# Operating System Security and Secure Operating Systems

By Cui-Qing Yang

Version 1.4b, Option 1 for GSEC
January 2003

## *Abstract*

One of the fundamental concerns in the security of cyberspace and e-commerce is the security of operating systems that are the core piece of software running in all information systems, such as network devices (routers, firewalls, etc), Web servers, customer desktops, PDAs, and so on. Many of known vulnerabilities discovered so far are rooted from the bugs or deficiency of underneath operating systems.

This paper discusses the security (or lack of security) of most commercial operating systems like Unix and Microsoft Windows, and its effect to the overall security of Web based applications and services. Based on DOD's trusted computer system model, the current effort toward development of secure operating systems is presented, and as a case study, the publicly available security enhanced Linux, SE-Linux, is also analyzed.

## 1. Introduction

Every modern computer system, from network servers, workstation desktops, to laptops and hand-held devices, has a core piece of software, called kernel or operating system, executed on the top of a bare machine of hardware that allocates the basic resources of the system (e.g., CPU, memory, device driver, communication port, etc), and supervises the execution of all applications within the system. Some popular commercial and Open Source operating systems are Microsoft Windows, different flavors of Unix (BSD, AIX, HP-UX, Solaris, etc), Mac OS, and Linux.

Because of the crucial role of the operating system in the operation of any computer systems, the security (or lack of security) of an operation system will have fundamental impacts to the overall security of a computer system, including the security of all applications running within the system. A compromise of the underneath operating system will certainly expose danger to any application running in the system. Lack of proper control and containment of execution of individual applications in an operating system may lead to attack or break-in from one application to other applications.

Based on the "Trusted Computer System Evaluation Criteria" of US government [1], the security level of most commercially available operating systems are no higher than C2 class, which requires Discretionary Access Control (DAC) protection at a per user granularity. Although this level of protection provides safeguard of certain extent among different applications in a multi-tasking, time-sharing environment that is typical for current mainstream operating systems, no mechanisms are supported by operating systems in this class to enforce strict security policies of individual applications. As a result, in a C2 class operating system the security of applications and users are responsible for their own fates.

With the ever-growing connectivity and E-commerce through the Internet, application security is an ultimate goal for millions of merchants and consumers who turn their business and service electronic and to the public world of cyberspace. On the other hand, efforts to achieve total security of such systems continue to be based on the flawed promise that adequate security can be achieved in applications with the current security mechanisms of mainstream operating system [2]. The reality is that secure applications demand secure operating systems, and tackling application compromises at the OS level by kernel-enforced controls should probably be considered as an attractive and effective approach.

In order to raise the security level of operating systems to next class – B class, the requirement of Mandatory Access Control (MAC) is a necessity. A typical MAC architecture needs the ability to enforce an administratively set security policy over all subjects and objects (users, processes, memory, files, devices, ports, etc) in the system, basing decisions on labels containing a variety of security-relevant information. MAC provides strong separation (or containment) of applications that permits the safe execution of untrustworthy applications, and enables critical processing pipelines (trusted path) to be established and guaranteed. Therefore, it offers critical support for application security by protecting against the tampering with, and bypassing of, secured applications. The benefits derived from MAC would never be possible with the existing DAC operating systems.

Many efforts have been devoted in defining and developing security model of trusted computer systems, requirements and architecture of secure operating systems. The results of some earlier research projects, such as Flask [3], and DTOS [4] were widely available in public. The emerging of more secure operating systems as commercial products and public domain software, e.g., HP-LX [5], SE-Linux [6], and Trusted Solaris, in recent years may indicate a new trend that attentions to the overall security of applications are duly focusing more on the root causes of the security of underneath operating systems.

The remainder of this article begins with a general examination of potential risks resulting from the compromise of an application due to the lack of proper operating system security; and followed by a summary of the security model of DOD's trusted computer system evaluation criteria. Then, based on the discussion of security requirements and general architecture of secure operating systems, a case study of the publicly available security enhanced Linux, SE-Linux, is presented at the end.

## 2. Security of Operating Systems

Most modern information computer systems provide concurrent execution of multiple applications in a single physical computing hardware (which may contain multiple processing units). Within such a multitasking, time-sharing environment, individual application jobs share the same resources of the system, e.g., CPU, memory, disk, and I/O devices, under the control of the operating system. In order to protect the execution of individual application jobs from possible interference and attack of other jobs, most contemporary operating systems implement some abstract property of containment, such as process (or task) and TCB (Task Control Block), virtual memory space, file, port, and IPC (Inter Process Communication), etc. An application is controlled that only given resources (e.g., file, process, I/O, IPC) it can access, and given operations (e.g., execution or read-only) it can perform.

