 1) It is one of the most popular Linux distributions.
- 2) SELinux comes installed by default on FC3.
- 3) SELinux was built based on RedHat style utilities.⁵⁵
- 4) The FC3 DVD has everything required to get started.

Installation

NSA is the root source of SELinux just like kernel.org is the root source of Linux. SELinux is under active development as evidenced by the developers’ mailing.⁵⁶ This discussion will focus on Fedora, which is both a leading distribution of Linux and SELinux. While version numbers may differ, Fedora-distributed components are generally aligned with the NSA distribution.⁵⁷ The most notable exceptions are that that NSA’s example policy equates to the FC 3 strict policy, and NSA does not have a targeted policy.

When SELinux is run in enforcing mode⁵⁸, the power of root is removed. It is possible to be logged in as root, without having significant administrative privileges. In fact Russell Coker has run several SELinux boxes with root as the guest account⁵⁹.

SELinux in FC3 comes installed by default enforcing the targeted policy. During the installation of Fedora Core, the option is provided to set the operating mode. Figure 2 shows the installation screen:

⁴⁸ SE Linux for Distributions Project, “SELinux for Distributions”

⁴⁹ Lemuria.org, “Installing a 2.6SELinux Kernel/System”

⁵⁰ Gentoo.org, “Installing Gentoo SELinux”

⁵¹ Bleher, “Thomas’ SELinux-Pages”

⁵² Wood, “SE Linux”

⁵³ RedHat, “SELinux”

⁵⁴ Fedora Project, “SELinux”

⁵⁵ RedHat is the sponsor of the Fedora Core project.

⁵⁶ National Security Agency, “SELinux Mailing List”

⁵⁷ National Security Agency, “Download 2.6-based SeLinux” Note that the development is very active and version numbers of NSA components change frequently.

⁵⁸ Enforcing mode actually “enforces” the security policy as described in figure 2.

⁵⁹ Coker, Russell “SELinux Play Machine”

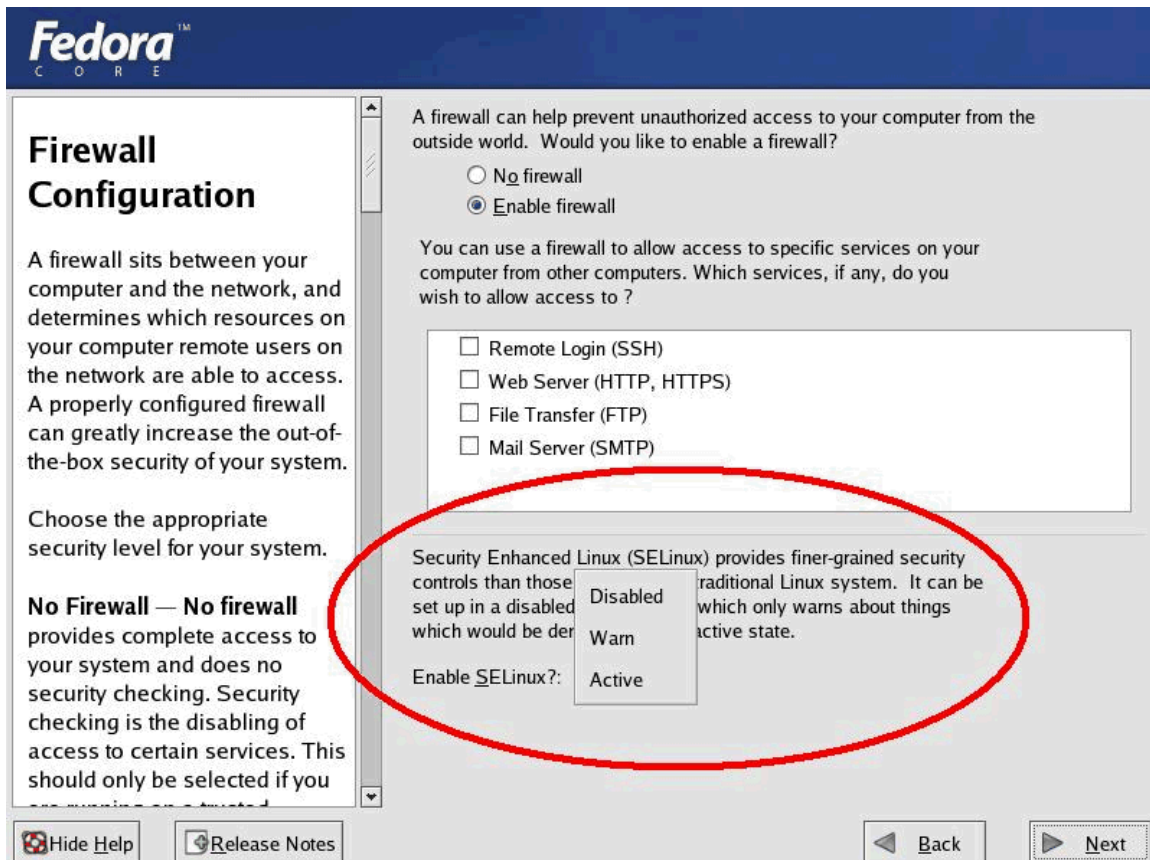


Figure 2: Fedora Core Installation

If the default of *Active* is chosen, SELinux will boot in enforcing mode. Any actions not explicitly permitted by the SELinux security policy will be prohibited. The *Warn* setting configures SELinux to boot into permissive mode. In this mode actions that would have been prevented by the security policy will only be logged. Permissive mode is designed for troubleshooting and testing the security policy. If *Disabled* is chosen, the security server and policy are not loaded into the kernel at all. Any files created during this time will not be labeled with a security context. If SELinux is needed at some later point, the system should be booted into permissive mode and the file system should be relabeled prior to attempting to run in enforcing mode.

The default installation of Fedora Core 3 includes the following packages:

- libselinux-1.17.14-1: provides interfaces for security-aware applications
- libselinux-devel-1.17.14-1: required to recompile policy
- libsepol-1.17.1-2: library for policy creation
- checkpolicy-1.17.5-1: the policy compiler
- policycoreutils-1.17.6-2: the core command line administrative utilities
- selinux-policy-targeted-1.17.30-2.19: the unrestrictive default policy
- setools-1.4.1-5: policy administration utilities created by Tresys

Additional packages included with the Fedora Core distribution can be installed using the **rpm** command.⁶⁰ Additional RPM packages on the Fedora Core installation media are as follows:

- **libsepol-devel-1.1.1-2.i386.rpm**: contains static libraries and header files needed to develop SELinux enabled applications. Also includes the *chkcon* and *genpolbools* tools
- **selinux-doc-1.14.1-1.noarch.rpm**: documentation including build instructions, porting information, and some NSA white papers
- **selinux-policy-strict-1.17.30-2.noarch.rpm**: strict policy for FC3; based on NSA's example policy
- **selinux-policy-strict-sources-1.17.30-2.noarch.rpm**: source files required to customize the NSA
- **selinux-policy-targeted-sources-1.17.30.2-19.noarch.rpm**: source files required to recompile the targeted policy
- **setools-gui-1.4.1-5.i386.rpm**: advanced GUI policy administration utilities created by Tresys. Also installs TCL support for the tools

To modify the targeted policy on FC3, the **selinux-policy-targeted-sources-1.17.30.2-19.noarch.rpm** must be installed. The package is located in a directory similar to */media/cdrom/Fedora/RPMS* on the installation DVD or fourth CD in the set. Once it is installed, the */etc/selinux/targeted/src* directory tree will be created. The targeted policy can be customized using the files under this tree.

To work with the NSA example policy and truly experience the access controls offered by SELinux, the strict policy and sources can be customized. The **selinux-policy-strict-1.17.30-2.noarch.rpm** package can be installed from the DVD or first CD in the set. Installing this RPM creates the */etc/selinux/strict* directory tree. The policy as loaded into the security server is binary and resides in the file */etc/selinux/strict/policy/policy.18*. As with the targeted policy, the sources must be installed before customization or experimentation can occur. After the binary policy is installed, the sources can be added by installing **selinux-policy-strict-sources-1.17.30-2.noarch.rpm** from the DVD or fourth CD in the set. This will create the */etc/selinux/strict/src* directory tree as expected.

Basic Administration

This section will provide an overview of the most basic SELinux administration concepts.

⁶⁰ The RedHat Package Manager (RPM) automatically labels files with appropriate security context information. This behavior is referred to as "SELinux aware."

Operating Modes

Essential to administering SELinux is knowledge of how to enable it, disable it, and set the operating mode as previously discussed. The files `/etc/sysconfig/selinux` and `/etc/selinux/config` define the SELinux mode at boot time. If the defaults were accepted during installation, the default entries would be as follows:

SELINUX=enforcing (other possible values: disabled, permissive)
SELINUXTYPE=targeted (other possible values: strict)

The `/selinux` filesystem is similar to the `/proc` filesystem in that it is a virtual filesystem that provides access to process state. It is primarily used for loading policy, setting the mode, and querying the kernel policy database.⁶¹ `/selinux/enforcing` indicates the current mode (0 for permissive; 1 for enforcing). The **getenforce** command can be used to retrieve the enforcing mode. Likewise, the **setenforce** command can be used to set it by passing a '0' for permissive or a '1' for enforcing.

There are a number of ways to improperly configure SELinux when using the strict policy. In order to recover from the most severe configuration errors, SELinux must be disabled⁶² at boot time. Passing **selinux=0** to the kernel will achieve this. Below is a simple procedure for disabling SELinux at boot time on Fedora Core 3:

- 1) Hit the <Esc> key at the Grub boot loader screen to stop the boot sequence.
- 2) Press the <e> key to edit the configuration.
- 3) Add **selinux=0** to the end of the second line, which begins with the word 'kernel'.
- 4) Press <Enter> to save the change for this boot session.
- 5) Press the key to boot using the specified configuration.

⁶¹ Coker, Russell (2004); "SE Linux Implementation LINUX20"; p. 12

⁶² This will not work on kernels compiled without the **NSA SELinux boot parameter** option. McCarty, (2004); p. 67

Status

SELinux provides the **sestatus** command line tool to provide relevant status information. Sample output is below:

```
[root@localhost policy]# sestatus
SELinux status: enabled
SELinuxfs mount: /selinux
Current mode: enforcing
Policy version: 18

Policy booleans:
allow_ybind active
<... output truncated ...>
```

The output above was truncated for the sake of brevity. The **sestatus** command can be run with the **-v** switch to view security context information relevant to system files and processes.

