

FORMAL MODEL DESIGN FOR SECURE OPERATING SYSTEMS*

Sihan Qing

qsihan@ercist.iscas.ac.cn

Qingguang Ji

Engineering Research Center for Information Security Technology, Chinese Academy of Sciences, China.

Abstract: This paper reviews some of the difficulties that arise in the secure operating system design. Then a formal framework different from and more powerful than the existing ones in the literature has been proposed aiming at supporting multiple security policies in modern computing environments. Finally, three novel models in the different areas, namely the confidentiality policy model DMLR_MLS, the integrity policy model DTE_IPM, and the extended role-based access control model PCM_RBPC for controlling process privileges have been introduced, all of which have a number of desirable new features.

1 INTRODUCTION

Due to the explosive growth of various electronic applications carried on the Internet, the need for secure operating systems turns out to be increasingly important. As stated by Loscocco et al. [1], a secure system must provide a framework for defining the operating system's mandatory security policy. The necessity of operating system security to overall system security is undeniable; the underlying operating system is responsible for protecting application-space mechanisms against tampering, bypassing, and spoofing attacks. If it fails to meet this responsibility, system-wide vulnerabilities will result.

It is generally agreed that, multiple policies may be necessary whenever there are multiple security goals such as confidentiality, privacy, and

* Supported by the National Natural Science Foundation of China under Grant No. 60083007 and the National Grand Fundamental Research 973 Program of China under Grant No. G1999035810.

integrity; whenever users with different values and traditions must share a common system; whenever a system is composed of separately evaluated pieces, and whenever policies must adapt to changing circumstances. The paradigm of a unified security policy has the following major shortcomings: it's inflexible, if a user wants to modify build-in aspects of the system security policy, the whole system must be reevaluated; exchanging sensitive data with systems with other security policies is difficult or impossible in real-time; it's unrealistic, a computer security officer creating an automated security policy must often integrate diverse and contradictory security policies together into a single coherent policy to meet TCSEC criteria; performance is poor, adding security to existing systems seriously slows down throughput. Hence the single-policy paradigm must be extended into a more flexible, more interoperative, better-performing multipolicy paradigm [2].

To enforce multiple security policies in a secure operating system, one has had to overcome some obstacles, such as how policy conflicts should be articulated and resolved, how new policies should be dynamically added into the system effectively, and how useless policies should be revoked from the system effectively, etc. So far, much work has been done in this area, just to mention a few: GFAC[5], Flask architecture[4], and flexible authorization manager(FAM)[6]. All this work has been developed to propose a coherent framework for using diverse security mechanisms to meet the requirements of mandatory policies. Therefore the above research provided frameworks to implement multiple policies rather than formally reasoned about security aspects for coexistent policies. For example, the FLSK architecture can provide revocation support mechanism, but it does not address the security problems stemmed from adding (or modifying) and revoking policies. The conflict resolution mechanism presented in GFAC is too simple to be used in practice. Moreover, typical capability architectures are inadequate for supporting mandatory access control with a high degree of flexibility and assurance.

Kuhnhauser and Ostrowski discussed the general concepts of a formal framework for supporting security policies in a multipolicy environment, and proposed a framework that provides a view for a formal foundation for reasoning about policy equivalence, policy cooperation and policy conflicts [3]. But their framework is too complex and universal to be used in constructing a formal framework for reasoning about policy security issues in a secure operating system. Bell also suggested a conceptual framework for reasoning about policy cooperation and policy conflict problems [7], but the lack of generalization of definition for policy makes his multipolicy machine insufficient for analysis of multiple security policies in a secure operating

system. As a consequence, it is necessary to establish a formal framework for reasoning about security aspects stemming from the coexistence of multiple policy models based on the related work. [3][7].

The foundation of an operating system's security is the security policy, containing all security related requirements. A security model is a precise representation of a security policy and thus is the starting point of any security policy implementation. Thus success in achieving a high degree of security in an operating system depends not only on the multipolicy paradigm that replaces the single-policy paradigm, but also on the improvement of the existing models and policies, such as confidentiality models and policies, authorization models and policies, and integrity models and policies etc.

As stated by Landwehr et al. [9], although the BLP model has dominated efforts to build secure operating systems for many years, using the BLP model in real applications has the following deficiencies:

1. Use of the BLP axioms in conjunction with trusted processes makes it difficult to determine the exact nature of the security rules that a system enforces. For example, the actual security rules enforced by the MME system include both the axioms of the Bell-LaPadula model and the exceptions allowed by the trusted processes.
2. Absence of multilevel objects. BLP recognizes only single-level objects; some message system data objects (e.g., messages and message files) are inherently multilevel.
3. No structure for application-dependent security rules. An example is a rule that allows only users with release authority to invoke the release operation. Such application-dependent rules are not covered by the BLP model.

To address the first problem, Rushby [12] observes that, in the absence of any precise formulation of the role of trusted processes within a model of secure system behavior, and in the absence of any formal understanding of how properties proved of trusted processes combine with those proved of a security kernel in order to establish the security of the complete system, there is no real justification for speaking of the 'verification' of the security of such systems at all.

To our knowledge, the first step towards formalizing trusted subject was taken by Bell [10], where he replaced the current-level function of a subject by two new functions, $\text{view-max}(S)$ and $\text{alter-min}(S)$, for an interpretation of the BLP model for networks. The subset of security levels defined by the

inclusive range between $\text{view-max}(S)$ and $\text{alter-min}(S)$ is called the level range, and denoted by $\text{ran}(S)$. Mayer [13] generalized this idea further and defined the set of trusted subjects in terms of those subjects where $\text{view-max}(S) \neq \text{alter-min}(S)$. That is, the set of security levels over which a trusted subject may have simultaneous view and alter access is the subset of security levels defined by $\text{ran}(S)$. The above argument has been generally accepted and used in the system design.

As for the second problem, Mayer [13] and Secure computing corporation [11] proposed an alternative design of multilevel security policy to support multilevel objects. But this design can only be regarded as an informal description of security rules rather than a formal model, and it is worth to discuss thoroughly the soundness of discarding maximum security levels in the design.

To solve the third problem, we believe that one will benefit from a multipolicy framework rather than direct addition of security rules like SMMSM. Ott [23] introduces a dynamically mediated mechanism for the current security level, improving flexibility in the BLP model at the expense of resource cost. In section 3, we will combine Ott's work with Bell's, obtaining a more practical model.

