Policy-Enhanced Linux

Paul C. Clark

Naval Postgraduate School
833 Dyer Rd., Code CS
Monterey, CA 93943-5118
E-mail: clarkp@cs.nps.navy.mil

Abstract

Graduates from the various computer fields need to have a better education in the area of computer security problems and their solutions. In particular, there appears to be little exposure to the enforcement of non-discretionary or Mandatory Access Control (MAC) policies in automated systems. One cause of this deficiency is the expense, limited availability, and limited functionality of operating systems that support such policies. This paper provides a detailed description of an effort to modify the Linux operating system to support MAC policies and overcome these obstacles, with the hope that it can be used to improve computer security education.

KEYWORDS: Education, Linux, MAC

1 Introduction

This section describes the necessity of having widely available and inexpensive systems that support Mandatory Access Control (MAC) policies at educational institutions. It also provides background material, requirements, and the supporting reasons for using Linux to achieve this goal. Section 2 describes the design of the new policy interface in an operating-system-independent manner. Section 3 describes the necessary Linux modifications, while section 4 provides conclusions and future work.

1.1 Need For Secure Operating Systems

In recent years a lot of emphasis, research and investment has been put into computer security products such as firewalls and intrusion detection systems (IDS's). Such products serve a useful purpose in supporting a multi-layer network defense. However, interest in these security products should not allow interest in the security aspects of operating systems to wane. Some of the reasons for continued diligence are listed below:

- With few exceptions firewalls and IDS's are applications installed on top of modern commercially available operating systems. If the operating system foundation is weak, then the mightiest security fortress built on top of it will crumble and fall.
- Firewalls are applied at network entry points to try to keep unwanted traffic off of internal networks. If a malicious user is able to bypass the firewall, then there must be another line of defense to thwart the attack; otherwise the network is "crunchy on the outside, yet soft and chewy on the inside."
- An IDS is used to detect attacks on an internal network and (hopefully) trigger defensive actions before the attack has been completed. This is certainly a useful feature, but such products are reactive in nature. In addition, an installation must be regularly updated with new attack signatures and scanning engines.

- From an Internet hacker's point of view, the operating system is the last line of defense against malicious outsiders.
- Additionally, the operating system is the first **and** last line of defense against insider attacks, universally accepted as the source of most security breaches.

1.2 Overview of Access Control Policies

Given any kind of information access control policy, the elements of the policy can be factored into their basic properties, which in turn can be grouped into one of three categories: a Discretionary Access Control (DAC) policy, a Mandatory Access Control (MAC) policy, or a Supporting Policy [1]. A Supporting Policy is one that is used to support the proper enforcement of a DAC or MAC policy.

A DAC policy is one where an object (e.g., file or directory) has a named user, or set of users, who can decide, at their discretion, who can have access to the object [2]. A MAC policy, on the other hand, implies that the owner of an object does not have control over who has access to it [2]. A good example is a file with a SECRET classification; the owner of the document does not have the discretion to control who has access to it because the policy states that a person must have the necessary clearance to view it.

1.3 Historical Background on MAC Systems

Systems that can withstand active and passive attacks from hostile users and outsiders have been research topics for decades. The result of some of this activity was the establishment of the Department of Defense (DoD) Computer Security Center and the publication of the Trusted Computer System Evaluation Criteria (TCSEC) in 1983, commonly referred to as the "Orange Book" [3], which was revised and republished in 1985. The Orange Book described the minimum requirements for various levels of assurance, and supported the evaluation of products that claimed to meet a level of assurance.

After the Orange Book was published, commercial companies became interested in meeting the stated needs of the DoD, as put forth in the criteria, with the idea that this was a valid market. Unfortunately, such ventures generally did not experience financial success over the next five to ten years, leaving some companies with the impression that MAC systems are not marketable. On the other hand, the impression left with users was that MAC systems are expensive and unusable.