Changing Policies

There are several ways to change policies; two will be discussed here. The first is an example of a procedure that can be used to select the policy and set the operating mode manually:

- 1) In `/etc/sysconfig/selinux` set **SELINUXTYPE=strict** and **SELINUX=permissive**. This will select the strict policy and set the mode to permissive at the next reboot. (The system could hang if the mode is not set to permissive.)
- 2) Type **touch /.autorelabel** from the command line. This file is read by `/etc/rc.sysinit` during boot. If it exists, the filesystem is relabeled early in the boot process.⁶³
- 3) Log in as a normal user and assume root privileges. Type **setenforce 1** from the command line to enter enforcing mode. Run the **sestatus** command to ensure the system is running in enforcing mode.
- 4) Reboot to ensure that processes start normally as a precautionary measure.⁶⁴

A new policy could be created using one of the others as a template. Running **cp -R /etc/selinux/strict /etc/selinux/tester** and setting `INSTALLDIR=$(DESTDIR)/etc/selinux/test` in the *Makefile* creates a new policy called *tester*.

Typing **system-config-securitylevel** in FC3 will launch a GUI tool to assist in configuring SELinux and Netfilter as shown in Figure 3.

⁶³ Coker, Faye (2004)

⁶⁴ Historically in Fedora Core 2 and other SELinux implementations there were interoperability problems with certain graphical environments. Things have gotten much better in Fedora Core 3.

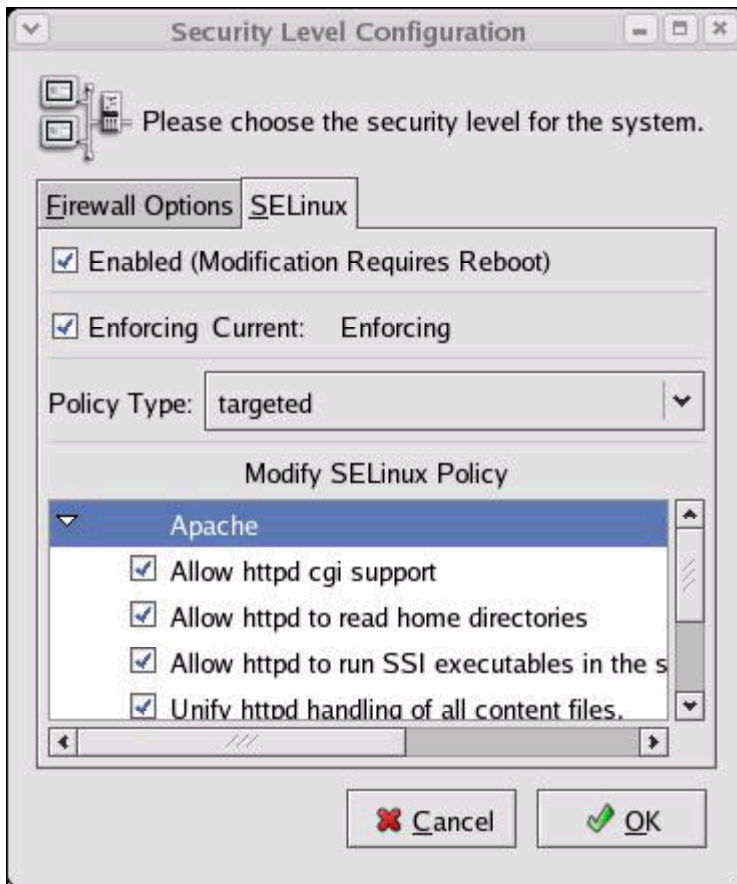


Figure 3: system-config-securitylevel

Deselecting the *Enabled* checkbox will completely disable SELinux. Deselecting the *Enforcing* checkbox will set the operating mode to permissive. The items under *Modify SELinux Policy* are policy-specific boolean values that may be set for common customizations. There are booleans available for tweaking the strict policy for the following services: Apache, Cron, FTP, Mozilla, DNS, NFS, NIS, SSH, spam assassin, Xserver, and various user privileges. Lastly, changing the policy type from targeted to strict yields the self-explanatory message in Figure 4 below.

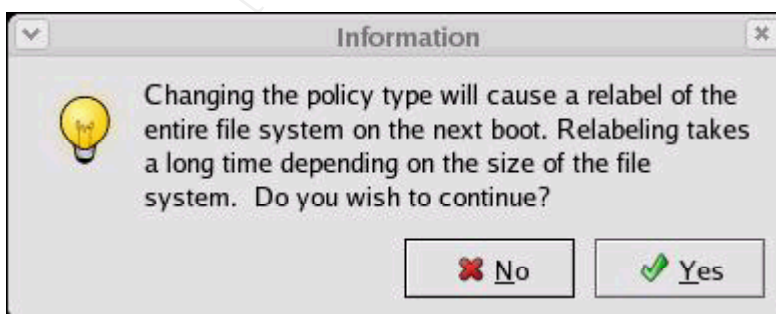


Figure 4: Security Level Confirmation

Security Policy Administration

Like programs, security policy is compiled from many source files into a single binary file. The two digit numeric suffix at the end of the **policy.xx** file indicates the version of the policy language. The default binary policy file for FC3 is displayed below:

```
[root@localhost policy]# pwd
/etc/selinux/targeted/policy
[root@localhost policy]# file policy.18
policy.18: SE Linux policy v18 6 symbols 7 ocons
```

The *sysadm_r* role must be assumed in order to modify the security policy. The main directory tree for the policy is */etc/selinux/targeted/src/policy*, where *targeted* is the name of the policy. The various source files are concatenated into the *policy.conf* file. The **checkpolicy** command is used to compile the policy from the *policy.conf* file into object form. The **load_policy** command is used to load a newly compiled policy into the kernel. Both the *checkpolicy* and *load_policy* commands are called from the *Makefile*. The default target for **make** is *install*. The descriptions of the primary targets below are taken from the *Makefile* and *README*:

clean	enables 'dontaudit' rules
enableaudit	disables 'dontaudit' rules
install	compile and install the policy configuration
load	compile, install, and load the policy configuration
reload	compile, install, and load/reload the policy configuration
relabel	relabel filesystems
policy	compile the policy only
check-all	check policy files in <i>domains/program/unused</i>
checkunused/FILE.te	check a single FILE from <i>domains/program/unused</i>

The following describes some of the policy configuration files located in the base directory of the policy source (for example */etc/selinux/targeted/src/policy*):

README	useful information about the files in the policy directory tree
tmp/all.te	defines the Type Enforcement configuration (auto-generated)
flask	the files in this directory define sids, classes and access vectors used by the kernel
macros	global, user, admin, and program macros
type	general types – not associated with a particular domain
domains	user, administrator, program, kernel and

assert.te	miscellaneous domains defined in .te files
rbac	assertions evaluated after the entire TE configuration
	role based access control
	configuration
mls	multi-level security configuration ⁶⁵
users	users recognized by the security policy
constraints	used to restrict changes in identity or role
initial_sid_contexts	security context for each initial SID
fs_use	defines labeling behavior for supported file systems
genfs_contexts	defines security contexts for non-persistent file
	system types like proc, vfat, iso9660, ntfs, etc.
file_contexts	security contexts for persistent files. The program
	subdirectory includes contexts such as <i>mozilla.fc</i> ,
	<i>snort.fc</i> , <i>ssh.fc</i> , etc.

Policy may be tuned by using macros in the */etc/selinux/targeted/src/policy/tunable.te* file or by setting booleans. The M4 macros used by the *tunable.te* file should look familiar to sendmail administrators. Various macros and booleans are set for common tweaks. Available booleans may be viewed in the */selinux/booleans* directory. As previously discussed, booleans may be conveniently set through *system-config-securitylevel* on Fedora installations.

Logging

During initial deployment of SELinux, much of the work is put into browsing the logs for AVC denied entries. AVC messages will appear in */var/log/messages* courtesy of syslog.⁶⁶ Once some experience is gained in using these messages to troubleshoot, policy customization is quite easy.

Format of AVC messages:

perm op pid exe path dev ino scontext tcontext tclass

perm	either <i>granted</i> or <i>denied</i>
op	one of nearly two hundred operations such as <i>read</i> , <i>load_policy</i>
pid	the process that caused the message
exe	executable that spawned the process
path	what the process is trying to access (could also be <i>name</i>)
dev	related device
ino	inode number of <i>path</i>
scontext	source security context; context of the source process

⁶⁵ If SELinux and checkpolicy were built with MLS enabled, the policy can be configured with MLS by setting *MLS = y* in the Makefile. MLS support is still considered experimental. It is not enabled by default.

⁶⁶ AVC messages can also appear in */var/log/dmesg*. These are the messages that occurred during the boot process; however they are also duplicated in */var/log/messages*.

tcontext target security context; context of the target object
tclass target security class

One helpful way to think of these messages is: **pid** was **perm** **permission** to perform **op** on **class/path/name** because **pid**'s context of **scontext** was **perm** access to **class/path/name** in the context **tcontext**.

A sample message is shown below:

```
Mar 25 01:51:17 localhost kernel: audit(1111733477.244:0): avc:
denied { relabelfrom } for pid=5509 exe=/bin/mount
scontext=user_u:user_r:user_mount_t
tcontext=system_u:object_r:dosfs_t tclass=filesystem
```

Plugging the values from the sample message into the introductory formula yields:

pid 5509 was *denied* permission to perform *relabel* on *filesystem*
because pid 5509's context of *user_u:user_r:user_mount_t* was *denied*
access to *filesystem* in the context *system_u:object_r:dosfs_t*.

It appears that a process run by a typical user was denied permission to relabel or mount a filesystem. The **audit2allow**⁶⁷ utility can be used to suggest policy rule for prevent denied AVC messages. The **seaudit** utility can be used to analyze the policy file for statements that may relate to an AVC message. Use of these tools to tweak the policy is demonstrated in latter sections.

For initial policy troubleshooting, SELinux maintains a cache and logs each denial entry only once in permissive mode. The cache can be cleared by reloading the security policy (*make reload*) or by switching to enforcing mode and then back to permissive. SELinux developers hope to implement a logging facility designed specifically for SELinux rather than relying on syslog.⁶⁸

Policy Tweaking

Essentially any program configured to run in the strict policy needs two types of configuration files. The file context (fc) files specify how objects such as files and directories are labeled. For the strict policy, these files are located in the */etc/selinux/strict/src/policy/file_contexts/program* directory. The type enforcement (te) files specify the domains, allowed transitions, access vector rules and role authorizations. These are essentially the meat of the security policy for each individual program. For the strict policy, they are located in */etc/selinux/strict/src/policy/domains/program*.

⁶⁷ The audit2allow utility is a Perl script.

⁶⁸ McCarty, (2004); p. 91

With the strict policy installed, some applications, especially those not natively supported by FC3, will need the policy customized in order to function properly. To demonstrate the types of issues one is likely to encounter with the strict policy, a simple example is in order.

A laptop loaded with FC3 was configured with the strict policy in enforcing mode. Afterwards it was discovered that the USB drive was not automatically mounted after logging in using the 'pike' username. Although this problem is just a minor annoyance, correcting it will provide an excellent introductory example of policy tweaking.