Integrity policy is another critical aspect in secure operating system design. As stated by Abrams et al. [14], there has been surprisingly little work done on integrity in recently years. To the authors' knowledge, no implementations of Clark-Wilson integrity have appeared. The very definition of integrity remains elusive. Biba developed an integrity policy model that is a dual of the Bell-LaPadula model in that it inverts the dominance relation. While Biba and BLP are isomorphic, they are sometimes treated as separate policy models. But Biba model cannot effectively prevent malicious codes from attacking against the system [15]. In addition to the Biba integrity model, Clark and Wilson introduced a commercial integrity model. Although the Clark and Wilson model effectively addresses all goals of integrity in a computer system, the experience shows that it is difficult or impossible that this model can be completely implemented in practice. As a practical technique, DTE can be used to implement integrity protection [16], and it has been proved successful in applying it to several prototype systems, such as SELinux, DTE-Linux and SAIC-DTE. Yet there is no formal DTE based integrity policy model in the literature. As such, it is worthwhile to develop a new model like this.

The mandatory security mechanisms of an operating system should be designed to support the principle of least privilege. In POSIX.1e, a dynamically mediated capability mechanism has been introduced to enforce

the principle of least privilege on processes. As stated in the POSIX standard, in order to support implementations that support the concept of least privilege to a finer level of granularity, one needs to provide the means by which a program can enable a capability only during the scope of execution for which it is actually required. But there is no real implementation following the above guidance. One of our goals in this paper is to design a good scheme to control privileges by means of the combination of RBAC [17], DTE and capability mechanism described in POSIX.1e.

In this paper, we propose a metapolicy based formal framework for reasoning about security issues resulted from the coexistence of multiple policy models, including at least discretionary access control model (DAC), and the following three novel models, namely, dynamically mediated level range mechanism based multilevel security model (DMLR_MLS), DTE based integrity protection model (DTE_IPM), and POSIX capability mechanism based role-based privilege control model (PCM_RBPC).

Based on Ott's work, Bell's work and work in [13], we then present a practical multilevel security model with mechanism to mediate level ranges dynamically. Based on DTE technique, Thomsen's work [20], and Lee's work [19], the DTE_IPM model is introduced with new integrity invariants and certification rules. Based on POSIX standard, Hoffman's work [21], role-based access control (RBAC) model, and lessons learned from UNIX system flaws analysis [22], the PCM_RBPC model is proposed with a new capability inheritance formula and a set of new invariants.

To make it into the frame of a conference paper, we only outline the main results and omit detailed arguments and proofs.

The remainder of this paper is organized as follows. In the next section we introduce the formal framework in which the conflict relation and cooperation relation different from the ones defined in [3] are presented. In section 3, 4, and 5 we describe DMLR_MLS, DTE_IPM, and PCM_RBPC respectively. Finally in section 6 we provide concluding remarks.

2 A FORMAL FRAMEWORK TO SUPPORT MULTIPLE SECURITY POLICIES

In this section, we introduce a formal framework that supports multiple security policies, followed by an illustrative example.

2.1 General Framework

Suppose multiple security policies are presented by n formal models $\{M_i \mid i = 1, \dots, n\}$, each of which can be invoked, and other new models can be added into the system. The framework, as shown in Figure 1, mainly consists of a family of model interfaces, and a model combiner.

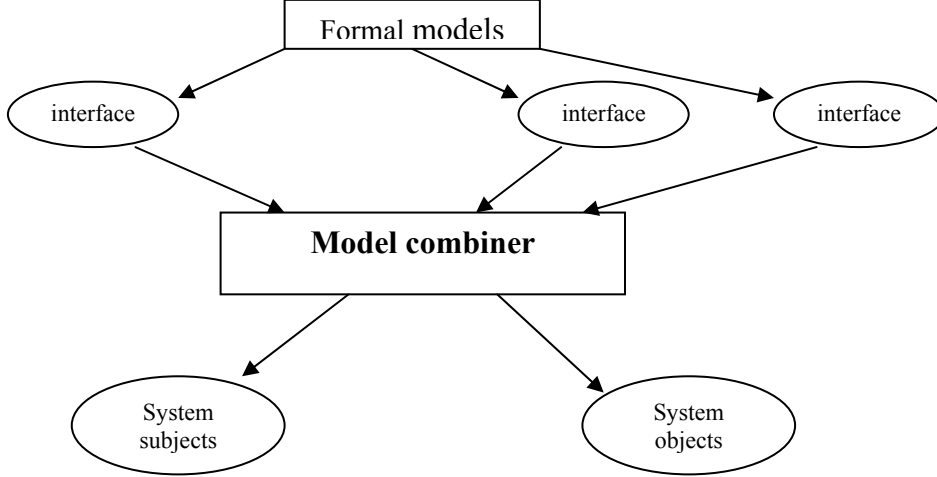


Figure 1: The general framework for multipolicy environment

For any formal model M_i , there exists a relation:

Interface $_{i,j} \subseteq \{V_{ij}\} \times D$, where V_{ij} is the state space of model M_i , D is the interface values set of the model (also called decision set).

Denoting $\{M_i \mid i = 1, \dots, n\}$ by \mathfrak{R} , the conflict relation Φ among models is a predefined relation on the model set. We say $\{M_{i_1}, M_{i_2}, \dots, M_{i_j}\} \in \Phi$, if we can derive contradictory invariants from these model's invariants, and lacking anyone of them cannot produce these contradictory invariants. By definition, we can obtain the following proposition:

If p and $\neg p$ are contradictory invariants belonging to $\{M_{i_1}, M_{i_2}\}$, then for any model M_{i_j} different from M_{i_1} and M_{i_2} , p and $\neg p$ are not contradictory invariants belonging to $\{M_{i_1}, M_{i_2}, M_{i_j}\}$. If p and $\neg p$ are contradictory invariants belonging to $\{M_{i_1}, M_{i_2}, M_{i_j}\}$, and there exist two of them, for example M_{i_1}, M_{i_2} , such that p and $\neg p$ belonging to $\{M_{i_1}, M_{i_2}\}$, then $\{M_{i_1}, M_{i_2}, M_{i_j}\} \notin \Phi$.

Corresponding to the conflict relation Φ , we define the conflict resolution relation Ψ as shown below:

For any given $\{M_{i_1}, M_{i_2}, \dots, M_{i_j}\} \in \Phi$, Ψ is a function from $\text{Interface}_{i_1}(k) (\{V_{i_1,k}\}) \times \dots \times \text{Interface}_{i_j}(k) (\{V_{i_j,k}\})$ to D , that is, $\Psi \subseteq (\text{Interface}_1(k), \dots, \text{Interface}_n(k)) \circ \Phi \times D$. In order to define the conflict resolution relation, we must find out all possible contradictory invariants. Obviously, the conflict matrix described in [3] is only a special conflict resolution relation, it is inadequate or impossible to characterize the general conflict resolution relation Ψ .

Next we introduce another relation in the model combiner, cooperation relation, based on the conflict relation.

