A Compositional Multiple Policies Operating System Security Model

Lei Xia, Wei Huang, and Hao Huang

State Key Laboratory for Novel Software Technology,
Department of Computer Science and Technology,
Nanjing University, 22 Hankou Road, Nanjing 210093, China
xiaxlei@gmail.com, whuang.nju@gmail.com, hhuang@nju.edu.cn

Abstract. Multilevel security policies aim at only confidentiality assurance, with less consideration on integrity assurance and weakness in expressing channel control policies. Besides, the trusted subjects it introduces to handle the information flow "downgrade" have many security flaws. Moreover, increasing diversity of the computing environments results in various security requirements. However, current mainstream security models are aiming at only one or few requirements of them each. The Multi-Policy Views Security Model is presented, which is based on the MLS model, combining the domain and role attributes to the model, to enforce the expression power in channel control policies, make permission management more fine-grained and enhance the ability of confining the permission of the trusted subjects. Moreover, MPVSM has integrated the properties and functions of MLS, Domain-Type and Role Based models into one unified model. It is able to enforce multi-policy views in operating system in a flexible way.

Keywords: security model, multiple policy views, integrity assurance, confidentiality assurance, least privilege, separation of duties.

1 Introduction

Multilevel Security Model (MLS) [1] is one of the most widely used security model in various system. MLS aims at confidentiality assurance of the information, preventing the unauthorized leakage of the data in high security level. However, it rarely considers the integrity assurance, which is also a very critical security requirement [2,7,11].

Biba model is a mathematical dual of BLP, intending to protect integrity of information. However, they are both based on lattice mechanism, and information policies based on the lattice are transitive. Therefore, MLS models are weak in expressing channel control policies [22]. A firewall's control policy is a typical channel control policy. As an example in Figure 1.1, Information is allowed to flow from the Outside component to the Inside component via the Access Control, but is forbidden to do so directly. Such policies are hardly expressed in the MLS models.

S. Kim, M. Yung, and H.-W. Lee (Eds.): WISA 2007, LNCS 4867, pp. 291–302, 2007.
© Springer-Verlag Berlin Heidelberg 2007

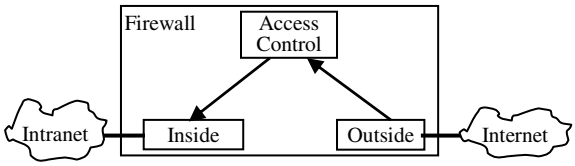

Fig. 1. Information channel control in a firewall

In addition, in MLS models, information is not allowed to flow "downward", such as from high confidentiality level to lower, or from lower integrity level to higher. However, in many situations, information flow "downward" is needed. Such an example of information flow from lower integrity level to the higher is the system call between user process and operating system. Data at a higher integrity level is more accurate and/or reliable than data at a lower integrity level. And the integrity level of user's data is usually lower than the operating system's data. Therefore, according to multilevel policies, user's data is not allowed to flow to the operating system data objects. However, in actual computer system, though user applications are not permitted to violate the integrity of operating system data, they should be given appropriate ways to pass the information or parameters to operating system.

To handle these problems, many MLS systems (such as HP-UX/CMW [8]) introduce some special trusted subjects outside the TCB. These special subjects are given privileges to bypass the MLS access control mechanisms. For they are not controlled by the access control mechanisms, they have almost all of privileges, which is far more than what they need to do they jobs. It is obviously a violation to the Principle of Least Privileges. These subjects turn to be the potential targets for malicious attacks. Once they are compromised, they can bring huge damages to the system.

Moreover, various security requirements are coming up with the sharply increased diversity and complexity of the computing environments. To satisfy these security requirements, a variety of security models were proposed in last twenty years. Widely-used security policies in current mainstream systems include multi-level military security model (Bell-LaPadula model, BLP) [1] and its variants (Biba [6], Dion model), Domain and Type Enforcement (DTE) [9], Role-based access control (RBAC) [14], and etc. Each of these models aims mainly at one or several security requirements, such as BLP aiming at the confidentiality of the information, Biba aiming at data integrity assurance, DTE aiming at confining the information flow channels, etc.

