

A horizontal decorative bar consisting of five segments: a solid red segment on the left, followed by four segments with abstract, colorful patterns in shades of blue, orange, green, and purple.

Security Features in Red Hat Enterprise Linux 4

Abstract

This white paper provides a summary of new security features provided in the Red Hat Enterprise Linux v.4 product family.

Credits

The material in this paper is condensed from in-depth papers and presentations by Mark Cox, Chris Runge, and Arjan van de Ven. Additional material was provided by Charlie Bennett.

Revision 4b. February 2005



Table of Contents

Introduction.....	3
Delivering Security.....	3
Buffer Overflow.....	5
ExecShield.....	6
NX/XD.....	6
Application Segmentation.....	6
Position Independent Executables (PIE).....	7
Effectiveness.....	7
Compiler and Library Enhancements.....	8
Advanced Glibc memory corruption checks.....	8
Printf format string exploit prevention.....	8
GCC buffer bound checking.....	8
Security Access Control Overview.....	9
Security-Enhanced Linux	10
Red Hat and SELinux.....	11
SELinux architecture and operation.....	12
Examples of SELinux in Action: Apache HTTP Server.....	14
Deciding Where to Deploy SELinux.....	15
Auditing.....	15
Red Hat Network.....	16
Red Hat Enterprise Linux Support.....	18
The Secure Foundation for Complete Solutions.....	19
Conclusion.....	19
Resources.....	20



Introduction

The world of computer security has changed dramatically in the last few years, and 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 throughout the Internet infrastructure.

Companies such as Red Hat are releasing new technologies and tools to address the needs of system administrators responsible for managing the security of large numbers of geographically dispersed systems. For example, technologies such as Position Independent Executables (PIE) and Exec Shield help protect against buffer overflows, a tactic frequently employed by attackers to infiltrate and compromise flawed software programs. Another technology, Security-Enhanced Linux (SELinux), prevents users and applications from damaging an entire system by enforcing security policies at the kernel level.

On the tools front, Red Hat Network, a key part of Red Hat Enterprise Linux, offers system administrators a way to review information about security vulnerabilities and pro-actively apply relevant security and other updates to large numbers of Red Hat Enterprise Linux systems easily and efficiently.

This whitepaper examines the benefits of these new security technologies and describe how Red Hat is working to make those benefits available to mainstream enterprise computing.

Delivering Security

Network computer security used to be about a dedicated hacker trying to get into a single government computer; nowadays it often is about automated mass attacks. The SQL Slammer and Code Red worms were the first such wide scale computer security incidents to get mainstream press coverage. Another relatively common phenomenon is that compromised computers are primarily being used for other purposes such as for sending spam or for participating in Distributed Denial of Service (DDOS) attacks. A big contributing factor to this mass-compromise problem is that a large portion of users and system administrators generally do not apply security fixes as provided by the operating system vendor, leaving a significant number of vulnerable machines connected to the Internet at all times. The combination of these factors leads to the conclusion that just providing security updates after the fact is not sufficient. Operating system providers need to be more proactive in combating security problems.

Recent technology enhancements directly target these issues by ensuring that operating systems provide a more secure environment for applications. They allow applications to be cocooned in an operating environment that does not allow them to compromise the entire system regardless of:

- Potential flaws that may be inherent in the application.



- Security weaknesses in the application that may be exploited by an attacker.

The Red Hat Enterprise Linux family of products has provided a highly secure environment since its original delivery in early 2002. It is worth noting that a recent survey of the security of Microsoft Windows XP showed that the time taken for an unpatched system connected to the Internet to be compromised had fallen from approximately 40 minutes to 20 minutes.