Suppose $\Omega_1, \Omega_2, \dots, \Omega_s \in \Phi$, and $\bigcup_{j=1, \dots, s} \Omega_j = \{M_i \mid i = 1, \dots, n\}$, the

cooperation relation Σ is a function from $\Psi(\Omega_1) \times \Psi(\Omega_2) \times \dots \times \Psi(\Omega_s)$ to D , where Ω_i is called the set of conflict class.

The structure of the model combiner is as shown in Figure 2 below.

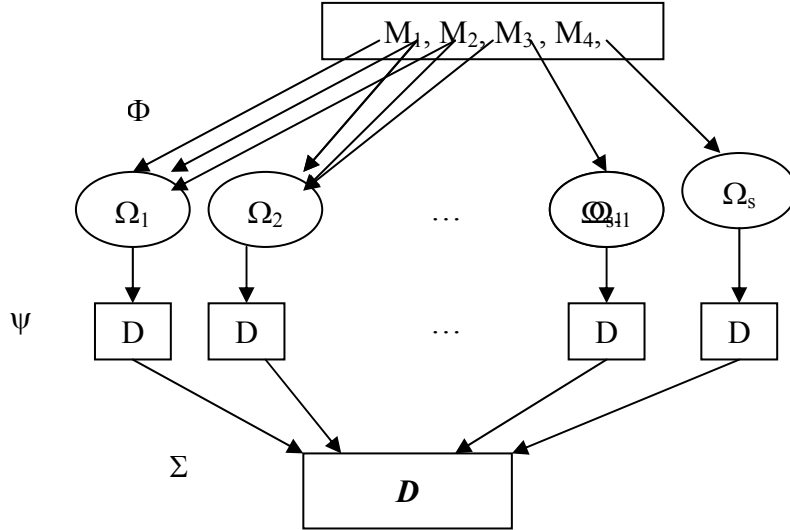


Figure 2: The structure of the model combiner

2.2 Case Study

In this section, a practical example is specified to illustrate our general framework. In this example, the model set is $\{DAC, DMLR_MLS, DTE_IPM, PCM_RBPC\}$; the decision set D is $\{YES, NO, DC, UNDEFINED\}$; the set of conflict class includes $\{DAC, PCM_RBPC\}$,

{ DMLR_MLS , PCM_RBPC }, and { DTE_IPM }. $\text{interface}_{DAC}(s)$ can be defined as follows:

$$\text{interface}_{DAC}(s) = \begin{cases} YES & \text{if } DAC \text{ can evaluate and grant an operation on states} \\ NO & \text{if } DAC \text{ can evaluate, but cannot grant an operation on states} \\ DC & \text{if } DAC \text{ recognizes an operation, but does not require checks for it} \\ UNDEFINED & \text{if } DAC \text{ does not recognize an operation} \end{cases}$$

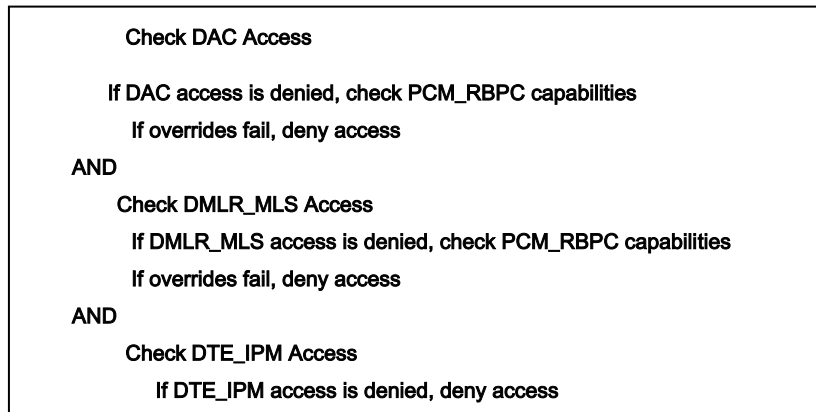
$\text{interface}_{PCM_RBPC}(s)$, $\text{interface}_{DMLR_MLS}(s)$, and $\text{interface}_{DTE_IPM}(s)$ can be similarly defined.

Next, we define the conflict resolution relation Ψ as follows:

$$\Psi(DAC, PCM_RBPC) = \begin{cases} YES & \text{if } \text{interface}_{DAC}(s) = YES \text{ and } \text{interface}_{PCM_RBPC}(s) = DC \\ & \text{or } \text{interface}_{DAC}(s) = NO \text{ or } DC \text{ and } \text{interface}_{PCM_RBPC}(s) = YES \\ NO & \text{if } \text{interface}_{DAC}(s) = NO \text{ and } \text{interface}_{PCM_RBPC}(s) = NO \text{ or } DC \\ & \text{or } \text{interface}_{DAC}(s) = NO \text{ or } DC \text{ and } \text{interface}_{PCM_RBPC}(s) = NO \\ DC & \text{if } \text{interface}_{DAC}(s) = DC \text{ and } \text{interface}_{PCM_RBPC}(s) = DC \\ UNDEFINED & \text{others} \end{cases}$$

$\Psi(DMLR_MLS, PCM_RBPC)$ can be defined analogously.

There are many ways to define a cooperation relation, for example, LaPadula's approach [5] could be used to define the cooperation relation. The detailed description of this cooperation relation is omitted here. Instead, an algorithm associated with the cooperation relation is given below.



3 DMLR_MLS MODEL

To the authors' knowledge, the idea of having rules that mediate integrity levels dynamically was first introduced by Biba in his low-water mark integrity model long time ago. The new idea of having rules that mediate current confidentiality levels dynamically was first introduced by Ott [23] recently in 2001. He's approach was successfully utilized in a prototype system RSBAC (Rule Set Based Access Control). But his proposal leads to cost increasing of the system. In addition, his method of introducing an additional level range for every subject is inconsistent with Bell's suggestion that replacing current level by a level range for trusted subjects.

In the following, we present a new model, called the DMLR_MLS model, laying emphasis upon some new rules and concepts.

In this model, a subject \underline{s} is assigned the following three security levels: a maximum security level $f_s(\underline{s})$, a maximum view level $f_{S-\max}(\underline{s})$, and a minimum alter level $f_{S-\min}(\underline{s})$ satisfying $f_{S-\min}(\underline{s}) \lesssim f_{S-\max}(\underline{s}) \lesssim f_s(\underline{s})$, where \lesssim is a dominance relation defined on the security level set. $(f_{S-\min}(\underline{s}), f_{S-\max}(\underline{s}))$ is referred to as \underline{s} 's write range, and is denoted by the set $\text{ran}(\underline{s})$. For every object \underline{o} , it is assigned the following two security levels: a maximum security level $f_{O-\max}(\underline{o})$ and a minimum security level $f_{O-\min}(\underline{o})$ satisfying $f_{O-\min}(\underline{o}) \lesssim f_{O-\max}(\underline{o})$. $(f_{O-\min}(\underline{o}), f_{O-\max}(\underline{o}))$ is referred to as \underline{o} 's security label range, and is denoted by the set $\text{ran}(\underline{o})$. In the case of $f_{O-\min}(\underline{o}) = f_{O-\max}(\underline{o})$, we denote $L(\underline{o}) = f_{O-\min}(\underline{o}) = f_{O-\max}(\underline{o})$.