Systems that enforce a DAC policy are easily obtained on the market, but it is much harder to find systems that enforce a MAC policy. The reasons for this disparity are not easily described, and tend to evoke strong disagreement between various camps of the government, research, and commercial sectors. The fact remains that there are few MAC systems, and they tend to be at least an order of magnitude more expensive than non-MAC systems.

1.4 Secrecy and Integrity Policies and Models

There are many MAC policies, but the two most common policies are some form of secrecy and/or integrity policies. A security model is a simple, abstract, precise and unambiguous representation of a security policy [1]. Modeling a policy before it is implemented is one way of increasing the assurance of the implementation. For systems that have a high degree of confidence in their correct enforcement of security policies, (i.e., high assurance systems), the model is typically expressed in mathematical terms, allowing it to be subjected to mathematical

proofs to expose any inconsistencies in the policy before it is implemented. For secrecy and integrity policies, the two models that often found their way into implementations were the Bell and LaPadula model[1] and the Biba model, respectively [2].

1.5 Advancing Computer Security

It is vital to our country's political and economic future to adequately protect corporate and government information from unauthorized disclosure and modification. Unfortunately, the current state of computer security is weak, especially when novice adversaries can perform successful infiltrations of sensitive systems. Systems that enforce Mandatory Access Control (MAC) policies are known to reduce some of the security weaknesses, but such systems have seen limited use within the DoD, and they have seen little or no use in the private sector. Some of this limited use is caused by a lack of awareness of the benefits of MAC systems, and the expense of current systems that do enforce a MAC policy.

Lack of exposure can be resolved through better education. There are probably many computer professionals who have never even heard of Mandatory Access Control. This is evidenced in the fact that few universities even offer an introductory course in computer security [4].

Lack of exposure could also be resolved if the cost of MAC systems was not so prohibitive. Ideally, every educational institution needing a MAC system to support its educational objectives could easily afford one, and every company that wanted to buy a MAC system could find one at the same price as a similar non-MAC system. In other words, the removal of cost as a deciding factor would be the ideal situation.

1.6 MAC Systems at Educational Institutions

The Computer Security Track of the Computer Science Department of the Naval Postgraduate School (NPS) teaches a class called *Introduction to Computer Security*, among other computer security courses [5]. This class is supported with a series of nine laboratory exercises, or tutorials, to re-enforce what the student is learning in class. Three of these tutorials are related to MAC policies. These three tutorials are critical in helping students fully understand MAC concepts.

The MAC tutorials are based on systems that are expensive to buy and maintain. This limits the track's ability to support the tutorials at distance learning sites, and prohibits other institutions from benefiting from our successes in this area because they are unable to buy the specialized hardware and/or software. Affordable MAC systems are absolutely necessary to provide the exposure and understanding of MAC systems within educational institutions.

1.7 Solution Requirements

The effort described in this paper researched the options for supporting an inexpensive Mandatory Access Control system. The requirements for such a system are given in the following subsections, along with a justification for each requirement.

1.7.1 Relatively Inexpensive

As expressed above, the current high cost of MAC systems is a barrier to greater exposure,

[1] Though it is often referred to as THE Bell and LaPadula model, the two authors actually published four distinct, yet related, models referred to as Volume I, Volume II, Volume III, and the Multics Interpretation.

acceptance, and demand. Lowering the cost of MAC systems will increase their exposure, which will then increase their acceptance, which will cause consumers to demand such functionality from operating system vendors.

1.7.2 Runs on PC Hardware

Requiring a MAC system to run on a PC may be viewed as a refinement of the previous requirement, though the previous requirement is directed at the cost of software; this requirement is directed at the cost of the hardware. By requiring the software to run on a PC, this eliminates the need for expensive and/or specialized hardware.

1.7.3 Dual-Boots with Other Operating Systems

This requirement is also cost-related. If, when a computer is booted, the user can choose between some number of operating systems to boot, the computer can have multiple uses. If the MAC system can be installed in a multi-boot configuration, then additional hardware and/or counter space is potentially not required to install and use it.

