SELINUX:
A NEW APPROACH TO SECURE SYSTEMS

CHRIS RUNGE

ABSTRACT

In this whitepaper, we will examine Security-Enhanced Linux® (SELinux), the benefits it brings, and how Red Hat is working to make those benefits available to mainstream enterprise computing.

INTRODUCTION

One of the greatest challenges now facing CIOs and IT directors is the task of maintaining the security of their IT environments. The effects of a security breach can be catastrophic, including unplanned downtime and the resulting loss of service–a potentially significant financial impact–and the loss of sensitive and confidential information. This problem has been compounded by the proliferation of networked PCs and servers as well as the growing intelligence of malicious software that seeks to exploit and expand through the Internet infrastructure.

The software industry is releasing new tools to address the needs of system administrators responsible for managing the security of large numbers of geographically dispersed systems. For example, Red Hat® Network, a key part of Red Hat Enterprise Linux and Red Hat's Open Source Architecture, offers system administrators a way to review information about security vulnerabilities and proactively apply relevant security and other updates to large numbers of Red Hat Enterprise Linux systems.[1]

Tools like Red Hat Network can help system administrators react to and take action to thwart security breaches. At the same time, new technologies are emerging at the core operating system level to protect against and contain the effects of a security breach until security updates can be applied. For example, NX technology (available in processors from AMD and Intel) and exec-shield (a technology develop by Red Hat's Ingo Molnar), help protect against buffer overflows, a tactic frequently employed by attackers to infiltrate and compromise flawed software programs. Another technology—the focus of this whitepaper–is Security-Enhanced Linux, or SELinux as it is most commonly known.

ACCESS CONTROL MECHANISMS

Multi-user operating systems such as Linux, UNIX®, Microsoft® Windows, and Mac OS X use access control mechanisms to determine whether and how the users and programs on that system can access objects (e.g. files, directories, sockets) on the system. Such actions as whether users or programs running on the system can read, write, create, or delete files (such as log files, configuration files, or data files), and whether they can start new programs are determined by an access control mechanism. Given this, the characteristics of the access control mechanism being used become very important for the security of a given system.

Most Linux and UNIX-based operating systems use an access control mechanism known as Discretionary Access Control (DAC)[2]. Under a DAC scheme, files and directories on the system are labeled with a set of permissions indicating which user and group the file belongs to and what that user, group, or others can do with that object, such as reading, writing, or executing it. This relatively simple yet largely powerful scheme allows multiple users and programs to coexist on the same system, and when the permissions are properly set, ensures that users have control over their objects but no others.

The scheme gets its name because users and programs have discretion over the objects in their control. The root user, or a program running as root, has discretion over everything on the system. It follows that if a user owns files containing confidential information (e.g. financial information, patient data, trade secrets), that user can intentionally or mistakenly share that data with other users or programs that should not have access.

1 For more information on Red Hat Network, see www.redhat.com/red_hat_network/

2 This generally applies to Windows as well, although the particulars of how the Discretionary Access Control scheme is implemented may be different than on a Linux or UNIX-based operating system.

Programs run with the level of privilege of the user starting them. Accordingly, a program started by the user "chris" would run as the user "chris" and have the same level of control and access as "chris." That same program, if started by root, would run with full root permissions. Some programs may be marked "setuid root," meaning that they run with full root permissions regardless of the user starting them. If a flawed or malicious program is run as root, the entire system can be compromised; setuid root programs are of particular concern since they can be started by non-root users.

Attackers who desire to bring a system down, re-purpose it (by running programs the owner did not intend to run), or gain access to sensitive information on a system generally seek to exploit the all-powerful nature of the root user. By exploiting a program, such as a daemon, that runs with full root permissions, they can use that program as a launching pad for their desired malicious actions on the system.

Mandatory Access Control (MAC) is an alternative (and far less frequently used) access control mechanism. Found on what are known as trusted operation systems, mandatory access control takes security decisions out of the hands of the user. While users and groups may still own files and directories on the system, permission is ultimately governed by a security policy, which defines which users and programs can access which objects on the system. This security policy is enforced over all processes and objects on the system, not just a subset. In addition, policy decisions are based on all security-relevant information, not just the user identity. The security policy trumps the intentions of the user and is the final authority to determine what takes place on the system. The mandatory nature of the security policy is what gives this access control mechanism its name.

Mandatory access control systems lack the necessary core concept of an all-powerful root user[3]; instead, the defining principle of MAC is the principle of least privilege. This principle dictates that programs are granted only the minimum amount of privilege in order to function rather than inheriting the privileges of the user starting them. This configuration is set via the security policy present on the system. The end result is a higher level of security. If a program is compromised under a DAC scheme, the attacker gains the ability to run programs and access objects on the system at the same level of privilege as the user account the program is running under; however, under a MAC scheme, the attacker is limited to the actions allowed by the system's security policy.

SELinux and Multi-Level Security

Multi-Level Security (MLS) is a specific mandatory access control security model that is focused on confidentiality. Traditionally, the separation of multiple classifications of information (e.g. Top Secret, Secret, Confidential, Unrestricted, etc.) was enforced through physical separation–a dedicated system would be used for each security classification to ensure that, for example, Top Secret information was not mistakenly or intentionally made visible to users possessing only a Secret (or lower) security clearance. A Multi-Level Security operating system is able to enforce both the separation of multiple classifications of information as well as manage multiple users with varying levels of information clearance. This allows one system to be used where multiple systems might otherwise be required. The MLS capability originally supplied with SELinux has been significantly updated to allow MLS to be dynamically enabled and meet certification requirements. A new model has been added, Multi-Category Security (MCS), which provides a discretionary labeling scheme to end users.