Previous trusted operating system usually enforced only one kind of mandatory access control model, for instance, Multics[3] implemented only BLP model in it. However, as mentioned above, the security goals in different applications are various. The different security requirements of applications result in different security models needed for them. How operating system to support this kind of multiple security model views--the access control model different applications can perceive in the system is different.

Recently, as a policy neutral security model, RBAC provides a valuable level of permission abstraction. However, using RBAC to simulate multi-level security level or

discretionary access control models [12] is over complex and therefore unpractical in real-world operating system.

In this paper, a Compositional Multiple Policy Security Model (MPVSM) is presented. MPVSM is a hybrid security model, which is based on Multi-level Security models. It combines confidentiality and integrity lattices into a unified security lattice for confidentiality and integrity assurance. It then divides the subjects and objects in the same security level into different domains and types, using access control mechanisms in DTE to make the permission assignment and management more fine-grained and flexible, meantime enforce the separation of duties between subjects in the same security level. In addition, using the thought of RBAC, role is added in MPVSM. Roles are assigned the extra permissions, which are independent of MLS and Domain-Type parts of the model. MPVSM makes use of the flexible permission assignment and revocation mechanisms in RBAC to confine the permissions of those special "trusted subjects", provides a way to make them do they job out of control range of the MLS and Domain-type access control parts, but meanwhile prevent them from too powerful to be potential security holes of the system.

MPVSM has integrated the properties and functions of Multiple Level Security, Domain-Type and Role Based models into one unified model. By combining the elements and attributes of DTE and RBAC to the MLS model, MPVSM avoids the drawbacks of MLS. MPVSM is able to enforce channel control and assured pipelines policies, with providing fine-grained permissions management. In addition, MPVSM owns an enhanced ability of policy expression. It can ensure the enforcement of least privilege to these special "trusted subjects" in the MPVSM model. Moreover, MPVSM provides a framework to enforce multiple policy views in operating system. It can not only enforce the equivalent functions of these three kinds of models independently in the system, but also can enforce multi-policy views between different applications in system.

The remainder of the paper is organized as follows. Section 2 describes the MPVSM formally. Section 3 gives the example policy configurations in MPVSM. Section 4 is the related works. And section 5 is the conclusion.

2 Multiple Policy Security Model

2.1 Overview

The architecture of the MPVSM is shown in figure 2. MPVSM comprises of elements, relations and mappings. A *user* in the framework is a system user. A *role* is a job function or job title within some associated semantics regarding the authority. *Subjects* are active entities in the system, usually processes or transactions. *Objects* are data objects as well as resource objects. *Domain* is a control access attribute associated with each subject. And *type* is the other control attribute associated with objects. Two global matrixes are defined to represent allowed access or interaction modes between domains and types or domains and domains respectively. *Permission* is an approval of a particular mode of access to object or interaction to subject. *Security label* is a 2-tuple, containing a *confidentiality label* and an *integrity label*.

There are several relations and mappings between elements. The relation between users and roles are defined in *user-role* assignment relation. The *user-subject* relation gives relation between subjects and users, while *subject-role* mapping figures out a subject's current running role. Permissions in system can be authorized to roles. Roles' authorized permissions are given in the *role-permission authorization* relation. Each role in system can have many authorized domains. *Role-domain authorization* relation gives the authorized domains of each role. Each subject has only one running domain, which is given in *subject-domain* mapping. Besides, each subject has a security label. The security labels are assigned to roles, and subject's security label is determined by the label of its running role. Each object has a *security attribute* which includes the *type* and *security label* of that object.

