

## Short Paper

---

# Managing User Associations and the Factors of Laws and Cultures in an Information Flow Control Model for Object-Oriented Systems

SHIH-CHIEN CHOU

*Department of Computer Science and Information Engineering  
National Dong Hwa University  
Hualien, 974 Taiwan  
E-mail: sczhou@mail.ndhu.edu.tw*

Users who play roles in an executing application may be in different security levels. In the application, information in high security levels should not be leaked to users in low security levels. To prevent information leakage, information flow control models can be used. We developed a series of models for the control, in which new features are added to newer models. This paper presents the newest model we developed, which offers the following two new functions: (1) correcting permissions invalidated by user association change and (2) adapting to different laws and cultures in different locations.

**Keywords:** information flow control, security, role-based access control (RBAC), prevent information leakage, law and culture management

## 1. INTRODUCTION

Information security issues such as network security, authentication, and access control have been recognized as important for modern applications. We developed models to prevent leaking information managed by an application when the application is being executed. This prevention is a sub-issue of access control. Other security issues are not with the scope of this paper.

Preventing information leakage is essential because low security level users may obtain high security level information when the *users* play *roles* in an executing application. For example, when a variable is output and the variable can be accessed by the role “*r1*”, then the user playing the role “*r1*” can obtain the output information. If the security level of the output information is higher than that of the user, information leakage occurs. To prevent the leakage, *information flow control* models can be used [1]. We developed information flow control models in the past years. The earliest model we developed is OORBAC (object-oriented role-based access control) [2]. It offers the features of: (a) controlling information flows among objects, (b) controlling method invocation through argument sensitivity, (c) allowing purpose-oriented method invocation and prevent leakage within an object, (d) controlling write access precisely, and (e) avoiding Trojan

---

Received August 22, 2002; revised September 14, 2006; accepted January 23, 2007.  
Communicated by Tsan-sheng Hsu.

horses (*i.e.*, preventing indirect information leakage). We then added new features and developed new models. For example, we added flexibility to OORBAC by allowing non-secure but harmless information flows and blocking secure but harmful information flows [3]. As another example, we added functions to control foreign objects [4], to manage dynamic role change [5], to control information flows *among* communicating applications [6], and to manage role relationships [7]. We are now adding the following new functions to the models we developed: (1) correcting permissions invalidated by user association change and (2) adapting to different laws and cultures in different locations. The former function is similar to that of managing role relationships [7]. Nevertheless, user relationships (user associations) instead of role relationships are used in the new function because different users may play the same roles. We claim that user relationship is more precise. For example, suppose a manager in a company can give a special discount to his customer friends. Suppose both John and Joe play the manager role and Mary plays the customer role. Also suppose that Mary is John's friend but not Joe's. If role relationship is used, Joe may incorrectly give the special discount to Mary because both John and Joe play the manager role. Below we explain the importance of the newly added functions.

- (1) Correcting permissions invalidated by user association change. In general, *associations* (relationships) may exist among users. User associations may change and the change may cause user permission change. For example, suppose friends can read one another's general information such as phone number. Also suppose that Joe and Mary are not friends initially. Then, Mary and Joe cannot read each other's general information. If Joe and Mary become friends a certain time later, Mary and Joe can read each other's general information this time. The above example reveals that user association change causes user permission change. Our research reveals that user association change may *invalidate existing user permissions*. Since the invalidation is related to preventing indirect information leakage, we first describe the leakage. Indirect information leakage occurs when information is leaked via the third ones. In general, indirect information leakage can be prevented by a join operation [8-11]. For example, suppose the information *inf1* can be read by one set of users and *inf2* can be read by users in another set. Then, the join operation allows the information derived from *inf1* and *inf2* to be read by users in the intersection of the two sets.

Having described indirect information prevention, we use an example to explain the necessity of correcting user permissions invalidated by user association change. Suppose Joe and Mary are initially not friends. Then, Joe cannot read Mary's general information. Suppose at this time, the information *phoneNoSet* is derived from Mary's phone number and others' that can be read by Joe. Then, according to the join operation mentioned above, Joe is not allowed to read *phoneNoSet*. Suppose Joe and Mary become friends a certain time later. Then, Joe can read Mary's phone number this time. Making friends between Joe and Mary results in user association change. After the change, Joe should be allowed to read *phoneNoSet* produced before because *phoneNoSet* is derived from Mary's phone number (which can be read by Joe this time) and others' numbers that can be read by Joe. Allowing Joe to read *phoneNoSet* invalidates the previous join operation because the previous operation disallowed Joe to read *phoneNoSet*. The invalidation re-

quires one or more previous join operations to be corrected when user associations change. The correction is a join redoing process. In the process, the current user associations should be used as a reference because user associations may affect user permissions.

