NPS-CS-06-001
August 2006

Trustworthy Commodity Computation and Communication

SecureCore Technical Report

An Analysis of Three Kernel-based Multilevel Security Architectures

Timothy E. Levin, Cynthia E. Irvine, Thuy D. Nguyen

This material is based upon work supported by the National Science Foundation under Grant No. CNS-0430566 and CNS-0430598 with support from DARPA ATO. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or of DARPA ATO.

Author Affiliations

Naval Postgraduate School:

Center for Information Systems Security Studies and Research
Computer Science Department
Naval Postgraduate School
Monterey, California 93943

An Analysis of Three Kernel-based Multilevel Security Architectures[1]

Timothy E. Levin, Cynthia E. Irvine, and Thuy D. Nguyen
Naval Postgraduate School

Abstract: *Various system architectures have been proposed for high assurance enforcement of multilevel security (MLS). This paper provides an analysis of the relative merits of three architectural types – one based on a traditional separation kernel, another based on a security kernel, and a third based on a least-privilege separation kernel. We show that by taking advantage of commonly available hardware features, and incorporating security features required by the nascent "Separation Kernel Protection Profile" (SKPP), the latter architecture can provide several aspects of security and assurance that are not achievable with the other two.*

After years of experimentation with alternate techniques, it is apparent that the existence of different levels of data sensitivity in automated environments with heterogeneously-trusted users still calls for multilevel-secure IT systems. Various system architectures have been proposed for high assurance enforcement of multilevel security, which would be suitable for protection of highly valued resources. Many of these systems utilize common kernel constructs such as the *separation kernel* and the *security kernel*. However, we lack metrics or even a common framework for understanding which architecture might be best to use in a given situation. The purpose of this paper is to (1) compare and contrast several idealized system architectures that are based on these kernels, and (2) provide a basis for future work in the construction of architectural security metrics. In particular we examine the "multiple independent levels of security" (*MILS)* architecture [Alves-Foss04, Vanfleet05] that is based on a generic separation kernel, a classic *"Evaluated Policy"* architecture based on a security kernel (e.g., XTS-400 system [DigitalNet04], and GEMSOS kernel [Thompson87]), and a new architecture [Irvine05] based on a kernel that is compliant with the security features defined in the Separation Kernel Protection Profile (SKPP) [SKPP04]: the *Least-Privilege* architecture.

We describe four basic functions of a usable MLS system:

- Establishment and isolation of *policy equivalence classes* – providing a set theoretic commonality underlying the three architectures, i.e., they provide, separate, and monitor flows between policy equivalence classes.
- MLS policy interpretation and enforcement – which includes allowing high to read from low

[1] This material is based upon work supported by the National Science Foundation under Grant No. CNS-0430566 and CNS-0430598 with support from DARPA ATO. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or of DARPA ATO,

- Mediation of a relaxed MLS policy – whereby high information can be securely transitioned to low domains, when appropriate
- Ability to enforce the principle of least privilege on the activity of subjects – to support accountability and confinement of damage

A fundamental distinction of the three security architectures is how these four functions are allocated to system components. We show that the allocation strategies result in several significant differences: the granularity with which the principle of least privilege can be enforced, factoring and reusability of components, dynamicity of runtime resource management, scalability of label space, dynamicity of security policies, assurance (i.e., confidence) of trustworthiness that is achievable during product evaluation (versus the accreditation phase of product deployment), and the trustworthiness and granularity of both controlled interference and intransitive information flow.

In the study of high assurance computer security, nomenclature has varied over time, so we will present our understanding of the underlying concepts and terminology in the Background section. In Section 2 we present a framework for comparative analysis of the security architectures to be examined. In Section 3 we compare the selected architectures. In Section 4 we discuss how functionality regarding information flow is managed in these architectures, and we conclude in Section 5.

1 Background

The basic characteristics of different kernels and the three architectures are described in this section. More detailed differences are described in the following sections. But first, several underlying security policy concepts are described.

1.1 Security Policies

In this paper, we will regard *Mandatory access control* (MAC) policies as those that regulate users' access to information objects based on the sensitivity level of the object and the authorization ("session") level of the individual user. The policies are mandatory because their enforcement is intended to be both global in scope and persistent in time, such that individual users cannot override the policy during normal use. Often, this amounts to ensuring that the user and her surrogate subjects are constrained to create or interact with objects that are at the same sensitivity level as the user's session.

The primary security policy of interest, historically, is that regarding the handling of sensitive documents with hierarchical confidentiality labels such as TOP SECRET (TS), SECRET (S), and UNCLASSIFIED (U). However, MAC policies are also used for managing information based on integrity labels, as well as non-hierarchical labels such as would identify an area of interest which must be kept separate within a hierarchical level, like a defense capability program (e.g., TS *CRYPTO*), or finance processing. Research has shown that MAC policies have a label space that forms a lattice with respect to the reads and writes (i.e., information flows) that are allowed [Denning76].

A secure computer system provides a translation of *organizational security policies* [Stern91] by way of automated mechanisms that control the system's resources, the *automated security policy*. The controlled resources include the security mechanisms' internal code and data, as well as the *exported resource* abstractions of the system (e.g., processor time, files, devices, subjects). This description assumes that inter-process communication mechanisms (e.g., pipes) can be sufficiently modeled, for the purpose of security analysis, as objects that subjects read from and write to. *Subjects* are the active entities of a system (e.g., processes, or programs in execution) and may act as surrogates for users (e.g., administrators), or may be independent system "daemons." Note that "exported resources," as used here, subsumes both subjects and objects. To implement an MAC policy, each exported resource is associated, explicitly or implicitly, with a sensitivity level. The levels *partition* the system's exported

resources into *equivalence classes* with respect to the MAC security policy. The equivalence classes of a partition are also called *blocks*, in mathematical terminology, and the two terms are used synonymously, here. Note that a block is not a subject, as it are not an active entity, but rather represents a set of resources. The policy enforcement mechanisms then ensure that the only accesses allowed in the system are those in which the interactions between equivalence classes are consistent with the MAC policy.

Here, *Multilevel security* (MLS) policies are those MAC policies in which subjects may freely access objects in parts of the lattice that: (1) can be different than that of their related user session level, and (2) are allowed by normal "paper world" information handling policies. In other words, subjects are **not** constrained to interact with objects at their own level. They may read "down" and write "up" with respect to the confidentiality lattice, as well as read objects of greater integrity and write to objects of lesser integrity. These operations are consistent with the Bell and LaPadula [] and Biba [] models. Note that in this document, "read" (as in "subject reads object") indicates a simple observation whereby information flows from the object to the subject; whereas "write" indicates modification, whereby information flows from the subject to the object. This semantics for write is sometimes called a "blind write," e.g., append, but in any case, it does not imply read. Mechanical problems of implementing "blind writes" are left to the system engineers.

1.2 Trusted subjects

History has shown that MLS systems must include various functions for two-way interaction between levels, which implies the presence of subjects whose labels logically span multiple nodes of the lattice [Schell85, Denning87, Landauer89, Schellhorn95]. These are called *trusted subjects* because they may be depended on to help enforce the usual policy, or at least to not violate a well-defined extension to the usual security policy (here, the MLS policy). Given an MLS system and a partial ordering of blocks based on the "normal" flows allowed between them, a trusted subject is one that is allowed to perform one or more inter-block read or write operations (viz., to or from objects in another equivalence class) that contradict that partial ordering.

We introduce the concept of the *relaxed MLS policy* to describe the overall policy that results from the introduction of trusted subjects into a system that enforces the "normal" or *strict MLS policy*. Trusted subjects are relied upon to conform to the relaxed MLS policies, if not help enforce the strict MLS policy. For example, consider a down-grader or guard, which is a TS trusted subject that can read and write data at both TS and S; specifically, it can "write down" from TS to S. This program needs to be carefully engineered so that only the intended information is written down, and it is written only to the intended objects.

