Analysis of Three Multilevel Security Architectures

Timothy E. Levin
Naval Postgraduate School
1411 Cunningham Rd.
Monterey, CA, 93943
levin@nps.edu

Cynthia E. Irvine
Naval Postgraduate School
1411 Cunningham Rd.
Monterey, CA, 93943
irvine@nps.edu

Clark Weissman
Northrop Grumman Corp.
One Hornet Way
El Segundo, CA,
clark.weissman@ngc.com

Thuy D. Nguyen
Naval Postgraduate School
1411 Cunningham Rd.
Monterey, CA, 93943
tdnguyen@nps.edu

ABSTRACT
Various system architectures have been proposed for high assurance enforcement of multilevel security. This paper provides an analysis of the relative merits of three architectural types – one based on a security kernel, another based on a traditional separation kernel, and a third based on a least-privilege separation kernel. We introduce the Least Privilege architecture, which incorporates security features from the recent "Separation Kernel Protection Profile," and show how it can provide several unique aspects of security and assurance, although each architecture has advantages.

Categories and Subject Descriptors
D.4 [**Software**]: Operating Systems – *security and protection.* D.2 [**Software Engineering**]: Software Architectures – *Data abstraction; Domain-specific architectures.*

General Terms
Measurement, Performance, Design, Security, Verification.

Keywords
Principle of Least Privilege, Security Kernel, Separation Kernel Partitioning Kernel, Multilevel Security, Architecture.

1. INTRODUCTION
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 [32][33]. Various system architectures have been proposed for *high assurance* enforcement of multilevel security. Many of these systems utilize common kernel constructs such as the *security kernel* [3] and the *separation kernel.*[36] However, we lack metrics or even a common framework for understanding the relative security characteristics of the different approaches. The purpose of this paper is to (1) compare and contrast several idealized *MLS system architectures* that are based on these kernels and (2) provide a basis for the construction of architectural security metrics.

We examine the classic "Evaluated Policy" *(EP)* architecture (e.g., XTS-400 [12], and GEMSOS[39]) based on a *security kernel*, the "Multiple Independent Levels of Security" (*MILS)* architecture [1][27][48] that is based on a basic *separation kernel*, and a new architecture [18] based on a kernel that is fully compliant with the security features defined in the Separation Kernel Protection Profile (SKPP): [34] the *Least-Privilege (LP)* architecture.

Comparing apples and oranges, such as these three architectures, requires abstraction of common characteristics. A policy *equivalence class* is one of those common characteristics. We describe a generic MLS environment as including these necessary functions, the absence of any of which would undermine the realization of the MLS policy:

- *Establishment and isolation of equivalence classes* of system resources – whereby all of the active resources in a given class are to be provided with the same security policy privileges
- *Rules for how interaction may occur* between equivalence classes – for MLS systems, the rules define a partial order of flows
- *MLS policy enforcement* – mechanisms that ensure the rules
- *A method for ensuring that the enforcement mechanisms themselves conform to the rules*.
- *A mapping between equivalence classes and MLS human-readable labels* – providing a link to National or other policy for handling of sensitive information.

We describe the enhancement of the basic MLS environment with functions for *controlled relaxation, restriction, and dynamic modification of the MLS policy*. These functions are commonly required for practical high assurance MLS systems. With controlled relaxation, high information can be securely transitioned to low domains, *when appropriate*. Restrictions to the strict MLS policy enable the application of the *principle of least privilege (PoLP)*, as well as certain specialized sub-policies; whereas dynamic policy modification supports adaptive response to emergencies and other environmental changes[32][33].

The three architectures differ in their approaches to these functions, and the differences are used to compare the assurance that the architectures embody. Additionally, they are compared with respect to several usability factors: scalability, reusability of components, and performance. While each architecture has advantages, in summary, our analysis shows that the Least Privilege and Evaluated Policy architectures provide better assurance.

2. BACKGROUND

To establish the context for the comparative analysis that follows, several underlying security policy concepts are described.

2.1 Security Policies

We regard *mandatory access control* (MAC) policies as those that regulate all users' access to information based on the sensitivity level of the information 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 his/her surrogate programs are constrained to create or interact with information that is at the same sensitivity level as the user's session.

Historically, The primary security policy of interest has been 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 [9].

A secure computer system protects the security mechanisms' *internal* code and data and hardware resources. Such a system provides a translation of *organizational security policies* [46] 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, processes). This description assumes that inter-process communication mechanisms (e.g., pipes) can be sufficiently modeled, for the purpose of security analysis, as objects that are read from and written 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 ordinary users), or may be independent system "daemons." Note that "exported resources," as used here, subsumes both subjects and objects. To implement a MAC policy, each exported resource is associated, explicitly or implicitly, with a sensitivity level. The policy enforcement mechanisms then ensure that the only accesses allowed in the system are those in which the interactions between levels are consistent with the MAC policy.

Multilevel security (MLS) policies are those MAC policies in which subjects may have access to 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 [4] and Biba [7] 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, sometimes called a "blind write," does not imply read. Mechanical problems of implementing "blind writes" are left to the system engineers.