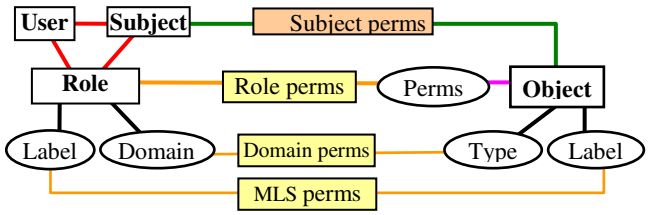

Fig. 2. The MPVSM

The *final permissions* that a subject gets are based on three kinds of permissions corresponding to that subject: *MLS permissions*, *Domain permissions* and *Role-based permissions. MLS Permissions* are coarse-grained base permissions, indicating the subject has the read-related permission or write-related permission. *Domain permissions* are the fine-grained permissions based on the *MLS Permissions. Role-based permissions* is independent from *MLS permissions* and *domain permissions.*

Correspond to the three kinds of permissions, MPVSM model contains three access control views: **1**) Multi-level Security Access Control. The *MLS permissions* given to the subject are based on the security level of the subject and the target object. **2**) Domain Access Control. Subjects run in different domains. A subject's *Domain permissions* are based on its running domain and the target objects' types. **3)**Role-Based Access Control. The subject has the permissions of its running role as its *Role-based permissions*.

In addition, MPVSM model is also an extensible security model, by adding other attributes to the role and object, besides the label and domain or type attributes of current role and object, more functions or properties can be added.

2.2 Formal Definitions

The following definitions formalize the discussion above:

Definition 2.1. Elements Sets

- Users: *U*
- Subjects: *S*

- Objects: O
- Roles: R
- Domains: D
- Types: T
- Confidentiality Labels: C
- Integrity Labels: I
- Security Labels: $SL \subseteq C \times I$
- Access modes: P, containing of two disjointed subsets: Read-related Modes RP={*read(r)*, *execute(e)*, *getattr(g)* **...** }; Write-related Modes WP={ *write(w)*, *append(a)*, *create(c)*, *delete(d), setattr(s)* **...** }; $P = RP \cup WP$.
- Domain transfer operation: *transfer(t)*□*transfer* denotes the subject can transfer from one domain to another domain.
- Role Permissions $CAP \subseteq P \times O$, $(p, o) \in CAP$ denotes the permission to access o in mode p.

Definition 2.2. $US \subseteq U \times S$, *user-subject* relation. More than one subject can run on behalf of one user at the same time. But each subject can only run on behalf of one user, called its running user.

- *user*: $S \rightarrow U$, mapping from subject to its running user. *user*(s) =u if and only if $u \in U \wedge (u, s) \in US$.

Definition 2.3. $UA \subseteq U \times R$, *user-role assignment* relation. Each user can be assigned many roles and each role can be assigned to many users.

- UR: $U \rightarrow 2^R$, mapping from user to the set of roles assigned to that user. $UR(u) = \{r \in R | (u, r) \in UA\}$.
- SR: $S \rightarrow R$, *subject-role* mapping, from the subject to its running role. Each subject has a running user and running role at anytime, and the role must have been assigned to that user: $SR(s) \in UR(user(s))$.

Definition 2.4. role's *security label*

- RL: $R \rightarrow SL$, mapping from role to its security label.
- Ssl: $S \rightarrow SL$, mapping from subject to its security label. Subject's security label is the same as its running role's security label: $Ssl(s) = RL(SR(s))$.

Definition 2.5. $RD \subseteq R \times D$, *role-domain authorization* relation. Each role has many authorized domains and each domain can be authorized to many roles.

- $RDom$: $R \rightarrow 2^D$, mapping from role to its authorized domains set. $RDom(r) = \{d \in D | (r, d) \in RD\}$.
- $SDom$: $S \rightarrow D$, mapping from subject to its running domain. Each subject is running in only one domain at anytime, and the domain is authorized to that subject's running role: $SDom(s) \in RDom(SR(s))$.

Definition 2.6. object's *security attribute*

- OT: $O \rightarrow T$, mapping from object to its type.
- OL: $O \rightarrow SL$, mapping from object to its security label.

Definition 2.7. $RCAP \subseteq R \times CAP$, role-permission authorization relation. $(r_1, cap) \in$ RCAP denotes that role r_1 has the role permission *cap*.