1.3 The Principle of Least Privilege

In their 1975 review, Saltzer and Schroeder [Saltzer75] identified least privilege as the restriction that "every program and every user of the system should operate using the least set of privileges necessary to complete the job. Primarily, this principle limits the damage that can result from an accident or error. It also reduces the number of potential interactions among privileged programs to the minimum for correct operation, so that unintentional, unwanted, or improper uses of privilege are less likely to occur. Least privilege mechanisms supported by operating systems include access control lists or capabilities relative to individual users and objects, and *privilege rings* (i.e., hardware-enforced hierarchical privilege domains).

The principle of least privilege provides a clear foundation for understanding why a system should not be configured to allow unfettered access to resources, such as results from the use of "super-user" or "root" programs in a Unix-like system[Sibert87]. To arrive at a securely deployed system, it must be capable of supporting least privilege, and it must have been administratively configured such that the

programs that might execute will be accorded "the least set of privileges necessary to complete the job." If a system does not allow individual users and programs to be so configured, the accountability mechanisms (e.g., audit) will be less able to accurately discern the source of various actions, e.g., individual modifications within a file. Thus, the ability of a secure system to realize the goals of accountability, as well as the confinement of damage, is limited by the level of granularity with which the system is able to invoke the principle of least privilege[Levin06]. The ideal from a security perspective is that a system should be able to apply least privilege at the same granularity as the resources that it exports (e.g., individual files).

1.4 Kernels

A kernel is a software program that manages hardware computing and communication resources. From these resources, it creates abstractions (e.g., subjects/processes and memory objects) that it makes available at (exports to) its interface. The desired security characteristics of a *high assurance* kernel are those of a "reference monitor" [Anderson72]: *always invoked*, *tamperproof*, and "*small* enough to be subjected to analysis and tests, the completeness of which can be assured."

If the kernel is always invoked, scurrilous processes cannot bypass its protection mechanisms to access resources. If the kernel is tamperproof, the protection mechanisms cannot be modified to perform falsely when they are invoked. Complete and consistent mediation of access results. In order to be "completely analyzable," a corollary to smallness is that the design must also be *simple*, as well as *minimized* to contain only the necessary functionality [TCSEC].

1.4.1 Security Kernels

A security kernel binds MAC sensitivity labels to all exported resources, thus assigning them to equivalence classes. A kernel-resident reference validation mechanism mediates access by subjects to objects according to an MLS policy [Ames83], based on those labels. The MAC label space may include confidentiality and integrity labels as well as non-hierarchical categories [Lunt89]. A security kernel usually provides a hardware-supported ring abstraction, and can host trusted subjects. Some security kernels can limit the scope of trusted subjects within the lattice by assigning, for example, a maximum level for reading and a minimum level for writing, with respect to confidentiality [Schell85].

The kernel mediates external communication, usually via network devices that are each dedicated to a given sensitivity level, or via *multilevel devices*, in which a sensitivity label is bound to each network protocol entity (e.g., packet), based on its sensitivity.

Security kernels generally support full resource and resource-allocation configurability during runtime. This dynamicity presents a difficulty with respect to covert channels. [Lampson72, Millen87]

1.4.2 Separation Kernels

A generic separation kernel [Rushby81] *partitions* exported resources into *equivalence classes* (also known as "blocks," in the mathematical literature, and "partitions" in the MILS literature) as defined in its configuration data. The separation kernel enforces separation of equivalence classes, and allows interactions within and between the equivalence classes based on further configuration data. The allowed interactions can be modeled as a "partition flow matrix" with equivalence class IDs as the labels of the columns and rows, and *read* and/or *write* as table entries. Typically, a separation kernel does not interpret the semantics of MAC labels, although they may appear in configuration data to indicate intended use of blocks in the application domain. Nevertheless, a separation kernel may be configured to allow flows between blocks in a manner that is consistent with an MLS policy

A common strategy in the design of separation kernels is to minimize the potential covert channels between blocks by simplifying the runtime functionality of the kernel. Sensitivity labels for subjects

and objects are not considered – although equivalence-class designations could be an abstract form of a label – and runtime reconfiguration of resources is minimized. Taken to its logical conclusion, the goal of a simplified functionality results in a completely static exported-resource environment (e.g., the runtime creation of exported resources is not allowed, and the allocation of exported resources to blocks and subjects is static); the alternative is greater complexity and correspondingly less assurance that covert channels have been avoided or completely documented. Different separation kernels will have different degrees of runtime configurability.

1.4.3 Least Privilege Separation Kernels

A Least Privilege Separation Kernel (LPSK), in addition to the characteristics of a generic separation kernel, also supports the full feature set of the NSA Separation Kernel Protection Profile (SKPP), which is a description of functional and assurance requirements that conform to level EAL6+ defined by the Common Criteria[CCIMB05]. For our purposes, the most significant additional characteristic of a LPSK is its ability to restrict information flow at the level of granularity of the resources that it exports. This can be modeled as a "subject-resource" flow matrix, which restricts flows in conjunction with the partition flow matrix described above.

An LPSK may mediate network communication like the security kernel, via single-level and multilevel devices, although that functionality is not required in the SKPP. The LPSK kernel [Irvine04] is an "open source" reference implementation of an LPSK, which is designed with a static exported resource environment and hardware supported rings. This kernel will be used as an example in some of the figures to follow.

1.5 System Architectures

Each of these kernels form the basis for a system *architecture* which comprises a kernel and various application programs that execute on the kernel. This level of architecture is our primary focus. Additionally, each architecture can support a network of similar systems, each of which shares a common sensitivity-label lattice. We have not considered heterogeneous-system networks. System components outside of the kernel may translate the kernel-exported resources into new abstract data types at the system interface, but the kernel's policy restrictions will still hold over those resources. All three architectures provide the capability for a trusted subject to span security levels. Figure 1 provides a legend for the figures that follow.

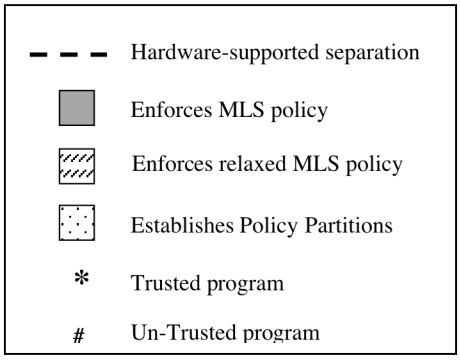


Figure 1. Legend for Figures

Architectural layers, processes, and functions that are separated with the support of hardware mechanisms are indicated with the dashed line between them. For example, horizontal dashed lines are provided by processor's privileged execution mechanism and ring mechanisms; verticle lines can indicate processor "task state" mechanisms. Each of the other features called out in Figure 1 is described

below, but it is worth noting at this point that a given architectural component can be allocated more than one feature. For example, as seen in Figure 2, the TS-U component provides MLS policy enforcement (gray), relaxed MLS policy enforcement (hashed lines), and is a trusted program.

1.5.1 MILS Architecture

As a term of art, MILS is based on several previous approaches to the problems of managing labeled data in an IT environment, such as the separation kernel and the use of trusted subjects to manage inter-level communication. Because MILS itself is evolving, we summarize our current understanding of it as follows.

The kernel of the MILS architecture is a generic separation kernel. MAC sensitivity labels are assigned to each block by system-level applications or system (vice kernel) configuration data. The label space consists of hierarchical confidentiality labels. At least one of the blocks contains a trusted subject that is allowed to read and write data in all of the other blocks. The trusted subject provides the only means by which other subjects can communicate between blocks in the local or a remote node, and it mediates these communications per the policy defined in its configuration data.