3 Depending on the configuration of the security policy, the system administrator domain (sysadm_t) on an SELinux system may have a wide variance in the processes and objects that are placed under its control. Accordingly, sysadm_t could be quite powerful, approaching the level of control of the root user under a discretionary access control model. At the same time, sysadm_t could be restricted to just actual authenticated administrator sessions, and not be use as a domain for any daemons or other privileged programs. The key point here is that an omnipotent superuser is not a fundamental assumption of mandatory access control.

Mandatory access control offers clear security benefits, so why has it historically only had limited adoption? There are several reasons. First, the trusted operating systems that provided mandatory access control had largely been separate offerings from their more mainstream counterparts. Due to a traditionally limited potential audience consisting of largely specialized military and intelligence applications, these trusted systems had received less focus from operating system vendors and third-party vendors alike. They tended to lag behind their commercial cousins in terms of functionality as well as hardware and application support. Moreover, MAC systems could be difficult and time-consuming to implement and use. All of this resulted in a small community of use, which further served as a disincentive for vendors to pour additional resources into the product. Trusted operating system has been largely confined to specific applications and deployments in the military and intelligence communities. Even then security features were often disabled or weakened; users have a natural preference for functionality and ease-of-use over increased security.

SECURITY-ENHANCED LINUX

Enter into this backdrop Security-Enhanced Linux, or SELinux, an open source research project sponsored by the National Security Agency (NSA) to implement mandatory access control in Linux. The NSA chose Linux as the basis for this project because of its large community of use and open source development model. This would allow the NSA to receive valuable feedback during the development process. At the same time, it provided an opportunity to improve the security of a rapidly growing and increasingly popular operating system.[4]

Recognizing that secure applications require a secure operating system5, SELinux provides security at the kernel level through the added implementation of a security policy and a separate enforcement mechanism. With this approach, SELinux is able to support most commonly used applications, generally without modification, while largely retaining full application functionality. The security policy on the system, rather than the configuration of a particular application, will ultimately dictate the boundaries of the application's capability.

SELinux provides several benefits. By leveraging the principle of least privilege and through the institution of a security policy on the system, SELinux prevents the compromise of an entire system due to the compromise of a single application running with what would otherwise be elevated privileges. Programs are placed into individual sandboxes, isolating them from one another and from the underlying operating system (see Figure 1). A second benefit is that SELinux protects the confidentiality and integrity of data. By removing discretion from users over how data may be manipulated, sensitive data can be restricted from willful or inadvertent sharing, modification, or deletion.

4 See www.nsa.gov/selinux

5 See "The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments," by Peter A. Coscocco, Stephen D. Smalley, et. al. www.nsa.gov/selinux/papers/inevit-abs.cfm

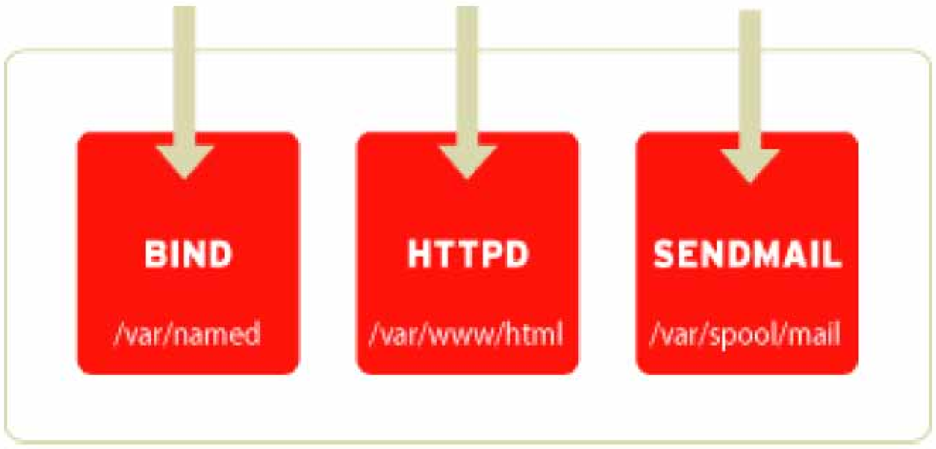


Figure 1. SELinux isolates programs from the operating system and from one another.

SELINUX ARCHITECTURE AND OPERATION

SELinux is a Linux security module that is built into the Linux kernel. SELinux is driven by loadable policy rules. When security-relevant access is taking place, such as when a process attempts to open a file, the operation is intercepted in the kernel by SELinux. If an SELinux policy rule allows the operation, it continues; otherwise, the operation is blocked and the process receives an error.

SELinux decisions, such as allowing or disallowing access, are cached. This cache is known as the Access Vector Cache (AVC). Caching decisions decrease how often SELinux policy rules need to be checked, which increases performance. Remember that SELinux policy rules have no effect if DAC rules deny access first.

SELinux is made up of both kernel and user-space components. With respect to the kernel, SELinux adds a security server containing the security policy. It also adds a separate enforcement mechanism that receives and applies the policy decision. Because the policy and enforcement mechanism are independent, policy changes do not require changes to the enforcement mechanism. SELinux also modifies or adds several user-space component for recognizing and handling security contexts, roles changes, policy development, and other tasks necessary for implementing mandatory access control on the system.