1.7.4 Easy to Use

As explained earlier, one of the complaints against MAC systems is their un-usability. This has two perspectives, both of which must be properly addressed: user and administrator. Users cannot be expected to learn many new commands or other interfaces, so the MAC system interface must be fairly intuitive. In addition, the administrative interfaces must be well documented with good security defaults. If an administrator is confused or frustrated with the system, then it will not be recommended for use.

1.7.5 Supports Secrecy and Integrity Policies

All current MAC systems support a secrecy policy because this policy is easily understood, both intellectually and in its application to an organization. Integrity, on the other hand, is not as easy to understand or apply, which may seem contrary to the previous requirement. From an educational point of view, however, a system supporting integrity is of great value, because it gives students hands-on experience with a somewhat difficult topic.

1.7.6 Supports the Setting of a Session Level

There are two basic types of user interfaces provided by MAC systems: 1) those that allow a user to read and write to any file during a session, as long as it falls within the user's clearance; and 2) those that require the user to specify the level at which reading and writing will be allowed during a session. The setting of a level as described in the latter interface is known as setting a session level. Experience in the NPS Computer Security Lab has shown that the former interface can be confusing to new users.

1.8 Proposed Solution

There is no product that meets the stated requirements. Building an operating system from scratch to meet the requirements would be an enormous undertaking, beyond the abilities of most people (and companies) to produce. The proposed solution is to modify an existing operating system. The Linux operating system has been chosen for this effort because it meets all the requirements except those related to MAC, (although some would argue that it is not user friendly) and because the source code is freely available.

The remainder of this paper describes modifications made to the Linux operating system to support additional access control policies [6]. This effort started as an attempt to support secrecy and integrity policies, but it soon became obvious that a flexible design would allow any new policy to be easily added to Linux. In this paper, the final design is referred to as "Policy-Enhanced Linux." Note that this effort does not attempt to provide a high-assurance product, nor one that is ready to be used in a "live" environment.

2 Linux Mandatory Access Control Design

This section describes the design of the new policy interface with the goal of providing a design that is independent of the operating system used to implement it. Section 3 continues by providing a design that incorporates these ideas into a Linux distribution. The designs presented in this section are intended to provide a somewhat generic interface to allow different policies to be "plugged into" an installation of Policy-Enhanced Linux, thus making a change in policy relatively easy to achieve.

2.1 New Databases

This section describes the new databases that must be added to Linux to support a new policy. In this context the term "database" refers to a passive object holding information.

In addition to the fields described, each new database has a field to track its version. This provides a mechanism for modules managing a database to identify older versions at run-time, and to act accordingly.

All the databases, with the exception of the Policy Label, will be stored in separate text files. This approach follows the standard Unix practice of having human-readable configuration files that can be modified using a text editor. It also has the added benefit of not requiring special administrative commands to be implemented. Such interfaces can be added later as a convenience for the System Administrator, but are not necessary for an initial implementation.

2.1.1 Policy Label

To support any kind of MAC policy, there must be a way of "attaching" some kind of label that describes the mandatory permission properties of the subject or object it is associated with. In a secrecy policy, for example, there must be a way to associate labels such as "Secret" and "Top Secret" with files. An efficient way to implement such a label is with a set of bits, instead of a character string like "Top Secret". The Policy Label is a set of bits that becomes an immutable[2] property of each subject and object. By comparing the label of the subject with the label of the object, the system can easily determine whether the desired access should be allowed or not.

To create a flexible policy environment, the Policy Label is viewed as a container for several labels. In this initial design, the Policy Label will contain a secrecy label component and integrity label component, leaving some amount of the Policy Label unused. To add another enforced policy, the first step is to reserve some piece of the unused portion of the Policy Label for its use. Note that the label is not called an "Access Class" or "Access Label" due to the fact that the implementation is so generic that the label could be used to enforce a policy that is not related to access control, e.g., an Audit Policy.

[2] Immutability is certainly the desired property for labels, but no extra effort is expended in this design, above and beyond what Linux already provides, to guarantee that a policy label cannot be modified.