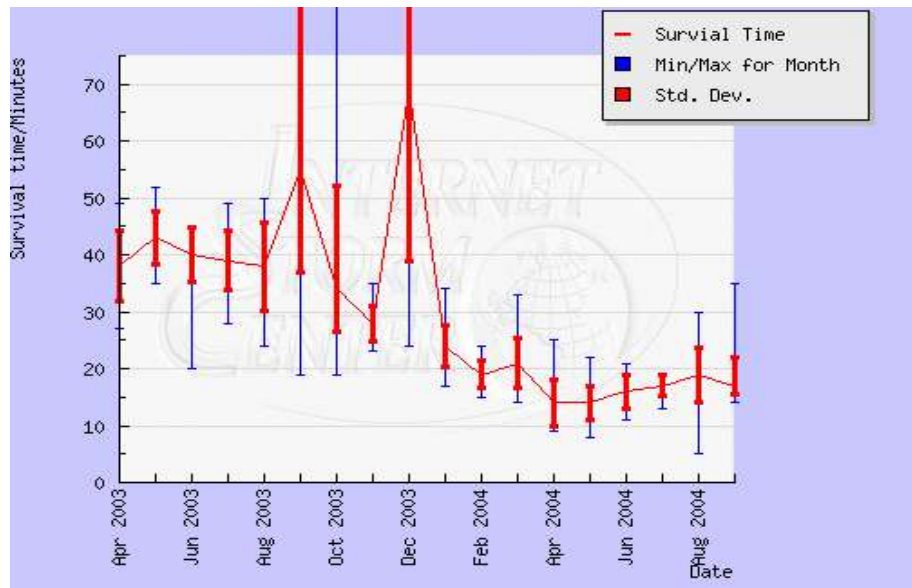


Figure 1. Average survival time for a Windows system connected to the Internet . (SANS Institute, www.sans.org)

At a recent security seminar a Microsoft presenter admitted that a system would typically be compromised before its initial installation was complete. A similar study performed by Red Hat on a Red Hat Enterprise Linux system showed that an original default installation from early 2002 would still be uncompromised. Nevertheless, Linux systems are not immune from attack, and Red Hat has continued to provide security enhancements as they become available. The following features have recently been made available:

Red Hat Enterprise Linux v.3 Update 3, shipped in September 2004:

- Two ExecShield features:
 - No eXecute (NX), or eXecute Disable (XD)
 - Segmentation
- Position Independent Executables (PIE)

Red Hat Enterprise Linux v.4, shipped in February 2005:

- Security-Enhanced Linux
- Compiler and library enhancements

- Advanced glibc memory corruption checker
- printf format string exploit prevention
- gcc buffer bound checking

The following sections look at the technologies in further detail. To start, however, a brief description of the most common form of security exploit, the *buffer overflow*, is required.

Buffer Overflow

The stack frame image shown in Figure 2 shows the memory layout of a typical function (or subroutine) that uses a fixed size I/O buffer.

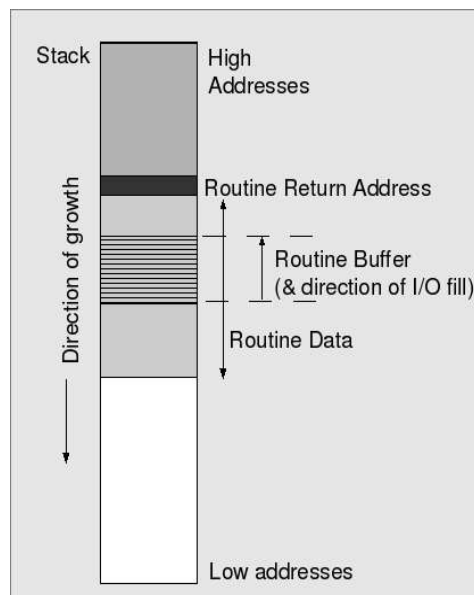


Figure 2. Memory layout of a typical function that uses a fixed size I/O buffer

This routine buffer is stored on the stack and is located before the memory buffer containing the address of the program code that invoked the subroutine (the routine return address). When the subroutine is finished, this address is used to resume the program at the point of the subroutine invocation. On Intel-compatible processors, the stack grows in a downward direction over time (as seen in Figure 2). This is why the buffer is stored below the return address. A buffer overflow exploit operates by virtue of a defect in an application. For example, an exploit can trick a subroutine to put more data into the buffer than there is space available. This surplus of data is stored beyond the fixed size buffer, including the memory location that has the return address stored. By overwriting the return address (which holds the address of the memory location of the code to execute when the subroutine is complete), the exploit has the ability to control which code is executed when the subroutine finishes. The simplest and most common approach is to make the return address point back to malicious code that has been transferred into the buffer.