The default SELinux implementation encompasses multiple security models: Type Enforcement (TE), Role Based Access Control (RBAC), Multi-Level Security (MLS), and Identity Based Access Control (IBAC). The combined security mechanisms achieve a flexible security policy model. Type Enforcement implements a mechanism whereby processes (running programs) are placed in what are known as domains, which govern what these processes can do. For example, BIND (Berkeley Internet Name Domain) running in the named_t[6] domain, can access its zone files under /var/named but not the system's web pages under /var/www/html (which would be the province of the Apache HTTP server, running in the httpd_t domain). Each domain is an isolated sandbox on the operating system where an application resides. Applications are only allowed to play in their own individual sandboxes; they are contained by the security policy on the system from interfering

6 The _t suffix indicates a domain or type.

with other application or with the underlying operating system. In a similar fashion, types are assigned to objects (files, directories, sockets, etc.) on the system. Types determine who gets to access the object and can be used to protect the integrity and confidentiality of data on the system.

Role-based access control helps to simplify policy creation and management. Instead of creating rules for every user as it relates to every object on the system, roles are used to determine what domains can be used. Common roles include user_r[7] (the standard unprivileged user role) and sysadm_r (the system administrator role). Those roles are then in turn assigned to users. Users can belong to multiple roles, with access dependent on the current role being used.

How does this work in actual operation? Files and process are labeled with a security context, which is made up of the user, the role, and the domain or type. For example, the security context of the Apache HTTPD daemon is

```
system_u: system_r: httpd_t
```

while the security context of the Apache HTTPD configuration file is:

```
system_u: object_r: httpd_config_t
```

Where system_u identifies the user, system_r and object_r identify the role and httpd_t and httpd_config_t identify the type.

When access is requested, the traditional UNIX permission system used by a discretionary access control scheme is referenced first. If the action is denied by that mechanism, the request immediately ends. If access is permitted, the mandatory access control scheme implemented by SELinux is used. The security server in the kernel checks the security context of the requesting user or program with the security context of the requested object in light of the security policy on the system. The resulting policy decision is received and applied by the enforcement mechanism, which either allows access or denies it (see Figure 2). All access requests and their corresponding policy decisions may be audited, if required.

4 The _r suffix indicates a role.

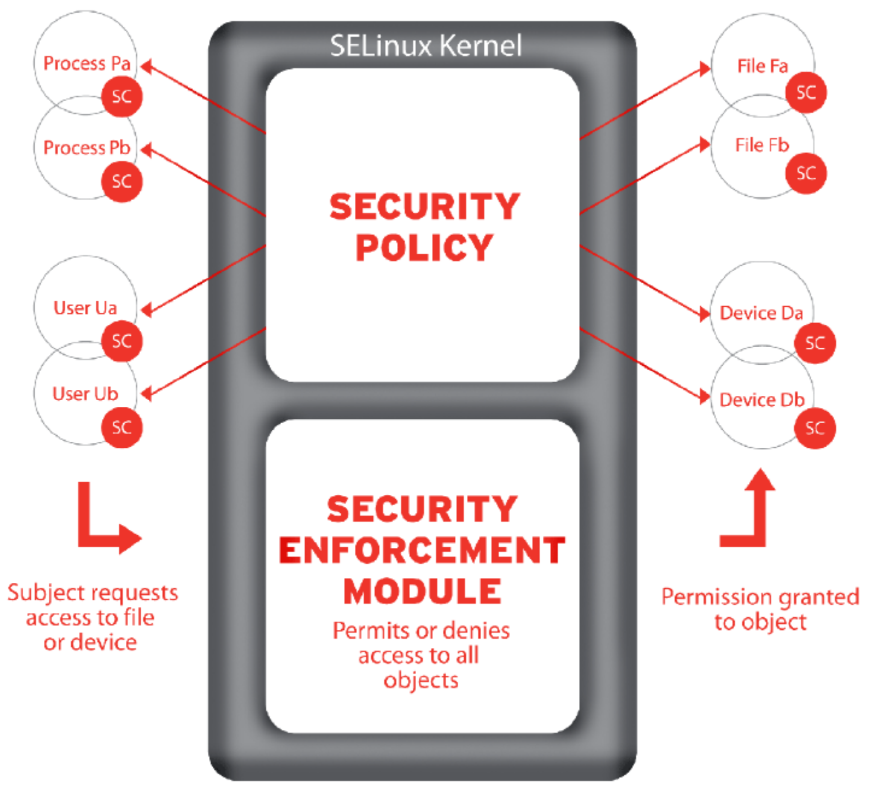

SELinux Kernel
Process Pa
SC
Process Pb
SC
User Ua
SC
User Ub
SC
SECURITY POLICY
File Fa
SC
File Fb
SC
Device Da
SC
Device Db
SC
Subject requests access to file or device
SECURITY ENFORCEMENT MODULE
Permits or denies access to all objects
Permission granted to object

Figure 2. SELinux access control mechanism

EXAMPLES OF SELINUX DEPLOYMENTS

The best way to understand the benefits that SELinux provides is to examine two real-world usage scenarios: an Apache HTTP server and a BIND DNS server. Because these are popular network-based applications, they are particularly susceptible to attack, and consequently stand to benefit from SELinux protection.

EXAMPLE: APACHE HTTP SERVER

The term "secure Web server" can have multiple meanings. It commonly refers to the security that a Web server may provide by encrypting incoming and outgoing traffic. However, that encryption process alone does not ensure the underlying security of the Web server software in its own right. An attacker can still exploit a flaw or misconfiguration of the Web server to gain access to the data it manages, allowing the attacker to deny service, deface websites, or capture confidential information. The ability of a Web server, if so configured, to execute scripts and other program code can exacerbate the situation.

Web server administrators can improve security by configuring the Web server to run as a user other than root, and by reducing functionality such as disabling the ability of the Web server to execute program code. However, these steps do not necessarily provide the level of containment desired if the Web server is exploited. In addition, it may be desirable in some situations to allow the Web server run scripts and use other functionality-enhancing capabilities. This is where SELinux can help.