- *Rolecap*: $R \rightarrow 2^{CAP}$, mapping from role to its authorized *Role permissions* set. *Rolecap*(r)={capl(r,cap)$\in$*RCAP*}.

Definition 2.8. Two control matrixes
- *DTM*: $D \times T \rightarrow 2^{P}$, *domain-type access control matrix*. $p \in DTM(d, t)$ denotes that the subjects in domain d can access objects with type t in mode p.
- *DDI*: $D \times D \rightarrow \{\Phi, \{transfer\}\}$, *domain-domain interaction control matrix*. $transfer \in DDI(d_1, d_2)$ denotes that subjects in domain d_1 can transfer into domain d_2.

Definition 2.9. Multiply Levels Security rule: *MLS_rule*: $SL \times SL \rightarrow 2^{P}$, $a \in MLS\_rule$ (ls, lo) implies that subjects with security label ls can access objects with security label lo in mode a. This rule combines the BLP confidentiality and Biba integrity lattices. Let ls=(cs, is), lo=(co, io):

- If cs$\geqslant$co, permit all read-related operations, that is: $RP \subseteq MLS\_rule$(ls, lo).
- If cs$<$co, deny all read-related operations, that is: $\forall$ p$\in RP$, p$\notin$ *MLS_rule* (ls, lo).
- If is$\geqslant$io, permit all write-related operations, that is: $WP \subseteq MLS\_rule$(ls, lo).
- If is$<$io, deny all write-related operations, that is: $\forall$ p$\in WP$, p$\notin$ *MLS_rule* (ls, lo).

2.3 Permission Decision

Definition 2.10
- *MLS permission* (*MLP*): a subject's *MLP* on an object is determined as follow: *mlp*(s, o)={(o, p)|p$\in$*MLS_rule* (*Ssl*(s), *OL*(o))}
- *Domain permission* (*DP*): a subject's *DP* on an object is determined as follow: *dp*(s, o)={(o, p)|p$\in$*DTM*(*SDom*(s), *OT*(o))
- *Role permission* (*RP*): a subject's *RP* on an object is determined as follow: *rp*(s, o)={(o, p)|(o, p)$\in$*Rolecap*(*SR*(s))}

A subject's *Final permissions* on an object is determined as: *fp*(s, o)=(*mlp*(s, o) $\cap$*dp*(s, o)) $\cup$ *rp*(s, o).

3 Examples of Policy Configuration

3.1 Trusted Subjects' Permission Confinement

We take the information interaction between user process and operating system as an example on the information flow from lower integrity level to higher integrity level. As shown in Figure 3. User data is in lower integrity level, and operating system data in higher integrity level. In order to satisfy the needs of system calls, User process is permitted to write to the buffer data space of the operating system, but no permission to the other data object of the OS. Similarly, operating system writes the data to the buffer object of the user process.

To enforce the policy described above, each process is assigned a security attribute {*role*, *domain*}, denoting the process's running role and running domain. And each data object is assigned a security attribute {(c, i), t}, denoting the object's confidentiality level c, integrity level i and type t, as shown in Figure 3. And *Rolecap*(usr_r)={(w, kerbuffer)}, role *usr_r* has *write* permission to the *kerbuffer* object, its security label is (0,1),

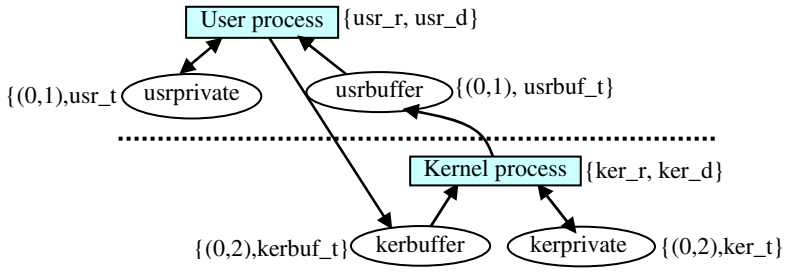

Fig. 3. Information interaction between user process and operating system

Table 1. *DTM*

	ker_t	Kerbuf_t	usr_t	usrbuf_t
ker_d	*r,w*	*r*		*w*
usr_d			*r,w*	*r*

and its authorized domains set is {usr_d}. *Rolecap*(ker_r)= Φ, ker_r has no role permission, its security label is (0,2), and its authorized domains set is {ker_d}. The *DTM* and *DDI* between the domains and types are shown in Table 3.1.