ExecShield

The ExecShield feature in Red Hat Enterprise Linux supports two technologies that protect applications from being compromised by most buffer exploits. The goal of these features is to prevent code that is maliciously deposited in data areas of an application from being executed. These features—NX/XD and Segmentation—use different techniques to achieve the same result.

NX/XD

No eXecute and eXecute Disable are two terms for the same processor capability. NX is the term used by AMD for its Opteron/Athlon64 processors, while XD is the term used by Intel for its Itanium2 and recent x86/EM64T processors. The capability provides a new memory management feature that allows individual pages of an application's memory to be marked as not executable. Previously the only level of control over memory pages was read and write, and a page that was enabled for read could also be executed. This meant that data areas such as the stack and I/O buffers, which are typically only used for read/write activity, could also be used to execute code. As described above, a common form of hacker exploit involves depositing code in a stack buffer and then executing it, so the ability to disable execution allows a significant increase in application and system security.

NX/XD support is available for most modern processors, including recent model x86 CPUs. Red Hat Enterprise Linux will utilize this feature whenever possible. Note that some applications are actually designed to execute code from buffers and on the stack so, by default, this feature is disabled on a per-application basis. By rebuilding the application with the appropriate compiler option, an image flag can be set to control the enabling of the feature. Red Hat Enterprise Linux v.3 Update 3 provided a number of network-facing applications that were NX/XD enabled; Red Hat Enterprise Linux v.4 will enable NX/XD for all possible applications.

Application Segmentation

Application Segmentation provides a capability that is very similar to NX/XD but is designed for the vast installed base of x86 systems that do not have the NX/XD hardware feature.

As with NX/XD, the goal is to prevent execution of code from stack or data buffer areas. Segmentation uses the little-known x86 processor memory management segmentation feature to split an application into two segments—executable and non executable as shown in Figure 3. As with NX/XD, Segmentation makes application exploits much more difficult to construct. It is enabled/disabled in the same manner as described for NX/XD earlier. Segmentation provides a less granular approach to preventing execution of data as code at the segment level as opposed to NX/XD, which operates at the per-page level, but it is equally effective.

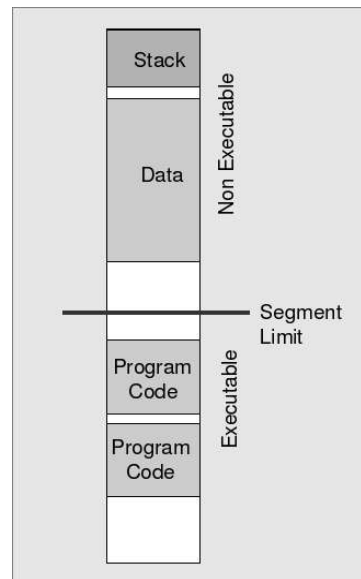


Figure 3: Segmentation

Position Independent Executables (PIE)

PIE is a feature that further complicates life for the aspiring hacker. NX/XD and ExecShield prevent data in buffers or on the stack from being executed as code, but they do not prevent exploits that, for example, change the return address of a routine on the stack so that program control is transferred to an incorrect part of the application. This type of exploit might allow routines such as security checks to be bypassed.

Without PIE, any given application is typically loaded into the same memory addresses each time it runs. Security exploits that manage to trigger incorrect sections or routines of an application to run are only effective if they know where the sections are located in memory. With PIE enabled, different sections of an application (and most applications have many sections) are randomly loaded into different memory locations each time the application runs. This makes it significantly harder for these exploits to succeed.

Effectiveness

The effectiveness of the ExecShield and PIE features has been measured against the known Linux security issues over the past year. In the period from November 1, 2003 to August 11, 2004, there were 16 security issues published for Linux that are more severe than a Denial of Service problem and for which an exploit was made available. Out of these 16 exploits, four were for kernel security holes which were not addressed by these features. Eleven were stack buffer overflows, and one was a heap buffer overflow all of which were stopped by ExecShield and PIE. So, using these technologies yielded a success rate of 75%. Note that these exploits were stopped even on processors without NX/XD technology.

As these examples prove, ongoing enhancements to the Red Hat Enterprise



Linux operating system continuously improve its security. All the previously mentioned features are provided by the operating system itself and do not rely on application enhancements (beyond recompiling). By ensuring that the core operating system provides a high level of security, one poorly written application does not represent a threat to the entire system.