2.1.2 Human-Readable Label (HRL) Databases

There must be a way to map the human-readable labels, such as "Secret" to the binary format of a Policy Label, and vice versa. The Human-Readable Label Databases help to satisfy this requirement. In order to support the addition of policies to the system, there is a separate human-readable label database for each enforced policy. Because the initial implementation will support the Bell and LaPadula secrecy policy and the Biba integrity policy, two Human-Readable Label Databases are required.

2.1.3 User Clearance Database

To support a mandatory policy, there must be a way of associating a clearance with each user, i.e., what the user is limited to read and write. These settings need to be stored in a database separate from the usual Unix user attribute file (/etc/passwd) for compatibility purposes. This database is known as the Clearance Database. Once again, in order to provide for flexibility, there is a separate database for each enforced policy. This database must hold the following information for each authorized user of the system: minimum session level; clearance; default session level. The following dominance relation must be true:

minimum ≤ default ≤ clearance.

2.1.4 Range Database

To be able to bound the level of the data being produced and the level of the subjects being executed (despite what the user clearances are), there exists a database known as the "Range Database." This database allows a System Administrator to constrain the range at which a system will operate. This range can change over the lifetime of a system so that there can exist objects on a system that are outside of the currently set range. There exists one database for each enforced policy, storing two pieces of information: the system high label for the associated policy and the system low label for the associated policy. The following dominance relation must be true:

System low ≤ System high

2.2 New Modules

This section describes the new modules to be added to Linux to support flexible policies. In this context, the term "module" refers to an active part of the system that manages a particular database or flow of control. The design of the modules has been layered in such a way that higher layer modules are dependent on the lower layers in a loop-free construct; modules in a particular layer do not call modules in the same or higher layer.

2.2.1 Policy Modules

For each enforced policy there must exist a module which enforces the policy. These modules provide an interface, as described below, with respect to the individual policy:

- Determine whether one label dominates another.
- Determine whether a read or write access should be allowed based on the subject and object labels involved.
- Change or query attributes of a label.
- Publish properties of the policy that are needed by other modules, e.g., the number of secrecy levels supported by the secrecy policy implementation.

2.2.2 Meta-Policy Manager

The Meta-Policy Manager is a replaceable module that is responsible for calling the individual policy modules and returning the net result of the query. It is called the "Meta-Policy Manager" because it implements a policy on policies, deciding which policy

modules are called first and whether some combination of results can result in an approved or declined access.

2.2.3 Label Modules

For each supported policy there exists a module that manages its associated Label Database. It provides an interface as described below:

- Map a human-readable label for the policy to a binary label for the policy.
- Map a binary label for the policy to a human-readable label for the policy.

2.2.4 Label Manager

The Label Manager is a replaceable module that defines the Policy Label, and is responsible for calling the individual Label modules to provide an interface as described below:

- Map a human-readable label for the system to a binary Policy Label for the system.
- Map a binary Policy Label for the system to a human-readable label for the system.
- Extract binary policy data from a Policy Label for any of the policies represented in the label.
- Set binary policy data in a Policy Label for any of the enforced policies.

2.2.5 Range Modules

For each enforced policy there must exist a module which manages its associated range database. Each range module provides an interface to the following:

- Return the system low label for the associated policy.
- Return the system high label for the associated policy.

2.2.6 Range Manager

The Range Manager is a replaceable module that is responsible for calling the individual Range Modules to provide an interface for doing the following:

- Return the combined system low label for all the enforced policies.
- Return the combined system high label for all the enforced policies.

2.2.7 Clearance Modules

For each enforced policy there exists a module which manages the associated User Clearance Database. It provides an interface to do the following:

- Return the maximum clearance for a given user ID.
- Return the minimum clearance for a given user ID.
- Return the default session level for a given user ID.

2.2.8 Clearance Manager

The Clearance Manager is a replaceable module that is responsible for calling the individual Clearance Modules to provide an interface to do the following:

- Return the maximum session level (clearance) allowed for a given user, in the form of a complete Policy Label.
- Return the minimum session level for a given user, in the form of a complete Policy Label.
- Return the default session level for a given user, in the form of a complete Policy Label.