An Apache HTTP server using SELinux carries a couple of principle advantages from an implementation standpoint. First, the standard Apache HTTP Server software (such as that provided with the operating system) can be used; no specially compiled binaries are required. Second, the full functionality of the Web server software, including running scripts and program code is available. By placing the Web server in a domain, typically called httpd_t, the system's security policy will provide a box around the software and what it can do—even if the Web server is compromised.

SELinux's flexible policy model allows a system administrator or security officer to implement a security policy tuned to the requirements of the application. SELinux's approach to policy is that everything not expressly allowed is denied. Using this principle of least privilege, an example policy might therefore look as follows:

- The Web server is allowed to bind to port 80
- The Web server is allowed to read its configuration files in /etc/httpd/conf and read/write its log files
- The Web server is allowed to execute the programs and libraries it needs in order to function
- Read-only access to the system's web pages in /var/www/html is allowed
- User pages may be read; in addition user scripts may be executed that read from and write to those pages

If a Web server running with this policy were compromised, the potential damage would be confined. The system's web pages in /var/www/html could not be defaced. The attacker could not use the Web server to read files on the system other than web pages, the Web server's log files, and the Web server's configuration files. Only user scripts could be run, and only user pages modifiable by those scripts could be defaced. Finally, the underlying operating systems as well as other applications running on the system would remain unaffected.

EXAMPLE: BIND

BIND is a popular DNS server used on a number of Linux and UNIX operating systems. BIND has been a popular target because of its ubiquity, and because it typically runs with superuser (root) privileges on the system, enabling attackers who compromise it to execute commands with superuser privileges.

The approach for using SELinux to lock-down BIND is similar to the approach taken above to secure the Apache HTTP server. The standard BIND software can be used and a chroot jail[8] – a commonly used tactic to improve the security of BIND – is not required. Again, the security policy on the system will provide the boundaries around the software and what an attacker who potentially exploits it can do on the system.

An example policy for BIND would be to create a domain, named_t, with the following characteristics:

- BIND can bind to port 53 on the system's external network interface
- BIND can read its configuration file, /etc/named.conf, and read/write its log files
- BIND can read and write its dynamic zone files in /var/named
- BIND can execute the programs and libraries it needs in order to function

Everything else would be implicitly denied.

If BIND were compromised, once again the potential for the attacker to execute malicious action would be limited. In this case the only thing the attacker could corrupt or modify would be the dynamic zone files in /var/named. The attacker could not install new binaries (such as a root-kit), and would be prevented from altering or removing existing system files. Moreover, the security policy would prevent the attacker from attaching to the system's internal network interface, blocking any attempts to use the compromised software to begin attacks on the internal network. The underlying operating system and any other applications on the system would remain untouched.

DECIDING WHERE TO DEPLOY SELINUX

As seen above, Internet-facing edge servers running daemons such as the Apache HTTP server and the BIND DNS server stand to gain immediate benefits from the use of SELinux. Other mission-critical infrastructure systems – mail servers, firewalls, authentication and authorization servers – are also good candidates.

SELinux also offers valuable information assurance by protecting the integrity and confidentiality of sensitive information. This includes not only classified information in a more traditional sense that is hosted on system used by the military and intelligence community, but also any confidential or sensitive information that needs to be protected. Examples include human resource data (e.g. payroll and personnel files), patient data (e.g. that protected by HIPAA), and financial data (e.g. credit card information).

The ultimate goal is to make SELinux capabilities valuable for every system. Due to its flexible policy model, SELinux has the potential to provide security for other classes of systems, such as an end-user desktop. For example, an SELinux policy could be created that would largely allow the end-user to work on the desktop as they normally would under a discretionary access control model. However, any network daemons (such as openssh, portmap, etc.) would be isolated into their own domains, as well as any application (such as Mozilla or Evolution) that might potentially be misused as an agent for sending and receiving malicious software such as viruses and trojans.

8 A chroot jail provides an increased level of security by moving the binaries, libraries, and other files used by BIND to a particular area on the filesystem, which in turn is treated as though it were the root directory of the operating system for the running BIND server. The idea is to contain a successful attacker to this limited filesystem, and in doing so prevent the attacker from compromising the operating system or other applications. A chroot jail may not provide complete security, however. See www.stahl.bau.tu-be.de/~hildeb/bind/chroot.shtml for more information.

SELINUX AND THE REAL WORLD

In the years since SELinux was introduced as a component of Red Hat Enterprise Linux, there have been several documented cases where SELinux provided protection for systems under attack.

Don Marti gave the following examples (LinuxWorld.com,02/24/08)

> The announcements of several recent security holes tell a new story: SELinux, if turned on, can prevent an attacker from using an exploit to its full destructive potential. For example, one vulner ability in the Hewlett-Packard Linux Imaging and Printing Project's software would have allowed an attacker to run arbitrary commands as root. However, according to the company's security advisory on the bug, "On Red Hat Enterprise Linux (RHEL) 5, the SELinux targeted policy for hpssd which is enabled by default, blocks the ability to exploit this issue to run arbitrary code."
>
> Dan Walsh, an SELinux developer at Red Hat, covered another, higher profile mitigation on his blog. Samba, the software that acts as a file server for Microsoft Windows systems, had a vulnerability that would have allowed an attacker to run commands as root. However, "while the exploit might be able to take advantage of a buffer overflow, when the attacker tries to execute the code, SELinux would stop it," he wrote.

Tresys Technology (www.tresys.com), one of Red Hat's partners focused on security and SELinux, maintains a web site with information about SELinux Mitigations, available at www.tresys.com/innovation.php.

Software with security bugs that SELinux mitigated includes the Apache web server, the Mambo content management system, the Sendmail MTA, and a commonly-installed PHP module.