Compiler and Library Enhancements

During late 2004, Red Hat developed a new group of features that improve buffer management and security for inclusion in Red Hat Enterprise Linux v.4. At the time of writing these features are unique to Red Hat environments.

Advanced GLIBC memory corruption checks

The GLIBC memory allocator functions now perform a set of internal sanity checks to detect double freeing of memory and heap buffer overflows. With these checks, regular application bugs and security exploit attempts that use these techniques are detected, and the program will be instantly aborted to avoid the possibility of the exploit succeeding. With these checks, double free exploits become entirely impossible, and all standard, generic heap overflow techniques are blocked.

These detection routines execute prior to the ExecShield features described earlier. In many cases exploit attempts of this type would ultimately be blocked by ExecShield, but these memory corruption checks provide an extra level of security because the sooner an exploit attempt is detected and aborted the better.

Printf format string exploit prevention

Printf format string exploits abuse a bug in programs that have a faulty call to the standard printf() function, caused by a very rarely used formatting parameter. When applications are compiled with the "-D_FORTIFY_SOURCE=2" compiler parameter, the printf function will check that this rare formatting comes from guaranteed trusted sources and will abort the program if that is not the case, thus preventing printf format exploits entirely.

Printf format exploits were popular about 2 years ago when the technique was first promulgated. Today most of the flawed code has been fixed but it is still possible that some applications suffer from this problem.

GCC buffer bound checking

This feature detects many potential buffer overflow conditions and is the most important of the buffer management improvements.

An enhancement has been added to the GCC compiler such that if the size of the destination buffer can be detected at compile time, functions such as strcpy(), memcpy(), strcat() will use a checking variant of these functions that detects if the buffer will actually overflow. If that happens the program is aborted immediately. While gcc cannot always detect the size of the destination buffer (for example, it is not possible for dynamically allocated buffers), buffer allocation errors usually occur with the types of buffer that can be detected by



gcc. The result is that a large percentage of buffer overflow errors are prevented immediately. Once again, it is probable that the ExecShield feature would have stopped execution of the buffer later on, but gcc buffer bound checking occurs much earlier and is thus more secure. Additionally, the feature does not require applications to be PIE or ExecShield enabled.

These checks are enabled when "-D_FORTIFY_SOURCE=1" or "-D_FORTIFY_SOURCE=2" is added to the gcc command line options.

Security Access Control Overview

Multi-user operating systems such as Linux, UNIX, Windows, and Mac OS X use access control mechanisms to determine whether and how the users and programs on that system can access objects (for example: files, directories, and sockets) on the system. Actions such 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 operating systems use an access control mechanism known as *discretionary access control*. Under a discretionary access control (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 in turn what that user, group, or others can do with that object, such as reading, writing, and/or executing it. This relatively simplistic yet powerful scheme allows multiple users and programs to coexist on the same system, and if the permissions are properly set, ensures that users have control over their objects but no others.

This 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 full discretion over everything on the system. It follows that if a user owns files containing confidential information (such as financial information, patient data, and trade secrets) that user can intentionally or even mistakenly share that data with other users or programs that should not have access.

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, 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 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 be 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 is an alternative—and far less frequently used—access control mechanism. Found on what are known as trusted operating systems, mandatory access control, or MAC, 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. The security policy trumps the intentions of the user and is the final determiner of what takes place on the system. This mandatory nature of the security policy is what gives this access control mechanism its name.

Mandatory access control systems lack the core concept of an all-powerful root user; 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 done 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.

Mandatory access control offers clear security benefits, so why has its adoption been limited? There are several reasons. First, the trusted operating systems that provide mandatory access control have largely been separate offerings from their more mainstream counterparts. Due to a traditionally limited audience that consists largely of specialized military and intelligence applications, these trusted systems have received less focus from operating system and application vendors. They tend to lag behind their commercial cousins in terms of functionality as well as hardware and application support. Moreover, MAC systems can be difficult and time-consuming to implement and use. All of this results in a small community of use, which further serves as a disincentive for vendors to pour additional resources into the product. Today, trusted operating system use is largely confined to specific applications and deployments in the military and intelligence communities. Even then, security features may be disabled or weakened with users having a natural preference for functionality and ease-of-use over increased security.