However, the limited containment supported by most commercial operating systems (MS Windows, various flavors of Unix, etc) bases access decisions only on user identity and ownership without considering additional security-relevant criteria such as the operation and trustworthiness of programs, the role of the user, and the sensitivity or integrity of the data. As long as users or applications have complete discretion over objects, it will not be possible to control data flows or enforce a system-wide security policy. Because of such weakness of current operating systems, it is rather easy to breach the security of an entire system once an application has been compromised, e.g., by a buffer overflow attack. Some examples of potential exploits from a compromised application are [5]:

- Use of unprotected system resources illegitimately. For example, a worm program launches attack via emails to all targets in the address book of a user after it gets control in a user account.
- Subversion of application enforced protection through the control of underneath system. For example, to deface a Web site by gaining the control of the Web server of the site, say changing a virtual directory in Microsoft IIS.
- Gain direct access to protected system resources by misusing privileges. For example, a compromised "sendmail" program running as root on a standard Unix OS will result in super user privileges for the attacker and uncontrolled accesses to all system resources.
- Furnish of bogus security decision-making information. For example, spoof of a file handle of Sun's NFS may easily give remote attackers gaining access to files on the remote file server.

It is not possible to protect against malicious code of an application using existing mechanisms of most commercial operating systems because a program running under the name of a user receives all of the privileges associated with that user. Moreover, the access controls supported by the operating systems are so coarse – only two categories of users: either completely trusted super users (root) or completely un-trusted ordinary users. As the result, most system services and privileged applications in such systems have to run under root privileges that far exceed what they really needed. A compromise in any of these programs would be exploited to obtain complete system control.

## 3. Model of Security

Generally, in an access control based security model, there are set of objects, and set of subjects (a subject itself can also be an object). Every object has an associated security attribute, or security label; every subject also has a security label, or security clearance; and a defined set of control rule, or security policy that dictates which subject is authorized to access which object.

For example, in military security model [7], a security label consists of two components: a security level with one of the four ratings: unclassified, confidential, secret, and top secret, where unclassified < confidential < secret < top secret, and "<" means "less sensitive than"; a set of zero or more categories (also known as compartments) that describe kinds of information, for instance, the names CRYPTO, NUCLEAR might mean information about cryptographic algorithms, and nuclear related technology.

Given two security labels, (X, S1) and (Y, S2), (X, S1) is defined as being "at least as sensitive as" (Y, S2) iff X • Y and S2 ⊆ S1. For example,

(TOP SECRET, {CRYPTO, NUCLEAR}) > (SECRET, {CRYPTO})

where ">" means "more sensitive than".

In general, security labels are partially ordered. That is, it is possible for two labels to be incomparable, in the sense that neither is more sensitive than the other. For example, neither of the following is comparable to each other:

(TOP SECRET, {CRYPTO})
(SECRET, {NUCLEAR})

A more generalized hierarchy of security classes (or levels) with a mathematical basis was presented by Bell and La Padula in 1973 [8].

In its effort to address computer security safeguards that would protect classified information in remote-access, resource-sharing computer systems, the National Computer Security Center (NCSC), later DOD (Department of Defense), published an official standard called “Trusted Computer System Evaluation Criteria” [1], universally known as “the Orange Book”.

The Orange Book defines fundamental security requirements for computer systems and specifies a series of criteria for various levels of security ratings of a computer system based on its system design and security feature. A brief summary of all the ratings and their main characteristics are given as follows with a basic condition that each subsequent higher ratings must meet all the requirements of its lower ones.

***D – Minimal Protection***: no security is required; the system did not qualify for any of the higher ratings.

***C1 – Discretionary Security Protection***: the system must identify different users (or jobs) running inside the system, and provide mechanisms for user authentication and authorization to prevent unprivileged user programs from interfere each other (e.g., overwriting critical portions of the memory).

***C2 – Controlled Access Protection***: the system meets additional security requirements than that of C1 that include access control at a per user granularity (access control for any subset of the user community); clearing of newly allocated disk space and memory; and ability of auditing (logging) for security-relevant events such as authentication and object access, etc.