We isolate the operating system data and user data from each other by dividing them into different integrity level. The *usrbuffer* and *usrprivate* data objects which are in the same integrity level are divided into different types, therefore User process can have different fine-grained permissions to these two objects. According to the MLS policy, User process has no write permission to the *kerbuffer* objects for its integrity level is lower than the object. However, this write permission is necessary for getting job done. So, we assign write permission on object *kerbuffer* to the role *usr_r*, this permission is independent of MLS and Domain permissions. In this way, the function of system call is achieved without giving too much permission to the User process to bring potential security flaws to system.

3.2 Channel Control

We design a simplified firewall system to demonstrate the use of MPVSM in configuring channel control policies. The firewall is shown in Figure 4. The security policy of the firewall is that all information flow from outside network to inside network or in reverse direction must be checked by the access control component. It can be described as follow: information is only allowed to flow from the Outside to the Inside or in reverse direction via the Access Control. It can not be flowed directly between them. Besides, all components are permitted to read the Config without modifying it. And all components can append information to the Log, but can not read it.

Our configurations to enforce this policy are shown in Figure 3.2. The *DTM* and *DDI* between domains and types are shown in Table 3.2. And *Rolecap*(fw_r) =Φ, role

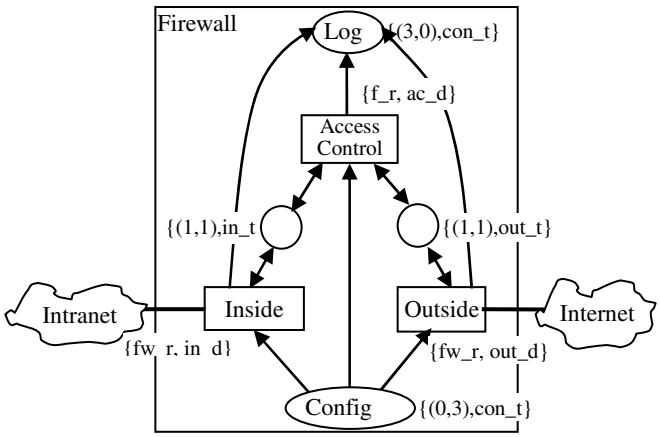

Fig. 4. Policy configuration of the Firewall

Table 2. The DTM and DDI

	in_t	out_t	con_t	in_d	out_d	ac_d
in_d	*r,w*		*r,a*			
out_d		*r,w*	*r,a*			
ac_d	*r,w*	*r,w*	*r,a*			

fw_r has no *Role permissions*. The security label of the *fw_r* is (1,1), and its authorized domains are {ac_d, in_d, out_d}, there is no *transfer* permission between any two of these three domains.

We upgraded the confidentiality level of the Log to make it unreadable to the components, upgraded the integrity level of the Config to make it unmodifiable by components. Then we divided the subjects of the same security level to different domains, and objects of the same security level to different types. By controlling the fine-grained permissions between the domains to types, information channel control policy between the inside and outside network is enforced.

3.3 Enforcing Multiple Security Policies

Through different configuring ways, Multi-Level security model, DTE and RBAC can be enforced separately in the MPVSM, and multi-policy views between different user groups can be enforced too.