RED HAT AND SELINUX

As mentioned earlier, the NSA originally developed the SELinux project and made it available as open source. The NSA also recognized Red Hat's experience and involvement in the open source community and asked Red Hat to partner with them. Red Hat recognized the immediate and potential security benefits that SELinux brings to the open source community. As a result, an open source community was built around SELinux, and SELinux has been accepted as a core component of Linux. As part of its goal to enable higher levels of security for all systems, Red Hat made a strategic decision to adopt, incorporate, and further the development of the SELinux project.

As a core part of the effort, Red Hat commercialized and provided the necessary commercial support for SELinux in its Red Hat Enterprise Linux product family. With the release of the 2.6 Linux kernel, the kernel components of SELinux became available in the upstream open source kernel, a key step in gaining wider adoption and long-term supportability in the open source community.

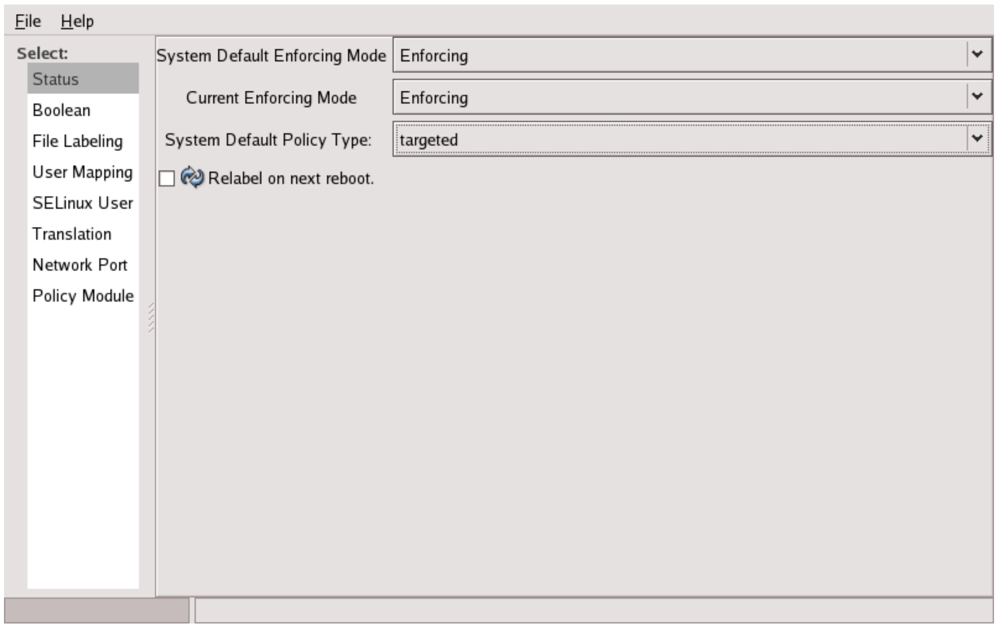
Red Hat understood that to release SELinux only as a part of a separate trusted operating system would have been repeating the mistakes of the past. For SELinux–and the security benefits that it provides–to be more widely adopted, SELinux needed to be integrated into the mainstream operating system, which is exactly what has happened. As a result, those who require systems with mandatory access control will inherit the same operating system capabilities, commercial support offerings, and support from third-party vendors as those deploying systems without it.

System administrators have several options around the configuration of SELinux on their systems. For those wanting to adhere to the traditional discretionary access control model, SELinux can be completely disabled or run in a Permissive (reporting only) mode. On systems where SELinux in enabled, administrators will be able to take advantage of its flexible policy model and tunable configuration options, enabling them to loosen or tighten the security on the system to meet their specific requirements. As part of the continuing evolution of SELinux there have been several recent improvements including:

1. Policy support for hundreds of applications
2. A "booleans" facility to allow certain policy features to be enabled or disabled without having to reload policy
3. A more robust Loadable Policy Module architecture that allows modules to loaded and modified while the system is running
4. The introduction of a Reference Policy to simplify policy development

Also as part of the effort to improve functionality and usability, there have been several additions to the available SELinux tools. These include several command line and GUI based tools to simplify use:

1. semanage – a command line tool to unify low-level SELinux administration
2. system-config-selinux – a GUI tool for comprehensive SELinux system management (see figures 3, 4, and 5)
3. setroubleshhot – a GNOME facility that provides users with information upon a policy violation. It also provides help in resolving the issue.
4. Audit2allow – which parses the audit log and converts access denial records into security policy



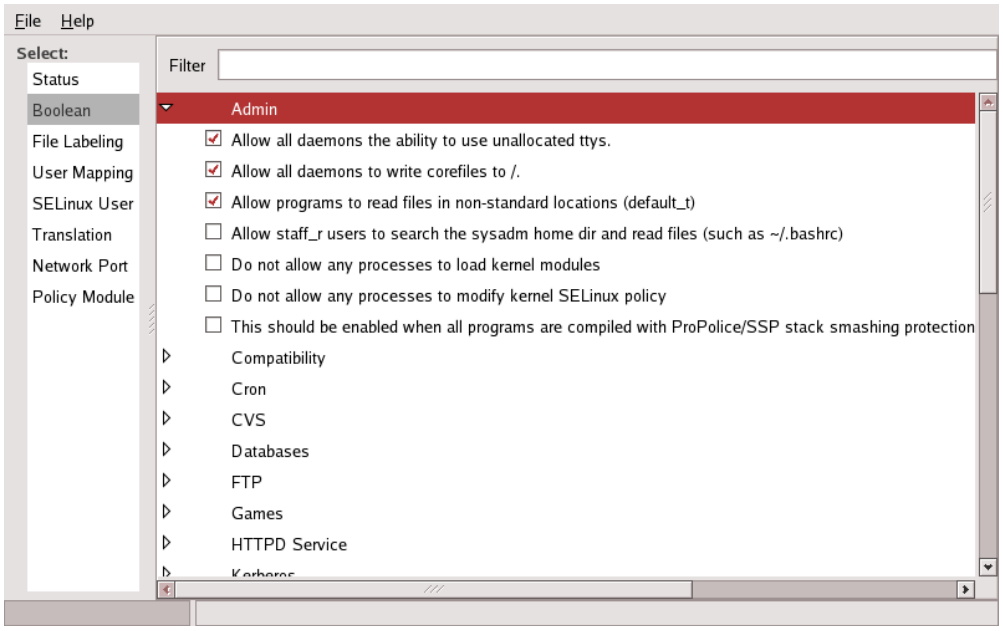
System Default Enforcing Mode: Enforcing