2.2 Trusted Subjects

Useful MLS systems must include various functions for two-way interaction between levels. These multi-level functions can be modeled by *trusted subjects* that have a label range spanning multiple nodes of the lattice.[47][10][20][40] It is a significant distinction that trusted subjects do not violate the security policy: rather their defined behavior is an explicit part of the security policy, as follows. The set of flows allowed by the *system MLS security policy* is the union of: (a) the flows allowed by the *strict MLS policy*; and (b) the set of flows defined to be allowed for the system's trusted subjects (here, called the *relaxed MLS policy*). As a result, the "incorrect or malicious execution [of a trusted subject] is capable of violating the system security policy."[31]

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.

2.3 The Principle of Least Privilege

In their 1975 review, Saltzer and Schroeder [38] 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. As applied to internal mechanisms of a secure system, the principle of least privilege also helps to minimize and clarify the design. 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 [45]. 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 [22]. 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).

3. KERNELS

A *system security architecture* of enough functionality to be of interest has layers of functions, services, and interfaces. At the center of the architectures examined here is a small *kernel*, which is simple and minimized.[49] The kernel manages hardware resources, from which it creates, exports and protects abstractions (e.g., subjects/processes and memory objects) and related operations. The desired security characteristics of a *high assurance* kernel are those of a *reference monitor*:[3] *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 [11].

3.1 Security Kernel

A security kernel binds internal sensitivity labels to exported resources, and mediates access by subjects to other resources according to a partial ordering of the labels defined in an internal policy module.[2] The label space may support confidentiality and integrity policies as well as non-hierarchical categories,[26] A security kernel usually provides a hardware-supported ring abstraction [43][44] and can host trusted subjects. [39] The rings can separate applications within a process. Thus, a *subject* is a process-ring pair. All high assurance security kernels to date have utilized segmented memory, which provides persistent hardware based process-local memory-protection attributes [12] [13] [39] [41] as opposed to dynamic, global, hardware attributes based on memory paging mechanisms.

The security kernel mediates external communication 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., datagram). Security kernels generally support full resource and resource-allocation configurability during runtime.

3.2 Separation Kernel

A separation kernel [36] *maps* the set of exported resources into *partitions*[1]: `resource_map: resource → partition`. There may be multiple "subject" resources and multiple "object" resources within a given partition, but a partition is not itself a subject. Resources in a given partition are treated equivalently with respect to the inter-partition flow policy, and specifically, subjects in one partition can be allowed to access resources in another partition.

Separation kernels enforce the separation of partitions, and allow (subjects in those) partitions to cause flows, each of which, when projected to partition space (per the `resource_map` function), comprises a flow between partitions (which may be between different or identical partitions). The allowed inter-partition flows can be modeled as a "partition flow matrix" whose entries indicate the *mode* of the flow, `partition_flow: partition × partition → mode`. The mode indicates the direction of the flow, so that `partition_flow(P1, P2) = W` means that subjects in `P1` are allowed to write to any resource in `P2`.

The assignment of resources to partitions and the access control or "flow" rules are passed to the separation kernel in the form of *configuration data* that the kernel interprets during system initialization. Since configuration data correctness is critical for the enforcement of the intended security policy, a *configuration tool* is often described for the construction of flow rules. Although not part of the kernel itself, this tool can help the security administrator or system integrator to organize and visualize complex data. This helps to ensure that user inputs reflect the intended policy.

Another term for a separation kernel is a *partitioning kernel* (PK)[25], and we will use that in the remainder of this document to distinguish the basic separation kernel from the *least privilege separation kernel*, discussed next.

3.3 Least Privilege Separation Kernel

Least privilege separation kernels (LPSKs) extend the PK concept to support the greater privilege granularity described in the NSA Separation Kernel Protection Profile (SKPP) [34]. The LPSK also generalizes the PK's reference monitor aspect to be the locus of control for *all* inter-partition flows. In addition to the `resource_map` and `partition_flow` functions of a PK, an LPSK supports the principle of least privilege in a manner than can be represented as a "subject-resource" flow matrix, `subj_res_flow: subject × resource → mode`. The SKPP allows the subject-resource flow matrix to override the rules of the partition flow matrix, however we prefer a more restrictive interpretation where a given flow is allowed by the LPSK only if both matrixes allow it: [34][21]

```
allow_flow(subject, resource, mode) →
   mode ∈ subj_res_flow(subject, resource)
   &
   mode ∈ partition_flow(subject.partition,
                          resource.partition)
```

While the allowed partition and subject-resource flows can be unstructured and arbitrarily complex, the SKPP requires that (1) each secure configuration include an identification of a "base" partial ordering of flows between partitions to identify the *strict MLS policy*, and (2) subjects allowed to cause flows between partitions in addition to those base flows are treated as *trusted subjects*. Figure 1 shows a *system MLS security policy* with circular flows between the partitions (thick arrows) and between the subject and object resources (thin arrows), and illustrates how a base partial ordering (the thick, *solid* arrows) determines which of the subjects must be trusted. The subject in Partition 3 must be trusted, since its flow to the object in Partition 1, a thin dashed arrow, is not consistent with the base flows (the projection of this flow in partition space is represented by the thick *dashed* arrow).