3 Linux Modifications

This section is divided into two major parts to describe the required Linux modifications: Operating System Modifications and Application Modifications.

3.1 Operating System Modifications

3.1.1 Basic Policy Enforcement

At the core of every access control policy is a description of how subjects can read or write objects. Therefore, any system call that checks the kind of access given to a subject must be modified to call the Meta-Policy Manager to perform the additional policy checks. The following system calls are affected:

- open
- opendir
- access

Other system calls are not affected, such as read() and write(), because the open() call determines the permission that is given to the opening subject when the file is opened.

3.1.2 Inode Changes

Linux is designed to simultaneously provide up to 15 different kinds of file systems. The native Linux file system is known as the Second Extended File System (EXT2) and has been available since 1993. It is this file system which has served as the starting point for the design of Policy Enhanced Linux, viz., the EXT2 file system has been modified to provide MAC.

Every object in the Linux file system has a unique structure associated with it, called an inode, which keeps track of various properties of the object, e.g., its owner, creation time and access rights, as well as the location on the disk where the object is being stored. The inode is the obvious place to store the Policy Label for file system objects. Therefore, all objects managed by Linux via inodes will be subjected to the new policies.

The non-MAC inode structure currently requires 128 bytes when compiled on an Intel CPU. The current structure of the inode has 8 bytes of reserved space. The initial implementation of the MAC policies will use these reserved bytes to store the Policy Label.

3.1.3 File Statistics

Many programs need to obtain information about file system objects. A prime example is the command shell when it needs to list the contents of the current directory. To get the necessary information, the shell makes one call to the operating system for each object in the current directory. The information returned for each object needs to include the Policy Label. This requires a change to the following data structure:

- stat

in addition to the following system calls:

- stat
- fstat
- lstat

3.1.4 Subject Changes

Now that labels are associated with file system objects, it is necessary to design the other half of the policy-enforcement mechanism: associating Policy Labels with subjects. If a subject passes the usual Linux DAC check, it must then pass a MAC check such that the Policy Label of the object is compared with the Policy Label of the subject.

Every subject in Linux has a data structure associated with it called task_struct. It provides the information needed (1) by the subject to run properly and (2) by the kernel to make DAC decisions. For example, it contains the User ID and Group ID of the subject, which are compared with the User ID and Group ID of the file system objects the process tries to access. This structure is the obvious location for associating Policy Labels with subjects.

It is necessary to give each subject two Policy Labels in order to support trusted subjects [7].

The two Policy Labels assigned to each subject are called the Read Label and the Write Label, which have the following relationship:

Write Label ≤ Read Label

The Read Label is the highest level that the subject can read, whereas the Write Label is the lowest level that the subject can write. For single-level (untrusted subjects) these two labels are equal.

3.1.5 Creating Objects

When a subject is running at a single level, objects created by the subject are assigned the same Policy Label as the subject. When a subject is a multilevel subject, the subject needs to communicate with the kernel to inform it of the Policy Label to assign to each new object. This can only be accomplished by modifying the creat() system call to accept the additional Policy Label parameter. This presents the following three requirements for creating new objects:

- Subject Read Class ≥ Requested Object Class
- Subject Write Class ≤ Requested Object Class
- Directory Class ≤ Requested Object Class

The first two restrictions simply mean that the multilevel subject must be able to both read and write at the requested access class of the new object. The latter restriction is given so the file hierarchy descends from the root to its leaves in non-decreasing levels. This restriction is necessary to avoid a "dangling object" that cannot be accessed by subjects at its level. This restriction is referred to as the "compatibility property" [7].