3.3.1 Enforcing Multi-level Security model

The way configuring MPVSM to enforce Multi-Level Security model, which based on both confidentiality and integrity lattices, is described as following:

1. $|R|=|SL|$, number of roles in the system is the same as the number of the security labels. Each role corresponds to one security label.
2. $D=\{gen\_d\}$, $T=\{gen\_t\}$, there are only one domain and one type in the system. $RD=\{(r, gen\_d)|r\in \mathrm{R}\}$, indicates that all roles' authorized domain is *gen_d*. The type of all objects is *gen_t*: $OT=\{(o, gen\_t)|\ o\in O\}$.
3. $DTM=\{(d, t, p)|d\in D, t\in T, p\in P\}$, which indicates domain *gen_d* have all domain permissions to type *gen_t*.
4. *Rolecap*(r:*R*)=Φ, no role permission is authorized to every role.

3.3.2 Enforcing DTE

1. $R=\{gen\_r\}$, only one role in system. $UA=\{(u, gen\_r)|u\in U\}$, role *gen_r* is assigned to every users.
2. $RD=\{(gen\_r, d)|d\in D\}$, indicates that all domains in system are authorized to the role *gen_r*.
3. $SL=\{(Only\_C,Only\_I)\}$, only one security label in the system, therefore: $RL=\{(r, Only\_C, Only\_I))|r\in R\}$.
4. *Rolecap*(r:*R*)= Φ.

3.3.3 Enforcing RBAC

1. $D=\{gen\_d\}$, $T=\{gen\_t\}$, only one domain and one type in system. $RD=\{(r, gen\_d)|\ r\in R\}$, *gen_d* is authorized to every role in system. The type of all objects is *gen_t*, $OT=\{(\mathrm{o}, gen\_t)|\ \mathrm{o}\in O\}$.
2. $DTM(gen\_d, gen\_t)=\Phi$, denotes subjects in *gen_d* domain have no domain permissions to objects in type *gen_t*.
3. $SL=\{(Only\_C, Only\_I)\}$, only one security label. $RL=\{(r,(Only\_C,Only\_I))\ |r\in R\}$.

3.3.4 Enforcing Multiple Model Views

Assume all users in the system can be divided into three groups: *Grpa*, *Grpb* and *Grpc*. Now we may hope that users in each group can perceive different access control model views. For instance, users in *Grpa* think that the security model enforced in system is MLS, users in *Grpb* think that is RBAC and users in *Grpc* think that is DTE. The configuring method that enforces this multi-model views in one system is given as below.

1. $U=Grpa\cup Grpb\cup Grpc$, the three sets are disjointed each other.
2. $R=\ mls\_rs\cup rbac\_rs\cup\{dte\_r\}$. *mts_rs* is the roles set corresponding to MLS model. *rbac_rs* is the roles set corresponding to RBAC model. And *dte_r* is the role corresponding to DTE model.
3. $D=\ \{mls\_d\}\cup\{rbac\_d\}\cup dte\_ds$.
4. $(u,\ r)\in UA\wedge(u,\ r')\notin UA$, where $u\in Grpa$, $r\in mls\_rs$, $r'\notin mls\_rs$, the roles in *mls_rs* are only permitted to be assigned to users in *Grpa*. $(u,\ r)\in UA\wedge(u,\ r')\notin UA$, where $u\in Grpb$, $r\in rbac\_rs$, $r'\notin rbac\_rs$, the roles in *rbac_rs* can only be assigned to users in *Grpb*. Similarly, $(u,\ dte\_r)\in UA\wedge(u,\ r)\notin UA$, where $u\in Grpc$, $r\neq dte\_r$, every user in *Grpc* is assigned the only role *dte_r*.
5. $|mls\_rs|=|SL|$, number of roles in set *mls_rs* is the same as the number of security labels in system. Each role in *mls_rs* corresponds to one security label. $MLS\_rule(Ssl(r), tsl)=M$, $r\in rbac\_rs$, $tsl\in SL$, have all of possible *MLS permis-*

sions to other subjects or objects. *MLS_rule*(*Ssl*(dte_r), *tsl*)=*M*, *tsl*$\in$*SL*, role *dte_r*'s *MLS permissions* to other subjects or objects include all of possible permissions too.