The SKPP defines various degrees of runtime configuration change, as a means for altering the TOE security policy, although it does not provide advice or detailed requirements for how a

[1] A mathematical *Partition* of a set, *S*, divides *S* into subsets, called *blocks*, such that each element of S belong to exactly one block and the blocks do not overlap. Blocks form *equivalence classes* of S with respect to the elemental equivalence relation "in the same block as."[35] In separation kernel and MILS literature, blocks are called "partitions," and we will use that terminology here.

vendor should provide evidence of the "continuity of secure state during the course of dynamic configuration changes."

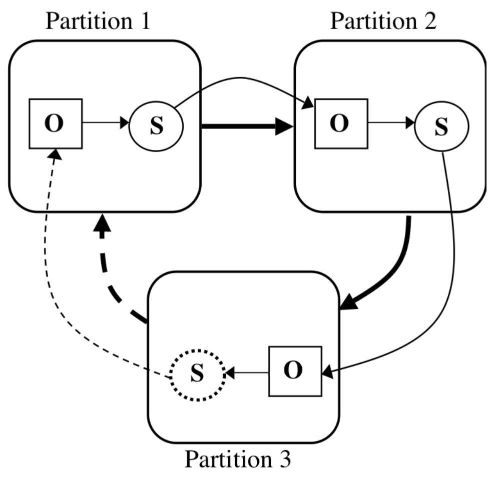

Figure 1. Partially Ordered Flows and Trusted-subject

4. SYSTEM ARCHITECTURES

In the class of architectures examined here, the Trusted Computing Base (TCB) includes a kernel, which is implemented using a combination of hardware and software mechanisms, and all applications essential for security policy support. We assume that such supporting policies, such as authentication and audit, are correctly provided and focus on how the MLS policy itself is realized. Thus, the TCB of each MLS architecture comprises a kernel that is configured to manage resources in a specific way, and in one case, an application program that is essential to enforcing the MLS policy.

In addition to the *active* mediation of subjects' access to exported resources, realization of an MLS policy may require *passive* design measures, to ensure that information does not flow through *internal resources* in violation of the rules (i.e., covert channels[19][28]). In other words, an enforcement mechanism [42] cannot include an effective procedure for completely determining its own consistency [14][29] with respect to the policy it is enforcing; so passive design techniques or externally applied methods are required. These techniques can include for example, physical separation of resources, and design disciplines that result in temporal or spatial partitioning of internal resources.

Due to their use of minimized kernels, all three architectures require either a customized operating system or a translation layer between their respective kernels and any complex, commercial-grade applications. System application components outside of the TCB may also translate the TCB-exported resources into new abstract data types, but the TCB's policy restrictions will still hold over those resources. For example, resource management services such as an ORB or DBMS can be provided, but these kernel-hosted services would not enforce or support *multilevel* security functions, so are not significant to an understanding of the architectures' multilevel security issues.

All three architectures can support trusted subjects and can be configured as a network of similar systems, assuming a common sensitivity-label lattice in the network. Architectures built upon separation kernels, be they PK or LPSK, will be referred to as "PK-based architectures." In Sections 6, 7, and 8, the three architectures are compared with respect to security functionality, assurance, evaluatability and usability

4.1 MILS

The **MILS architecture** is based on the PK and the use of trusted subjects to manage inter-level communication. Because MILS itself is evolving [1][48][27], we summarize our current understanding of it as follows.

The kernel of the MILS architecture is a *real-time* PK. MLS sensitivity labels are assigned to each partition by system-level applications or system (vice kernel) configuration data. Hierarchical confidentiality labels comprise the label space. At least one of the partitions is allowed (via PK configuration data) to read and write data in all of the other partitions. Enforcement of inter-partition flow policy is allocated to a trusted subject in this partition, which is called for example, a *Block Access Controller* (BAC),[27] or *Partitioning Communication System* (PCS).[1][48] This *AC-subject,*[2] provides the only means by which other subjects can communicate between local partitions or between partitions in distinct network nodes. Thus, subjects read and write information in other partitions by making requests to the AC-subject. Similar to the PK's configuration tool, an *AC-subject configuration tool* may be used to generate AC-subject configuration data.

The architecture statically allocates one process per partition (dynamic changes are described in Section 6). Hardware support for rings is not provided, so there is one subject per process. Subject isolation is based on the PK's partition-separation mechanism, and the general paradigm for program services is the use of multiple partitions (vs. multiple subjects in a process). For example, a database service could be provided by a dedicated process/partition, where database accesses are accomplished via inter-partition communication.

Several partitions may be assigned the same confidentiality label: here we refer to these as *sub-partitions*, since together, they comprise an equivalence class with respect to the system MLS policy. Least privilege can be provided by restricting sub-partitions, over and above the MLS constraints, through the AC-subject. We understand that one MILS vendor may be working on incorporation of a kernel least privilege mechanism in their PK, but that development has yet to be published. Protection from covert channels is supported by the static assignment of resources to a given partition.