Security-Enhanced Linux

Security-Enhanced Linux (SELinux) is an open-source research project sponsored by the National Security Agency (NSA) to provide mandatory access control in Linux. The NSA chose Linux as the basis for this project because of its large community of use and its open-source development model.¹ 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 increasing popular operating system.

Recognizing that secure applications require a secure operating system, SELinux provides security at the kernel level through the added

¹ <http://www.nsa.gov/selinux/>

implementation of a security policy and a separate enforcement mechanism. By adopting 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 4). 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.

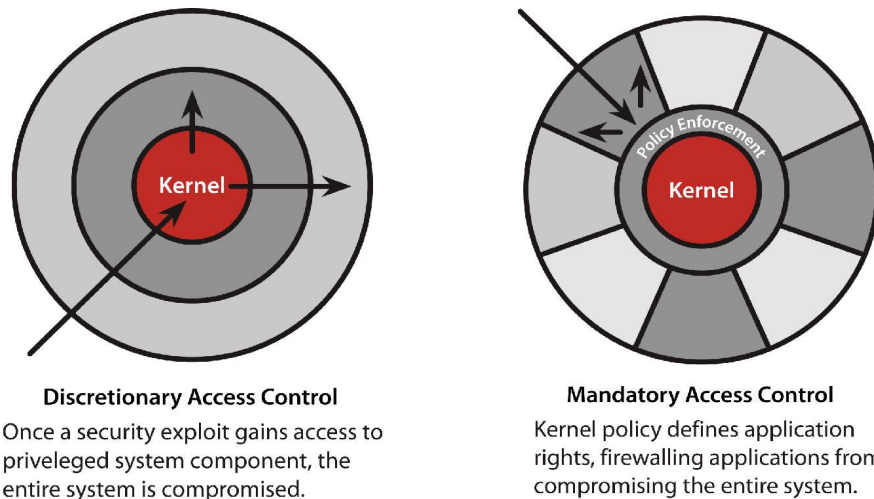


Figure 4: Difference between Discretionary Access Control and Mandatory Access Control environments

Red Hat and SELinux

Enter into this backdrop Red Hat Enterprise Linux v.4, released in February 2005, which includes a full implementation of SELinux that is integrated with all the existing security features. It is important to realize that all the security features inherent in Red Hat Enterprise Linux continue to play a vital role alongside the new SELinux capability.

Red Hat recognized the immediate and potential security benefits that SELinux offered in attaining its goal of providing higher levels of security for all systems, so Red Hat made a strategic decision to adopt, incorporate, and further the development of the SELinux project.

As a core part of this effort, Red Hat commercialized it by providing the necessary commercial support for SELinux across the full Red Hat Enterprise



Linux product family. Until recently, deploying an SELinux-enabled system required patching the kernel and several user-space tools from an existing Linux distribution using code provided from the NSA's SELinux website. 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. Nonetheless, many organizations, especially those requiring commercial support for mission-critical systems, delayed implementation until these capabilities were included in a generally available, enterprise-class Linux distribution, such as Red Hat Enterprise Linux v.4.

Red Hat realized that to release SELinux only as a part of a separate trusted operating system would be to repeat the mistakes of the past. For SELinux—and the security benefits that it provides—to be more widely adopted, SELinux needs to be integrated into the mainstream operating system. 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 will have several options around the configuration of SELinux on their systems. For those who desire to adhere to the traditional discretionary access control model, SELinux can be completely disabled. On systems where SELinux is 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.

The latest release of Red Hat Enterprise Linux, version 4, released in February 2005, features the 2.6 Linux kernel and SELinux as two of its major features. With this release, organizations are able to deploy a commercially supported SELinux-enabled operating system for production and mission-critical use.

SELinux architecture and operation

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 components for recognizing and handling security contexts, role changes, policy development, and other tasks necessary for implementing mandatory access control on the system.