Definition 3.1 An operation \underline{op} on an object \underline{o} , which is performed by a subject \underline{s} , is called multi-level-property-preserving, if $f_{S-\min}(\underline{s}) \neq f_{S-\max}(\underline{s})$, and this operation \underline{op} is related to all security levels in \underline{o} 's security label range. Otherwise, this operation is called single-level-property-preserving.

Remark: By saying: " \underline{op} is related to all security levels in a security label range," we mean that this operation must take consideration of the effect stemming from all security levels. For example, multilevel directory is generally an object satisfying $f_{O-\min}(\underline{o}) \neq f_{O-\max}(\underline{o})$, the operation delete multilevel directory is a multi-level-property-preserving operation, because this operation deletes all objects appearing in this directory, while these objects have all possible security levels in the label range of the directory.

Definition 3.2 If an arbitrary model variety in current state is denoted by x , then it is denoted by x^* in the successive state. By this convention, subject \underline{s} is called a trusted subject, if it satisfies:

- (1) $f_{S-\min}(\underline{s}) \neq f_{S-\max}(\underline{s})$
- (2) for any state, we have $f_{S-\min}^*(\underline{s}) = f_{S-\min}(\underline{s})$, $f_{S-\max}^*(\underline{s}) = f_{S-\max}(\underline{s})$,
 $f_{\max}^*(\underline{s}) = f_{\max}(\underline{s})$

The two key invariants in this model are:

- (1) Simple Security Property (ss-property): for any state, if a subject \underline{s} has read or write access to an object \underline{o} , then $f_{O-\max}(\underline{o}) \lesssim f_S(\underline{s})$.
- (2) Star Security Property (*-property): suppose b is the current access set, for any state, we have:
 - ① If $(\underline{s}, \underline{o}, \text{read}) \in b$, and read has single-level property, then $f_{O-\min}(\underline{o}) \lesssim f_{S-\max}(\underline{s})$.
 - ② If $(\underline{s}, \underline{o}, \text{append}) \in b$, and append has single-level property, then $f_{S-\min}(\underline{s}) \lesssim f_{O-\max}(\underline{o})$.
 - ③ If $(\underline{s}, \underline{o}, \text{write}) \in b$, and write has single-level property, then $f_{O-\min}(\underline{o}) \lesssim f_{S-\max}(\underline{s})$, and $f_{S-\min}(\underline{s}) \lesssim f_{O-\max}(\underline{o})$.
 - ④ If $(\underline{s}, \underline{o}, \text{read}) \in b$, and read has multi-level property, then $f_{O-\max}(\underline{o}) \lesssim f_{S-\max}(\underline{s})$.
 - ⑤ If $(\underline{s}, \underline{o}, \text{append}) \in b$, and append has multi-level property, then $f_{S-\min}(\underline{s}) \lesssim f_{O-\min}(\underline{o})$.
 - ⑥ If $(\underline{s}, \underline{o}, \text{write}) \in b$, and write has multi-level property, then $\text{ran}(\underline{o}) \subseteq \text{ran}(\underline{s})$.
 - ⑦ If $(\underline{s}_1, \underline{s}_2, \text{signal}) \in b$, then $f_{S-\max}(\underline{s}_1) \lesssim f_{S-\max}(\underline{s}_2)$.
 - ⑧ If $(\underline{s}_1, \underline{s}_2, \text{connect}) \in b$, then $\text{ran}(\underline{s}_1) \subseteq \text{ran}(\underline{s}_2)$.

The rules for dynamically mediating $\text{ran}(\underline{s})$ are as follows:

- (1) If $(\underline{s}, \underline{o}, \text{read}) \in b$, and read has single-level property, then $\text{ran}(\underline{s})$ evolves as follows:
 - 1) If $f_{O-\min}(\underline{o}) \lesssim f_{S-\min}(\underline{s})$, then $f_{S-\min}^*(\underline{s}) = f_{S-\min}(\underline{s})$, $f_{S-\max}^*(\underline{s}) = f_{S-\max}(\underline{s})$, $f_{\max}^*(\underline{s}) = f_{\max}(\underline{s})$
 - 2) If $f_{S-\min}(\underline{s}) \lesssim f_{O-\min}(\underline{o}) \lesssim f_{S-\max}(\underline{s})$, then $f_{S-\min}^*(\underline{s}) = f_{O-\min}(\underline{o})$, $f_{S-\max}^*(\underline{s}) = f_{S-\max}(\underline{s})$, $f_{\max}^*(\underline{s}) = f_{\max}(\underline{s})$
- (2) If $(\underline{s}, \underline{o}, \text{append}) \in b$, and append has single-level property, then $\text{ran}(\underline{s})$ evolves as follows:
 - 1) If $f_{S-\max}(\underline{s}) \lesssim f_{O-\max}(\underline{o})$, then $f_{S-\min}^*(\underline{s}) = f_{S-\min}(\underline{s})$, $f_{S-\max}^*(\underline{s}) = f_{S-\max}(\underline{s})$, $f_{\max}^*(\underline{s}) = f_{\max}(\underline{s})$
 - 2) If $f_{S-\min}(\underline{s}) \lesssim f_{O-\max}(\underline{o}) \lesssim f_{S-\max}(\underline{s})$, then $f_{S-\min}^*(\underline{s}) = f_{S-\min}(\underline{s})$, $f_{S-\max}^*(\underline{s}) = f_{O-\max}(\underline{o})$, $f_{\max}^*(\underline{s}) = f_{\max}(\underline{s})$
- (3) If $(\underline{s}, \underline{o}, \text{write}) \in b$, and write has single-level property, then $\text{ran}(\underline{s})$ evolves according to (1) and (2), that is, the operation write is decomposed into two operations: read and append.
- (4) If $(\underline{s}, \underline{o}, \text{read or append or write}) \in b$, and read or append or write has

multi-level property, then $\text{ran}(\underline{s})$ evolves as follows: $\text{ran}^*(\underline{s}) = \text{ran}(\underline{s})$, that is, only trusted subjects can perform these multi-level-property-preserving operations.