4.2 Evaluated Policy

The **Evaluated Policy (EP) architecture** is based on a security kernel. The name Evaluated Policy architecture derives from the fact that the flow policy enforced by the system is validated as part of product evaluation.

There may be several subjects within the same process (as stated above) The EP architecture uses each ring to identify a *privilege domain*, across all processes. 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. Within a process, more privileged subjects can also be given security labels that

[2] We will use this term, since at the time of this writing we could not find a common name for the component.

provide more access than the less-privileged subjects (e.g., to support an MLS service in the most-privileged ring), so a process can span equivalence classes, which is not possible in the PK-based architectures, where subjects are labeled homogeneously within partitions. Hence the "interference" of a trusted subject can be limited to occur only within its privilege domain, as well as within its label range. The hardware rings support 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 (e.g., call gates).

The EP architecture uses CPU scheduling and memory allocation schemes that can dynamically interleave security levels, rather than allocating those resources statically.

4.3 Least Privilege

Least Privilege (LP) architectures [18] inherit characteristics from both the MILS and EP architectures, and use an LPSK. Thus, like MILS, an LP system gets its definition of equivalence classes and flow policy from its initialization environment, and is run statically with those definitions (dynamic changes are described in Section 6). As with the EP architecture, hardware-supported rings are used to protect privileged applications and to define subjects of a process; and there may be many subjects within a given equivalence class.

Unlike the MILS architecture, the LP architecture does not require an AC-subject to perform its basic functions, but can host trusted subjects in a multi-level partition, if one is needed for other purposes. The LP architecture utilizes the partition, least privilege, and ring mechanisms of LPSK to provide the ability to securely manage complex processing environments, protecting and confining privileged applications, while limiting covert channels through the static allocation of resources.

5. COMPARISON FACTORS

The comparison of system security architectures includes functionality, assurance, usability and evaluatability factors. As discussed in the following sections, several architectural properties are inherited from their respective kernels.

6. FUNCTIONALITY

A high assurance MLS system *enforces* the MLS policy and may include capabilities for the *relaxation, restriction, and dynamic modification* of that policy. Inherent to the maintenance of a strict MLS policy is the establishment and isolation of *equivalence classes, rules* for interaction between those classes, *enforcement* of the rules, *design of the enforcement mechanism* to conform to the rules (see Section 7.2), and a mapping between the equivalence classes and *human-readable MLS labels*. Incorrect realization of any of these can subvert the security policy.

6.1 Establishment of Equivalence classes

In all three architectures, the kernel defines equivalence classes for MLS policy enforcement by associating attributes with exported resources, and it is axiomatic that the equivalence classes are provided to the kernel correctly to reflect the organizational security policy. For PK-based architectures, the binding of partition attributes to resources is imported via configuration data during initialization. In EP architectures, the kernel binds machine-readable security labels to resources as they are created. In all cases, exemptions to isolation are allowed per the policy *rules*.

6.2 Specification of Security Rules

The security rules define the MAC policy. For PK-based architectures, the rules for interaction across equivalence classes are defined in the kernel configuration data. In MILS systems, rules in the AC-subject configuration data restricts the transitivity of flows allowed by the PK configuration data. The rules can be configured to result in partially ordered inter-partition flows (modulo flows caused by trusted subjects).

In EP architectures, the kernel security module specifies rules for partially ordered interaction across equivalence classes (modulo flows caused by trusted subjects).

6.3 Locus of Policy Enforcement

In all three architectures, cross-equivalence-class information flows are enforced based on both the rules and the equivalence-class attributes associated with resources. In the MILS architecture, rules are enforced by a combination of the PK and the AC-subject. In the EP and LP architectures, the kernel enforces the rules.

6.4 Mapping to Human Readable Labels

None of the kernels of the three architectures require human-readable labels in their implementation: the mapping from human-readable labels to the corresponding equivalence classes is performed outside of the kernel in a manner consistent with policy.

6.5 Relaxation of Policy

The architectures differ in the ways that relaxation of the strict MLS policy is achieved. The relaxed MLS policy behavior of trusted subjects has been called "intransitive noninterference,"[37] a policy where, for example, a trusted subject may move data from TS to S, as an intermediary, but "untrusted" TS and S subjects cannot do so directly. We refer to the constrained behavior of a trusted subject as *controlled interference*, to avoid the double negative. Traditional difficulties in trusted subject development have been in modeling their correct behavior, as well as implementing useful constraints to guarantee correct behavior.[5]

The PK-based architecture configuration data determines which subjects may interact (both read and write) between security levels, and are therefore "trusted." The EP architecture assigns the individual trusted subject a label range (e.g., U to TS), allowing it to perform the necessary functions within that range (e.g., downgrade).