Current Enforcing Mode: Enforcing

System Default Policy Type: targeted

☐ Relabel on next reboot.

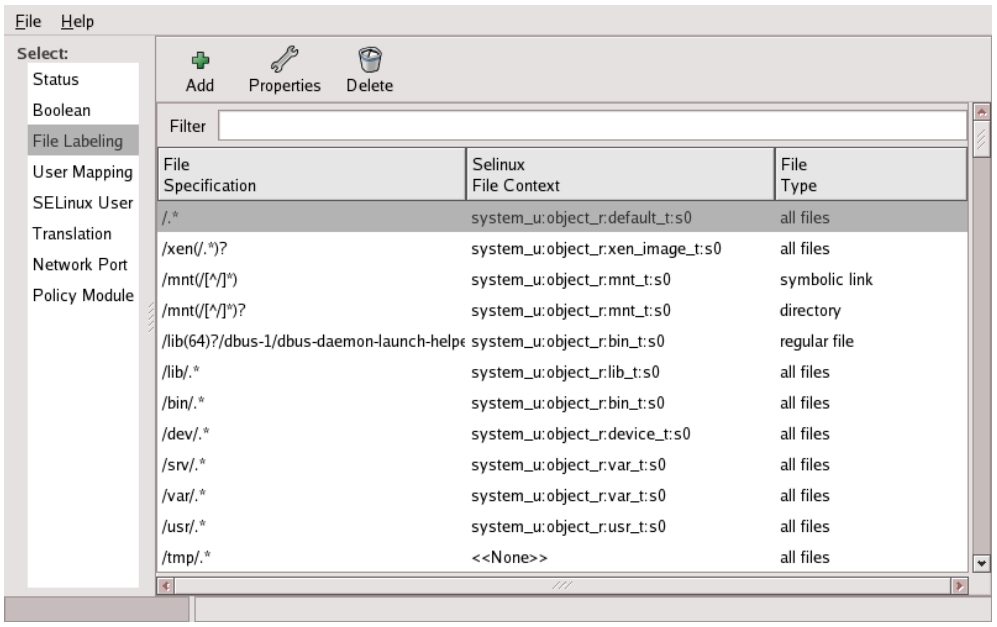
Figure 3: system-config-selinux (Status)



Admin
- [✓] Allow all daemons the ability to use unallocated ttys.
- [✓] Allow all daemons to write corefiles to /.
- [✓] Allow programs to read files in non-standard locations (default_t)
- [ ] Allow staff_r users to search the sysadm home dir and read files (such as ~/.bashrc)
- [ ] Do not allow any processes to load kernel modules
- [ ] Do not allow any processes to modify kernel SELinux policy
- [ ] This should be enabled when all programs are compiled with ProPolice/SSP stack smashing protection

Compatibility
Cron
CVS
Databases
FTP
Games
HTTPD Service

Figure 4: system-config-selinux (Booleans)



File Specification	Selinux File Context	File Type
/.*	system_u:object_r:default_t:s0	all files
/xen(/.*)?	system_u:object_r:xen_image_t:s0	all files
/mnt(/[^/]*)	system_u:object_r:mnt_t:s0	symbolic link
/mnt(/[^/]*)?	system_u:object_r:mnt_t:s0	directory
/lib(64)?/dbus-1/dbus-daemon-launch-helpe	system_u:object_r:bin_t:s0	regular file
/lib/.*	system_u:object_r:lib_t:s0	all files
/bin/.*	system_u:object_r:bin_t:s0	all files
/dev/.*	system_u:object_r:device_t:s0	all files
/srv/.*	system_u:object_r:var_t:s0	all files
/var/.*	system_u:object_r:var_t:s0	all files
/usr/.*	system_u:object_r:usr_t:s0	all files
/tmp/.*	<<None>>	all files

Figure 5: system-config-selinux (File Labeling)

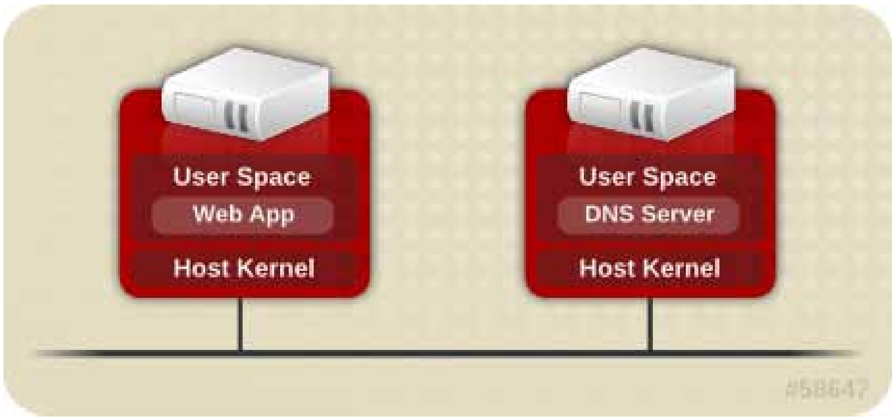
SELINUX AND VIRTUALIZATION

On of the more significant recent advances in computer technology has been the growth of virtualization. While bringing many benefits, virtualization also requires some special security considerations. SELinux is uniquely qualified to securely isolate multiple virtual guests on the same host machine from interaction with each other and with the host. An open source project, sVirt, focused on just that capability. sVirt is possible because both SELinux and Kernel based Virtual Machine (KVM) are part of the kernel. sVirt is simply an extension of SELinux with specific policy and tooling.