Convention: if \underline{s} is not a trusted subject, then for any object \underline{o} created by \underline{s} , the level function $f_{S-\min}(\underline{s})$ is assigned to \underline{o} as its security level; in this situation $\text{ran}(\underline{o}) = \{f_{S-\min}(\underline{s})\}$. If \underline{s} is a trusted subject, then for any object \underline{o} created by \underline{s} , one can assign any label range $\text{ran}(\underline{o})$ to \underline{o} , where $\text{ran}(\underline{o}) \subseteq \text{ran}(\underline{s})$, according to the specific requirement.

The main theorem that achieves our goal is listed below. Obviously, it is a nice result, and it captures our intuitive notion very well.

Theorem 3.1 Suppose the system satisfies the *-property and follows the rules for dynamically mediating the subject's security clearance range described above, if \underline{o}_i ($i=1,2$) only has single security level, i.e. $f_{O-\min}(\underline{o}_1) = f_{O-\max}(\underline{o}_1)$, $f_{O-\min}(\underline{o}_2) = f_{O-\max}(\underline{o}_2)$, and there exists a subject \underline{s} , such that $(\underline{s}, \underline{o}_2, \text{append}) \in b$ and $(\underline{s}, \underline{o}_1, \text{read}) \in b$, then either we have $f_{O}(\underline{o}_1) \lesssim f_{O}(\underline{o}_2)$, if \underline{s} is not a trusted subject; or we have $f_{O}(\underline{o}_1) \lesssim f_{O}(\underline{o}_2)$ or $f_{O}(\underline{o}_1), f_{O}(\underline{o}_2) \in \text{ran}(\underline{s})$, if \underline{s} is a trusted subject.

4 DTE_IPM MODEL

The DTE_IPM model consists of two main parts: the certification rules and the integrity protection state transition model. The purpose of the certification rules is to show how domains and types should be configured to implement the security invariants, and what assurance measures should be taken to achieve the integrity goals. The integrity protection state transition mode is a typical state machine model that defines the system states, state transition rules and security invariants. In [24], a useful concept -- event was introduced. An event is defined as a place in system code where a security-relevant decision is made or recorded. There are four types of events defined by the system: *operational event*, *access event*, *access override event*, and *audit event*. Operational event is a check to determine whether a subject has appropriate privilege to perform a restricted operation (e.g., mounting a file system, or adding a new user). Access event is a check to determine whether a subject can gain a requested mode of access (e.g., read) to an object. Access override event is a check to determine whether a subject has appropriate privilege to override a denial of access by an access control policy (e.g., override an ACL denying read access to a file). Audit event is a check to determine whether a security-relevant occurrence should be recorded in the system audit trail. If so, the record is created and added. This model only

copies with integrity aspects relevant to access events, operational events and access override events are left to be dealt with by the PCM_RBPC model.

4.1 The Certification Rules

- R₁** Identify each class of non-administrative (operational) users to be supported by the system. Define different domains for different classes.
- R₂** Identify each class of data. Define different integrity types for different classes.
- R₃** Identify each class of administrative users to be supported by the system. Define different domains for different classes. None of these domains can be the same as any of those defined in R₁.
- R₄** Identify relationship between domain and domain (DDT), and relationship between domain and type (DTT), these relations must be guaranteed to prohibit incorrect information flow. If information flow among some types in the system is inevitable, the information flow must be implemented via well-formed transactions (WFT).
- R₅** The transactions must be certified to be functionally correct.
- R₆** All of the constrained Data Items (CPIs) within the system conform to the integrity specification at the time the integrity verification process (IVP) is executed.

4.2 Model Overview

(1) State variables and constants:

- **SUBJECTS, OBJECTS, ROLES, TYPES, DOMAINS, USERS, OPS** and **WFT** (processes, objects, roles, types, domains, users, operations and well-formed transactions).
- **DDT** \subseteq **DOMAINS** \times **DOMAINS** \times **OPS**, a relation between two domains to show how to perform domain transitions, current domain transition methods include *auto*, *exec*, and *signal*. If $(d_1, d_2, auto$ or *exec* or *signal*) \in **DDT**, we say d_2 is controlled by d_1 , and denoted by *control*(d_1, d_2).
- **DTT** \subseteq **DOMAINS** \times **TYPES** \times **OPS**, a relation between domains and types to show which type of a process in a domain may operate. Abstractly, the operations can be classified as two classes, *view* and *alter*. If there exist a domain d and two types t_1 and t_2 , such that $(d, t_1, view) \in DTT$ and $(d, t_2, alter) \in DTT$, we say t_1 clings to t_2 , and denoted as *clingto*(t_1, t_2).
- $\neg control \subseteq$ **DOMAINS** \times **DOMAINS**, $(d_{source}, d_{target}) \in \neg control$, iff there exist no domains d_1, d_2, \dots, d_k , such that *control*(d_{source}, d_1),

- $control(d_1, d_2), \dots, control(d_k, d_{target})$.
- $\neg clingto, \subseteq TYPES \times TYPES, (t_{input}, t_{output}) \in \neg clingto$, iff there exist no types t_1, t_2, \dots, t_k , such that $clingto(t_{input}, t_1), clingto(t_1, t_2), \dots, clingto(t_k, t_{output})$.
 - $T \subseteq seq(TYPES \times OPS \times DOMAINS \times OPS \times TYPES)$, called assured pipeline set, a sequence of the form $(t_{11}, view, d_1, alter, t_{12}), (t_{21}, view, d_2, alter, t_{22}), \dots, (t_{k1}, view, d_k, alter, t_{k2})$ satisfying $t_{i2} = t_{(i+1)1}, I=1,2,\dots, k-1$.
 - $b = \{(s, d, o, t, y) \text{ or } (s, s_1, d, d_1, control) \mid y = \text{view or alter}, s, s_1 \in SUBJECTS, d, d_1 \in DOMAINS, t \in TYPES\}$, is called current access set.
 - $T_WFT(t:T) \rightarrow seq(WFT)$, an assured-pipeline to a sequence of well-formed transactions mapping, which makes an assured pipeline corresponding to a sequence of well-formed transactions to embody this pipeline.
 - $current_T(s:SUBJECTS) \rightarrow T$, a subject-to-assured-pipeline mapping, indicating that which assured pipeline is the one to be used currently by the subject.
 - $role_authorized_T(r:ROLES) \rightarrow 2^T$, a mapping of a role onto assured pipeline sets, which are authorized to the role.
 - $subject_authorized_domains \subseteq SUBJECTS \times DOMAINS$, a many-to-many mapping, subject-to-domain assignment relation.
 - $object_type(o:OBJECTS) \rightarrow TYPES$, an object-to-type mapping, which assigns type for any object.
 - $subject_role(s:SUBJECTS) \rightarrow 2^{ROLES}$, a subject-to-role mapping.
 - $user_roles(u:USERS) \rightarrow 2^{ROLES}$, a mapping of a user onto a set of roles.
 - $subject_user(s:SUBJECTS) \rightarrow USERS$, a subject-to-user mapping.
 - $active_domain(s:SUBJECTS) \rightarrow DOMAINS$, a subject-to-domain mapping, which assigns a currently executable domain to a subject.
 - $role_domains(d:DOMAINS) \rightarrow ROLES$, a mapping of a set of domains onto a set of roles.