6.6 Restrictions to Policy

There are several "least privilege" mechanisms for constraining subjects from unfettered access to resources. EP and LP architectures, using kernel ring support, can constrain subjects. For example, a TS subject in ring 2 would be able to access only TS objects in rings 2 and 3. Although the MILS architecture does not utilize hardware ring separation, PoLP can be applied by subdividing a given partition into privilege-specific sub-partitions, some of which may be allowed less access to resources than the partition would allow. In the LP architecture, the LPSK's `subj_res_flow` function constrains subjects, with kernel assurance, to access only specific objects in a partition, thus supporting least privilege at the same level of granularity as the resources that the kernel exports – a granularity of control not possible in a PK or a security kernel.

Another restriction to information flow of interest is that of being able to control how "far" in the sensitivity-label lattice information may directly flow. We refer to this property as "intransitive information flow." It is different from controlled interference in that it is a *restriction* to the strict MLS policy rather than an extension or relaxation. Intransitive information flow has several applications; such as to provide integrity control in systems without explicit integrity labels, when there is a concern that low confidentiality information might *corrupt* high confidentiality subjects. Intransitive information flow can be achieved in the PK-based architectures by configuring the partition flow matrix to disallow direct reading of U information by TS subjects, and direct writing of TS objects by U subjects; but allow read-down/write-up from (e.g., an integrity filter program in) a SECRET partition. In the LP architecture, least privilege can limit the transitive flow associated within the SECRET partition to a single subject; this can also be achieved in the MILS architecture by configuring the system to have only one SECRET subject.

The LPSK kernel can also be configured to strictly prohibit transitive flow from U to TS. This can be accomplished in the subject-resource flow matrix by ensuring that the resources in S (including subjects) that can interact with TS are distinct from those in S that can interact with U. A similar separation can be created with the EP architecture through the use of non-hierarchical categories.

6.7 Dynamic Security Policies

The need for secure systems to respond to changing conditions [32][33] implies the support for dynamic policies, i.e., where mandatory access control semantics may be changed or over-ridden during extraordinary circumstances. The three architectures address dynamic policy changes in different ways.

The PK-based architectures' configuration data can include pre-configured, pre-verified policy "vectors" [34] to enable a runtime kernel policy change. It is also possible for a MILS system to support dynamic changes to the AC-subject's code or configuration data. For the EP architecture, the system must be halted, the internal policy module replaced, and the kernel recompiled and then restarted.

A serious consideration regarding dynamic security policies is their lack of formal foundations, industry standards, and accepted National policy. We know of no formal security policy model for dynamic MLS. It is clear from previous work [16] that *ad hoc* changes to an access control policy can prevent the understanding of the ultimate policy. Of particular concern are the revocation of access to data that is no longer accessible under a new policy, and how to "get the genie back in the bottle" after an emergency. If a given partition or domain has been "polluted" with higher confidentiality data, cleaning it up may be intractable, short of drastic action.

7. ASSURANCE

This section summarizes several architectural factors the affect the assurance of secure systems.

7.1 Persistence of Policy

The primary conceptual difference between EP kernels and separation kernels is in the nature and persistence of their security policies. The policy of a security kernel is to control flows in a lattice. In contrast, the policy of a PK is to enforce the policy that is input to it: a *second order* effect. An EP system's policy is determined at kernel *compile time*, whereas SK systems enforce (or *interpret*, in compiler terminology) a policy that is determined at *boot time*, through configuration data. As a consequence, the security policy of an EP system is verified when it (including the security kernel) is evaluated, but the policy of the PK-based system can only be verified after the construction of the configuration data and the integration of the data with the system. The lack of persistence of a security policy in PK-based architectures implies that there is inherently less assurance of its enforcement, compared to EP systems.

7.2 Design to Avoid Covert Channels

The design of the TCB to avoid covert channels, and the analysis to demonstrate their absence, is essential for all MLS systems. These activities are more difficult with EP architectures (than with PK-based architectures) due to their use of dynamic runtime resource allocation.

The LP architecture is less problematic for covert channels than the MILS architecture, since in the latter the analysis must extend to the AC-subject.

7.3 Structural Abstractions

The architectures differ in the abstractions available to support development. The "second-order" policy of PK-based architectures is not as concrete as the EP architecture's MLS model. Also, the MILS architecture lacks an overarching abstraction like rings for organizing program integrity and privilege[43]: those considerations must be handled in an *ad hoc* fashion via restrictions defined in AC-subject configuration data. As a result, it is more likely that the intended MLS policy can be undermined in a MILS- or LP-based system through programmer and verifier confusion (e.g., in constructing either the AC-subject or the configuration tool).

7.4 Reliance on Trusted Subjects

Reliance on trusted subjects is an architectural weakness for several reasons, including the problem that they require analysis and demonstration of functional correctness over and above what is required for the basic MLS security model. A principle for the use of trusted subjects is to minimize their range of trust.[30][6] Unfortunately, the AC-subject requires a range over all partitions, so it is a single point of failure. That the fundamental MLS policy functions of MLS enforcement and mediation of relaxed MLS policy are performed in a trusted subject complicates this situation. However, it is clear that the addition of kernel-enforced constraints to trusted subject behavior reduces the amount of "blind faith" (viz., trust) that the users must have in them, and commensurately increases the assurance in the system compared to one without such constraints. The increased granularity with which the LP architecture can apply constraints on trusted subjects provides is an assurance advantage in this regard.