***B1 – Labeled Security Protection***: the system must implement the Mandatory Access Control in which every subject and object of the system must maintain a security label, and every access to system resource (objects) by a subject must check for security labels and follow some defined rules.

***B2 – Structured Protection***: few new security features are added beyond B1; rather the focus is on the structure (design) of the system to maintain greater levels of assurance so that the system behaves predictably and correctly (such as, a minimal security kernel, trusted path to user, and identified covert channels, etc).

***B3 – Security Domains***: more requirements to maintain greater assurance that the system will be small enough to be subjected to analysis and tests, and not to have bugs that might allow something to circumvent mandatory access controls, e.g., support of active audit, and secure crashing, etc.

***A1 – Verified Design***: no additional features in an A1 system over a B3 system; rather there are formal procedures for the analysis of the design of the system and more rigorous controls on its implementation.

Most existing commercial operating systems are with the ratings of C2 or below.

## 4. Requirements of Secure Operation Systems

As discussed in Section 2 and 3 above, most current operating systems provide discretionary access control, that is, someone who owns a resource can make a decision as to who is allowed to use (access) the resource. Moreover, because the lack of built-in mechanisms for the enforcement of security policies in such systems, the access control is normally a one-shot approach: either all or none privileges are granted, rarely supporting the “principle of least privilege” (without limiting the privileges a program can inherit based on the trustworthiness).

The basic philosophy of discretionary controls assumes that the users and the programs they run are the good guys, and it is up to the operating system to trust them and protect each user from outsiders and other users. Such perception could be extremely difficult to hold true and no longer be considered as secure enough for computer systems of “information era” with broad connectivity through the Internet and heavily commercialization of e-commerce services. Systems with stronger security and protection will require evolving from the approach of discretionary control towards the concept of mandatory (non-discretionary) control where information is confined within a “security perimeter” with strict rules enforced by the system about who is allowed access to certain resources, and not allow any information to move from a more secure environment to a less secure environment.

Some of basic criteria or requirements of a secure operating system are discussed below.

*Mandatory security* – a built-in mechanism or logic within the operating system (often called system security module or system security administrator) that implements and tightly controls the definition and assignment of security attributes and their actions (security policies) for every operation or function provided by the system. Generally, a mandatory security will require:

- A policy independent security labeling and decision making logics. The operating system implements the mechanism, whereas the users or applications are able to define security policies.
- Enforcement of access control for all operations. All system operations must have permission checks based on security labeling of the source and target objects. Such enforcement requires controlling the propagation of access rights, enforcing fine-grained access rights and supporting the revocation of previously granted access rights, etc.
- The main security controls include permission or access authorization, authentication usage, cryptographic usage, and subsystem specific usage, etc.

*Trusted path* – a mechanism by which a trustworthiness relationship is established among users and application software so that:

- A user or application may directly interact with trusted software, which can only be activated by either user or trusted software
- Mutually authenticated channel is needed to prevent impersonation of either party.
- The mechanism must be extensible to support subsequent addition of trusted applications.

*Support of diverse security policies* – traditional MAC mechanisms (such as the multi-level security – MLS [8]) are usually based its security decisions strictly on security clearances for subjects and security labels for objects (see Section 3), and are normally too restricted to serve as a general security solution. A secure architecture requires flexibility for support of a wide variety of security policies:

- Separation of security policy logic from the mechanism of policy enforcement, so that a system can support diverse security policies.
- Support for policy definition and policy changes with well-defined policy interfaces and formats.
- Provide of default security behavior of the system so that to maintain tight system security without requiring detailed system configuration.

*Assurance* – a process or methodology to verify the design and implementation of the system that should actually behave as it claims to be and meet the security requirements:

- The process generally involves two elements, (i) statement of the security properties a system is claimed to satisfy; and (ii) some kind of argument or evidence that the system does satisfy those properties.
- The structure of such systems normally requires a small security kernel or module so that the system behavior would relatively easy be verified.
- One of the concerns for a secure operating system is the so-called covert channels, which are the means to circumvent the security barrier enforced by the system in prevention of passing information from one security domain to a less secure domain. For example, one possible covert channel is a “timing channel”, where a Trojan horse program alternately loops and waits, in cycles of, say one minute per bit, and a program outside the perimeter that constantly tests the loading of the system may sense the information the Trojan horse intended to send. There is no general way to prevent all covert channels. It is more practical to introduce enough noise or reduce the bandwidth of such channels in the system so that they won’t be useful to an intruder.