(2) Invariants and constraints:

Integrity invariant-1: A subject's currently executable domain must be assigned to the subject, i.e.

$$\forall s \in SUBJECTS, \quad d = active_domain(s) \Rightarrow (s, d) \in SUBJECTS \times DOMAINS$$

Integrity invariant-2: A subject's current assured pipeline must be assigned to the role that the subject plays currently, i.e.

$\forall s \in \mathbf{SUBJECTS}, p = \text{current_T}(s) \Rightarrow r \in \text{subject_role}(s) \wedge p \in \text{role_authorized_T}(r).$

Integrity invariant-3: A subject has current access to an object, if in the executable domain the subject is authorized to perform the specific operation on the object of the specific type, or there exists an assured pipeline that can be used to access this object, and the subject is authorized to use this assured pipeline, i.e.

$(s, d, o, t, y) \in b \Rightarrow (d, t, y) \in \mathbf{DTT} \vee (\exists p \in \mathbf{T}, (t = \text{input}(p) \vee t = \text{output}(p)) \wedge p = \text{current_T}(s)).$

Integrity invariant-4: A subject transits from current domain to another domain, if there exist domains authorized for the subject: d_1, d_2, \dots, d_k , such that $\text{control}(d_{\text{source}}, d_1), \text{control}(d_1, d_2), \dots, \text{control}(d_k, d_{\text{target}})$, where d_{source} is the subject's current domain, and d_{target} is the target domain to be transited to, i.e.

$(s, s_1, d_{\text{source}}, d_{\text{target}}, \text{control}) \in b \Rightarrow \exists (d_1, d_2, \dots, d_k) (\text{control}(d_{\text{source}}, d_1), \text{control}(d_1, d_2), \dots, \text{control}(d_k, d_{\text{target}})) \wedge \{(s, d_{\text{source}}), (s, d_{\text{target}}), (s, d_1), (s, d_2), \dots, (s, d_k)\} \subseteq \mathbf{SUBJECTS} \times \mathbf{DOMAINS}.$

Information flow invariant: If there exist types t_1, t_2, \dots, t_k satisfying $\text{clingto}(t_0, t_1), \text{clingto}(t_1, t_2), \dots, \text{clingto}(t_k, t_{k+1})$, then there exist $j_1 < j_2 < \dots < j_n$, such that for any i , types $t_{j_i}, t_{j_{i+1}}, \dots, t_{j_{i+1}-1}$ satisfy one of the two conditions below :

(1) there exists an assured pipeline p , such that $\text{input}(p) = t_{\text{input}}$ and $\text{output}(p) = t_{\text{output}}$

(2) for any domain d , if $(d, t_{\text{output}}, \text{view}) \in \mathbf{DTT}$, then $(d, t_{\text{input}}, \text{view}) \in \mathbf{DTT}$ and if $(d, t_{\text{input}}, \text{alter}) \in \mathbf{DTT}$, then $(d, t_{\text{output}}, \text{alter}) \in \mathbf{DTT}$.

where $t_{j_i} = t_{\text{input}}, t_{j_{i+1}-1} = t_{\text{output}}$.

Integrity constraint: When new types or domains are added to the system, the original relations $-\text{clingto}$ and $-\text{control}$ are left unchanged.

5 PCM_RBPC MODEL

As previously mentioned in section four, the PCM_RBPC model is to be used to deal with operational events and access override events. In order to effectively control operational events and access override events it is necessary to enforce the least privilege principle in the system, which means any process in the system should be allocated only the absolutely minimal set of privileges needed to successfully perform the desire task.

We believe that the implementation of least privilege principle in the system should have a hierarchy with three layers: the first layer (the top level), called administration layer, is exploited to control the maximum

privilege that a subject may obtain in the system. In this layer, the static separation of duty and the dynamic separation of duty (see [17]) can be implemented effectively based on the role model. The second layer (the middle level), called functionality layer, is utilized to control the functional privileges that are required to perform specific functions in the system. A subject having a role can perform many functions, but is not allowed to perform them simultaneously. For example, an administrator may have both administration function and normal user function, but he cannot perform them simultaneously otherwise his privileges could be misused. Therefore these functions must be partitioned into different groups according to the system needs. In this layer, the functional separation, which is different from those separations of duty described in [25], can be implemented effectively based on the extended DTE model. The last but not the least layer, called performance layer, is used to control effective privileges. This layer can be implemented successfully by means of the capability mechanism resulted from POSIX. The proposed model is intended to formally reflect this hierarchy and to implement the least privilege principle.

Before describing the model, we start with introducing a new capability inheritance algorithm.

5.1 A New Capability Inheritance Formula

It is well known that POSIX 1003.6 has proposed a good capability mechanism, but it does not define either the notion of executable file capabilities or the notion of user capabilities [26]. Most importantly, it does not provide a capability inheritance algorithm which is essential to compute capability sets.

Capability inheritance algorithm currently implemented in Linux kernel has three steps [27]:

$$I_1 = I_0$$

$$P_1 = P_f \vee (I_0 \wedge I_f)$$

$$E_1 = P_1 \wedge E_f$$

where:

- I_0 , P_0 , and E_0 are current process current capability sets: *inheritable set*, *permitted set*, and *effective set* respectively.
- I_1 , P_1 , and E_1 are process capability sets that replace I_0 , P_0 , and E_0 respectively for the current capability sets after the execution of the algorithm.
- I_f , P_f , and E_f are file capability sets: *inheritable set*, *permitted set*, and *effective set* respectively.

The above capability inheritance algorithm is determined completely by the process inheritable set, and three file capability sets. Moreover, the process inheritable set remains unchanged, and is determined by user's uid. In the sequel, this algorithm is not enough to claim POSIX compliance.

Another improved algorithm was specified by Dragovic in [26]:

$$\begin{aligned} I_1 &= I_0 \\ P_1 &= P_f \vee ((I_1 \vee P_u) \wedge I_f) \wedge B_u \wedge B_g \\ E_1 &= P_1 \wedge E_f \end{aligned}$$

where the following three new capability sets have been introduced:

- user permitted set (P_u): capability set used to reflect user privileges in the system (via processes run under the user's uid).
- user bounding set (B_u): maximum capability set that a process running under the user's uid can obtain during its lifetime.
- global bounding set (B_g): B_g has system wide effect and describes the maximum possible set of privileges any process can reach during its lifetime in the system.