7.5 Verification of Configuration Data

Verification of configuration data in PK-based architectures is a significant concern, as this data determines the system's security policy. The verification will likely be supported by the presence of evaluated configuration tools.. The need for validation of each change to configuration data may result in lesser assurance than that provided by the evaluation of EP architecture systems, whose rules are verified once during the evaluation of each security kernel.

We do not yet have standards or accepted practices regarding PK policy and AC-subject configuration tools, such as for the properties they are supposed enforce and the criteria[8] for their robust construction. Until these standards and practices are established, the tools themselves can be a source for weakening the assurance of PK-based architectures.

8. EVALUATABILITY AND USABILITY

To ensure system trustworthiness, all of the mechanisms providing the basic security functions in a high assurance system need to be carefully assessed (i.e., evaluated).Security evaluation is an expensive, time-consuming effort that requires highly skilled personnel, so architectural factors that ease or get in the way of evaluation of a system are of interest. Also, performance, scalability and re-usability can help to determine the suitability of a security architecture for a particular use and are briefly summarized here.

8.1 TCB Code and Data

The assessment effort of a TCB module is roughly the same whether it is in the kernel or in a trusted subject. So for a given architecture, you "pay now" with kernel analysis or "pay later" with AC-subject analysis. Additionally, the evaluation of the configuration data for PK-based kernels, and AC-subjects for MILS architectures, will require extra effort, as, *for each separate configuration*, an evaluator, accreditor or other analyst must examine the rules for interactions between partitions. Frequent re-evaluation of configuration data changes, or evaluation of configuration data in the field, may be problematic.

8.2 Configuration Tools

Relative to EP architectures, which do not require configuration tools, the evaluation of MILS and LP configuration tools will require additional time and expense. Additionally, standard methods for the assessment of configuration tools do not yet exist, so the evaluations will be more expensive, initially.

8.3 Factoring and Reusability

The PK offers a "policy neutral" reference validation mechanism, which can be reused in a conceivably broader range of systems than a security kernel. The LP architecture also provides 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 little reusability advantage in moving the MLS policy enforcement function out of the kernel, as it is necessary to duplicate that functionality in every MILS system.

8.4 Scalability

PK-based architectures must provide a separate partition (or set of sub-partitions) for each supported MAC label upon initialization. Worse still, MILS systems, with single-process partitions, must initialize a partition for each application that may ever need to run. Since scheduling and memory allocation are static, all applications are scheduled and fully occupy memory even when they are not being used. This resulting pre-configuration of partitions would not be scalable to highly diverse security environments.[24] In contrast, LP architectures with multiple applications per partition can dynamically schedule those applications within the partition's time slice and memory domain. EP architectures are more dynamic still, creating new processes on the fly, and easily support a large number of different levels without requiring all applications and resources to be initialized at boot time.

8.5 Performance

We provide a few observations, but do not yet have in depth analysis. The overhead of a context switch is generally inversely proportional to the amount of context attributes to be managed. Thus, changing contexts within a process (i.e., between subjects or between a subject/process and the kernel) is faster than changing processes. The MILS services and program interactions are achieved through the use of separate processes. For example, inter-partition communication involves switches between three partitions: the source, the AC-subject, and the destination; the result is that the AC-subject creates a star network among partitions. The EP and LP architectures use multiple subjects within a process, and perform access control via a subject/kernel interaction, which would generally be more efficient than the MILS task-switching approach, all other things being equal.

To avoid covert timing channels, all MLS architectures must maintain a consistent allocation of time to each equivalence class, so interrupt-driven interactions with the MILS AC-subject would be constrained to be serviced within the MLS-aware scheduling sequence, which could adversely affect performance (of course some optimization is possible, via ordered scheduling, [17] etc.). On the other hand, MILS requires its PK to be "real time," to some definition, which may help to mitigate this problem.

9. SUMMARY AND FUTURE WORK

Three MLS security architectures were analyzed: MILS, Evaluated Policy, and Least Privilege. The EP and LP approaches appear to have an overall advantage over MILs with respect to the factors addressed. However, it is likely that some of these factors would be more important than others, depending on the context in which a particular architecture might be applied, which could result in a different overall ranking. Table 1 summarizes how the three architectures differ.

Further research is needed to better quantify and create metrics for these architectural differences.[23] For example, we have no data at this point regarding how the architectures differ in their performance or prevention of covert channels.

The flow-matrix model discussed in Section 3 might be shown to represent a noninterference property[15] if the "subject-resource" flow matrix were extended to include internal resources,[21] thus representing all state in the system. We are working on analysis of the security properties of that extension.

10. ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Grant No. CNS-0430566 with support from DARPA ATO, and by the Office of Naval Research. 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, of DARPA ATO, or of the Office of Naval Research. We thank Rance Delong for his helpful comments and suggestions regarding a previous draft of this document.