Included in Red Hat Enterprise Linux 6, sVirt integrates SELinux and virtualization. sVirt applies Mandatory Access Control (MAC) to improve security when using virtual machines. The main reasons for integrating these technologies are to improve security and harden the system against bugs in the hypervisor that might be used as an attack vector aimed toward the host or at another virtual machine.

NON-VIRTUALIZED ENVIRONMENT

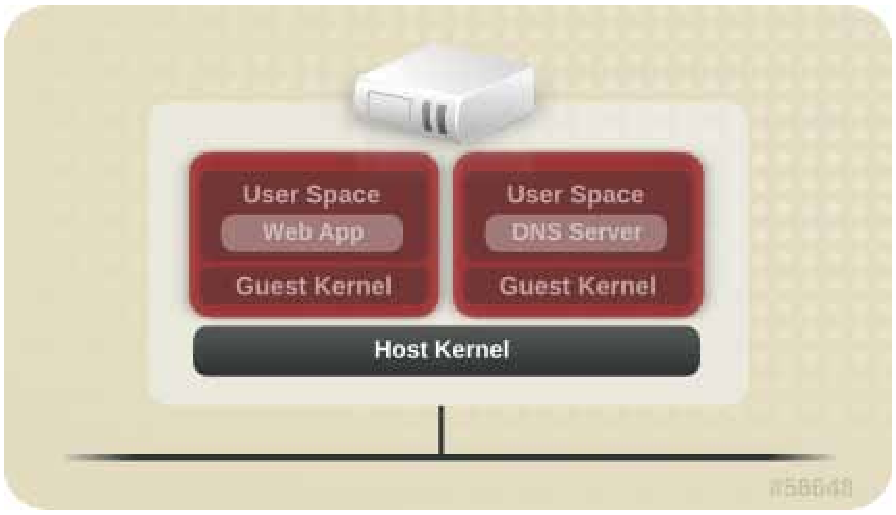
In a non-virtualized environment, hosts are separated from each other physically and each host has a self-contained environment consisting of services such as a Web server or a DNS server. These services communicate directly with their own user space, host kernel, and physical host, offering their services directly to the network. The following image represents a non-virtualized environment:



Non-virtualized environment

VIRTUALIZED ENVIRONMENT

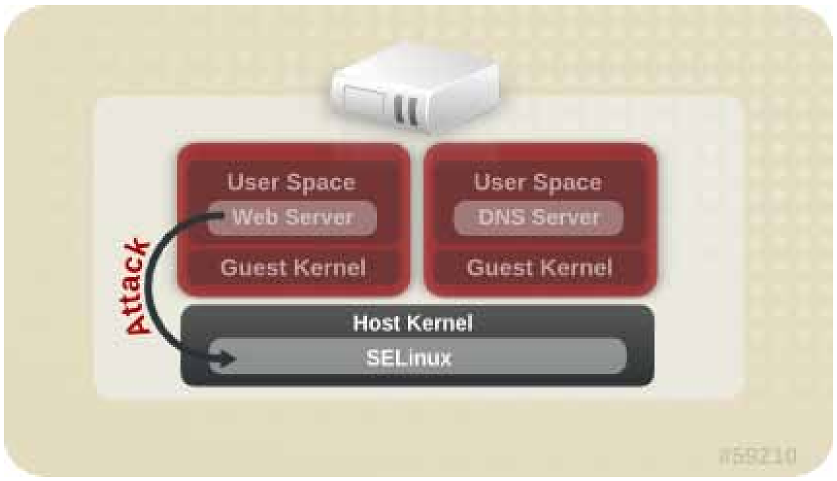
In a virtualized environment, several operating systems can be housed (as "guests") within a single host kernel and physical host. The following image represents a virtualized environment:



Virtualized environment

When services are not virtualized, machines are physically separated. Any exploit is usually contained to the affected machine, with the obvious exception of network attacks. When services are grouped together in a virtualized environment, extra vulnerabilities emerge in the system. If there is a security flaw in the hyper-visor that can be exploited by a guest instance, this guest may be able to not only attack the host, but also other guests running on that host. This is not theoretical; attacks already exist on hypervisors. These attacks can extend beyond the guest instance and could expose other guests to attack.

sVirt is an effort to isolate guests and limit their ability to launch further attacks if exploited. This is demonstrated in the following image, where an attack can not break out of the virtual machine and extend to another host instance:



Attacked virtual machine

SELinux introduces a pluggable security framework for virtualized instances in its implementation of Mandatory Access Control (MAC). The sVirt framework allows guests and their resources to be uniquely labeled. Once labeled, rules can be applied, which can reject access between different guests.