The MILS architecture, with confidentiality labels (TS, S, U, TS-U) for each block, is shown in Figure 2. Each blocks usually has a single process, although a process may contain multiple programs, such as in the TS-U block.

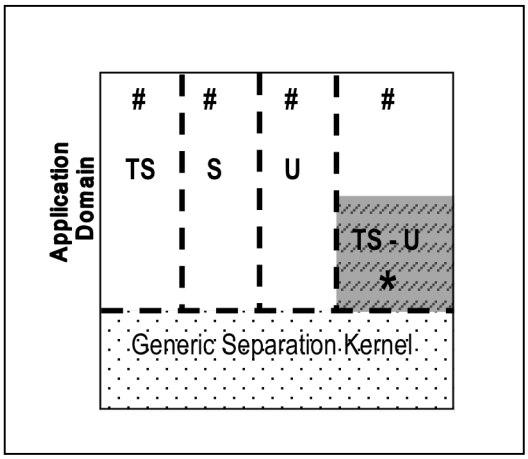


Figure 2. MILS Architecture

1.5.2 Evaluated Policy Architecture

The kernel of the *Evaluated Policy* architecture is a security kernel. There may be many processes in each equivalence class (one process per equivalence class is shown in Figure 3). Kernel-supported rings are used to separate privileged applications (resource management, etc.) from less privileged, within a process. A *subject* is defined by a process-ring pair, so the application in the S equivalence class is a different subject from the application in the S service domain.

Communication with remote systems can be managed with a single-level device on either end, or with a multi-level device on either end. Either way, the kernel mediates access to the networked data at both ends, according to the MLS policy.

The name Evaluated Policy architecture derives from the fact that the flow policy enforced by the system is determined at the time that the kernel is compiled, rather than at system runtime. Thus, the

enforced security policy is determined as part of product evaluation, rather than as part of system accreditation.

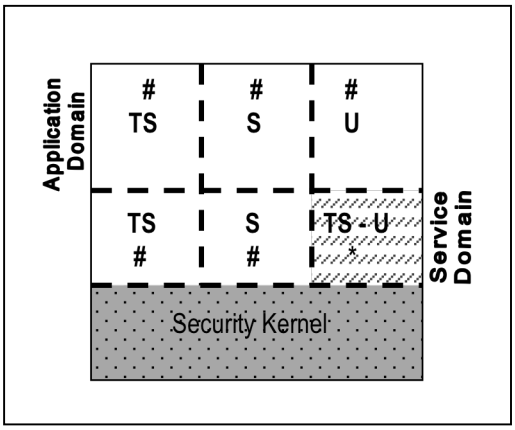


Figure 3. Evaluated Policy Architecture

1.5.3 Least Privilege Architecture

The kernel of the Least Privilege architecture[Irvine05] is an LPSK. Like the Evaluated Policy architecture, kernel-supported rings are used to separate privileged applications, and also to define subjects of a process. There may be many processes within an equivalence class. Unlike the MILS architecture, the Least Privilege architecture does not require a trusted subject to perform its basic functions, but can host one in a multi-level block, if it is needed. Network communication to other systems is handled just as the Evaluated Policy architecture.

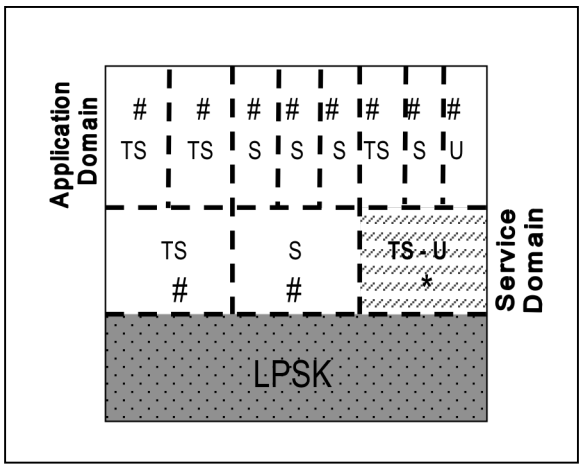


Figure 4. Least Privilege Architecture

1.5.4 Supporting Policies

In all of these system architectures support the assurance of both policy enforcement as provided by the reference monitor, and the limitation to damage defined by the principle of least privilege, mechanisms must be provided by the system to authenticate users, and to create an unforgeable binding between the users and the subjects or processes that execute on their behalf. Such identification and authentication (or, “I and A”) mechanisms, as well as those for audit, are outside the scope of this report.

2 Comparison and Metrics

Due to their use of minimized kernels that lack sophisticated operating system services such as file systems, all three of the architectures require either a customized operating system or a translation layer between their respective kernel and an unmodified OS, to support complex, commercial-grade applications. Other resource management services (e.g., RDBMSs and object request brokers) can be provided as applications in these architectures, as well. However, in general these hosted services do not provide multilevel security functions, so they are not significant to an understanding of the multilevel security issues of the architectures.

The following factors form the basis for our comparison of system security architectures:

- Allocation of Security Functionality
- Ability to Enforce the Principle of Least Privilege
- Factoring and Reusability
- Runtime Resource management
- Scalability of Label Space
- Dynamic Security Policies
- Trustworthiness and Evaluatability
- Controlled Interference
- Intransitive Information Flow

As shown below, many architectural properties are inherited from their respective kernels.

3 Contrast and Comparison of Architectures

In this section, the identified metrics will be applied in an analysis of the three kernel-based architectures.

3.1 Allocation of Security Functionality

Regardless of the particular architecture, several basic security functions are required to support high assurance enforcement of MLS policies: (1) establishment and isolation of policy equivalence classes, (2) MLS policy enforcement, (3) mediation of a relaxed MLS policy, and (4) the ability to enforce the principle of least privilege. In this regard, the primary differences between the architectures in question are that these functions are allocated differently:

- For the MILS architecture, the separation kernel provides function #1. Functions #2, #3 and #4 are packaged as trusted subjects. Note that this approach to function 2 is an architectural decision, not a limitation of the basic separation kernel.
- For the Least Privilege and Evaluated Policy architectures, the security kernel provides #1, #2 and #4; whereas #3 is implemented in a trusted subject that is constrained somewhat differently by the different types of kernel.

Only the Least Privilege architecture provides enforcement for the principle of least privilege at the granularity of the resources that it exports.

3.1.1 Establishment and Isolation of policy equivalence classes

In both separation-kernel-based architectures, the partition of exported resources, and resulting equivalence classes, are defined by kernel configuration data and maintained by the kernel. In an Evaluated Policy architecture, kernel-maintained sensitivity-level markings of exported resources establish the equivalence classes. The resource markings are bound to the resources as they are created.

In a separation kernel, the kernel enforces separation of, and restricts runtime interactions between, equivalence classes. The configuration data defines the allowed interactions. In a security kernel, process isolation mechanisms together with kernel-resident access rules ensure the separation of equivalence classes during runtime. Note that process isolation within an equivalence class is not an architectural requirement for separation kernels, although hardware task management functions can be used to implement and separate blocks.

Thus, all three architectures achieve basic partitioning and isolation in the kernel.

3.1.2 MLS Policy Interpretation and Enforcement

In a separation kernel, neither the configuration data nor the kernel have any "knowledge" of sensitivity levels or their manifested labels. In the MILS and Least Privilege architectures, system-level structures or mechanisms must embody any intended MAC semantics, i.e., provide a mapping of security levels to kernel-defined equivalence classes that is semantically consistent with the interactions allowed by the configuration data, in order for an MAC policy to be correctly enforced at the system interface. So, the MLS policy is interpreted outside of the kernel.