Table 1. Summary of MLS Security Architecture Differences

Factors	MILS	Evaluated Policy	Least Privilege
Functional			
Specification of rules	Configuration data	Hard-coded in kernel	Configuration data
Locus of MLS enforcement	Kernel and trusted subject	Kernel	Kernel
Relaxation of MLS policy	Configuration data	Assign subject a label range	Configuration data
Restrictions to MLS policy	Sub-partitions	Rings	Rings and subject-resource flow matrix
Dynamic MLS policies	Runtime Reconfiguration	None	Runtime Reconfiguration
Assurance			
Persistence of MLS policy	Dynamic with each boot	Static once kernel is compiled	Dynamic with each boot
Design to avoid covert channels	Static resource allocation leads to simpler design	Dynamic resource allocation requires greater care in design	Static resource allocation leads to simpler design
Structural Abstractions	Partition separation	Process separation, rings and static policy	Partition separation and rings
Constraints on trusted subjects	Partition-flow matrix	Kernel-enforced label restrictions	Subject-resource and partition-flow matrices
Verification of configuration data	Requires post-evaluation use of non-kernel tools	Not needed	Requires post-evaluation use of non-kernel tools
Evaluatability & Usability			
TCB Code and Data	Requires evaluation of kernel and AC-subject configuration data	Does not require evaluation of configuration data	Requires evaluation of kernel configuration data
Configuration Tools	Requires evaluation of kernel and AC-subject configuration tools	Does not require evaluation of configuration tools	Requires evaluation of kernel configuration tool
Factoring and reusability	Policy modifiable via reconfiguration (AC subject required)	Single policy kernel	Policy modifiable via reconfiguration
Scaleable process support	Statically defined by configuration	Dynamic instantiation of processes	Statically defined by configuration
Scalable label spaces	Label space proportional to total system resources	Large label space possible	Label space proportional to total system resources
Performance	Process context switches required for routing of data and services - slower	Some services and data available through ring crossing – generally faster	Some services and data available through ring crossing - generally faster

11. REFERENCES

[1] J. Alves-Foss, C. Taylor, and P. Oman. A Multi-layered Approach to Security in High Assurance Systems. *Proc. 37th Hawaii International Conference on System Sciences*. 2004

[2] 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.

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

[4] D. Bell and L. La Padula. *Secure Computer Systems: Unified Exposition and Multics Interpretation*. MITRE Corp. Technical Report MTR-2997. July 1975. URL: http://csrc.nist.gov/publications/history/bell76.pdf

[5] T. Benzel and D. Tavilla. "Trusted software verification: A case study." In *Proceedings of the Symposium on Security and Privacy*. pages 14–31, Oakland, CA, April 1985. IEEE Computer Society.

[6] T. V. Benzel, C. E. Irvine, T. E. Levin, G. Bhaskara, T. D. Nguyen, and P. C. Clark. *Design Principles for Security*. NPS-CS-05-010, Naval Postgraduate School, September 2005.

[7] Biba, K. J., *Integrity Considerations for Secure Computer Systems*. ESD-TR-76-372, April 1977. Electronic System Division, Air Force Systems Command, Hanscom AFB, Bedford, MA.

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

[9] Denning, D. A Lattice Model of Secure Information Flow, *Comm. ACM.*, Vol 19, No. 5, pp. 236-243, 1976.

[10] D. Denning, T. F. Lunt, R. R. Schell, M. Heckman, and W. Shockley. "A multilevel relational data model," *Proc. 1987 IEEE Symposium on Security and Privacy*, pp. 220–234, 1987.

[11] Department of Defense Trusted Computer System Evaluation Criteria. DOD 5200.28-STD, December 1985, Washington, DC. URL: http://csrc.nist.gov/publications/history/dod85.pdf

[12] DigitalNet Government Solutions. *Security Target Version 1.7 for XTS-6.0.E*. March 2004.

[13] L. Fraim, *SCOMP:* A Solution to the Multilevel Security Problem, *IEEE Computer*, Vol 16, No. 7, 1983.

[14] K. Gödel. "Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme," *Monatshefte für Mathematik und Physik*. Vol. 38 pp 173–198, 1931

[15] J. Goguen and J. Meseguer. "Security Policies and Security Models," *Proc. IEEE Computer Society Symposium on Research in Security and Privacy*, pp 11–20, 1982.

[16] M. Harrison, M. Ruzzo, J. Ullman. "Protection in Operating Systems," in *Communications of the ACM,* v 19 no 8 (Aug 1976), pp 461–471.

[17] W. Hu, "Lattice scheduling and covert channels." Proc. IEEE Computer Society Symposium on Research in Security and Privacy, 4-6 May, 1992. Pp. 52-61.

[18] C. E. Irvine. *Collaborative Research: SecureCore for Trustworthy Commodity Computing and Communications*. 31 Mar. 2005. https://www.fastlane.nsf.gov/servlet/showaward?award=0430566

[19] B. Lampson, "A Note on the Confinement Problem," *Comm ACM*, Vol. 16, No. 10, pp. 613–615, 1972.