6. The security label of the roles in *rbac_rs* is the lowest level label of the system, that is: $\forall$ r$\in$rbac_rs, *RL*(r)=(cs, is), $\forall$ c$\in$*C*, cs$\leqslant$c and $\forall$ i$\in$*I*, is$\leqslant$i. the security label of *dte_r* is the highese level label of the system, that is: *RL*(dte_r)=(cs, is), $\forall$ c$\in$*C*, cs$\geqslant$c and $\forall$ i$\in$*I*, is$\geqslant$i.
7. (*r*, *mls_d*)$\in$*RD*$\wedge$(*r*,*d*)$\notin$ *RD*, where *r*$\in$*mls_rs*, *d*$\neq$*mls_d*, every roles in *mls_rs* is authorized the only domain *mls_d*. (*dte_r*, *d*) $\in$*RD*$\wedge$(*r'*, *d*)$\notin$ *RD*, where *r'*$\neq$*dte_r*, *d*$\in$*dte_ds*, all domains in *dte_ds* are authorized to role *dte_r*. Simliarly, (*r*,*rbac_d*) $\in$*RD*$\wedge$(*r*,*d*)$\notin$ *RD*, where *r*$\in$*rbac_rs*, *d*$\neq$*rbac_d*, every role in *rbac_rs* is authorized the only domain *rbac_d*.
8. (*mls_d*,*t*,*p*)$\in$*DTM*,*t*$\in$*T*, *p*$\in$*P*. *DDI*(*mls_d*, *d*)=Φ, *d*$\in$*D*, subjects in domain *mls_d* can not transfer to other domains.
9. (*rbac_d*, *t*, *p*)$\in$*DTM*, *t*$\in$*T*, *p*$\in$*P*, the subjects in domain *rbac_d* have all *domain permission* to all types' objects in system. *DDI*(*rbac_d*, *d*)=Φ, *d*$\in$*D*, subjects in domain *rbac_d* can not transfer to other domains.
10. For every *r*$\in$*mls_rs*$\cup$\{*dte_r*\} and c$\in$*CAP*, (*r*, *c*)$\notin$ *RCAP*, there is no role permission authorized to roles in set *mls_rs* and the role *dte_r*.

4 The Related Works

Bell-LaPadula [1] (BLP) model mainly emphasizes the protection of confidentiality. It is able to limit flow of information and unauthorized information leakage. However, it does not care about the integrity, which is also important [2,7,11]. Besides, BLP is weak in channel control of information flow [22]. Biba Integrity Model [6] is the mathematical dual of BLP, intending to protect the integrity in system.

Type enforcement is a table-oriented mandatory access control policy for confining applications and restricting information flows. DTE [9] is an enhanced version of type enforcement designed to provide needed simplicity and compatibility. Role-based access control [14] provides a valuable level of abstraction to promote security administration at a business level.

The Flask [10] security architecture emphasizes diverse security policies support. However, it applies only MAC to the Fluke Microkernel. It provides the mechanisms for diverse policies without giving how to enforce multiple policies in system.

One of the earliest MAC mechanisms in operating system is Lattices [1, 20]. For instance, LOMAC [13] enforces Biba integrity. CMW [8] can dynamically relabel the current object for increased flexibility.

Recently, Asbestos [17] provides labeling and isolation mechanisms that can support applications to express a wide range of policies and make MAC more practical. KernelSec [18] aims at improving the effectiveness of the authorization model and the security policies that can be implemented.

In capability-based confinement systems, KeyKOS [21] achieved military-grade security by isolating processed into compartments and interposing reference monitors to control the use of capabilities. EROS [19] later successful realized this principles on

the modern hardware. And the Coyotes kernel [5] mainly explores use of software verification techniques to achieve higher confidence in the correctness and security of the kernel.

Mandatory access control can also be achieved with unmodified traditional operating system through virtual machines [16, 4].