With respect to enforcement, for the MILS architecture, at least one of the blocks contains a trusted subject that enforces the MAC policy regarding inter-block flows. The kernel is configured to ensure that all inter-block interactions are channeled through the trusted subject. Thus, subjects read and write information in other blocks by making requests to the trusted subject. This trusted subject can be configured to enforce an MLS policy, i.e., allowing read down or write up.

In the Least Privilege architecture, the kernel enforces MLS policy semantics, based on the kernel configuration data and the system-level MLS policy mapping.

In the Evaluated Policy architecture, kernel-resident access rules define the MLS policy semantics, such that the security kernel completely defines and controls the MLS policy.

In all three architectures, the configuration data that defines sensitivity labels of devices must be correct with respect to the intended system policy. Also, correct MLS policy enforcement depends on assigning the "right" label to other exported resources. For MILS and Least Privilege architectures, which have kernels with static resource allocation, this means that the kernel configuration data must be correct. Additionally, for the MILS architecture, the trusted subject must assign the right labels to each resource (e.g., partition connection) it controls. For Evaluated Policy architectures, which dynamically allocate resources, the implication is that the system-level subjects that create resources must do so correctly, to ensure that each resource is associated with the correct labels.

3.2 Ability to Enforce the Principle of Least Privilege

This section discusses how privilege is minimized in all three architectures. It also discusses trusted subjects and how they may be constrained, i.e., how the kernel may or may not share in establishing the trustworthiness of trusted subjects.

In addition to assigning no more than the necessary and sufficient security level to each subject (Evaluated Policy architecture), and configuring blocks with the least amount of access to resources

(separation-kernel-based architectures), there are several "least privilege" mechanisms for constraining subjects from unfettered access to resources.

In a MILS architecture, control of access to resources is at the granularity of the equivalence class (block). However, the principle of least privilege can be applied by subdividing a given system-level MLS equivalence class into privilege-specific kernel blocks, each of which has the same sensitivity label, but some of which (e.g., the less privileged) may be allowed less access to resources than the label would allow. Thus, in the absence of hardware-supported privilege rings, it is possible to support privilege domains.

All three architectures can constrain trusted subjects within an access class range, with kernel assurance. For example, in a system that is configured with resources that range from U to TS, if a given trusted subject need only access U and S information, then it can be assigned to an equivalence class with a range of U-S, and be prevented by the kernel from accessing TS information.

Evaluated Policy and Least Privilege architectures, using kernel ring support, can constrain subjects using hardware privilege rings. This is done by segregating programs that have different privileges (e.g., even those with the same labels or within the same process) into different rings – creating a privilege-based partitioning of resources that is orthogonal to the MLS-label based partitioning. Then, for example, the "interference" of a trusted subject can be limited to occur only within its privilege domain. The MILS architecture does not support this.

Utilizing hardware ring support, the Least Privilege and Evaluated Policy architectures can securely place privileged (including "trusted") and unprivileged programs in the same process. The hardware provides both an effective separation mechanism for protecting the privileged programs, and a secure and efficient mechanism for transitioning control from one program to the other. The MILS architecture does not support this.

A LPSK's subject-resource access restrictions constrain subjects to a degree that is not possible in a generic separation kernel or a security kernel. This allows subjects in the Least Privilege architecture to be constrained, with kernel assurance, to access only specific objects in the system, thus supporting least privilege at the same level of granularity as the resources that the kernel exports (see Least Privilege Separation Kernels).

The MILS architecture does not require the utilization of hardware ring separation, yet it also allows heterogeneously privileged programs in a process or block, for example to support a non-privileged object management function as well as a privileged relaxed MLS policy function (see Figure 2). Since there is no hardware support for privilege separation of these programs, they may be mutually vulnerable; so, either the unprivileged application must be evaluated to the same degree as the privileged program, or it should be moved to a different block where the separation kernel can isolate it.

3.3 Factoring and Reusability

Efficient factoring of system functionalities into commonly used components, e.g., which may be transposed to a lower system layer, is a fundamental computer design tool that can reduce redundancies in a given system and also increase component reusability across systems.

Regarding reuse, there seems to be a tension between endowing a low-level security component with enough functionality to be useful in many systems (i.e., the "principle of least common mechanism[Benzel05]) and restricting its functionality so that there would not be unused functions in some of those systems (e.g., a security kernel in a MILS architecture could have unused mediation functions in the kernel). A key indicator for judging whether over-restriction has occurred is if the systems that incorporate the minimized component all need to create the same set of additional

functions in support of the component or the system security policy – then, it would seem to be more useful for the additional functions to reside in the low-level component.

The separation kernel offers a "policy neutral" reference validation mechanism, which can be reused in a conceivably broader range of systems (i.e., greater reusability) than a security kernel, which is "hardwired" to enforce an MLS policy. The Least Privilege architecture offers the advantage of a policy neutral kernel, with the advantage of enforcing the MLS policy in the kernel. However, inasmuch as the purpose of the MILS architecture is MLS policy enforcement, there seems to be no reusability advantage to moving the MLS policy enforcement out of the kernel, as it is necessary to duplicate the MLS functionality in every MILS system, thus potentially negating the benefits of having a simpler kernel.

3.4 Runtime Resource management

Another fundamental difference between separation kernel and security kernel architectures is the runtime creation and allocation of resources. Since the separation kernel largely allocates resources only during initialization, it is simpler, and will require less effort to build, as well as to confirm its trustworthiness.

On the other hand, systems that use a separation kernel and require dynamic resource allocation will require applications to provide *virtual* dynamicity through management of the kernel's fixed exported resources, just as a kernel virtualizes its fixed hardware resources. Thus, dynamic configuration can be provided in any of the architectures. However, if trusted subjects in a separation kernel architecture need to be involved in creating dynamicity, greater complexity could result.

3.5 Scalable Label Space

The MILS and Least Privilege architectures, insofar as they are limited to a static resource allocation by their separation kernels, must provide a separate block for each supported MAC label upon initialization. This resulting pre-configuration of blocks would not be scalable to highly diverse security environments such as in the Intelligence Community or a diverse commercial context [Lipner82]. In contrast the Evaluated Policy architecture easily supports a large number of different levels.

3.6 Dynamic Security Policies

The Global Information Grid (GIG) vision [NSA06] includes the support for *dynamic policies*, i.e., where access control semantics may be changed, or over-ridden during extraordinary circumstances. The three architectures can address dynamic policy changes in different ways. In the MILS architecture, the policy enforcing application component could be made to be "hot swappable;" the Least Privilege architecture includes a runtime MAC policy change function at the kernel level (viz., interchangeable pre-configured policy "sets" are loaded into the kernel at boot time); for the Evaluated Policy architecture, policy-interpretation module must be replaced, and the kernel recompiled, in order to change policies (a "hot-swappable" policy module is conceivable, but has not been implemented to our knowledge).

It is conceivable that the GIG may ultimately require frequent or arbitrary changes to MAC policies. In this case the MILS architecture may be a more appropriate foundation than an Evaluated Policy or Least Privilege architecture. If the GIG is to only require periodic over-rides, or a small number of alternative MAC policies, then the Least Privilege architecture could be suitable, as well. By its nature, it is hard to see how the Evaluated Policy architecture could support dynamic security policies.

3.7 Trustworthiness and Evaluation

To ensure system trustworthiness, all of the mechanisms providing the basic security functions in a high assurance kernel-based system would need to be assessed, regardless of where they reside structurally – in the kernel or in trusted subjects. Additionally, evaluations of *high assurance* system by the US Government do not allow for the unexamined incorporation of pre-evaluated components [NSA05], which means that all of the components of a high assurance system must be evaluated together. So, "you pay now" with kernel analysis (e.g., where the MLS policy definition is located in the kernel, as in the Evaluated Policy architecture, and where the MLS policy enforcement is located in the kernel, as in the Evaluated Policy and Least privilege architectures) or "you pay later" with trusted subject (MILS) analysis: the evaluation effort would be roughly the same for the three architectures, regardless of how the functions are allocated.