[20] J. Landauer, T. Redmond, and T. Benzel. "Formal policies for trusted processes," in *Proc. Computer Security Foundations Workshop II*, (Franconia, NH), pp. 31–40, June 1989.

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

[22] T. E. Levin, C. E. Irvine, and T. D. Nguyen. Least Privilege in Separation Kernels, *Proc. International Conference on Security and Cryptography*, Setúbal, Portugal, August 2006, pp. 355-362. URL: http://cisr.nps.edu/downloads/06report_mps.pdf

[23] T. E. Levin, C. E. Irvine, and T. D. Nguyen. *An Analysis of Three Kernel-based Multilevel Security Architectures*, NPS Technical Report NPS-CS-06-001, August 2006.

[24] S. B. Lipner, "Non-discretionary controls for commercial applications," *Proc. IEEE Symposium on Security and Privacy*, (Oakland), pp. 2–20, IEEE Computer Society Press, 1982.

[25] Lockheed-Martin/The Open Group. *Protection Profile for PKs in Environments Requiring High Robustness*, Draft Version 1.3, submittal for NSA approval, 09 Jun 2003. http://www.csds.uidaho.edu/pp/PKPP1_3.pdf. Last Accessed 6/6/07.

[26] 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.

[27] D. McNamee , Scott Heller , Dave Huff. "Building Multilevel Secure Web Services-Based Components for the Global Information Grid." *CrossTalk*, Vol. 19, No. 5, pp. 15–19. May 2006.

[28] J. Millen. Covert Channel Capacity, *Proc. IEEE Symposium on Research in Security and Privacy*, Oakland, CA, pp. 60-66, April 1987.

[29] D. Myers. *Gödel's Incompleteness Theorem*. http://www.math.hawaii.edu/~dale/godel/godel.html, last accessed Feb 2, 2007

[30] P. G. Neumann. *Principled Assuredly Trustworthy Composable Architectures*. Final Report, DARPA Order No. M132. SRI International Menlo Park, California, December 2004. URL: http://www.csl.sri.com/neumann/chats4.pdf

[31] National Computer Security Center. *Glossary of Computer Security Terms*. NCSC-TG-004-88. 21 October 1988.

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

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

[34] National Security Agency. *U.S. Government Protection Profile for Separation Kernels in Environments Requiring High Robustness*, Version 1.03, 29 June 2007. URL:

[35] F. P . Preparata, and R. T. Yeh. *Introduction to Discrete Structures for Computer Science and Engineering*, Addison Wesley, Reading, MA, 1973.

[36] J. Rushby. Design and Verification of Secure Systems, *ACM Operating Systems Review,* Vol.15, No.5. December 1981

[37] J. Rushby. *Noninterference, Transitivity, and Channel-Control Security Policies*. Technical Report CSL-92-02, SRI International, Menlo Park, CA, 1992. URL: http://www.csl.sri.com/papers/csl-92-2/

[38] J. H. Saltzer, and Schroeder, M. D. The Protection of Information in Operating Systems, *Proc. IEEE*, Vol. 63, No. 9: 1278-1308, 1975.

[39] R. Schell, T. Tao, and M. Heckman. "Designing the GEMSOS Security Kernel for Security and Performance", *Proc. 8th National Computer Security Conference*, September 1985, pp.108.

[40] G. Schellhorn, 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.

[41] D. D. Schnackenberg. "Development of a Multilevel Secure Local Area Network," *Proc. 8th National Computer Security Conference*, October 1985, pp. 97 - 101.

[42] F. Schneider. “Enforceable Security Policies.” *ACM Transactions on Information and System Security*, Vol. 3, No. 1, February 2000. pp. 30–50.

[43] M. D. Schroeder and J. H. Saltzer, “A hardware architecture for implementing protection rings,” *Comm. A.C.M.*, vol. 15, no. 3, pp. 157–170, 1972.

[44] L. J. Shirley and R. Schell. "Mechanism Sufficiency Validation by Assignment," *Proc. IEEE Symp. Security and Privacy*, Apr. 1981, pp. 26-32.

[45] O. Sibert et. al, Unix and B2: Are the Compatible?, Proc. 10th National Computer Security Conference, National Bereau of Standards/National Computer Security Center, September 1987, Baltimore, MD, pp. 142-149.

[46] D. Sterne. On the buzz word ‘Security Policy.’ *Proc. IEEE Symposium Research on Security and Privacy,* Oakland, California, May 1991, pp. 219-230.

[47] M.F. Thompson, R.R. Schell, A. Tao, and T. Levin. Introduction to the Gemini Trusted Network Processor. *Proc. 13th National Computer Security Conference*, pp. 211-217, Baltimore, 1987.

[48] W. M. Vanfleet, R. W. Beckwith, B. Calloni, J. A. Luke, C. Taylor, and G. Uchenick. “MILS: Architecture for high assurance embedded computing,” *CrossTalk*, 18, pp. 12–16, August 2005.

[49] C. Weissman. “Security Controls in the ADEPT-50 Time Sharing System.” *Proceedings of the 1969 AFIPS Fall Joint Computer Conference*, pp 119-135. AFIPS Press, 1969.