1) The **tail -f** command was run against `/var/log/messages` as the USB drive was being connected. The following AVC entry was noted:

```
Mar 23 17:33:07 localhost kernel: audit(1111617187.068:0): avc:
denied { search } for pid=4799 exe=/bin/mount name=console dev=hda1
;ino=507953 scontext=user_u:user_r:user_mount_t
tcontext=system_u:object_r:pam_var_console_t tclass=dir
```

2) The noted AVC entry was copied and pasted into a file named 'file.' The **audit2allow** script was run on *file* to determine a recommended AV rule.

```
[root@localhost pike]# audit2allow -i file
allow user_mount_t pam_var_console_t:dir { search };
```

The *audit2allow* utility looks for denied rules from specified input files and suggests rules that would allow the denied operations. These rules can be added to the policy to permit a specific required function that the policy is preventing. Caution should be used when using *audit2allow*, because it indiscriminately suggests *allow* rules for all AVC *denied* entries it finds.⁶⁹ Each *allow* rule weakens the access control policy a tiny bit. It only makes sense to add an allow rule if the function it permits is absolutely necessary for business. Excessive use of *audit2allow* can weaken the security policy considerably. The principle of least privilege should always govern access control decisions.

⁶⁹ McCarty (2004); pg 181

3) The previously suggested AV rule was added to `/etc/selinux/strict/src/policy/domains/program/mount.te` using the **sepcut** tool⁷⁰ as shown in Figure 5.⁷¹

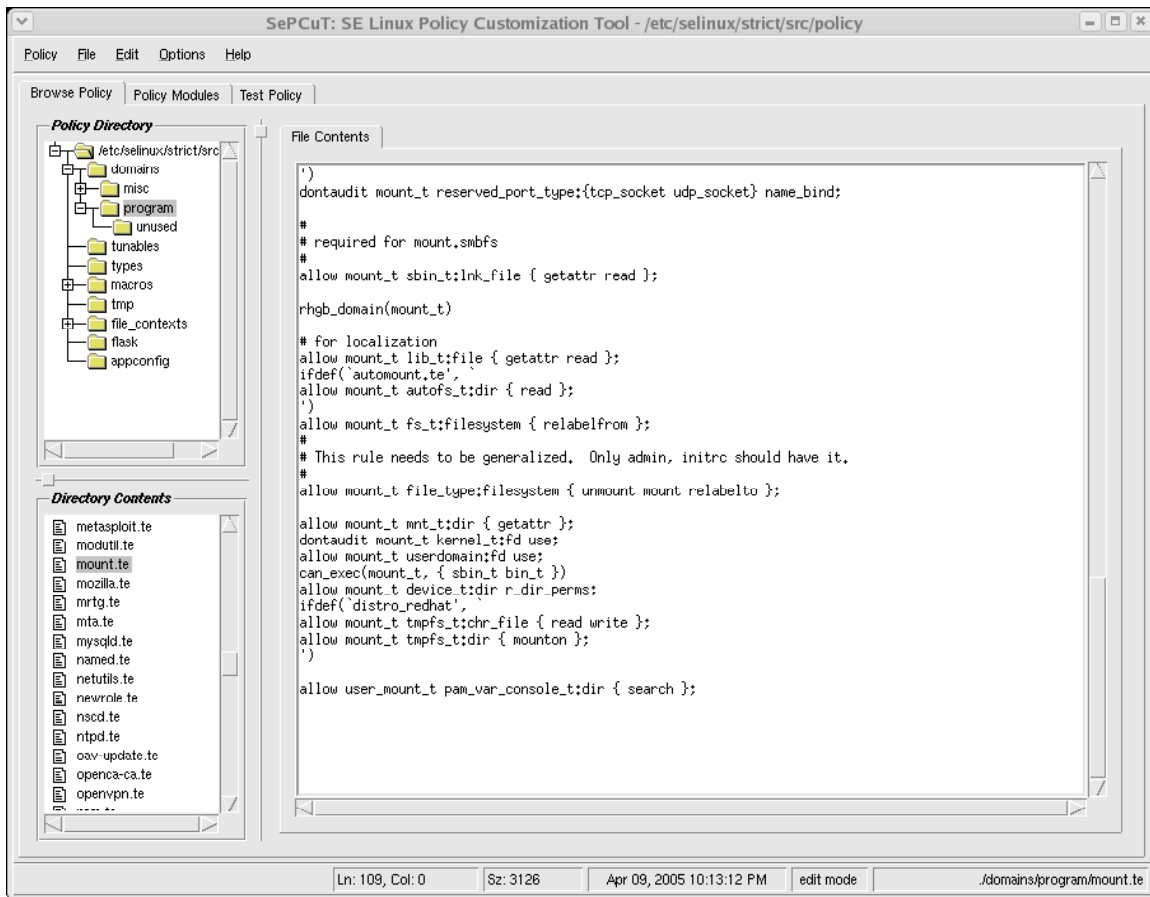


Figure 5: Sepcut – mount.te

⁷⁰ Tresys Technology, “SE Linux Policy Customization and Editing Tool Help File”

⁷¹ While the steps completed with sepcut can easily be accomplished using *vi*, *make policy*, and *make reload*, the GUI makes it easier to keep track of things when working on policy files.

4) After editing the TE file, the *Test Policy* tab was selected. Clicking the *Test Policy* button caused the policy to be compiled and checked for errors. Finally, clicking the *Load Policy* button caused the policy to be loaded into the kernel, producing the output shown in Figure 6.

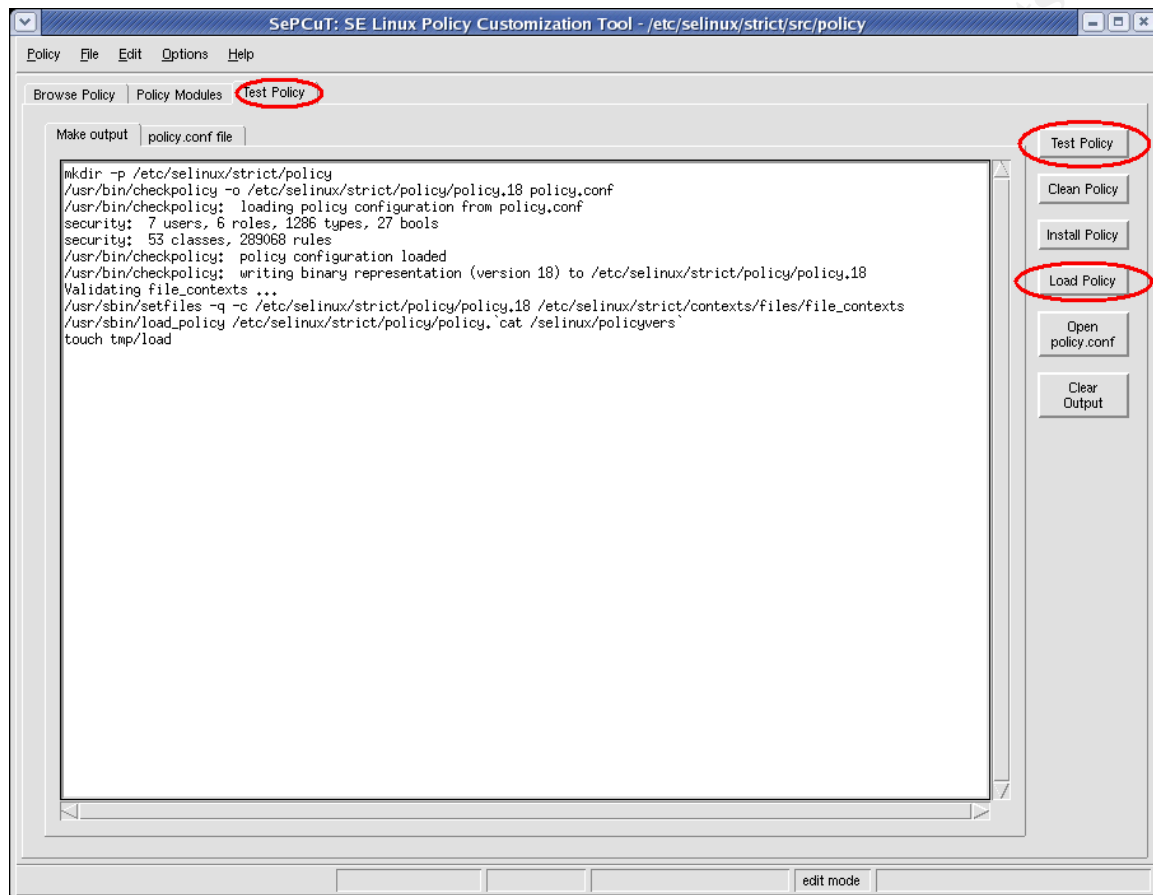


Figure 6: Sepcut – Test Policy

5) The USB drive was reseated, and the following AVC entry was noted:

```
Mar 25 01:48:39 localhost kernel: audit(1111733319.021:0): avc:
denied { getattr } for pid=4771 exe=/bin/mount
path=/var/run/console/pike dev=hda1 ino=512872
scontext=user_u:user_r:user_mount_t
tcontext=system_u:object_r:pam_var_console_t tclass=file
```

6) The observed entry was appended to the end of 'file' in the current working directory. The *audit2allow* utility was run once again.

```
[root@localhost pike]# audit2allow -i file
allow user_mount_t pam_var_console_t:dir { search };
allow user_mount_t pam_var_console_t:file { getattr };
```

7) The recommended allow rule was appended to *mount.te* and the policy was

reloaded. Upon reseating the USB drive, the following AVC denied entries were observed:

```
Mar 25 01:51:17 localhost kernel: audit(1111733477.244:0): avc:
denied { relabelfrom } for pid=5509 exe=/bin/mount
scontext=user_u:user_r:user_mount_t
tcontext=system_u:object_r:dosfs_t tclass=filesystem
Mar 25 01:51:17 localhost kernel: audit(1111733477.358:0): avc:
denied { relabelfrom } for pid=5509 exe=/bin/mount
scontext=user_u:user_r:user_mount_t
tcontext=system_u:object_r:dosfs_t tclass=filesystem
```

8) The output was appended to 'file' and *audit2allow* was run once more:

```
[root@localhost pike]# audit2allow -i file
allow user_mount_t pam_var_console_t:dir { search };
allow user_mount_t pam_var_console_t:file { getattr };
allow user_mount_t dosfs_t:filesystem { relabelfrom };
```

9) The suggested rule was appended to *mount.te* and the policy was reloaded. Upon reseating the USB drive, it automatically mounted with no AVC denied messages generated.

There are many methods that can be used to customize and tweak policy. It's more of an art than a science. Those experienced with day-to-day SELinux administration may prefer working from the command line. For those less experienced with SELinux, GUI tools like *seppcut*, *seaudit*, *seuserx*, and *apol* can help reinforce how SELinux works.