- (2) Adapting to different laws and cultures in different locations. The adaptation is necessary because an application executing in a location should obey the location's laws and cultures (we use the term "location" instead of "country" because different locations in the same country may possess different laws and cultures). For example, advertisements containing Bikini girls would not cause trouble in USA because the laws and cultures of USA generally allow the advertisements. Therefore, a server application can freely send advertisements with Bikini girls to a client application located in USA. Nevertheless, the same advertisements will cause trouble in an Islamic country because both laws and cultures generally do not allow Bikini girls to be publicly displayed in an Islamic country. Therefore, a server application is generally not allowed to send advertisements with Bikini girls to a client application located in an Islamic country.

The above example exhibits that both laws and cultures disallow certain information flows. There may be information flows that are allowed in some cases but disallowed in others. For example, selling alcohol beverage to users under the age of 18 contradicts the laws of Taiwan. Therefore, an e-commerce transaction that sells alcohol beverage from a server application to a client application is disallowed if the client application is operated by a user under the age of 18 and the client is located in Taiwan. Nevertheless, if the user is over or equal to the age of 18, the transaction is allowed. As another example, displaying an advertisement containing porn stars to users under the age of 20 destroys the good custom of Taiwan (*i.e.*, the display disobeys the culture of Taiwan). Therefore, a server application should not send the advertisement to a client application if the client application is operated by a user under the age of 20 and the client is located in Taiwan. Nevertheless, if the user is over or equal to the age of 20, the sending is allowed. Since different locations possess different laws and exhibit different good custom (*i.e.*, possess different cultures), an information flow control model should adapt to different laws and cultures in different locations. Note that when adapting to different laws and cultures, users should provide their attributes such as ages and locations. We suppose the information is always correct because authentication is not within the scope of our research.

We developed a model that offers the functions. We named the model *YAIFCM* (yet another information flow control model) because we believe that more new models will be developed in the future. *YAIFCM* inherits all the useful features of our previous models. This paper focuses on presenting the above mentioned new functions and ignores other features.

## 2. RELATED WORK

We cannot identify a model that offers the functions mentioned in section 1. Nevertheless, we still survey important information flow control models and point out their primary features.

Traditional access control is achieved by access control matrix (ACM) [12]. A subject can access an object if the required access right is recorded in an ACM. The ACM approach disallows subject permissions to be changed dynamically. Nevertheless, dynamically changing permissions is necessary to prevent indirect information leakage for an information flow control model. The DACM (dynamic access control matrix) approach [13] dynamically grants access rights to subjects under different situations. Therefore, it is an improvement of ACM.

The model in [14] uses access control lists of objects to compute ACLs of executions (which are composed of one or more methods). A message filter is used to filter out possibly non-secure information flows. Interactions among executions are categorized into five modes. Different modes result in different access control policies, which loosen the restriction of MAC (mandatory access control). Flexibility is added by allowing exceptions during or after method execution [15, 16]. More flexibility is added using versions [17].

MAC [18-20] is useful in access control. An important milestone of MAC is that proposed by Bell&LaPadula [18], which categorizes the security levels of objects and subjects. Access control in the model follows the “no read up” and “no write down” rules [18]. Bell&LaPadula’s model was generalized into the lattice model [19, 20]. Information can flow from an object to a subject if a “can flow” relationship exists among them.

The decentralized label approach [8-11] marks the security levels of variables using labels. A label is composed of one or more policies, which should be simultaneously obeyed. A policy in a label is composed of an owner and zero or more readers that are allowed to read the data. Both owners and readers are principals, which may be users, group of users, and so on. Principals are grouped into hierarchies using the act-for relationships. A principal possesses all access rights of the principals it acts for. Join operator is used to prevent indirect information leakage.

The purpose-oriented model [21, 22] proposes that invoking a method may be allowed for some methods but disallowed for others, even when the invokers belong to the same object. It uses existing objects to create a flow graph, from which non-secure information flows can be identified. The consideration of purpose orientation is correct because different methods may be in different security level [23] and therefore should be controlled independently.

Role-based access control (RBAC) models [24, 25] can also be used in access control. In the general cases, RBAC fails to offer the necessary features for information flow control because it was not developed for the control. Nevertheless, RBAC can be customized for the control. For example, the model in [26] applies RBAC for information flow control. It classifies object methods and derives a flow graph from method invocations. From the graph, non-secure information flows can be identified.