This algorithm has the following limitations. First, the user permitted set P_u , and the user bounding set B_u , newly introduced by the algorithm, are analogous to the existing inheritable capability set. Second, the necessity of having introduced B_g is unclear.

Overall, the existing capability inheritance algorithms are inadequate to reflect the spirit of the POSIX capability mechanism. To integrate the capability inheritance algorithm into our hierarchy implementation of the least privilege principle, a new capability inheritance formula must be given.

The new algorithm is as shown below.

$$\begin{aligned} I_1 &= I_0 \wedge I_f \\ P_1 &= (P_f \vee (I_1 \wedge P_b)) \wedge B_r \wedge B_d \\ E_1 &= P_1 \wedge E_f \end{aligned}$$

Where two new capability sets, namely B_r and B_d are introduced while other capability sets have their usual meaning. B_r is role capability set, which is determined solely by the role's id, irrelevant to the user's uid. B_d is domain capability set that determined solely by the domain's id. Due to the space constraints, the features of the newly proposed capability inheritance algorithm are omitted here.

5.2 Model Overview

(1) State variables and constants

- **SUBJECTS, USERS, ROLES, DOMAINS, EVENTS, TASKS, ExecutableObject, FileCapability,**

ProcessCapability, cardinality (subjects, users, roles, domains, operational events and access override events, tasks, executable objects, file capability, process capability, and the number of role members).

- $subdomain \subseteq \mathbf{DOMAINS} \times \mathbf{DOMAINS}$, a relation between domain and domain:
 $(d_1, d_2) \in subdomain$ iff the privilege set assigned to d_1 is the subset of the privilege set assigned to d_2 .
- $\alpha \subseteq \mathbf{ROLES} \times \mathbf{ROLES}$, a relation of roles hierarchy: if $(r_1, r_2) \in \alpha$, r_1 is called a subrole of r_2 .
- $RE \subseteq \mathbf{ROLES} \times \mathbf{EVENTS}$, a many-to-many mapping, role-to-event assignment relation.
- $DE \subseteq \mathbf{DOMAINS} \times \mathbf{EVENTS}$, a many-to-many mapping, domain-to-event assignment relation.
- $ET \subseteq \mathbf{EVENTS} \times \mathbf{TASKS}$, a many-to-many mapping, event-to-task assignment relation.
- $subject_user$, a subject-to-user mapping as specified in the DTE_IPM model.
- $user_roles$, a mapping of a user onto a set of roles as specified in the DTE_IPM model.
- $role_domains(d: \mathbf{DOMAINS}) \rightarrow \mathbf{ROLES}$, a mapping of a set of domains onto a set of roles as specified in the DTE_IPM model.
- $domain_executables \subseteq \mathbf{DOMAINS} \times ExecutableObject$, a many-to-many mapping, domain-to-executable_program assignment relation.
- $subject_roles$, a subject-to-role mapping as specified in the DTE_IPM model.
- $subject_authorized_domains$, a many-to-many mapping, subject-to-domain assignment relation as specified in the DTE_IPM model.
- $active_domain$, a subject-to-domain mapping, which assigns a currently executable domain to a subject as specified in the DTE_IPM model.
- $ER_a \subseteq \mathbf{ROLES} \times \mathbf{ROLES}$, $(r_1, r_2) \in ER_a$ iff roles r_1 and r_2 are a pair of mutually exclusive roles in authorization time.
- $ER_r \subseteq \mathbf{ROLES} \times \mathbf{ROLES}$, $(r_1, r_2) \in ER_r$ iff roles r_1 and r_2 are a pair of mutually exclusive roles in run time.
- $ED_r \subseteq \mathbf{DOMAINS} \times \mathbf{DOMAINS}$, $(d_1, d_2) \in ED_r$ iff functionally, the domains d_1 and d_2 are a pair of mutually exclusive domains in run time.
- $trace \subseteq \mathbf{SUBJECTS} \times seq(ExecutableObject)$, a function of process image sequence, a many-to-many mapping.

(2) State invariants

Here we only list some important new invariants, the normal ones are omitted. The invariants we have introduced in this paper will help in the understanding and analysis of the security model in SELinux.

Least privilege invariant-1: Two different domains assigned to the same process must have subdomain relation. i.e. $(\forall s \in \mathbf{SUBJECTS})(s, d_1) \in \text{subject_authorized_domains} \wedge (s, d_2) \in \text{subject_authorized_domains} \Rightarrow (d_1, d_2) \in \text{subdomain}$

Least privilege invariant-2: Two different roles which are mutually exclusive in authorization time cannot be assigned to the same user, i.e. $(\forall u \in \mathbf{USERS}) r_1 \in \text{user_roles}(u) \wedge r_2 \in \text{user_roles}(u) \Leftrightarrow (r_1, r_2) \notin \mathbf{ER}_a$.

Least privilege invariant-3: Two different roles which are mutually exclusive in run time cannot be assigned to different processes created by the same user, i.e. $(\forall u \in \mathbf{USERS}) r_1 \in \text{subject_roles}(s) \wedge r_2 \in \text{subject_roles}(t) \wedge \text{subject_user}(s) = \text{subject_user}(t) \Leftrightarrow (r_1, r_2) \notin \mathbf{ER}_r$.

Least privilege invariant-4 (Principle of operational separation): For a given user u , $\{event | (event, task) \in \mathbf{ET}\} \not\subseteq \{event | \forall s \in \mathbf{SUBJECTS}, \text{subject_user}(s) = u \wedge (r \in \text{subject_roles}(s) \wedge (r, event) \in \mathbf{RE}) \wedge (d \in \text{subject_authorized_domains}(s) \wedge (d, event) \in \mathbf{DE})\}$.

Least privilege invariant-5 (Principle of functional separation): The functionally different two domains can't be assigned to a process with the same user and role, i.e., $(\forall u \in \mathbf{USERS}, \forall r \in \mathbf{ROLES}) (\exists s, t \in \mathbf{SUBJECTS}) r \in \text{subject_roles}(s) \wedge r \in \text{subject_roles}(t) \wedge \text{subject_user}(s) = u \wedge \text{subject_user}(t) = u \wedge d_1 \in \text{subject_authorized_domains}(s) \wedge d_2 \in \text{subject_authorized_domains}(t) \Leftrightarrow (d_1, d_2) \notin \mathbf{ED}_r$.