On the other hand, it is clear that kernel-enforced constraints to the behavior of trusted subjects decrease the amount of faith (viz., trust) that the system must have in the correctness of trusted subjects, and commensurately increases the assurance of correct policy enforcement compared to a system without such constraints on trusted subjects. A quantification of the resulting differences in trustworthiness between the three architectures is outside of the scope of this document, but functional comparisons are provided in the Ability to Enforce the Principle of Least Privilege and Controlled Interference sections.

A key component of system security assurance is the elegance (by which we mean completeness, minimalism, and clarity) of the design. By definition, complexity is not *clear*, and it has long been recognized as the enemy of security [Corbato91]. The Evaluated Policy architecture provides an elegant mechanism for enforcing a strict MLS policy, as the evaluator need only examine the set of label and the set of resources that are to be labeled in order to understand the policy that is to be enforced (assuming that all of the supporting mechanisms are correct). In contrast, for each separate configuration of a MILS or Least Privilege system, an accreditor or other analyst must also closely examine the mapping of MAC labels to the rules for interactions between equivalence classes (as well as the MLS semantics of trusted subjects in the MILS architecture) to understand the strict MLS policy enforced by the deployed system. Depending on the design and how the MAC labels are managed, a MILS or Least Privilege architecture may present considerable more complexity than an Evaluated Policy architecture.

3.8 Controlled Interference

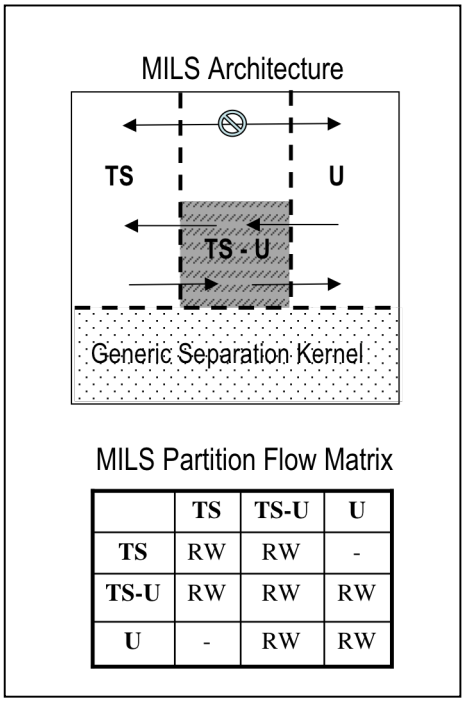
The secure management of information in an MLS system is sometimes modeled as "non-interference"[Rushby81, Goegen82], whereby processes at high confidentiality levels are prohibited from "interfering" with processes at low confidentiality levels (the prevention of low confidentiality subjects from directly reading high confidentiality information is implicitly assumed). As stated above, in most MLS systems there is a need to provide relaxed MLS policy functions, which are implemented with trusted subjects. In the literature, the relaxed MLS policy behavior of trusted subjects has been called "intransitive noninterference," [Rushby92] referring to a policy where, for example, a trusted subject may move data from TS to S, but "untrusted" TS and S subjects cannot directly do so. Here, we refer to the constrained behavior of a trusted subject as *controlled interference*, to avoid the double negative.

A difficulty in the development of trusted subject programs has been to model their correct behavior, as well as to implement useful constraints to guarantee correct behavior. A previous presentation has described the identification and control of trusted subjects [Levin04] in a formal model.

All three architectures can provide kernel-level assurance that only trusted subjects might define and enforce the relaxed MLS policy, within their label range (see Ability to Enforce the Principle of Least

Privilege). The MILS architecture manages relaxed MLS policies by routing all inter-block flows through the trusted subject in the partition flow rules as shown abstractly in Figure 5.

The flows allowed by the policy are shown with arrows and with *R* (read) and *W* (write) entries in the partition flow matrix. The flow semantics of the table are that (subjects in) a given block in the left hand column may perform the indicated flows with respect to (resources in) the blocks in the top row. Notice that the diagonal row, indicating flows allowed within each block are not illustrated with arrows. The upper bidirectional arrow is marked with a *diagonal bar inside of a circle*, indicating that those flows are not allowed.

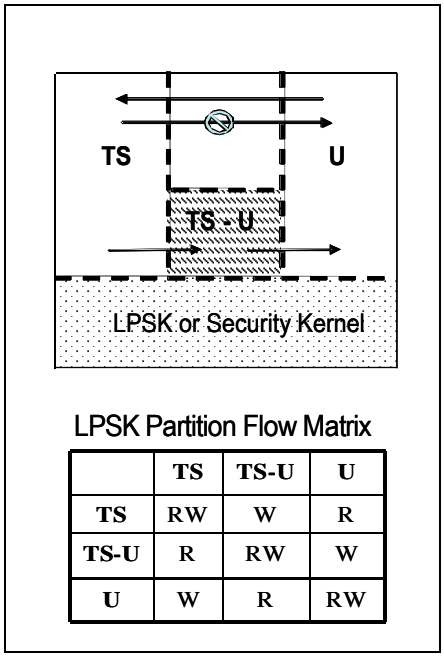


MILS Partition Flow Matrix

	TS	TS-U	U
TS	RW	RW	-
TS-U	RW	RW	RW
U	-	RW	RW

Figure 5. MILS use of Trusted Subjects for Controlled Interference

The Evaluated Policy architecture assigns the trusted subject a label range that includes TS, S and U, allowing the trusted subject to perform the necessary functions (e.g., downgrade), and otherwise allows TS to read S and U directly, as allowed by the MLS policy. The Least Privilege architecture provides similar kernel-enforced functionality through configuration of the partition flow rules. Both of these are shown in Figure 6.