SELinux combines two security mechanisms to achieve a flexible security policy model: *type enforcement* (TE) and *role-based access control* (RBAC). Type enforcement implements a mechanism whereby processes (running programs) are placed in what are known as *domains*, which govern what these processes can do. 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 constrained by the security policy on the system from interfering with other applications 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. Those roles are then in turn assigned to users. Users can belong to multiple roles, with access dependent on the current role being used.

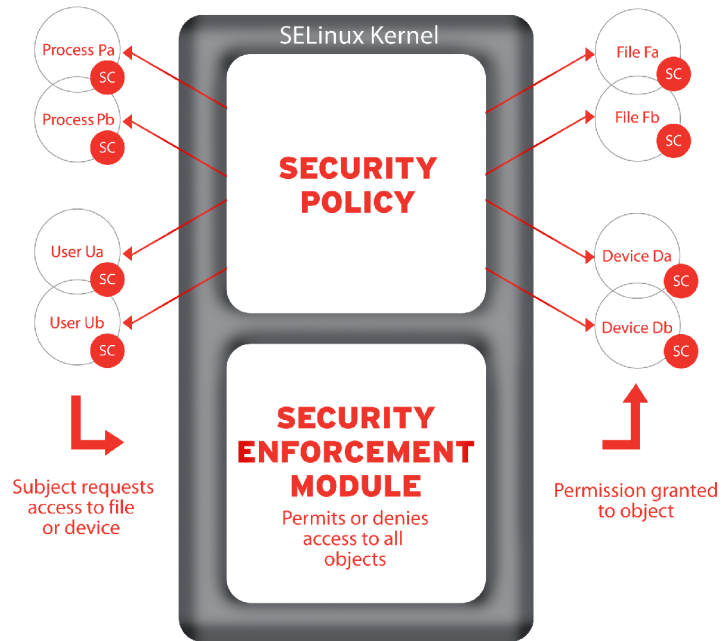


Figure 5: SELinux Access Control

How does all of this work in actual operation? Files and processes are labeled with a *security context*, which is made up of the user, the role, and the domain or type. When access is requested, the traditional UNIX permission system used by the 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 5). All access requests and their corresponding policy decisions may be audited, if required.



Examples of SELinux in Action: Apache HTTP Server

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

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 webpages, 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 to 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 private domain, 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 the 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 to function
- Read-only access to the system's webpages 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 was compromised, the potential damage would be confined. The system's webpages in `/var/www/html/` could not be defaced. The attacker could not use the web server to read files on the system other than webpages, 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 system as well as other applications running on the system would remain unaffected.

Deciding Where to Deploy SELinux

As seen above, Internet-facing edge servers running daemon, such as the Apache HTTP server, stand to gain immediate benefits from the use of SELinux. Other mission-critical infrastructure systems—bind, 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 systems used by the military and intelligence community but any confidential or sensitive information. Examples include human resource data (such as payroll and personnel files), patient data (such as that protected by HIPAA), and financial data (such as 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 applications (such as Mozilla and Evolution) that might potentially be misused as an agent for sending and receiving malicious software such as viruses and trojans.

Auditing

The ability to audit system behavior is a natural adjunct to the delivery of a highly secure environment. The ability to monitor, log, and analyze system events allows the overall security of the system to be understood and managed and then adapted to meet changing requirements. To meet this important requirement, Red Hat Enterprise Linux v.4 also provides a new lightweight audit framework, called *audit²*. Audit was developed by Red Hat and has been accepted into the upstream Linux kernel. It is intended to be a way for the kernel to provide various types of audit information to user space without impacting performance, especially when auditing is not being used. The framework, which replaces the previous auditing capability (LauS) is designed to audit both SELinux and non-SELinux systems. Major goals of audit are to provide a simple, elegant, generalized implementation that can be used by multiple audit record consumers (such as Snare and trace packages). Beyond simply auditing events, a set of reporting tools is also provided.

System call auditing is supported, although turned off by default. It can be turned on by a suitably privileged user-space process, which can then load a

² Audit will be provided for Red Hat Enterprise Linux v.4 in the first half of 2005.



set of rules describing what should be logged. Rules can test on various attributes of the calling process, including its process ID, user and group ID, etc. Rules can also be set to fire on accesses to particular devices or files. Finally, there are also tests on specific system call arguments, whether the call succeeds or for a specific return value. These rules can be adjusted at runtime.