The efforts for the development of secure operating systems can be dated to the earlier days of operating system development (e.g., Multics [9] and Hydra [10]).

With the rapid growth of Internet connectivity and e-commerce, recent development of secure operating systems spreads from traditional focus of defense or military related systems to more general commercial systems. As a case study, next section presents detailed discussions of a publicly available secure system from National Security Agency (NSA).

## 5. A Case Study of SE-Linux

In this section, NSA's SE-Linux is discussed as a case study of the recent efforts in the development of secure operating systems [6]. After a brief background introduction and a high level description of the architecture, a summary of how SE-Linux meets the general requirements of secure operating systems described in the previous section is presented.

### Background

As the ultimate gatekeeper of information security and assurance within USA, the National Security Agency (NSA) has been long involved in defining security criteria/requirements for information systems, and in the research of new model and architecture of secure information systems, including secure operating systems. The development of SE-Linux is indeed the results of several previous projects of NSA, especially the DTOS and Flask [3,4].

One notable feature of SE-Linux release is that it follows the same Open Source Initiative as that of the Linux. All documentation and source code of SE-Linux are publicly available at NSA Web site [6] under the same terms and conditions of Linux, hoping to reach a wide audience and to encourage further efforts and research of secure operating systems.

### Architecture

The SE-Linux is an adoption of the Flask security architecture in Linux operating system. Furthermore, the integration of the security architecture with Linux is accomplished in a way that a new kernel module, called the Security Server (SS) that implements the security policy decision logic, is added into a non-security-enhanced Linux (hereafter as ordinary Linux) that is patched with LSM (Linux Security Module) [11-13] for maintaining security attributes in kernel data structures and for the mechanism of security control enforcement. All policy-independent requests from operations of kernel objects (e.g., processes, devices, and files etc) are mandatorily enforced (via hooks of LSM) for decisions by SS based on the security labeling and security policies defined. Such a framework takes the advantage of existing functionality of the ordinary Linux, and keeps the required modifications to Linux kernel in minimum.

The SS of SE-Linux [11] implements the MAC in the form of identity and role based access control with type enforcement. A new data type called security context is maintained by the SS, which contains security attributes of identity, role, and type. The combinations of these attributes are used for access decisions with given security policies.

Security contexts are not directly bound to objects in the system. Instead, each object that requires a security label is assigned with a security identifier (SID) that is mapped to a security context. SID's are non-global and non-persistent, and the mapping between SID's and security contexts is maintained by SS at run time. With such security labeling, nearly every system operation of the ordinary Linux is now subject to security decisions or permission checks by Security Server's policy logic in respect to the security related attributes contained in the security contexts of the source and target objects that are associated with their SID's.

Every subject (process) of the system is assigned an identity that comes from a user when the user logs on to the system (this identity is orthogonal to Linux UID, and will remain unchanged even after a process changes its UID). A set of roles can be defined in security policies for individual users that may be entered by processes with the given user's identity, and the transition between roles may also be defined in the policy. As in role-based access control (RBAC), each role is specified by a security policy for allowable actions whenever a subject assumes the role. However, different from the typical RBAC in which permissions are directly granted to roles, type enforcement (TE) is used with roles for fine-grained access controls in SE-Linux. Each object in SE-Linux is assigned a type, and TE policies can be defined resulting in an access matrix that determines the permissions (e.g., transition, signal for process class, and create, write for file class) between a pair of types for each subject and object class that are supported in the kernel.

Security policies are specified in text-based policy configuration files using a simple language developed for SS. The policy configuration for a specific installation of SE-Linux is checked and compiled into binary and loaded at boot time into SS (if allowed by the policy, it may also be reloaded at runtime).

Some of policy templates used in SE-Linux are shown below:

*allow type_1 type_2: class { perm_1 … perm_n };*
*type_transition type_1 type_2 : process default_process_domain;*
*role rolename types { type_1 … type_ n};*
*user username roles { role_1 … role_n };*

## Meeting Requirements of Secure Operating Systems

The general requirements for a secure operating system (that is beyond the C2 class) are discussed in Section 3. Based on the analysis of SE-Linux architecture in this section, a summary of how SE-Linux meets the requirements of secure operating systems is listed as follows:

- Mandatory security – with the hooks in LSM and the decision-making policy logics of SS, all system operations in SE-Linux (even initiated by a super-user process) are subjected for mandatory access controls (permission checks) based on security labels of source and target objects.
- Trusted path – permission checks based on the security context of user identity, role definition, and type enforcement of the source and target objects involved in an operation ensures the security policies for transitions or transactions between user authentications, change of roles, and process domains/types, etc.
- Support of diverse security policies – all policy-independent security data types in the kernel and decision-making logics in SS make it a clear separation of mechanism from policy in the implementation of SE-Linux. A variety of diverse security policies can be defined to meet specific security objects. The example of a general-purpose security policy included in the release of SE-Linux could be used as default configuration for a normally secured system.
- Assurance – No formal assurance is provided for the security of SE-Linux, although the well-defined SS module and LSM framework could be a good basis for system assurance of SE-Linux. In addition, the issues of covert channels and security audit tails are not addressed in SE-Linux.

## 6. Conclusions

With ever growing security alerts and CERT Advisory for systems like Microsoft Windows and the ordinary Linux, people must be wandering how such games of cat-mouse-catching would ever be ended, and if there could be any better way to address the root causes of many of general vulnerabilities of information systems. The approach covered in this article – executing applications from a strongly guarded, secure operating system – certainly opens an alternative frontier in battling with many of existing cyber-space threats of the real world.

Although, the approach of using secure operating systems will not be a panacea for all the dangers of current cyber space, and the security of individual applications may still suffer from the vulnerabilities of their own, with the strong containment of a secure operation system, the damages caused from a compromise within one application would be much localized, and the impacts among various applications could be much well controlled.

As a demonstration of how mandatory access control can be integrated into a popular, main-stream operating system, the release of SE-Linux to general public assures that the usage of secure operating systems is not necessarily an expensive endeavor limited only to academic and defense related institutions, and encourages further efforts in research and development of secure operating systems. Not much testing result has been reported regarding the performance impacts and effectiveness of MAC of SE-Linux. It would be interesting to see some experimental deployment and test results using SE-Linux with real-world applications, such as Web servers for e-commerce services.

## 7. References

1. DOD 5200.28-STD, "DOD Trusted Computer System Evaluation Criteria" (Orange Book), 26 December 1985, http://www.radium.ncsc.mil/tpep/library/rainbow/5200.28-STD.pdf.
2. P. A. Loscocco, S. D. Smalley, P. A. Muckelbauer, R. C. Taylor, S. J. Turner, and J. F. Farrell, "The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments", *Proceedings of the 21st National Information Systems Security Conference*, pages 303-314, Oct. 1998, http://www.nsa.gov/selinux/doc/inevitability.pdf.
3. "Flask: Flux Advanced security Kernel", http://www.cs.utah.edu/flux/fluke/html/flask.html.
4. DTOS Technical Reports, http://www.securecomputing.com/randt/HTML/technical-docs.html.
5. Chris Dalton and Tse Huong Choo, "An Operating System Approach to Securing E-Services", *Communications of the ACM*, V. 44, No. 2, p. 58, 2001.
6. Security Enhanced Linux, http://www.nsa.gov/selinux/index.html.
7. Charlie Kaufman, Radia Perlman, and Mike Speciner, "Network Security: Private Communication in a Public World", *PTR Prentice Hall*, Englewood Cliffs, New Jersey, 1995.
8. D.E. Bell and L. J. La Padula, "Secure Computer Systems: Mathematical Foundations and Model", *Technical Report M74-244, The MITRE Corporation*, Bedford, MA, May 1973.
9. Ames, Stanley R., Jr., and J.G. Keeton-Williams, "Demonstrating security for trusted applications on a security kernel base", *IEEE Comp. Soc. Proc. 1980 Symp. Security and Privacy*, April 1980.
10. Wulf, W. A., Levin, R., and Harbison, S. P., "HYDRA/C.mmp: An Experimental Computer System", *McGraw-Hill*, New York, 1981.
11. Peter A. Loscocco and Stephen D. Smalley, "Meeting Critical Security Objectives with Security-Enhanced Linux", http://www.nsa.gov/selinux/doc/ottawa01.pdf.
12. Stephen Smalley and Timothy Fraser, "A Security Policy Configuration for the Security-Enhanced Linux", http://www.nsa.gov/selinux/doc/policy.pdf.
13. Linux Security Modules: General Security Hooks for Linux, http://lsm.immunix.org/docs/overview/linuxsecuritymodule.html.