Adding Users

Adding users is a common function. The standard *useradd*, *usermod*, and *userdel* utilities do not modify the SELinux policy after performing their functions. In order for the user to be recognized by the security policy, a newly created user would need to be manually added to the */etc/selinux/strict/src/policy/users* file, and the policy would need to be recompiled and reloaded.

User management tasks can be completed with greater ease using the *seuser*⁷² tools. The *seuseradd*, *seusermod*, and *seuserdel* command line utilities can be used in the same ways as their non-SELinux-enabled counterparts. Below is an example of adding an administrative user 'test' using the *seuserx* graphical utility.

1) Before running *seuserx*, login as the root user and assume the *sysadm_r* role.

```
[root@localhost ~]# newrole -r sysadm_r
Authenticating root.
Password:
```

⁷² Tresys, "SELinux User Manager Help File"

```
[root@localhost ~]# seuserx
```

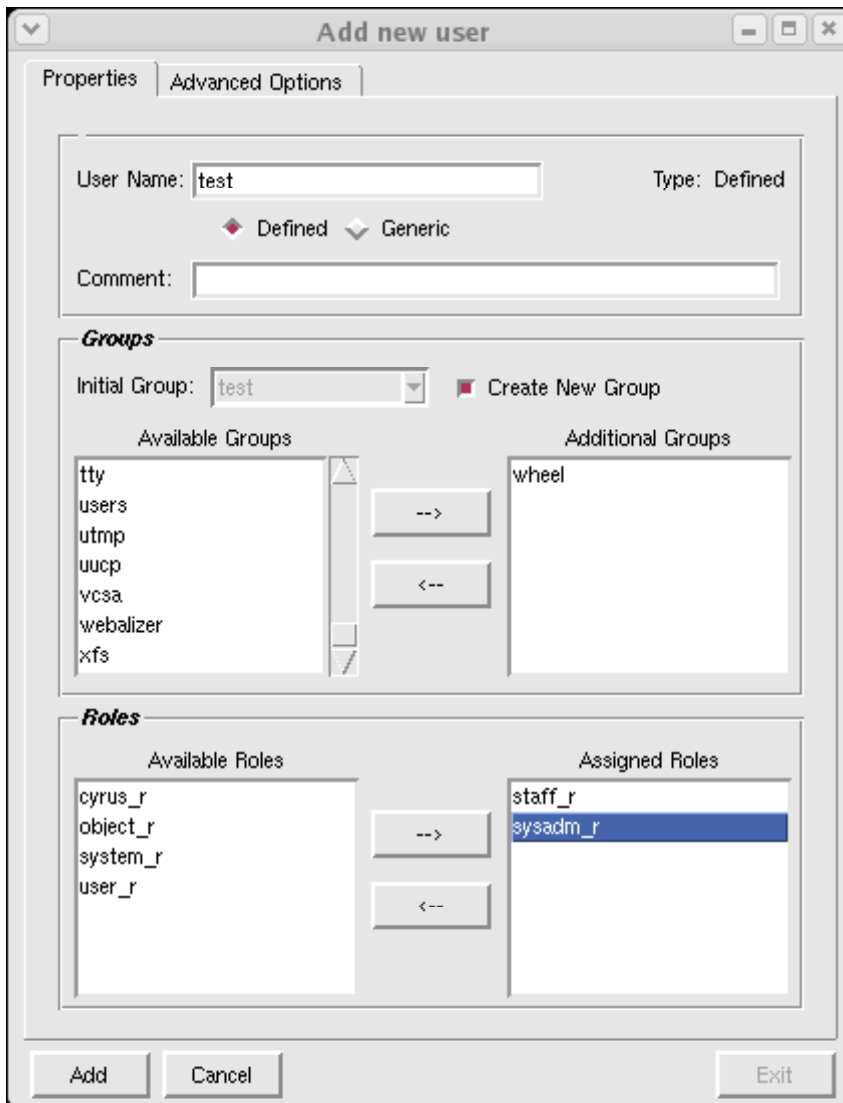
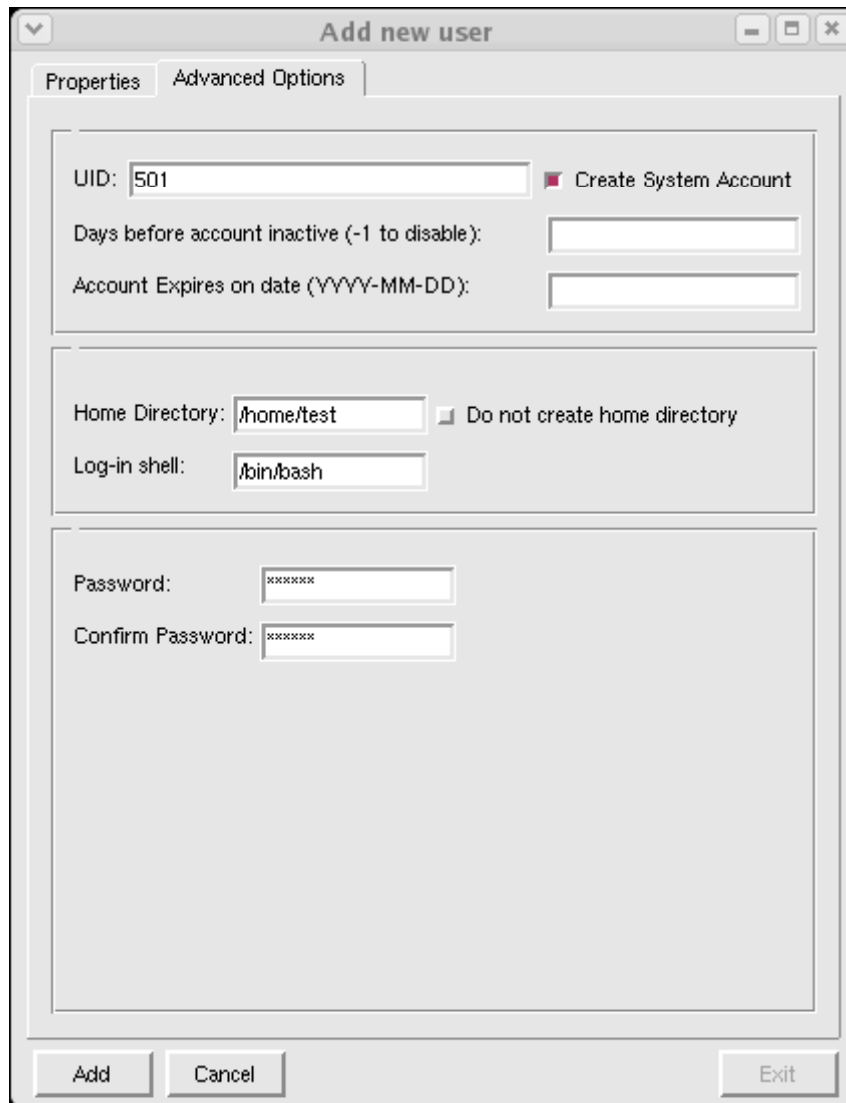


Figure 7: Seuserx – Add new user - Properties

2) After clicking Add from the main window, the appropriate values can be filled in under the *Properties* tab as shown in Figure 7. A generic account is a Linux user account not defined in the SELinux security policy. The roles assigned to the user are *staff_r*, and *sysadm_r*.⁷³

⁷³ The *staff_r* role is the default. The *sysadm_r* role is required for policy administration.

3) Figure 8 shows the *Advanced Options* tab. These options are largely self-explanatory. Note that the *Create System Account* checkbox would be deselected for user accounts existing on the system but not included in the policy.



The screenshot shows a window titled "Add new user" with two tabs: "Properties" and "Advanced Options". The "Advanced Options" tab is selected. It contains several input fields and checkboxes:

- UID:** A text box containing "501".
- Create System Account:** A checkbox that is currently checked.
- Days before account inactive (-1 to disable):** An empty text box.
- Account Expires on date (YYYY-MM-DD):** An empty text box.
- Home Directory:** A text box containing "/home/test".
- Do not create home directory:** A checkbox that is currently unchecked.
- Log-in shell:** A text box containing "/bin/bash".
- Password:** A text box with masked characters (xxxxxx).
- Confirm Password:** A text box with masked characters (xxxxxx).

At the bottom of the window, there are three buttons: "Add", "Cancel", and "Exit".

Figure 8: Seuserx – Add new user – Advanced Options



4) After creating the new user, click the *Exit* button from the *Add new user* window. After all users are added to the policy source, clicking the *Exit* button on the main *SE Linux User Manager* window in Figure 9 causes the policy to be updated before the window is closed.