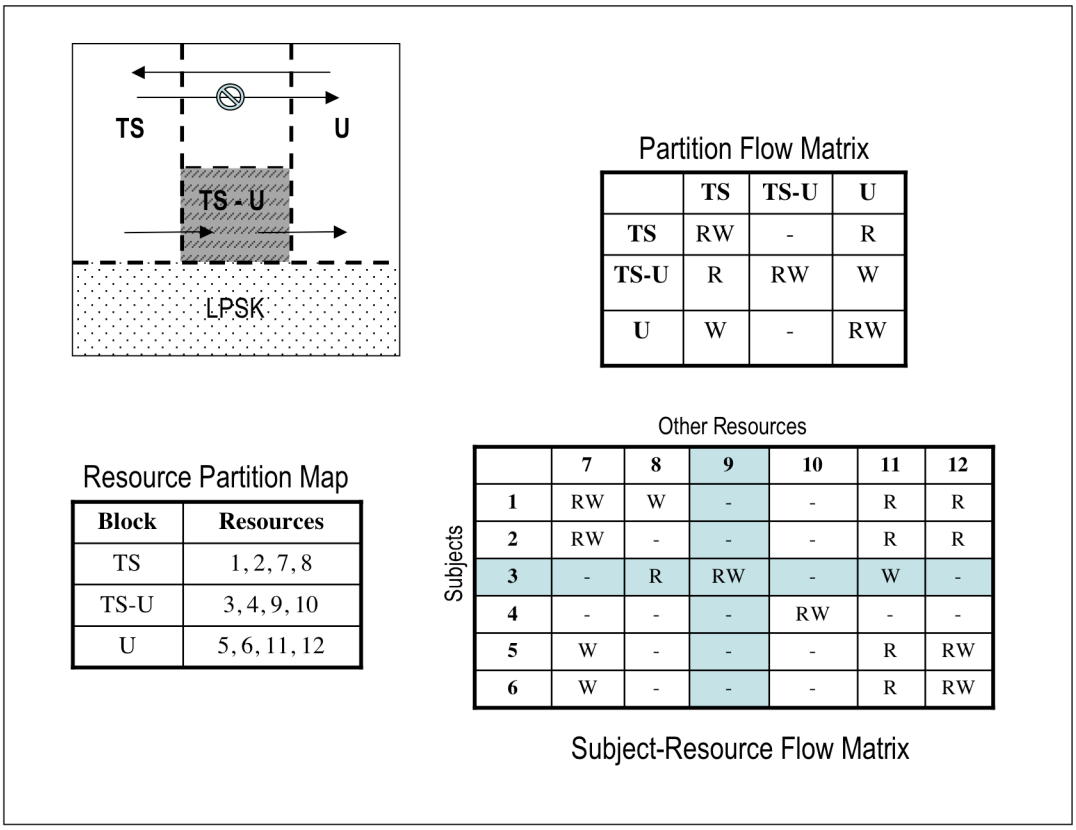
LPSK Partition Flow Matrix

	TS	TS-U	U
TS	RW	W	R
TS-U	R	RW	W
U	W	R	RW

Figure 6. Use of Trusted Subjects for Controlled Interference in Evaluated Policy and Least Privilege Architectures

However, these relaxed MLS policy mechanisms alone provide little capability for enforcing "least privilege" of trusted subjects beyond limitation of the label range, potentially allowing an over-reach of authority on the part of the trusted subject, and uncontrolled propagation of the effects of programming errors, developmental subversion, and runtime intrusions. To address this problem, the Least Privilege architecture applies kernel-enforced least privilege to a trusted subject, illustrated in the *subject-resource flow matrix* shown in Figure 7 (see also Section 1.4.3). The Resource Partition Map shows the resources identified with (viz., "bound" to) each block. Note that this binding does mean that a given subject is prohibited from accessing resources in a different block, which would require the appropriate flow is allowed by both the partition flow matrix and the subject-resource flow matrix. In the underlying security policy model [Levin04] the two matrices may disagree, which would only further restrict the allowed flows.

In Figure 7, the two matrices are configured to be consistent: any flow allowed in the subject-resource matrix is also allowed in the partition flow matrix, and the partition flow matrix is minimized to only include the flows required by the subject-resource matrix. Thus, the partition-to-partition flow policy is concisely articulated, and the ultimate subject-resource flows allowed can be seen in the subject-resource flow matrix. `subject 3` (TS-U) can read `resource 8` (TS) and write `resource 11` (U), but is not allowed any other accesses in the TS and S blocks, achieving "controlled interference," to the extent that `subject 3` is trusted.



Partition Flow Matrix

	TS	TS-U	U
TS	RW	-	R
TS-U	R	RW	W
U	W	-	RW

Resource Partition Map

Block	Resources
TS	1, 2, 7, 8
TS-U	3, 4, 9, 10
U	5, 6, 11, 12

Other Resources

Subjects

	7	8	9	10	11	12
1	RW	W	-	-	R	R
2	RW	-	-	-	R	R
3	-	R	RW	-	W	-
4	-	-	-	RW	-	-
5	W	-	-	-	R	RW
6	W	-	-	-	R	RW

Subject-Resource Flow Matrix

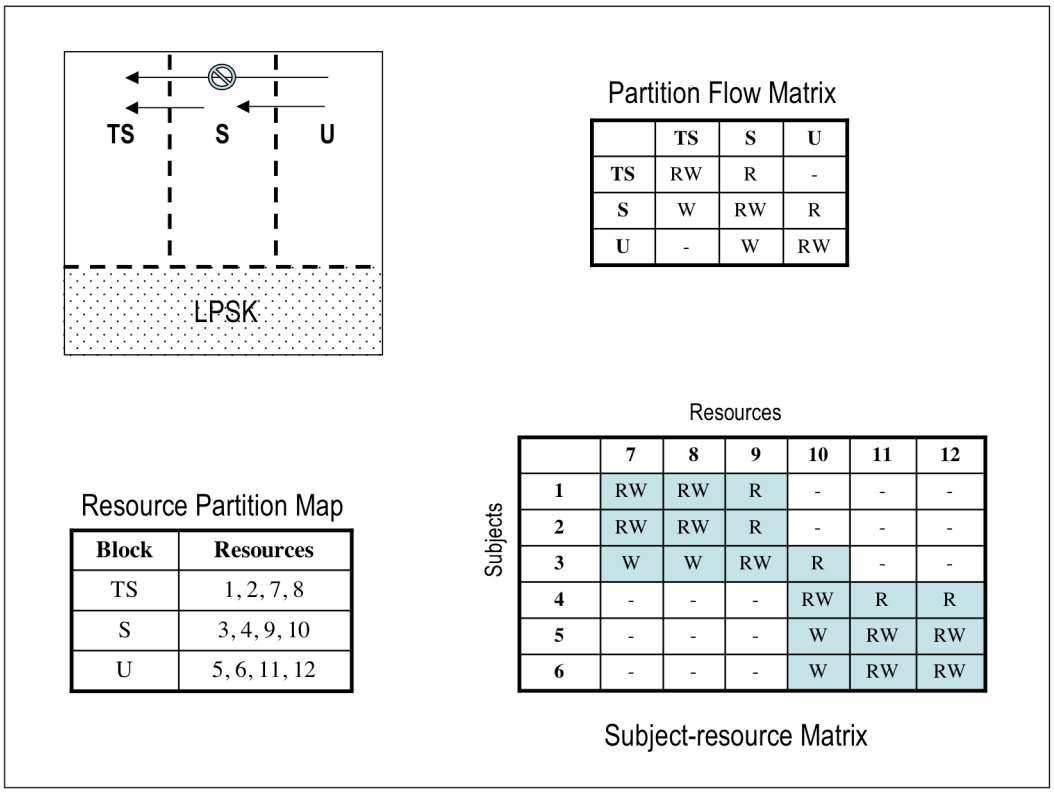
Figure 7. Kernel-based Controlled Interference in Least Privileged Architecture

3.9 Intransitive Information Flow

Given a high assurance kernel-based architecture that provides strict MLS policy enforcement, another information flow property of interest is that of being able to control how "far" in the sensitivity-label lattice the information may flow. We refer to this property as "intransitive information flow;" it can be considered to be another form of the principle of least privilege. It is different from controlled interference in that it is a *restriction* to the strict MLS policy rather than an extension or relaxation.

This sort of restriction has several applications. For example, in an MLS system that enforces a confidentiality policy, there may be a concern that low integrity data or programs in the U equivalence class do not *corrupt* the TS equivalence class, i.e., protecting data integrity in a confidentiality-policy system. Restriction of flows is also discussed in the context of the "cascade" problem.[Millen88]

Intransitive information flow can be achieved through the use of a "trusted subject" that is relied upon with respect to the intransitivity policy. In the separation kernel architectures, the partition flow matrix can be configured to disallow direct reading of U information by TS subjects, and direct writing of TS objects by U subjects (see Figure 8). Figure 8 also illustrates how the Least Privilege architecture can apply least privilege to limit the transitive flow to a single subject, such as a virus checker. To prevent transitive flow from U to TS via S, the S subjects must be trusted to enforce the intransitive policy.