The new audit mechanism gives system administrators a new tool for monitoring and managing what is going on between user space and the kernel and will also provide the necessary auditing infrastructure to allow Red Hat Enterprise Linux to achieve NIAP Common Criteria/EAL4+ certification.

Red Hat Network

As mentioned in the introduction, Red Hat Network (RHN) is Red Hat's Internet-based system maintenance and management infrastructure. RHN's core capability is to analyze all the packages on a Red Hat Enterprise Linux system and identify packages for which updates are available and resolving any dependencies that the packages require. RHN can then apply the updates as required. RHN's strength lies in being able to manage hundreds of systems automatically, with features such as system grouping, automatic updates, continuous monitoring and alerts, etc. Using RHN greatly simplifies the process of keeping multiple systems up-to-date and secure. As Red Hat continuously provides security enhancements, RHN provides the mechanism to ensure that they are applied to customer systems quickly and efficiently.

RHN provides several modes of operation, Hosted, Proxy, and Satellite:

- In Hosted mode each managed system connects across the Internet to an RHN server hosted by Red Hat. The RHN server will inventory the system's packages and apply updates as required.

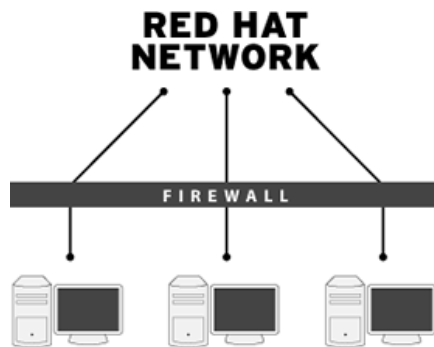


Figure 6: RHN Hosted Model

- Proxy mode is useful for reducing Internet bandwidth consumption when a customer has multiple systems. Updates are cached at the customer's site in a local Proxy server, so the customer only needs to download them from Red Hat RHN servers once. Package inventorying is still performed by Red Hat's RHN servers, allowing Red Hat Network to notify administrators immediately an update is available.

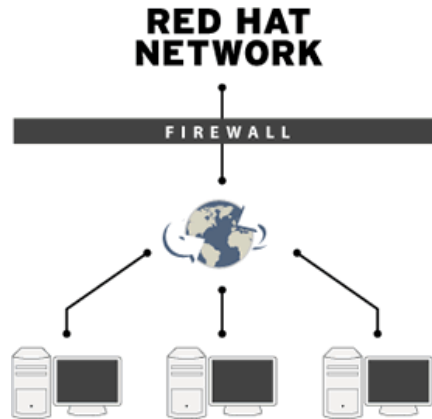


Figure 7: RHN Proxy Model

- Satellite mode provides a fully disconnected RHN environment, where package inventorying and caching is all performed using a Satellite Server located on the customer site. Updates from Red Hat are performed on an as-needed basis. A Satellite server can be used to deliver and update customer-specific applications.

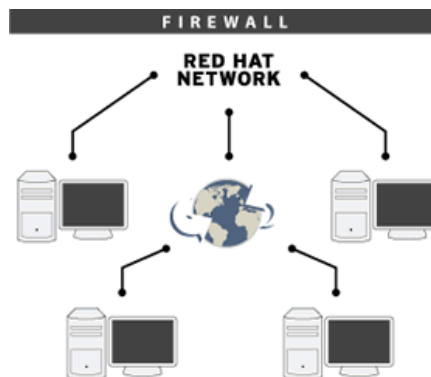


Figure 8: RHN Satellite Model

RHN also provides multiple management levels: Update, Management, and Provisioning. Update provides the basic ability to keep systems up-to-date; Management provides additional features such as system grouping and role-based permissions; Provisioning provides bare-metal system deployment capabilities (allowing new systems to be up-and-running in the shortest possible time) and multi-state rollback. For additional information on Red Hat Network please refer to <http://rhn.redhat.com/>.