5 Conclusions

The Compositional Multiple Policy Security Model is presented, which is based on the MLS model, combining the domain and type attributes to the model, to eliminate the limitations of MLS models. It has enforced expression power in channel control policies, and made permission management more fine-grained and enhanced the ability of confining the permission of the trusted subjects. MPVSM is also able to enforce multiple policy views in operating system in a flexible way.

References

1. Bell, D., La Padula, L.: Secure Computer Systems: Mathematical Foundations. Technical Report MTR-2547, vol. I, MITRE Corporation (1975)
2. Amoroso, E., Nguyen, T., Weiss, J., et al.: Towards an Approach to Measuring Software Trust. In: 1991 IEEE Symposium on Research in Security and Privacy, pp. 198–218 (1991)
3. Organick, E.: The MULTICS System: An Examination of Its Structure. MIT Press, Cambridge (1972)
4. Karger, P.A., Zurko, M.E., Bonin, D.W., et al.: A VMM security kernel for the VAX architecture. In: 1990 IEEE Symposium on Security and Privacy, pp. 2–19 (1990)
5. Shapiro, J., Doerrie, M.S., Northup, E., et al.: Towards a Verified, General-Purpose Operating System Kernel. In: 1st NICTA Workshop on Operating System Verification (2004)
6. Biba, K.: Integrity Considerations for Secure Computer Systems. Technical Report MTR-3153, MITRE Corporation (1977)
7. Eswaran, K., Chamberlin, D.: Functional Specifications of Subsystem for Database Integrity. In: The International Conference on Very Large Data Bases (1975)
8. Berger, J.L., Picciotto, J., Woodward, J.P.L., Cummings, P.T.: Compartmented mode workstation: Prototype highlights. IEEE Transactions on Software Engineering, Special Section on Security and Privacy 16, 608–618 (1990)
9. Badger, L., Sterne, D.F., Sherman, D.L., et al.: A Domain and Type Enforcement UNIX Prototype. In: 5th USENIX UNIX Security Symposium (1995)
10. Spencer, R., Smalley, S., Hibler, M., et al.: The Flask Security Architecture: System Support for Diverse Security Policies. In: 8th USENIX Security Symposium, pp. 123–139 (1999)
11. Lipner, S.: Non-Discretionary Controls for Commercial Applications. In: 1982 Symposium on Privacy and Security (1982)
12. Osborn, S., Sandhu, R., Munawer, Q.: Configuring Role-based Access Control to Enforce Mandatory and Discretionary Access Control Policies. ACM Transactions on Information and System Security 3, 85–106 (2000)
13. Fraser, T.: LOMAC–low water-mark mandatory access control for Linux. In: 9th USENIX Security Symposium (1999)

14. Sandhu, R., Coyne, E., Feinstein, H., Youman, C.: Role-Based Access Control. IEEE Computer 29 (1996)
15. Loscocco, P., Smalley, S.: Meeting critical security objectives with security-enhanced linux. In: Ottawa Linux Symposium 2001 (2001)
16. Goldberg, R.P.: Architecture of virtual machines. In: AFIPS National Computer Conference, vol. 42, pp. 309–318 (1973)
17. Efstathopoulos, P., Krohn, M., VanDeBogart, S., et al.: Labels and Event Processes in the Asbestos Operating System. In: 20th Symposium on Operating Systems Principles (2005)
18. Radhakrishnan, M., Solworth, J.A.: Application Support in the Operating System Kernel. In: ACM Symposium on Information, Computer and Communications Security (2006)
19. Shapiro, J.S., Smith, J.M., Farber, D.J.: EROS: A Fast Capability System. In: 17th ACM symposium on Operating systems principles (1999)
20. Saltzer, J.H., Schroeder, M.D.: The protection of information in computer system. Proceedings of the IEEE 63, 1278–1308 (1975)
21. Key Logic. The KeyKOS/KeySAFE System Design (1989), http://www.agorics.com/Library/KeyKos/ keysafe/Keysafe.html
22. Rushby, J.: Noninterference, Transitivity, and Channel-Control Security Policies. Technical Report CSL-92-02, Computer Science Lab, SRI International (1992)