	TS	S	U
TS	RW	R	-
S	W	RW	R
U	-	W	RW

	7	8	9	10	11	12
1	RW	RW	R	-	-	-
2	RW	RW	R	-	-	-
3	W	W	RW	R	-	-
4	-	-	-	RW	R	R
5	-	-	-	W	RW	RW
6	-	-	-	W	RW	RW

Figure 8. Intransitive Information Flow via Separation Kernel Trusted Subject

In the Evaluated Policy architecture, intransitive information flow can be achieved through the addition of an integrity policy enforcement mechanism in the kernel together with subjects that are trusted with respect to the integrity label lattice. As shown in the Figure 9 example, resources in the TS equivalence class have an H (high) integrity level, and resources in the U equivalence class have an L (low) integrity level. The MLS integrity policy prohibits the reading of information that is at a lower integrity level, so a H subject cannot directly read L information. The S subject could, for example, provide an inspection or modification of U-L data to ensure that it has H integrity before *upgrading* the data to the TS-H equivalence class. Again, to prevent transitive flow from U to TS via S, the S subjects must be trusted to enforce the intransitive policy.

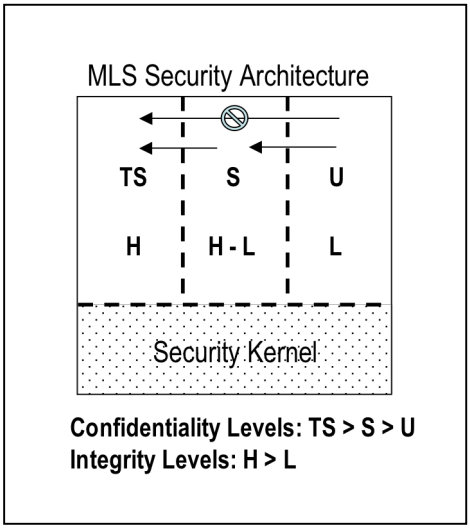
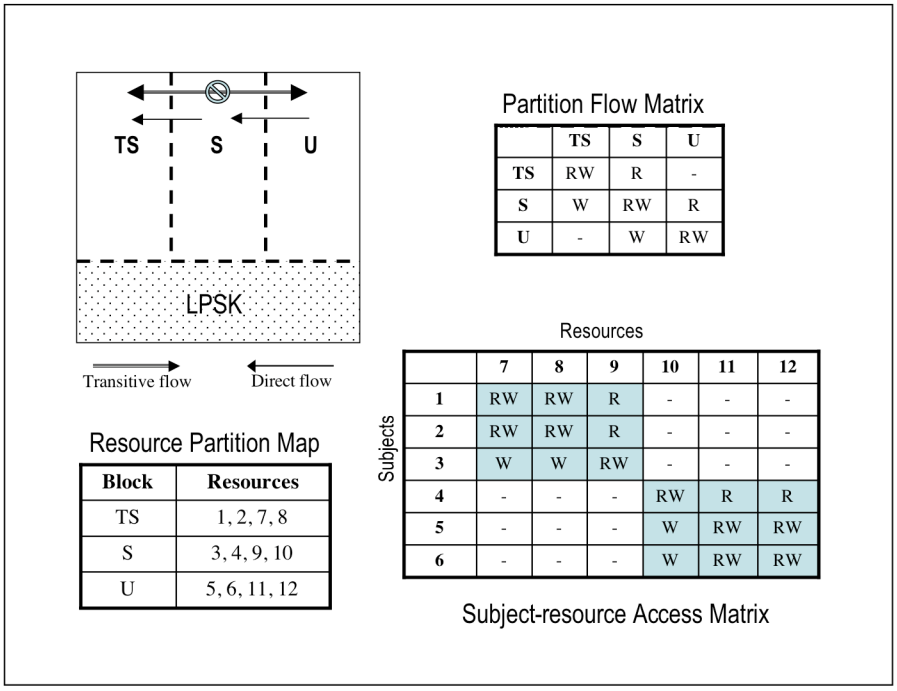


Figure 9. Intransitive Information Flow via Security Kernel Trusted Subject

The SKPP compliant LPSK kernel can also be configured to allow some inter-block flows but strictly prohibit transitive flow, with kernel assurance, by separating resources in the subject-resource flow matrix into an upper section with subjects 1, 2 and 3, and a lower section with subjects 4, 5 and 6, e.g., as shown in Figure 10. The kernel then ensures that no information can flow between the TS and U equivalence classes, even though subjects in S can read from U and write to TS. A similar partition can be created by the Evaluated Policy architecture through the use of non-hierarchical categories.



	TS	S	U
TS	RW	R	-
S	W	RW	R
U	-	W	RW

	7	8	9	10	11	12
1	RW	RW	R	-	-	-
2	RW	RW	R	-	-	-
3	W	W	RW	-	-	-
4	-	-	-	RW	R	R
5	-	-	-	W	RW	RW
6	-	-	-	W	RW	RW

Figure 10. Strict Intransitive Information Flow without Trusted Subjects

4 Summary and Future Work

We have compared three MLS security architectures: (1) MILS, (2) Evaluated Policy, and (3) Least Privilege. These architectures share some important security properties but each also offers unique features that may be significant for a particular security situation or deployment scenario, especially with respect to high assurance of policy enforcement. The performance tradeoffs in the architectures have not been examined in detail here, but several architecture-based advantages are apparent (a summary comparison is provided in the Appendix).

The Evaluated Policy architecture has advantages for:

- Assurance of MLS policy enforcement
- Dynamic resource management
- Label space scalability
- Cohesion and efficiency of evaluation activities

The MILS architecture's advantage is primarily:

- Support for dynamic policies

The Least Privilege architecture has advantages in the following areas:

- Assurance of MLS policy enforcement (similar to Evaluated Policy)
- Assurance of relaxed policy enforcement
- Granularity of least privilege
- Intransitive Information Flow
- Granular controlled Interference

Each architecture appears to have advantages and disadvantages with respect to factoring and re-usability of the kernel.

Further research is needed to better quantify these architectural differences. For example, we have no data at this point regarding how the architectures actually differ in their prevention of covert channels. Also, we need further research to understand the inherent differences in performance.

5 References

Alves-Foss04 Jim Alves-Foss, Carol Taylor, and Paul Oman. A Multi-layered Approach to Security in High Assurance Systems. Proceedings of the 37th Hawaii International Conference on System Sciences. 2004

Ames83 S. H. Ames, M. Gasser, and R. R. Schell, "Security kernel design and implementation: An introduction," IEEE Computer, vol. 16, no. 7, pp. 14–22, 1983.

Anderson72 Anderson, J. P. Computer Security Technology Planning Study, ESD-TR-73-51, vol. I, ESD/AFSC, Hanscom AFB, Bedford, Mass., October 1972 (NTIS AD-758 206)

Benzel05 Benzel, T. V., Irvine, C. E., Levin, T. E., Bhaskara, G., Nguyen, T. D., and Clark, P. C., "Design Principles for Security", NPS-CS-05-010, Naval Postgraduate School, September 2005. (PDF) (Also Available as ISI-TR-605.)

CCIMB05 Common Criteria for Information Technology Security Evaluation, Version 3.0, CCIMB-2005-06-[001, 002, 003], Common Criteria Project Sponsoring Organizations, June 2005.

Corbato91 Corbato, Fernando. On Building Systems That Will Fail, Communications of the ACM, Vol. 34, No. 9. ACM, September1991

Denning76 Denning, D., A Lattice Model of Secure Information Flow, Communications A.C.M., 19(5):236-243, 1976.

Denning87 D. Denning, T. F. Lunt, R. R. Schell, M. Heckman, and W. Shockley, "A multilevel relational data model," in Proceedings 1987 IEEE Symposium on Security and Privacy [2], pp. 220–234.DigitalNet04 Security Target Version 1.7 for XTS-6.0.E. DigitalNet Government Solutions, March 2004

Goguen82 J Goguen, J Meseguer, "Security Policies and Security Models," in procedings of the IEEE Computer Society Symposium on Research in Security and Privacy, 1982, pp 11–20.