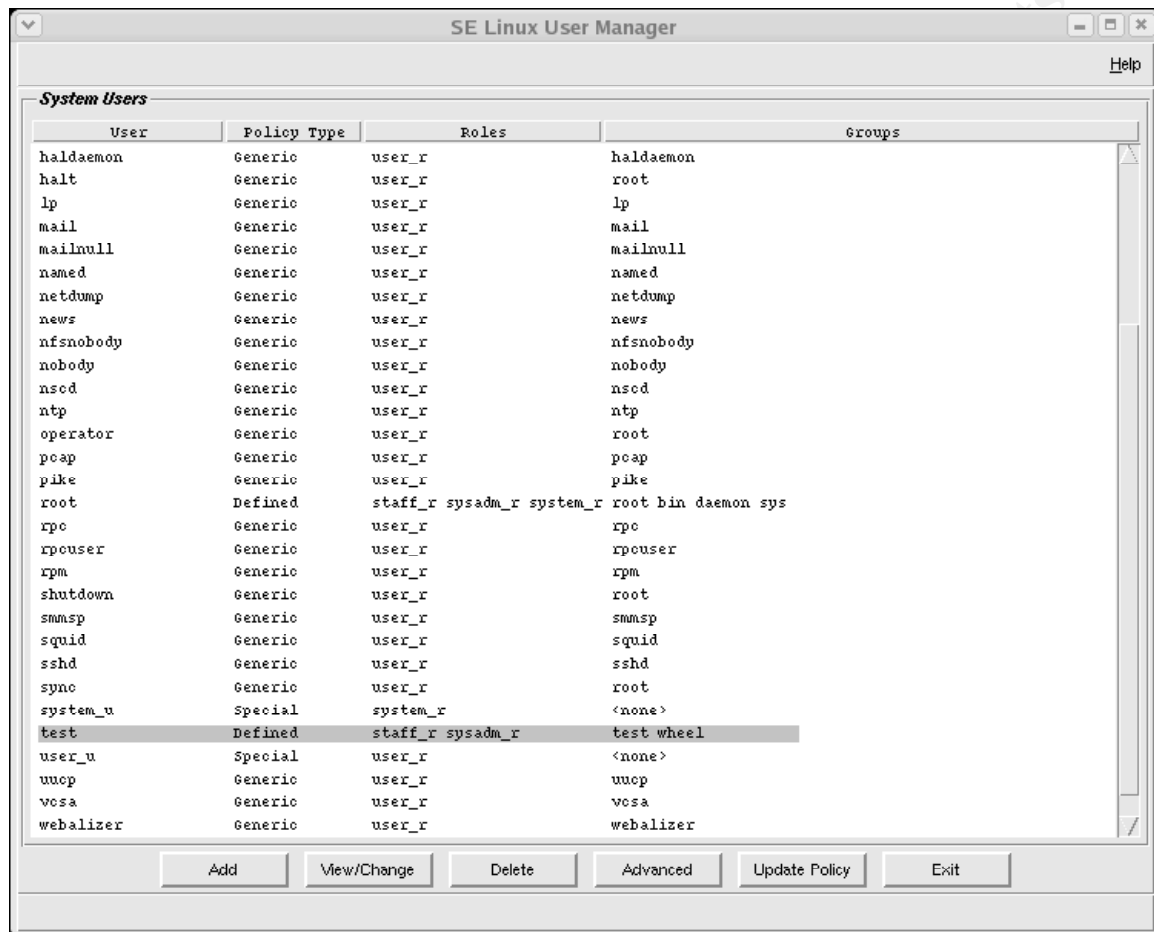


Figure 9: Seuserx – Main Window

5) After the account is added it can be tested.

```
root@localhost policy]# su - test
Your default context is test:staff_r:staff_t.

Do you want to choose a different one? [n]
[test@localhost ~]$ id -Z
test:staff_r:staff_t
[test@localhost ~]$ newrole -r sysadm_r
Authenticating test.
Password:
Warning! Could not open /dev/pts/0.
[test@localhost ~] id -Z
test:staff_r:staff_t
[test@localhost ~] cd /etc/selinux/strict/src
bash: cd: /etc/selinux/strict/src: Permission denied
[test@localhost ~]
```

6) Note above that the user 'test' is lacking permission to open the console. In order to accurately test the new user account it should be used to login. Once the 'test' account is actually used to login, it may assume the `sysadm_r` role.

```
[test@localhost ~]$ newrole -r sysadm_r
Authenticating test.
Password:
[test@localhost ~]$ id -Z
test:sysadm_r:sysadm_t
[test@localhost src]$ cd /etc/selinux/strict/src
[test@localhost src]$
```

Seaudit

The ***seaudit*** tool⁷⁴ can be used to analyze the policy for statements that may be relevant to specific AVC messages. To illustrate an example using this tool, recall the AVC denied message received in step 5 of the previous section. The output below shows a *grep* command that could be used to locate the entry and launch of the *seaudit* utility.

```
[root@localhost ~]# grep '/dev/pts/0' /var/log/messages
grep: /var/log/messages: Permission denied
[root@localhost ~]# newrole -r sysadm_r
Authenticating root.
Password:
[root@localhost ~]# grep '/dev/pts/0' /var/log/messages
Mar 29 17:39:23 localhost kernel: audit(1112135963.964:0): avc:
denied { ioctl } for pid=22934 exe=/bin/bash path=/dev/pts/0
(deleted) dev=devpts ino=2 scontext=root:staff_r:staff_t
tcontext=root:object_r:sysadm_devpts_t tclass=chr_file
[root@localhost ~]# seaudit
```

⁷⁴ Tresys, "Audit Log Analysis Tool for Security Enhanced Linux."

AVC messages displayed within *seaudit* are sortable by fields including source type, target type, class, date, permission and executable. Figure 10 is sorted by permission. There is only one denied *ioctl* action in this file. The *Query policy* button launches a tool in a separate window that allows for searching of statements related to selected entry⁷⁵ in the combined policy source file⁷⁶ using regular expressions.⁷⁷

Hostname	Message	Date	Source Type	Target Type	Object Class	Permissi	Executable	Other
localhost	Denied	Mar 29 17:16:28	user_t	unlabeled_t	dir	getattr	/bin/bash	dev=hda1
localhost	Denied	Mar 29 17:30:24	user_t	unlabeled_t	dir	getattr	/bin/bash	dev=hda1
localhost	Denied	Mar 29 17:12:05	useradd_t	staff_home_dir_t	dir	getattr	/usr/sbin/us	dev=hda1
localhost	Denied	Mar 29 17:39:23	staff_t	sysadm_devpts_t	chr_file	ioctl	/bin/bash	dev=devpts
localhost	Granted	Mar 29 17:04:15	load_policy.	security_t	security	load_policy	/usr/sbin/lo.	
localhost	Granted	Mar 29 17:05:11	load_policy.	security_t	security	load_policy	/usr/sbin/lo.	
localhost	Granted	Mar 29 17:10:09	load_policy.	security_t	security	load_policy	/usr/sbin/lo.	
localhost	Granted	Mar 29 17:10:11	load_policy.	security_t	security	load_policy	/usr/sbin/lo.	
localhost	Granted	Mar 29 17:12:30	load_policy.	security_t	security	load_policy	/usr/sbin/lo.	
localhost	Granted	Mar 29 17:20:08	load_policy.	security_t	security	load_policy	/usr/sbin/lo.	
localhost	Granted	Mar 29 17:22:22	load_policy.	security_t	security	load_policy	/usr/sbin/lo.	
localhost	Granted	Mar 29 17:22:24	load_policy.	security_t	security	load_policy	/usr/sbin/lo.	
localhost	Granted	Mar 29 17:24:23	load_policy.	security_t	security	load_policy	/usr/sbin/lo.	
localhost	Granted	Mar 29 17:28:16	load_policy.	security_t	security	load_policy	/usr/sbin/lo.	
localhost	Granted	Mar 29 17:28:22	load_policy.	security_t	security	load_policy	/usr/sbin/lo.	
localhost	Granted	Mar 29 17:28:24	load_policy.	security_t	security	load_policy	/usr/sbin/lo.	
localhost	Granted	Mar 29 17:30:44	load_policy.	security_t	security	load_policy	/usr/sbin/lo.	
localhost	Granted	Mar 30 14:02:53	load_policy.	security_t	security	load_policy	/usr/sbin/lo.	

Figure 10: Seaudit

Launching the *Query Policy* function automatically populates the *Source type*

⁷⁵ Only one entry may be selected at a time.

⁷⁶ */etc/selinux/strict/src/policy/policy.conf*.

⁷⁷ Friedl (2002)

regular expression, *Target type regular expression*, and *Object Class* fields based on the entries previously selected. Clicking the *Query Policy* button in this new window starts a search through the policy file⁷⁸ for related rules and produces output similar to that shown in Figure 11.

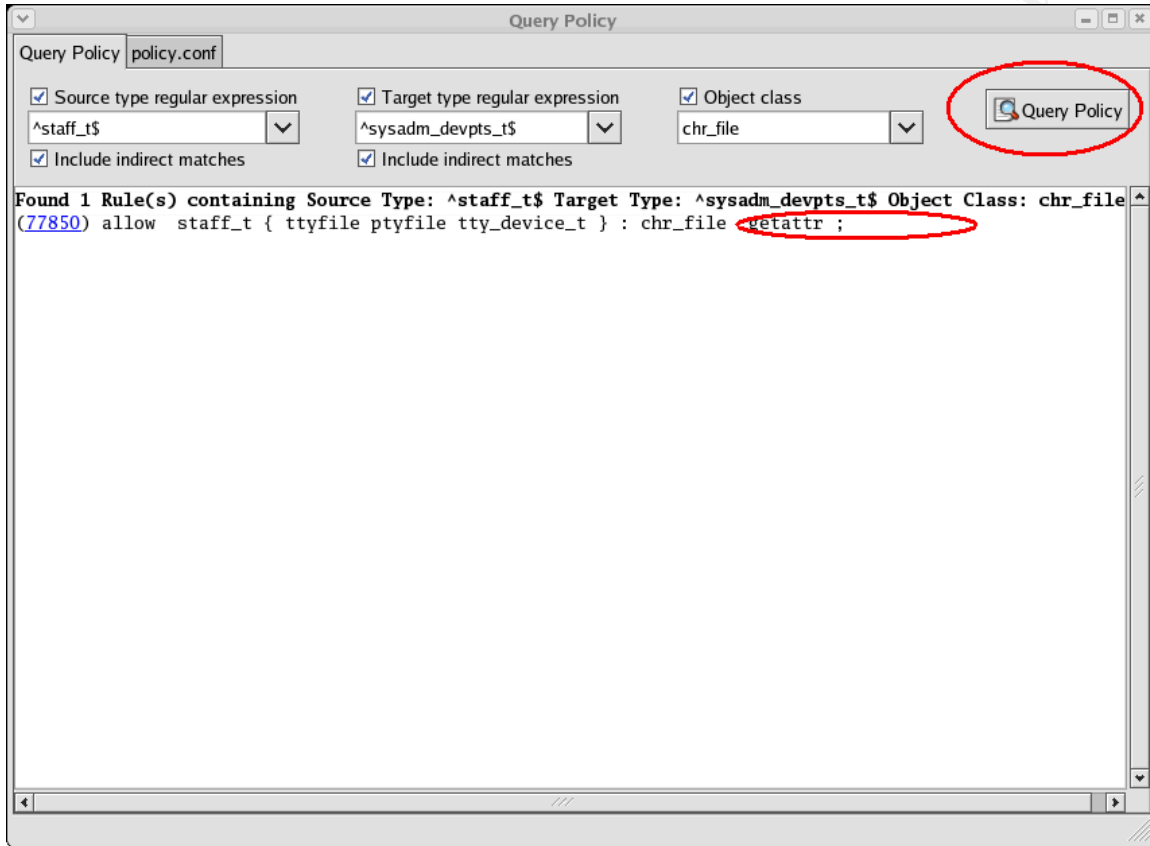


Figure 11: Seaudit – Query Policy

As shown line 77,850 of policy.conf allows processes in the *staff_t* domain to perform the *getattr* operation on objects in the *chr_file* class of the *ttyfile*, *ptyfile*, or *tty_device_t* types. Perhaps adding the *ioctl* operation and the *sysadm_devpts_t* types would prevent the denied message. If the *test* user actually required access to root's console this might be an option, however; the principle of least privilege dictates otherwise. AVC denied entries can be a good thing.

⁷⁸ In the author's case the concatenated policy.conf file is 227,244 lines of code.