Files are only classified at a single level. Directories, as objects that contain names and locations of other objects, are also single-level objects, but the objects that they point to may be at a different level than the level of the directory. However, there are constraints placed on the creation of objects whose class is different from that of its directory:

- Subject Read Class ≥ Requested Object's Directory
- Subject Write Class ≤ Requested Object's Directory

These last two restrictions mean that the multilevel subject must be able to both read and write to the new object's directory. Therefore, the only way to create an object in Policy-Enhanced Linux with a different access class than that of the directory it is being created in, is by using a trusted subject, e.g., a multilevel process whose Read and Write class spans a range that includes the level of the directory and the level of the object being created. When an object is created with an access class that is higher than the directory it is being created in, it is referred to as an upgraded object.

Other designs have shown that it is possible for low-level subjects to create upgraded objects [8], but it would require a major redesign of how object statistics are stored and managed.

The following system calls have been identified as being affected by the changes described above:

- open
- creat
- mkdir
- symlink

3.1.6 Deleting Objects

Deleting an object requires the same privileges as Creating an object because it requires the modification of a directory. With respect to deleting objects, the following system calls are affected:

- unlink
- rmdir

3.1.7 Deflection Directories

Existing Linux applications expect to be able to read and write to a temporary directory, located at "/tmp" in the file system hierarchy. Because applications will be running at a number of levels simultaneously, and because these applications are mostly running as single-level subjects, this creates a problem, given the constraints described in the previous subsections. Fortunately, this is not a new problem.

When other multilevel Unix designs were faced with this problem of either modifying every existing application to write temporary files somewhere else, depending on the level they were running at, or coming up with an alternative, they all came up with an alternative. The best solution is something known as a Deflection Directory, first proposed by Kramer [9], though it is also known by other names.

A Deflection Directory is a directory that contains sub-directories for each level that is needed. Access to these directories is transparent to the applications. For example, if a subject at the SECRET level writes a temporary file to a deflection directory, say "/tmp", the kernel will first create a SECRET sub-directory, transparent to the user, before creating the file in the SECRET sub-directory. From the user's point of view, the file was created in "/tmp" because the underlying system deflects every reference to "/tmp" to the "/tmp/SECRET" sub-directory for SECRET subjects. This allows existing applications to work without modification, no matter what level they are running. In order to prevent a covert channel, deflection directories can only be created and deleted by a System Administrator.

In order to simplify the semantics of the deflection directory, only the System Administrator is exempt from the deflection. The administrator sees the true directory structure and can therefore list and access any file within a deflection hierarchy. This has the negative side effect of not allowing higher-level subjects to read the lower-level objects in these directories. This is a trade-off between the complexity involved with determining when a subject wants to walk the deflected path, and when a subject wants to explicitly walk down a different path in the deflected hierarchy.

Over time, deflection directories can potentially grow quite large as new transparent directories are created. This leads to the design choice to delete all transparent directories under "/tmp" during system initialization.

The following system calls have been identified as being affected by the changes described above:

- mkdir
- chdir
- fchdir
- chroot
- open
- opendir
- stat
- lstat
- fstat

3.1.8 Updating Object Properties

Mandatory Access Control policies, such as the Bell and LaPadula policy, do not allow a high level subject to modify a file at a lower level; if this were allowed, a huge security hole would be created. A less obvious observation is that any change in an object's properties, such as the time of last access, creates a covert channel. Therefore, in addition to the usual MAC constraints, all object properties can only be changed by a subject at the same level of the object. This includes the following properties:

- Name
- Owner
- Group
- Size
- Last access

The following system calls must be modified in order to selectively update object properties:

- rename
- truncate
- ftruncate
- chmod
- fchmod
- chown
- fchown
- utime
- utimes

3.1.9 The Super User

Unix systems have a user known as the Super User, associated with any user with a User ID value of 0, typically only given to a user with the name of "root." This user bypasses all security checks on the system. One becomes the super user in one of two ways: 1) logging in as the root user; 2) logging in as a regular user, executing the "su" (super user) command and entering the password for the root user.

This ability of the super user to bypass security checks will continue to be supported in Policy-Enhanced Linux by including an exemption on the additionally enforced policies.