Irvine04 Irvine, C. E., Levin, T. E., Nguyen, T. D., and Dinolt, G. W., "The Trusted Computing Exemplar Project," Proceedings of the 2004 IEEE Systems, Man and Cybernetics Information Assurance Workshop, West Point, NY, June 2004, pp. 109-115.

Irvine05 Irvine, C. E., Collaborative Research: SecureCore for Trustworthy Commodity Computing and Communications. March 31, 2005. No expiration. Last viewed December 12th, 2005. https://www.fastlane.nsf.gov/servlet/showaward?award=0430566

Lampson72 Lampson, B., "A Note on the Confinement Problem," in Communications of the ACM, v 16 no 10 (Oct 1973), pp 613–615.

Landauer89 J. Landauer, T. Redmond, and T. Benzel, "Formal policies for trusted processes," in *Proceedings of the Computer Security Foundations Workshop II*, (Franconia, NH), pp. 31–40, June 1989.

Levin04 Levin, T. E., Irvine C. E., and Nguyen T. D., "A Least Privilege Model for Static Separation Kernels", NPS-CS-05-003, Naval Postgraduate School, October 2004

Levin06, Timothy E., Irvine, Cynthia E., and Nguyen, Thuy, D., Least Privilege in Separation Kernels, Proceedings International Conference on Security and Cryptography, Setubal, Portugal, August 2006, pp. 355-362.

Lipner82 S. B. Lipner, "Non-discretionary controls for commercial applications," in Proceedings 1982 IEEE Symposium on Security and Privacy, (Oakland), pp. 2–20, IEEE Computer Society Press, 1982.

Lunt89 T. F. Lunt, P. G. Neumann, D. E. Denning, R. R. Schell, M. Heckman, and W. R. Shockley, "Secure distributed data views security policy and interpretation for DMBS for a Class A1 DBMS," Tech. Rep. RADC-TR-89-313, Vol I, Rome Air Development Center, Griffiss, Air Force Base, NY, December 1989.

Millen87 J.K. Millen, Covert Channel Capacity, Proceedings of the IEEE Symposium on Research in Security and Privacy, Oakland, CA, pp. 60-66, April 1987.

Millen88 Jonathan Millen. The cascading problem for interconnected networks. In proceedings of the Fourth Aerospace Computer Security Applications Conference, Orlando, FL, December 1988, pp. 269–273.

NSA04 National Security Agency, (U) *Global Information Grid Information Assurance Capability/Technology Roadmap*, Version 1.0 (Final Draft), October 2004.

NSA05 NSA personnel, private conversation at the Naval Postgraduate School, 3 August 2005.

NSA06 Global Information Grid. National Security Agency. http://www.nsa.gov/ia/industry/gig.cfm?MenuID=10.3.2.2. Last Accessed June 2006.

Rushby81 Design and Verification of Secure Systems, John Rushby, ACM Operating Systems Review, Vol.15, No.5 (December 1981), p.12.

Rushby92 J. Rushby, Noninterference, Transitivity, and Channel-Control Security Policies, Technical Report CSL-92-02, SRI International, Menlo Park, CA, 1992.

Saltzer75 Saltzer, J. H., and Schroeder, M. D., The Protection of Information in Operating Systems, Proceedings of the IEEE, 63(9):1278-1308, 1975.

Schell85 Roger Schell, Tien Tao, and Mark Heckman, "Designing the GEMSOS Security Kernel for Security and Performance", Proceedings of the 8th National Computer Security Conference, September 1985, p.108.

Schellhorn95 Schellhorn, G., W. Reif, A. Schairer, P. Karger, V. Austel, and D. Toll. Verification of a Formal Security Model for Multiapplicative Smart Cards. in 6th European Symposium on Research in Computer Security (ESORICS 2000). 4-6 October 2000, Toulouse, France. Lecture Notes in Computer Science Vol. 1895. Springer-Verlag. p. 17-36.

Sibert87 W. O. Sibert et al., UNIX and B2: Are They Compatible?, Proceedings of the 10th National Computer Security Conference, National Bureau of Standards/National Computer Security Center, September 1987, Baltimore, MD, pp. 142-149.

SKPP04 U.S. Government Protection Profile for Separation Kernels in Environments Requiring High Robustness, Version 0.621, National Security Agency, 1 July 2004.

Stern91 Daniel Sterne, On the buzz word 'Security Policy', Proceedings of the IEEE Symposium Research on Security and Privacy, Oakland, California, May 1991, pp. 219-230.

Thompson87 M.F. Thompson, R.R. Schell, A. Tao, and T. Levin. Introduction to the Gemini Trusted Network Processor. In Proceedings of the 13th National Computer Security Conference, pages 211-217, Baltimore, 1987.

Vanfleet05 W. M. Vanfleet, R. W. Beckwith, B. Calloni, J. A. Luke, C. Taylor, and G. Uchenick, "Mils: Architecture for high assurance embedded computing," *CrossTalk*, vol. 18, pp. 12–16, August 2005.

Appendix: Summary Comparison

Table A-1 lists comparison factors discussed in the paper and for each, gives a rough estimation of the differential merits of the three architectures along with an abstract of the rationale for those estimates, reflecting the discussion in the section indicated. Merit increases from yellow/lowercase to green/ALLCAPS to blue/**BOLD**.

It is clear that some of these factors are more important for assurance of security policy enforcement, while others are important for usability, cost, etc., which may have a more indirect impact on security. A score is provided at the bottom of the table that reflects equal weights for each factor, and where yellow is 1, green is 2, and blue is 3. A column is provided in the table for readers to fill in their own weights in order to derive weighted scores that may be meaningful to a particular organization or security environment.

Table A-1. Summary Comparison of Security Factors

Architectural Security Factors		Weight	MILS	Evaluated Policy	Least Privilege
1	Locus of MLS policy enforcement point (3.1.2)		TRUSTED SUBJECT	**KERNEL**	**KERNEL**
2	Assurance via Kernel control of trusted subjects (3.7)		TRUSTED SUBJECT LABEL RANGE	TRUSTED SUBJECT LABEL RANGE	**SR MATRIX, TRUSTED SUBJECT LABEL RANGE**
3	Locus of Dynamic Resource Management (3.4)		APPLICATION	**KERNEL**	APPLICATION
4	Least Privilege Granularity (3.2		block sub-partitioning	RINGS	**RINGS AND SR MATRIX**
5	Controlled Interference (3.8)		UNCONSTRAINED TRUSTED SUBJECTS	UNCONSTRAINED TRUSTED SUBJECTS	**TRUSTED SUBJECTS CONSTRAINED BY SR MATRIX**
6	Intransitive information flow (3.9)		trusted subjects	CATEGORIES AND INTEGRITY TRUSTED SUBJECT	**SR MATRIX, CATEGORIES AND INTEGRITY TRUSTED SUBJECT**
7	Label-space scalability (3.5)		pre-configured set of labels	**DYNAMIC LABEL SPACE**	pre-configured set of labels
8	Support for Dynamic policies (3.6)		**UNLIMITED RUNTIME POLICY CHANGES**	limited – requires recompilation to change policy	LIMITED – REQUIRES RESTART FOR MAJOR POLICY CHANGES
9	Evaluation Effort (3.7)		trusted subjects, configuration data, kernel	**KERNEL**	CONFIGURATION DATA, KERNEL
10	Cohesion of evaluation (3.7)		product and product deployment analysis required	**PRODUCT ANALYSIS SUFFICIENT**	product and product deployment analysis required
	Security Score		15	23	23
	Weighted Score				