Red Hat Enterprise Linux Support

When designing a system for long-term deployment the availability of on-going vendor support is a critical consideration. In fact, during the 1990s the immaturity of Linux support offerings was a clear hindrance to its adoption for mission critical deployments. So, when Red Hat created its Red Hat Enterprise Linux family of products a key feature was the inclusion of comprehensive and flexible services. As a result, Red Hat Enterprise Linux is delivered on an annual subscription basis, which includes all security errata and updates (delivered using Red Hat Network), system upgrades to any new versions that are released during the subscription period, and support for unlimited service incidents. Support is available on a Monday-Friday 9am-9pm (USA) with 4 hour response basis or 24x7 with a 1 hour response (for the latest details please refer to <http://www.redhat.com/>).

Each version of Red Hat Enterprise Linux is supported for seven years from the date of initial release with updates provided on (approximately) a quarterly basis, while new versions of Red Hat Enterprise Linux are released approximately every 18-24 months. Updates include all the latest bug fixes and new technical features that are needed by customers and partners. Security errata are released as soon as they are available and also included in the regular updates. Red Hat offers a wide range of other services such as training and professional consulting services.

In terms of security fixes, Red Hat Enterprise Linux has achieved a remarkable track record. With security issues, the two metrics to consider are fix time and severity. Clearly, high severity issues need to be fixed rapidly, while low severity issues can be handled less urgently. Obscuring these numbers is a classic way for vendors to present their handling of security problems in a positive light (for example, by focusing on the fix time without regard to the severity of the issue). For Red Hat Enterprise Linux v.2.1 and v.3, over 70% of “critical” security issues and almost a third of all security issues were resolved in less than 1 day. Table 1 provides an overview of results to date.

Table 1: Fix time for security issues in Red Hat Enterprise Linux

Rating	Median Fix Time	<1 Day
Critical	0	77%
Important	4	42%
Moderate	11	24%
Low	31	16%
All	12	31%



The Secure Foundation for Complete Solutions

To round out the Red Hat Enterprise Linux story, the product family provides the most extensive set of application and hardware certifications in the industry, ensuring that the applications that customers need are available and that they can be deployed on the widest range of hardware systems. By late 2004, Red Hat Enterprise Linux had been certified for use with over 1000 applications supplied by more than 300 ISV partners on over 600 hardware platforms.

Lastly, Red Hat Enterprise Linux had garnered an impressive portfolio of industry benchmark results, including the overall world record TPC/C performance result and numerous other world records at many differing system sizes and configurations (such as TPC/H, SPECweb, SPECjAppServer, SPECComp, Oracle Applications Standard Benchmark, etc.)

Conclusion

This paper has outlined the many facets of security in a Red Hat Enterprise Linux environment from kernel technical features through monitoring/auditing capabilities to support services. It is a combination of all these capabilities that makes Red Hat Enterprise Linux the industry-leading commercial Linux solution.

- ExecShield and PIE technologies provide greatly increased resilience to security attacks, and are effective for all applications.
- Compiler and library enhancements improve buffer management capabilities, detecting coding flaws and making applications more secure.
- SELinux introduces 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.
- The new auditing capability enables systems to be closely monitored to ensure that the desired level of security is being achieved.
- Red Hat Network provides an efficient, low-cost infrastructure for keeping multiple systems updated and secure automatically
- Red Hat Enterprise Linux is offered with a range of support services and is provided with a full seven years of updates and security errata for every version.

In combination, these technologies and supporting services enable customers to deploy highly secure Red Hat Enterprise Linux solutions successfully in long-term, mission critical environments.



Resources

1. NSA SELinux web site.
<http://www.nsa.gov/selinux>
2. NSA SELinux FAQ.
<http://www.nsa.gov/selinux/info/faq.cfm>
3. Red Hat SELinux web page.
<http://www.redhat.com/solutions/security/SELinux.html>
4. Fedora Core 2 SELinux FAQ.
<http://people.redhat.com/kwade/selinux/selinux-faq/selinux-faq-en>
5. Mailing list: fedora-selinux-list.
<http://www.redhat.com/mailman/listinfo/fedora-selinux-list>
6. The Unofficial SELinux FAQ.
http://sourceforge.net/docman/display_doc.php?docid=14882&group_id=21266
7. Getting Started with SELinux HOWTO: The New SELinux.
http://sourceforge.net/docman/display_doc.php?docid=20372&group_id=21266
8. Writing SELinux Policy HOWTO.
http://sourceforge.net/docman/display_doc.php?docid=21959&group_id=21266