Unix also supports something known as "setuid" and "setgid" programs, where executable files can be configured to run as the owner or group of the executing file, respectively, instead of running as the user executing the file. This feature is typically used to allow a program to execute with root privileges, even when executed by a non-root user. Despite the fact that these features are recognized as a security weakness in Unix, the current version of Policy-Enhanced Linux will not change how setuid and setgid programs work.

3.2 Application Modifications

3.2.1 Login Program

The program called "login" presents the login interface to the user, checks a user's password, and starts up the user environment if the password is correct. This program must be modified to also prompt the user for the desired session level. If no level is given, then the user's default session level is used. This session level must pass the following tests before the user environment is set up:

- User's Minimum Session Level = Session Level
- System low = Session Level = System High
- Session Level = User's Clearance

The login program must also be modified to set the level of the user processes to the approved session level, which requires the addition of a new system call: setlevel.

3.2.2 Object Statistics (ls, stat)

There are two user-level programs that can be used to display statistics about file system objects: ls and stat. Both programs need to be modified to display the human-readable Policy Label when requested. The ls command is sometimes implemented as an internal shell command, and would therefore require the modification of such a shell.

The ls command must also be modified to indicate when a directory is a deflection directory when a full listing of a directory is made, i.e., when "ls –l" is used. The 'd' that normally indicates that the entry is a directory must be replaced with a 'D' if the directory is a deflection directory.

3.2.3 Process Status (ps)

The ps command must be modified to have a new command-line option to allow the displaying of process Policy Labels.

3.2.4 Process Identification (id)

The id command must be modified to display the session level, in addition to the group ID and user ID that are currently displayed with this command.

3.2.5 Directory Creation (mkdir)

A new option (-M) must be added to the supported command-line options of the mkdir command. This new option will create the specified directory as a deflection directory. A deflection directory can only be created by a System Administrator, i.e., the root user.

4 Conclusions and Future Work

Detailed specifications have been produced for all the new databases and modules presented in this paper, and all new modules have been implemented. A small amount of Linux code was modified to support the addition of policy labels to subjects and objects, as well as the comparing of labels to support the enforcement of the Bell and LaPadula secrecy policy. A demonstration of this new capability was produced, showing that a low-level subject could not read a high-level object, even when the DAC permissions allowed it.

The additional Linux changes can now be implemented with confidence that the system will work as designed. The finished project can then be distributed to other universities to enhance their computer security curriculum, or to other interested parties.

Additional research is now being performed to design and implement a Trusted Path, as well as a robust auditing mechanism.

5 References

1. Gasser, M., *Building a Secure Computer System*. New York: Van Nostrand Reinhold, 1988.
2. Pfleeger, C. P., *Security in Computing*. Second edition, Upper Saddle River: Prentice Hall, Inc., 1997.
3. Department of Defense, *Trusted Computer System Evaluation Criteria*, DOD 5200.28-STD, December 1985.
4. Higgins, J. C., "Information Security as a Topic in Undergraduate Education of Computer Scientists," *Proceedings of the 12th National Computer Security Conference*, NIST / NCSC, pp. 553-557, October 1989.
5. Irvine, C., Warren, D., Clark, P., "The NPS CISR Graduate Program in INFOSEC: Six Years of Experience", *National Information Systems Security Conference, NIST / NCSC*, Volume 1, pp. 22-29, October 1997.
6. Clark, P., "*A Linux-Based Approach to Low-Cost Support of Access Control Policies*", Naval Postgraduate School, September 1999.
7. Bell, D.E., LaPadula, L.J., Secure *Computer System: Unified Exposition and Multics Interpretation*, ESD, Air Force Systems Command, United States Air Force, Hanscom Air Force Base, Report MTR-2997 Rev. 1, March, 1976.
8. Irvine, C. E., "A Multilevel File System for High Assurance", *Proceedings of the 1995 IEEE Symposium on Security and Privacy*, pp. 78-87, 1995.
9. Kramer, S., "Linus IV – An Experiment in Computer Security", *Proceedings of the 1984 Symposium on Security and Privacy*, Institute of IEEE, pp. 24-31, 1984.