Miscellaneous Administration

Domains can be created to support new applications or to address specific permission problems on installed applications. A domain for a program within the strict policy would be defined in the `/etc/selinux/strict/src/policy/domains/` directory in a file such as `domain_name.te`. Roles can be created to address system and organizational operating requirements. However, according to McCarty, “It’s generally not necessary to create a new SELinux role.”⁷⁹ Unlike domain declarations, which only appear in one file for a given domain, role declarations appear in the `.te` file for each domain they are authorized to enter. Details on creating domains, types, and roles are beyond the scope of this introductory text.⁸⁰

The `seinfo` command can be used to provide detailed information about the current SELinux policy. To view each type and its associated attributes, the `-t` and `-x` switches can be used:

```
[root@localhost contexts]# seinfo -t -x |more
```

```
Types: 1287
      device_t
      file_type
      dev_fs
      null_device_t
      device_type
      dev_fs
      mlstrustedobject
      zero_device_t
<output truncated...>
```

The `seinfo` command supports the `-c` (classes), `-t` (types), `-a` (attributes), `-r` (roles), `-u` (users), `-b` (booleans), `-i` (initial-sids), `-A` (all), `-x` (expand), `-s` (stats), `-h` (help), and `-v` (version) switches. For example, the `-s` switch can be used:

```
[root@localhost ~]# seinfo -s
```

```
Statistics for policy file:
/etc/selinux/strict/src/policy/policy.conf
Policy Version: v.18
Policy Type: source
Classes:          53      Permissions:      192
Types:            1287    Attributes:       80
Users:            6      Roles:           6
Booleans:         27     Cond. Expr.:     30
Allow:            33173   Neverallow:      55
Auditallow:       14     Dontaudit:       3622
Type_trans:       1358   Type_change:     17
Role_allow:       8      Role_trans:      97
Initial SIDs:     27
```

⁷⁹ McCarty (2004); pg 168

⁸⁰ Smalley (2005); pp. 27-32 – provides details on creating domains, types, and roles

The **apol** utility can be used for advanced policy analysis and customization. Forward and reverse domain transition analyses allow the user to walk through a tree of domains that may be transitioned into from a given starting domain. Figure 12 shows a forward domain transition analysis in progress.

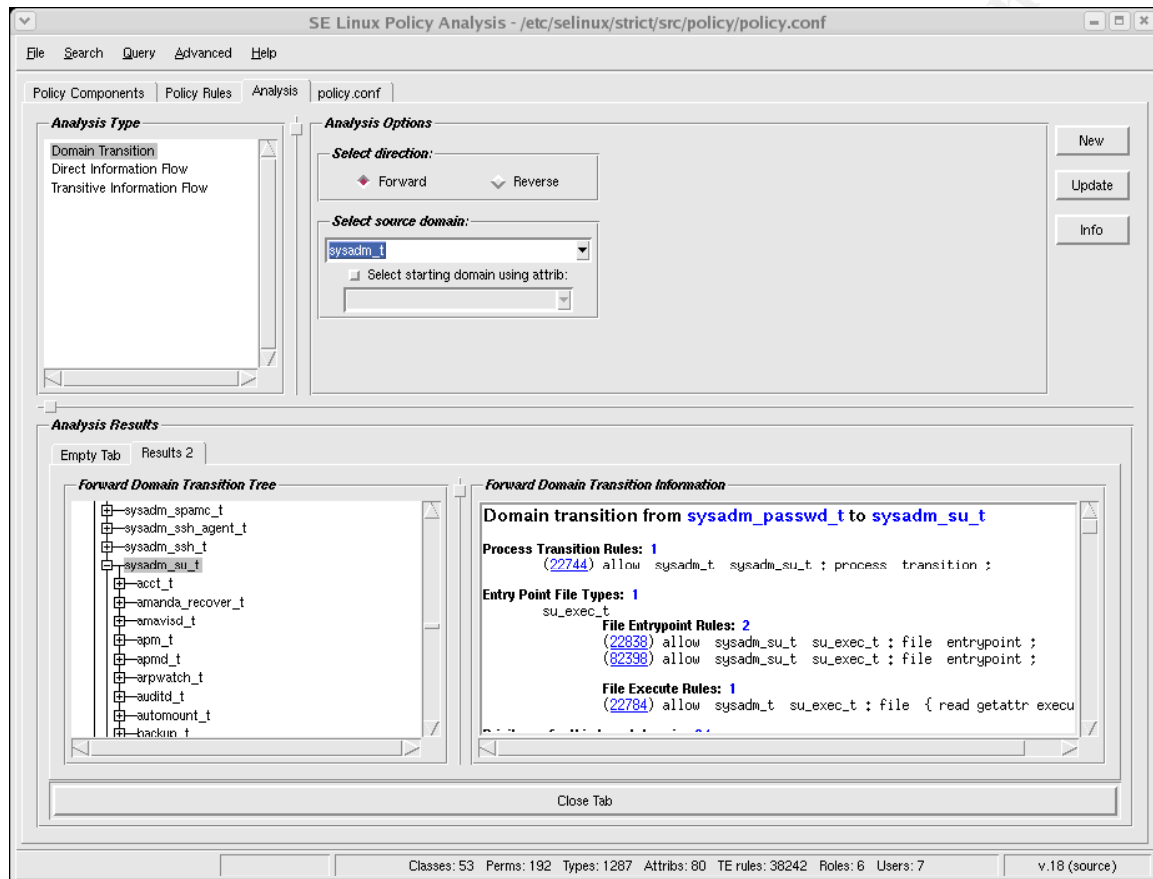


Figure 12: Apol – Domain Transition Analysis

Transitive (or indirect) and direct information flow analyses can be used to determine how objects in one domain can be accessed by subjects in another domain. Rule analysis functionality (under the *Policy Rules* tab) allows the policy to be searched for rules relating to selected types, attributes, AV types, classes, booleans, and roles. The *Policy Components* tab allows *apol* to be used like *sepcut* to dissect and analyze policy components.⁸¹ The *Advanced* menu contains an option to edit the permission map. Further specifics regarding the use of *apol* for policy analysis are beyond the scope of this introduction.⁸²

⁸¹ Whereas *sepcut* can traverse the source directory, *apol* can only be used on the *policy.conf* or *policy.xx* file. Although *apol* may be used on the *policy.xx* file, it is recommend to compile the policy first and analyze the *policy.conf* file.

⁸² RedHat, "Using apol for Policy Analysis;" Additional information may also be found under the */usr/share/doc/setools-1.4.1* directory in *apol_help.txt*, *dta_help.txt*, and *iflow_help.txt*.

Future Development and Conclusions

Future Development

As evidenced by the SELinux mailing list,⁸³ development of the OS is accelerating. Many are just getting their first exposure to SELinux. As interest increases, development of SELinux and SELinux-enabled applications will continue to increase.

A cursory exploration of Fedora Core 4 Test Candidate 1 reveals a few new tools and utilities including *genhomedircon*, *open_init_pty*, *seaudit-report*, *sediff*, and *sediffx*.⁸⁴ In six months version numbers of packages in the 1.17.x range have incremented to 1.22.x. So have the version numbers of the equivalent NSA core source libraries and components. The version number of the *setools* package has incremented to 2.0.0-1 from 1.4.1-5. A developer's package for *setools*, *setools-devel-2.0.0-1.i386.rpm*, makes its debut in Fedora Core 4. Clearly, development of SELinux is moving rapidly.

Faye Coker provides the following comments regarding future development:

Development work is currently underway on making the strict policy more flexible and on making defaults that will work more easily out of the box. Work also is being done on Security enhance X, where the aim is to have control over the X sessions so that, for instance, a hostile X program can't interfere with other X programs on the display. Examples of this are programs not being able to sniff the keyboard and seeing windows or concealing windows without the X user knowing.⁸⁵

Currently SELinux is not capable of kernel-level auditing like Solaris, which uses Sun's Basic Security Module (BSM). However, development is underway on this front as well. Secure Auditing for Linux is funded by DARPA to develop a Common Criteria / TCSEC C2-equivalent kernel-level auditing package for Red Hat Linux.⁸⁶ Adding BSM-like auditing capabilities to SELinux could yield a free "trusted" operating system.⁸⁷

Many exciting advancements are occurring within the SELinux community. The inaugural SELinux Symposium was held in March of 2005.⁸⁸ Russell Coker

⁸³ National Security Agency, "SELinux Mailing List: by date."

⁸⁴ See Appendix B for explanations of these utilities.

⁸⁵ Coker, Faye (2004)

⁸⁶ Secure Auditing for Linux

⁸⁷ According to the NSA SELinux FAQ, a trusted operating system is an operating system that provides sufficient support for Multi Level Security. MLS support for SELinux is still considered experimental. MLS Systems also have BSM-like auditing requirements.

⁸⁸ SELinux Symposium

even has SELinux patches for the ARM-based IPAQ PDA.⁸⁹

© SANS Institute 2000 - 2005, Author retains full rights.

⁸⁹ Coker, Russell “SELinux”

Conclusions

Commercial “trusted” operating systems, such as Trusted Solaris and other MAC-based systems, are not so prevalent today due to cost. Because the licenses are expensive, developers are less likely to experiment and develop new applications for these systems. Few people want to spend a couple thousand dollars just to play on an OS that is not widely deployed. SELinux is freely available, enabling motivated individuals to familiarize themselves with mandatory access controls. Because the limiting factors have been removed, SELinux and other MAC-based systems, such as TrustedBSD,⁹⁰ could receive increased consideration as secure hosting platforms in the future.

SELinux can be used to reduce exposure to all vulnerabilities, including zero-day vulnerabilities. SELinux is an excellent operating system for bastion hosts on a service network. Because the configuration of these systems should change rarely (excluding patch installation) the burden of security policy should be manageable. SELinux is capable of mitigating the risk associated with zero-day vulnerabilities because it truly enforces the principle of least privilege.

For those seeking more information, Appendix C is a comprehensive list of references. The whitepapers available on NSA’s website provide complete coverage of SELinux implementation and theory. Other particularly useful references include Bill McCarty’s book, RedHat’s SELinux Guide, Fedora’s SELinux FAQ, and Russell Coker’s SELinux page.