Least privilege invariant-6: If a role is assigned to a subject, then so is its subrole. i.e., $\forall s \in \mathbf{SUBJECTS}, r_1 \propto r_2 \wedge r_2 \in \text{subject_roles}(s) \Rightarrow r_1 \in \text{subject_roles}(s)$.

Least privilege invariant-7: If a domain is assigned to a role, then so is its senior role. i.e., $\forall d \in \mathbf{DOMAINS}, r_1 \propto r_2 \wedge (r_1, d) \in \mathbf{RD} \Rightarrow (r_2, d) \in \mathbf{RD}$.

Least privilege invariant-8: For any given process, its process image's *inheritable* capabilities and *permitted* capabilities are bounded by the role's capabilities assigned to this process.

6 CONCLUSION

In this paper we have made several contributions to the field of secure operating system design. First, a formal framework different from and more powerful than the one described in [3] has been proposed aiming at supporting multiple application-specific security policies in modern computing environments. Then three novel models in the different areas, namely the confidentiality policy model DMLR_MLS, the integrity policy model DTE_IPM, and the extended role-based access control model PCM_RBPC for controlling process privileges have been introduced, all of which have a number of desirable new features. It is our hope that methods such as the ones we present in this paper will help in the development of operating systems with higher assurance of security.

7 REFERENCES

- [1] P. A. Loscocco, S. D. Smalley, P. A. Muckelbauer, R. C. Taylor, S. J. Turner, and J. F. Farrell (Oct. 1998), “The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments”. In Proceedings of the 21st National Information Systems Security Conference, pages 303– 314.
- [2] H.H.Hosmer (1993), ‘The Multipolicy Paradigm for Trusted Systems’, Proceedings of the New Security Paradigms Workshop, Little Compton, R.I., IEEE Press.
- [3] Winfried E Kuhnhauser and Michael von Kopp Ostrowski (May,1995), “A Framework to Support Multiple Security Policies”, In Proceedings of the 7th Canadian Computer Security Symposium, Ottawa, Canada.
- [4] R. Spencer, S. Smalley, P. Loscocco, M. Hibler, D. Andersen, and J. Lepreau (Aug. 1999), “The Flask Security Architecture: System Support for Diverse Security Policies”. In Proceedings of the Eighth USENIX Security Symposium, pages 123–139.
- [5] L.J.LaPadula (1995), “Rule-set modeling of a trusted computer system, in Information Security”:An Integrated Collection of Essays ,Marshall D. Abrams Sushil Jajodia Harold J. Podell, IEEE Computer Society Press,Los Alamitos, California, USA.

- [6] S. Jajodia, P. Samarati, V. S. Subrahmanian, and E. Bertino (1997). “A Unified Framework for Enforcing Multiple Access Control Policies”. Proc. Of ACM-SIGMOD.
- [7] D.E. Bell (Aug. 1994), Modeling the Multipolicy Machine, In Proceedings of the New Security Paradigm Workshop, pages 2-9.
- [8] D.E.Bell and L.J.La Padula (July 1975), “Secure Computer systems:unified exposition and multies interpretation”, MTR-2997, the MITRE Corporation, Bedford, MA.,(ESD-TR-75-306).
- [9] Landwehr, Carl E., C.L. Heitmeyer, and J. McLean (Aug. 1984), A Security Model for Military Message Systems, ACM Trans. on Computer Systems, Vol. 9, No. 3,pp. 198-222.
- [10] D. E. Bell (1988), “Security policy modeling for the next-generation packet switch”, in Proceedings of IEEE Symposium on Security and Privacy, pp.212-216.
- [11] Secure Computing Corporation (1999), “Assurance in the Fluke microkernel: Formal top-level specification”, CDRL A004, Technical report, Secure Computing Corporation.
- [12] J.Rushby (1981), “Design and verification of secure systems”, ACM Operating System Review, Vol. 15 No.5 pp. 12-21.
- [13] F.L. Mayer (1988), “An interpretation of refined Bell-LaPadula model for the TMach kernel”, Fourth Aerospace Computer Security Applications Conference, IEEE Computer Society Press, Florida, pp. 368-378.
- [14] Marshall D. Abrams and Michael V. Joyce(1995),” Trusted System Concepts, Computers and Security”, Vol. 14 No.1 pp. 45-56.
- [15] F.Cohen (1988), “Computer viruses: theory and experiments, Advance in Computer System Security”, Volume 3, edited by Rein Turn, ARTECH HOUSE, INC.
- [16] D.J.THOMSEN and J.T.HAIGH (December 1990), “A comparison of type enforcement and Unix setuid implementation of well-formed transactions”. In Proceedings of Sixth Annual Computer Security Applications Conference, Tucson, Arizona, pp.304-312.
- [17]D.F.Ferraiolo,R.Sandhu,S.Gavrila, D.R.Kuhn and R.Chandramouli (August 2001), “Proposed NIST standard for role-based access control”, ACM Transactions on Information and System Security,Vol.4, No.3., pp. 224-274.

- [18] D.D.Clark and D.R.Wilson, "A comparison of commercial and military security policies", in Proceedings of 1987 IEEE Symposium on Research in Security and Privacy, IEEE Computer Society Press, April 1987, pp.184-238.
- [19] T.M.P.Lee (1988), "Using mandatory integrity to enforce 'commercial' security", in Proceedings of IEEE Symposium on Security and Privacy, , pp.140-146.
- [20] THOMSEN, D.J.(1991), "Role-based application design and enforcement" In S.Jajodia and C.E.Landwehr, editors, Database Security, IV: Status and Prospects, North-Holland, , pp.151-168.
- [21] J.Hoffman (December 1997), "Implementing RBAC on a type enforced system" Proc. 13th Annual Computer Security Applications Conference, pp.158-163.
- [22] M.Bishop(1995), "A taxonomy of UNIX system and network vulnerabilities", CSE-95-10,Department of Computer Science, University of California, Davis,CA.
- [23] Amon Ott (November 2001), "The Rule Set Based Access Control (RSBAC) Linux Kernel Security Extension", in the 8th International Linux Kongress, Enschede, available at <http://www.rsbac.org/documentation.htm>.
- [24] Data General (Nov. 1996), "Managing security on DG/UX system", manual 093-701138-o4, Data General Corporation, Westboro, MA01580.
- [25] V.D.Gligor, S.I.Gavrila, and D.Ferraiolo (May 1998), "On the formal definition of separation-of-duty policies and their composition", in Proceedings of IEEE Symposium on Security and Privacy, Berkeley, CA.
- [26] B.Dragovic (April 2002), "LinSec-Linux Security Protection System", available at <http://www.linsec.org/doc/final/final.pdf>.
- [27] Dave Wreski, "Linux Capabilities FAQ v0.2", available at <ftp.guardian.no/pub/free/capabilities/capfaq.txt>.