Dan Walsh, in his blog at danwalsh.livejournal.com/30565.html, recently described using SELinux in virtual environments:

> James Morris, Daniel Berrange, myself and others have added SELinux support to libvirt, in the form of sVirt. We added a security plug-in architecture to libvirt that defaults to SELinux protection. ... libvirt dynamically labels the image files and starts the virtual machines with the correct labels. This allows us to avoid the problem of the administrator having to remember to set the correct label on the image files and devices. By default all virtual machines l... get labeled with the svirt_t type and all image files get the svirt_image_t type.
>
> SELinux policy has rules that allow the svirt_t processes to read/write svirt_image_t files and devices.
>
> This protection allows us to protect the host machine from any of its virtual machines. A virtual machine will only be able to interact with the files and devices with the correct labels. A compromised virtual machine would not be allowed to read my home directory, for example, even if the virtual machine is running as root.
>
> ...When we were developing sVirt... we realized that we could use MCS to provide us separation between two virtual machines running with the same SELinux type, svirt_t. We designed libvirt to assign a different randomly-selected MCS label to each virtual machine and its associated virtual image. libvirt guarantees that the MCS fields it selects are unique. SELinux prevents different virtual machines running with different MCS fields from interacting with each other or any of their content.

For more information on sVirt, see s**elinuxproject.org/page/SVirt**.

Since the release of Red Hat Enterprise Linux 4 in early 2005, which featured the 2.6 Linux kernel and SELinux as two of its major features, organizations have been able to deploy a commercially supported SELinux-enabled operating system for production and mission-critical use[9]. Ongoing development of SELinux capabilities continues in the Fedora project–the development area and proving ground for future Red Hat Enterprise Linux capabilities.

RECENT ADDITIONS TO SELINUX CAPABILITIES

XGUEST: KIOSK MODE

The xguest package provides a kiosk user account. This account is used to secure machines that people walk up to and use, such as those at libraries, banks, airports, information kiosks, and coffee shops. The kiosk user account is very limited: essentially, it only allows users to log in and use Firefox to browse Internet websites. Any changes made while logged in with this account, such as creating files or changing settings, are lost when you log out.

CONFINED USERS

Traditionally, SELinux is used to define and control how an application interacts with the system. SELinux in Red Hat Enterprise Linux 6 introduces a set of policies that allows system administrators to control what particular users can access on a system.

SANDBOX

SELinux in Red Hat Enterprise Linux 6 features the new security sandbox feature. The security sandbox adds a set of SELinux policies that enables a system administrator to run any application within a tightly confined SELinux domain. Using the sandbox, system administrators can test the processing of untrusted content without damaging the system.

X ACCESS CONTROL EXTENSION (XACE)

The X Window System (commonly referred to as "X") provides the base framework for displaying the graphical user interface (GUI) on Red Hat Enterprise Linux 6. This release features the new X Access Control Extension (XACE), which permits SELinux to access decisions made within X, specifically, controlling information flow between window objects.

9 For more information on the Fedora Project, visit **fedora.redhat.com.** For more information on getting started with SELinux in Fedora Core 2, see the Fedora Core 2 SELinux FAQ in the Resources section of this paper.

CONCLUSION

SELinux introduced a new security paradigm-mandatory access control-to Linux. Mandatory access control uses a security policy to isolate applications from the operating system and from one another and to protect the integrity and confidentiality of information. While mandatory access control provides many benefits for security, it was traditionally limited to trusted systems that had limited adoption. Red Hat incorporated SELinux into its operating system releases, believing that this technology provides better security for all systems.

By default, over 200 core system services in Red Hat Enterprise Linux are protected by targeted policies. This enables organizations to quickly benefit from the security provided by SELinux. Red Hat Enterprise Linux also includes enhanced SELinux management tools that simplify the process of creating, customizing, managing, and troubleshooting SELinux policy. And SELinux development is continually ongoing in the Fedora open source community.

Ultimately SELinux should be viewed as a component of a larger set of security tools, technologies, processes, and procedures. Used in combination with these best practices, SELinux enables system administrators to isolate and minimize the effects of an increasing number of malicious attacks.

RESOURCES

RED HAT DOCUMENTATION

- **Red Hat Enterprise Linux 6 Security-Enhanced Linux Guide**

docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/6/html/Security-Enhanced_Linux/index.html

- **Red Hat Enterprise Linux 5 Deployment Guide**

redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/5/html/Deployment_Guide/index.html

SEE SECTIONS:

- Security and SELinux
- Working With SELinux
- Customizing SELinux Policy

NSA SELINUX WEB SITE

www.nsa.gov/research/selinux/

NSA SELINUX FAQ

www.nsa.gov/research/selinux/faqs.shtml

FEDORA SELINUX PROJECT PAGES

fedoraproject.org/wiki/SELinux

MAILING LISTS

www.redhat.com/mailman/listinfo/fedora-selinux-list

www.redhat.com/mailman/listinfo/gov-sec

DAN WALSH'S JOURNAL

danwalsh.livejournal.com/

RED HAT MAGAZINE

- What's New in SELinux for Red Hat Enterprise Linux 5

magazine.redhat.com/2007/05/04/whats-new-in-selinux-for-red-hat-enterprise-linux-5/

- Writing policy for confined SELinuxusers

magazine.redhat.com/2008/07/02/writing-policy-for-confined-selinux-users/

- Uli Drepper part 4: SELinux

magazine.redhat.com/2007/10/09/uli-drepper-part-4/

- A step-by-step guide to building a new SELinux policymodule

magazine.redhat.com/2007/08/21/a-step-by-step-guide-to-building-a-new-selinux-policy-module/

TRESYS TECHNOLOGY

oss.tresys.com/projects

RED HAT SALES AND INQUIRIES

NORTH AMERICA	EUROPE, MIDDLE EAST AND AFRICA	ASIA PACIFIC	LATIN AMERICA
1-888-REDHAT1	00800 7334 2835	+65 6490 4200	+54 11 4329 7300
www.redhat.com	www.europe.redhat.com	www.apac.redhat.com	www.latam.redhat.com
sales@redhat.com	europe@redhat.com	apac@redhat.com	info-latam@redhat.com