⁹⁰ TrustedBSD.org

Appendix A: Glossary

access decision – determines whether permission is granted based on a pair of SIDS, the class of the object, and the set of requested permissions.

access vector – a bitmap that is a set of permissions within a class; controls access to objects within

access vector cache (AVC) – caches access decision computations to improve performance of the security server

attribute – security-relevant information associated with a subject or object

auditallow – an access vector that audits an action allowed by the security policy

auditdeny – an access vector that logs an action denied by the security policy

Bell-LaPadula model – security model commonly associated with MAC. Also known for enforcing the simple security property and * (star) property

boolean – true/false values commonly used within *if* statements to test and tweak the policy

class – a logical grouping of objects

confinement – The confinement of an application to a unique security domain

discretionary access control (DAC) – traditional Linux access control model where users are allowed to set the permissions on the files they own

domain – security attribute; the space a process runs in; determines the access a process has; similar to type; label for a subject

domain and type enforcement (DTE) – access control type based on the Bell-LaPadula Model

file context – security context for files specified under `file_contexts` directory

identity – a security attributed associated with a subject or object that have the same name as the Unix user, but is distinctly separate; it determines what roles and domains can be used

identity based access control (IBAC) – access based on user identity

label – synonymous with security context

labeling decision – determines the security context of a newly-created object

Linux security modules (LSM) – an access control extension to the Linux kernel; SELinux is an LSM

mandatory access control (MAC) – OS-enforced access control using labels (domains and types) to enforce security policy

multi-level security (MLS) – multiple level security systems are used by the Department of Defense; possible levels include unclassified, confidential, secret, and top secret

object – anything that can be acted upon by subject; includes files and processes

object manager - the component of an access control mechanism that enforces the decision; synonymous with enforcer

policy – the collection of rules that are enforced by the system. Policy is

compiled from policy files

policy files – define security contexts and domains for the system

policy flexible – describes an OS that is capable of enforcing a wide variety of security policies

role – security attribute assigned to users that determines what domains they can enter or use

role based access control (RBAC) - access control through the use of roles and profiles

security context – a variable length policy-independent data type containing a set of labeling information of each object consisting of three components: identity, role, and type or domain; similar to label

security server – synonymous with decider; component that makes security policy decisions

security identifier (SID) – a fixed-size value that is associated with each labeled object; used by the security server to make for security decisions

subject - can perform an action on an object; usually a process or user

transition – determines the security context

trusted operating system – a system that provides sufficient support for MLS and meets certain other requirements such as Common Criteria / TCSEC C2 auditing requirements

type – security attribute; a logical grouping of objects; type is to files and directories as domain is to processes; label for an object

type enforcement (TE) – model in which subjects are associated with domains and objects are associated with types

user – sometimes used interchangeably with identity

Appendix B: Quick Reference

A quick reference of SELinux commands and declarations. Examples are given where syntax is not obvious. Explanations of examples given for examples that may not be self-explanatory.

Commands

These are the commands that are available in SELinux. Most support the **--help** option. Man pages are available for just a few of them.

apol – A GUI policy analysis tool from the setools-gui rpm

apol & (launches the apol tool)

audit2allow – A utility that analyzes AVC messages in syslog files and returns suggested rules that could permit currently denied actions

audit2allow /var/log/message > file (recommends allow rules to prevent denied messages and redirects output to 'file')

awish – from the setools-gui package; a TCL/TK wish interpreter with built-in support for setools libraries

chcon – changes security context of files

checkpolicy – policy compiler; checks and compiles the policy into a binary representation that can be loaded into the kernel

checkpolicy -d (loads policy configuration and enters debug mode)

chkcon – determines whether or not a context is valid; from libsepol-devel

chkcon /etc/selinux/targeted/policy/policy.18 root:system_r:unconfined_t

genhomedircon – replaces HOME_DIR and HOME_ROOT macros in .fc files with specific values

genpolbooleans – generates booleans for a new policy

getenforce – returns the current enforcing mode of SELinux

getfattr – get extended attributes⁹¹ of file system objects; can be used to see the security context of objects when SELinux isn't running

getfattr -m . -d /etc/xinetd.d/*

(gets extended attributes of all files in the /etc/xinetd.d directory)

getsebool – get the value of a boolean

getsebool -a (gets the value of all booleans)

fixfiles – fix the file security context database; relabel a files or filesystem

fixfiles check

id – modified to display security context information; new options: --context, -Z

id -Z (shows the security context of the current user)

load_policy – loads a policy into the kernel

load_policy policy.18 ../booleans

ls – modified to include security context information; new options: --context, --

⁹¹ "Extended attributes are a new feature in the 2.6 kernel for pairing name/value tags to files, similar to a database. Although a few new filesystems currently support them, they have not yet seen widespread use." Love (2003); p. 197

lcontext, **--scontext**, **-Z**
ls -aslZ (long listing showing security context info)
mount – modified to support labeling
open_init_pty – used by **run_init** to allocate pseudo terminals
newrole – transition to a new authorized role
newrole -r sysadm_r
 (transition to the **sysadm_r** role, which is required to administer policy)
restorecon – set security context on files
runcon – run a command with a specified security context
run_init – run an init script in proper context (/etc/rc.d scripts)
ps – modified to include security context information; new options: **--context**, **-Z**
ps -efZ |grep ssh
seaudit – GUI interface for analyzing syslog entries generated by SELinux
seaudit-report – provides customized audit reports in plain text or HTML
sediff – semantic policy difference tool; differentiates two policies
sediffx – graphical version of **sediff**
seinfo – displays summary info about the SELinux policy, including statistics about the number of roles, users, classes, booleans, attributes, etc.
seinfo -u -r (prints the users and roles in the current policy)
sepcut – A GUI policy editing tool from the **setools-gui** package
sesearch – search the policy for information on a particular type
sesearch -a -t default_t (searches for AV rules on **default_t**)
sestatus – (Fedora and Gentoo) status of SELinux
sestatus -v
setenforce – sets enforcement mode of SELinux; 0=permissive; 1=enforcing
setenforce 1 (immediately sets the SELinux mode to enforcing)
setfiles – set file security contexts; labels files or filesystems
setfiles /etc/selinux/strict/src/policy/file_contexts/file_contexts /tmp/test
 (labels /tmp/test as specified in .../file_contexts)
setsebool – sets the value of a boolean
setsebool secure_mode true (sets value of **secure_mode** to true)
seuser – A GUI (when run with **-X**) and command line user manager
seuser -X
seuseradd – SELinux-aware **useradd** command; part of the **setools** package
seuserdel – SELinux-aware **userdel** command; part of the **setools** package
seusermod – SELinux-aware **usermod** command; part of the **setools** package
seuserx – launches the SELinux User Manager GUI
system-config-securitylevel – launches a GUI that can be used to set the SELinux operating mode and policy; also can be used to set booleans for supported daemons

Replaced Commands

These commands appeared in older versions of SELinux.

avc_enforcing – returns current mode (enforcing or permissive); replaced with *getenforce*
avc_toggle – switches between enforcing and permissive modes; replaced with *setenforce*
change_bool – changes the value of a boolean; replaced with *setsebool*
show_bools – shows the value of a boolean; replaced with *getsebool*

RBAC/TE Policy

Except where noted these statements appear in .te files

allow – allow *role new_role*;
permits transitions between roles; appears in */etc/selinux/strict/src/policy/rbac*
allow staff_r sysadm_r; (allows staff_r to transition to sysadm_r)
bool - bool *identifier true|false*;
defines a boolean
bool httpd_unified false;
role – role *role_name types domain_or_type_name*;
(describes the domains/types the role is authorized to enter)
role system_r types httpd_php_t;
(allows the system_r role to enter the httpd_php_t domain)
type - type *name, attribute, attribute, attribute*;
defines a type/domain and associates it with attributes
type httpd_config_t, file_type, sysadmfile;
(associates the file_type and sysadmfile attributes with the httpd_config_t type name)
typealias – typealias *type_name alias alias_name*;
defines an alias for a type or domain
typealias httpd_sys_content_t alias httpd_user_content_t;
(defines httpd_user_content_t as an alias for httpd_sys_content_t)
type_transition – type_transition *source_type target_type:class new_type*;
defines what domain and type transitions occur by default⁹²
type_transition sysadm_t sysadm_home_dir_t:dir sysadm_home_t;
(when a process in the sysadm_t domain creates a directory (dir) under a directory of the home_dir_t type, the new directory is given the type sysadm_home_t)
user – user *username roles roles_assigned*;
assigns a role to a user; appears in */etc/selinux/strict/src/policy/users*
user test roles { staff_r sysadm_r }; (assigns staff_r and sysadm_r to test)

AV rules

Access vector rules, which appear in .te files, share the same syntax:

syntax: - av *domain type:class operation*

⁹² RedHat, “Te Rules – Types”

allow – defines actions allowed and not logged

auditallow – defines actions allowed and logged

auditdeny – defines actions not permitted that are logged

dontaudit – defines actions not permitted and not logged

allow httpd_t user_home_dir_t:dir { getattr search};

(Allows process associated with *httpd_t* domain to perform the directory related operations *getattr* and *search* on files of the *user_home_dir_t* type)

FC Files

The syntax of file context files is shown here:

regex *flags* *context*

regex: a standard Unix regular expression specifying a directory tree or file; by default it is anchored with **^** at the beginning and **\$** at the end; this can be overridden by using **.*** at the beginning or end

flags: **--** for files only, **-d** for directories only, or empty for both

context: security context to be applied to the files when policy is installed

/var(/.*)? system_u:object_r:var_t

(any files under /var get the specified context)

/var/tmp -d system_u:object_r:tmp_t

(the /var/tmp directory gets the specified context)

Generic Policy Notation

Some of the general logic of the policy files is described below

; terminates a statement
{ } group together arguments and code blocks
&& logical and
|| logical or
== equal to
!= not equal to
^ XOR
! not
~ used to complement an attribute (invert)
- used to subtract domains from an access vector
***** wildcard; expands to all classes, domains or types

Miscellaneous Policy Keywords

Some miscellaneous keywords found in SELinux policy files are described below

class defines a class
common used to define access vector components common to a class
constrain defines a constraint

genfscon	general file system context; contexts for non-persistent file systems
netifcon	defines network interface context
nodecon	defines the security context of a node on the network
portcon	defines the security context of a network port
self	the target and the source are the same
sid	defines initial sid for an object

© SANS Institute 2000 - 2005, Author retains full rights.

Appendix C: List of References

Anderson, Ross. Security Engineering. New York, NY: John Wiley & Sons, Inc., 2001

Badger, Lee, Daniel F. Sterne, David L. Sherman, Kenneth M. Walker, Shelia, A. Haghighat. "A Domain and Type Enforcement Unix Prototype." (1995) URL: <http://www.usenix.org/publications/library/proceedings/security95/badger.html> (9 Apr 2005)

Bleher, Thomas. "Thomas' SELinux-Pages." URL: <http://www.cip.ifi.lmu.de/~bleher/selinux/> (9 Apr 2005)

centos. "Red Hat Enterprise Linux 4: Red Hat SELinux Guide." URL: <http://beta.centos.org/centos/4.0beta/docs/html/rhel-selq-en-4/selq-part-0057.html> (9 Apr 2005)

Coker, Faye. "Getting Started with SE Linux HOWTO: the new SE Linux." (6 Dec 2003). URL: http://www.lurking-grue.org/gettingstarted_newselinuxHOWTO.html (9 Apr 2005)

Coker, Faye. "What's New in Fedora Core 3 SE Linux." (09 Nov 2004) URL: <http://www.linuxjournal.com/article/7887> (9 Apr 2005)

Coker, Russell. "NSA Security Enhanced Linux." URL: <http://www.coker.com.au/selinux/> (9 Apr 2005)

Coker, Russell. "SELinux Play Machine." URL: <http://www.coker.com.au/selinux/play.html> (9 Apr 2005)

Coker, Russell. "SE Linux Policy Writing LINUX21." (17 July 2004) URL: <http://www.coker.com.au/selinux/talks/ibmtu-2004/LINUX21.pdf> (9 Apr 2005)

Coker, Russell. "SE Linux Implementation LINUX20" (7 July 2004) URL: <http://www.coker.com.au/selinux/talks/ibmtu-2004/LINUX20.pdf> (9 Apr 2005)

Costales, Bryan, Eric Allman. sendmail. 3rd ed. Sebastopol, CA: O'Reilly Media, Inc, 2002

Dunne, Paul. LinuxWorld.com. "Using the m4 Macro Preprocessor." (25 Apr 2000) URL: <http://www.itworld.com/Comp/2378/LWD000425m4/> (9 Apr 2005)

Fedora Project. "SELinux." URL: <http://fedora.redhat.com/projects/selinux/> (9 Apr 2005)

Fedora Project. "The fedora-selinux-list Archives." URL: <https://www.redhat.com/archives/fedora-selinux-list/> (9 Apr 2005)

Ferraiolo, David, and Richard Kuhn. "Role-Based Access Control." (1992) URL: <http://csrc.nist.gov/rbac/ferraiolo-kuhn-92.pdf> (9 Apr 2005)

Friedl, Jeffrey E. F. Mastering Regular Expressions. 2nd ed. Sebastopol, CA: O'Reilly & Associates, 2002

Ganguli, Harpreet. Java Security. Cincinnati, Ohio (2002): Premier Press

Gentoo.org. "Installing Gentoo SELinux." URL: <http://www.gentoo.org/proj/en/hardened/selinux/selinux-x86-handbook.xml?part=1&chap=0> (9 Apr 2005)

grsecurity. "grsecurity." URL: <http://www.grsecurity.net> (9 Apr 2005)
Hally, Richard. "Re: Difference between policy-sources and policy-strict-sources." E-mail to: Fedora SELinux support list for users & developers. (12 May 2004) URL: <https://www.redhat.com/archives/fedora-selinux-list/2004-May/msg00104.html> (9 Apr 2005)
Hallyn, Sege Edward. "Domain and Type enforcement in Linux." (2003) URL: <http://www.cs.wm.edu/~kearns/dissertations.d/serge.pdf> (9 Apr 2005)
Immunix. "Linux Security Modules." URL: <http://lsm.immunix.org> (9 Apr 2005)
Kilpatrick, Doug, Wayne Salmon, Chris Vance. "Securing the X Window System With SELinux." (March, 2003) URL: http://www.nsa.gov/selinux/papers/X11_Study.pdf (9 Apr 2005)
Lemuria.org. "Installing a 2.6SELinux Kernel/System." URL: <http://selinux.lemuria.org/install-2.6.html> (9 Apr 2005)
Love, Robert. Linux Kernel Development. Indianapolis, IA: SAMS Publishing, 2004

Loscocco, Peter A., Stephen D. Smalley, Patrick A. Muckelbauer, Ruth c. Taylor, S. Jeff Turner, John F. Farrell. "The Inevitability of failure: The flawed Assumption of Security in Modern Computing Environments." (1998) URL: <http://www.nsa.gov/selinux/papers/inevitability.pdf> (9 Apr 2005)

Loscocco, Peter A., Stephen D. Smalley. "Meeting Critical Security Objectives with Security-Enhanced Linux." (2001) URL: http://www.nsa.gov/selinux/papers/sel_ottawa01.pdf (9 Apr 2005)
Loscocco, Peter, Stephen Smalley. "Integrating Flexible Support for Security Policies into the Linux Operating System." (June, 2001) URL: <http://www.nsa.gov/selinux/papers/freenix01.pdf> (9 Apr 2005)

McCarty, Bill. SELinux: NSA's Open Source Security Enhanced Linux. Sebastopol, CA: O'Reilly Media, Inc, 2005

Mecklenburg, Robert. Managing Projects with GNU Make. 3rd ed. Sebastopol, CA: O'Reilly & Associates, 2004
National Security Agency. "What's New With SELinux." URL: <http://www.nsa.gov/selinux/news.cfm> (9 Apr 05)
National Security Agency. "Download 2.6-based SELinux." URL: <http://www.nsa.gov/selinux/code/download5.cfm> (9 Apr 2005)
National Security Agency. "Download Experimental SELinux Code." URL: <http://www.nsa.gov/selinux/code/download6.cfm> (9 Apr 2005)
National Security Agency. "Historical Versions of SELinux." URL: <http://www.nsa.gov/selinux/code/download1.cfm> (9 Apr 2005)
National Security Agency. "Security-Enhanced Linux." URL: <http://www.nsa.gov/selinux/index.cfm> (9 Apr 2005)
National Security Agency. "SELinux Background." URL: <http://www.nsa.gov/selinux/info> (9 Apr 2005)
National Security Agency. "SELinux Frequently Asked Questions (FAQ)." URL:

<http://www.nsa.gov/selinux/info/faq.cfm> (9 Apr 2005)
National Security Agency. "SELinux Mailing List."
<http://www.nsa.gov/selinux/info/list.cfm?MenuId=41.1.19> (9 Apr 2005)
National Security Agency. "SELinux Mailing List: by date."
<http://www.nsa.gov/selinux/list-archive/date.cfm?MenuID=41.1.1.9.3> (9 Apr 2005)
Pomeranz, Hal, SANS. Issues and Vulnerabilities in Unix: 6.1. Oakland, CA: Deer Run Associates, 2003
RedHat. "SELinux." URL: <https://www.redhat.com/security/innovative/selinux/> (9 Apr 2005)
RedHat. "RedHat SELinux Guide."
<http://www.redhat.com/docs/manuals/enterprise/RHEL-4-Manual/selinux-guide/> (10 Apr 2005)
RedHat. "TE Rules – Types." URL:
<http://www.redhat.com/docs/manuals/enterprise/RHEL-4-Manual/selinux-guide/rhlcommon-section-0050.html> (10 Apr 2005)
RedHat. "Using apol for Policy Analysis." URL:
<http://www.redhat.com/docs/manuals/enterprise/RHEL-4-Manual/selinux-guide/rhlcommon-section-0104.html> (12 Apr 2005)
Secure Auditing for Linux. "Secure Auditing For Linux." URL:
<http://secureaudit.sourceforge.net> (10 Apr 2005)
Secure Computing Corporation. "Sidewinder G2 Firewall Type Enforcement Technology." URL:
http://www.securecomputing.com/pdf/type_enforcement_wp.pdf (10 Apr 2005)
SELinux for Distributions Project. "SELinux for Distributions."
<http://selinux.sourceforge.net/about.php3> (10 Apr 2005)
SELinux Symposium. "SELinux Symposium." URL: <http://www.selinux-symposium.org> (12 Apr 2005)
Smalley, Stephen. "Security-Enhanced Linux 2001 Kernel Summit Presentation." (2001) URL: <http://www.nsa.gov/selinux/papers/sel.summit.pdf> (10 Apr 2005)
Smalley, Stephen. "NSA Security Enhanced Linux (SELinux)." (2003) URL: <http://www.nsa.gov/selinux/papers/ols2003-selinux.pdf> (25 Apr 2005)
Smalley, Stephen. "Configuring the SELinux Policy." (February, 2005) URL: <http://www.nsa.gov/selinux/papers/policy2.pdf> (10 Apr 2005)
Smalley, Stephen, Chris Vance, Wayne Salmon. "Implementing SELinux as a Linux Security Module." (March, 2004) URL:
<http://www.nsa.gov/selinux/papers/module.pdf> (10 Apr 2005)
Smalley, Stephen, Timothy Fraser. "A Security Policy Configuration for the Security-Enhanced Linux." (February, 2001) URL:
<http://www.nsa.gov/selinux/papers/policy.pdf> (10 Apr 2005)
Spencer, Ray, Stephen Smalley, Peter Loscocco, Mike Hibler, David Andersen, Jay Lepreau. "The Flask Security Architecture: (August, 1999) System Support for Diverse Security Policies" URL: <http://www.nsa.gov/selinux/papers/flask.pdf> (11 Apr 2004)
Stallings, William. Operating Systems: Internals and Design Principles. 5th ed.

Upper Saddle River, NJ, 2005

Sun Microsystems. "Trusted Solaris Operating System."

<http://www.sun.com/software/solaris/trustedsolaris/index.xml> (11 Apr 2005)

Thompson, Kerry. "Security Enhanced Linux." URL:

<http://www.crypt.gen.nz/selinux/> (11 Apr 2005)

Thompson, Kerry. "SELinux." (March, 2003) URL:

<http://www.samag.com/documents/s=7835/sam0303a/0303a.htm> (11 Apr 05)

Tresys Technology. "Audit Log Analysis Tool for Security Enhanced Linux."

URL: http://www.tresys.com/Downloads/selinux-tools/seaudit/seaudit_help.txt

(11 Apr 2005)

Tresys Technology. "Security Enhanced Linux." URL:

<http://www.tresys.com/selinux/> (11 Apr 2005)

Tresys Technology. "SELinux Policy customization and Editing tool Help File."

URL: http://www.tresys.com/Downloads/selinux-tools/sepcut/sepcut_help.txt

(11 Apr 2005)

Tresys Technology. "SELinux User Manager Help File." URL:

http://www.tresys.com/Downloads/selinux-tools/seuser/seuser_help.txt (11 Apr

2005)

TrustedBSD.org. "About TrustedBSD." URL: <http://www.trustedbsd.org> (11 Apr

2005)

University of Utah. "The Distributed Trusted Operating System (DTOS) Home Page." (December, 2000) URL:

<http://www.cs.utah.edu/flux/fluke/html/dtos/HTML/dtos.html> (11 Apr 2005)

University of Utah. "Flask: Flux Advanced Security Kernel." (December, 2000)

URL: <http://www.cs.utah.edu/flux/fluke/html/flask.html> (11 Apr 2005)

User Mode Linux Community Site. "User Mode Linux." URL:

<http://usermodelinux.org> (11 Apr 2005)

User-mode Linux Kernel Home Page. "The User-mode Linux Kernel Home Page." <http://user-mode-linux.sourceforge.net> (11 Apr 2005)

Viega, John, Gary McGraw. Building Secure Software. Boston, MA: Addison-Wesley, 2002

Walker, Kenneth M., Daniel F Sterne, M. Lee Badger, Michael J. Petkac, and Karen A. Oostendrop. "Confining Root Programs with Domain and Type Enforcement (DTE)." (1996) URL:

<http://www.usenix.org/publications/library/proceedings/sec96/walker.html> (11

Apr 2005)

Walters, Colin. "Understanding and Customizing the Apache HTTP SELinux Policy (Beta Document)" URL: <http://fedora.redhat.com/docs/selinux-apache-fc3/> (11 Apr 2005)

Wood, Timothy. "SE Linux." URL: <http://www.diyab.net/selinux/> (11 Apr 2005)