### 3. YAIFCM

Since *user association* is important in YAIFCM, we declare it first. A user association is a user relationship. Nevertheless, not every user relationship is regarded as a user association. We define a user association as a user relationship that will affect user permissions. We use the relationship *assignment* between patients and doctors to describe

user association. Suppose a patient is assigned to one or more doctors. Also suppose that a doctor can read and change the case history of a patient assigned to him/her, and can read but not change the case history of a patient not assigned to him/her. In this example, the user relationship *assignment* between doctors and patients affect the permissions of a doctor. Therefore, *assignment* should be regarded as a user association.

YAIFCM should be embedded in an application to ensure secure information access. An application embedded with YAIFCM is defined as “(*App*, *YAIFCM*)”, in which *App* is an application and *YAIFCM* is the model to prevent information leakage. We give a definition to YAIFCM as follows:

**Definition 1**  $YAIFCM = (USR, RLE, UAS, PER, URA, RPA, JH, LawPer, CulPer)$ , in which

- a) *USR* is the set of users. Users play roles.
- b) *RLE* is the set of roles. YAIFCM regards an object as a role.
- c) *UAS* is the set of user associations. A user association is defined as “(*ua\_name*, *usr\_set*)”, in which *ua\_name* is the user association name and *usr\_set* is a non-empty user set. In the definition, users in *usr\_set* are within the user association *ua\_name*. For example, suppose *ua\_name* is *friend* and *usr\_set* is “{*John*, *Mary*}”. Then, John and Mary are friends.
- d) *PER* is a set of permissions. A permission is a read or write access right on a piece of information. A permission *per* is defined as follows:

$per = (inf, right, ua)$ , in which

- (1) *inf* is the information to be accessed.
- (2) *right* may be *R* or *W*, which respectively grant read and write access rights.
- (3) *ua* is a user association (*i.e.*,  $ua \in UAS$ ).

Permissions granted to a user can be identified from the user-role and role-permission assignments respectively described in items *e* and *f* below. A user  $usr_1$  is granted the permission  $per_1$  if the following conditions are true: (1) a user-role assignment between  $usr_1$  and the role  $rle_1$  exists, (2) a role-permission assignment between  $rle_1$  and  $per_1$  exists, and (3)  $usr_1$  is within the user set of  $per_1$ 's user association.

- e) *URA* is a set of user-role assignments, which is defined as “ $USR \rightarrow 2^{RLE}$ ”.
- f) *RPA* is a set of role-permission assignments, which is defined as “ $RLE \rightarrow 2^{PER}$ ”.
- g) *JH* records join history. It facilitates correcting permissions invalidated by user association change, as described in section 3.2.
- h) *LawPer* is a set of law permissions. A permission granting a right to access the information  $inf_i$  under control of the law  $law_i$  is defined as: “( $inf_i$ ,  $law_i$ , *right*, *condition*)”. The component *right* may be *R* or *W* that respectively grant read or write access rights. The component *condition* is the condition that should be true for the right to be granted. A condition is an expression consisting of user attribute values and user roles. For example, “( $Location == \text{“Taiwan”}$ ) && ( $Age \geq 18$ )” is a condition. Under the definition of *LawPer*, the law permission of buying cigarettes in Taiwan (see section 1) can be defined as: “( $BuyCigarettes$ , 100, *R*, ( $Location == \text{“Taiwan”}$ ) && ( $Age \geq 18$ ))”. In the permission, we suppose that the law number restricting the buying of

cigarettes is 100, which implies that every law related to information flow control should be numbered.

- i) *CulPer* is a set of culture permissions. A permission granting a right to access the information  $inf_i$  under control of the culture  $cul_i$  is defined as: “( $inf_i, cul_i, right, condition$ )”. The definition of *right* and *condition* are the same as those in a law permission. Under the definition of *CulPer*, the culture permission of observing porn start advertisements in Taiwan (see section 1) can be defined as: “(*ObservePornStarAdv*, 50, *R*, (*Location* == “Taiwan”) && (*Age* >= 20))”. In the permission, we suppose that the culture number restricting the observing of porn start advertisement is 50, which implies that every culture related to information flow control should be numbered.

### 3.1 Secure Information Access in YAIFCM

To ensure secure access of information, the two *secure access conditions* defined below should be fulfilled. The first condition controls read access and the second controls write access. To define the first secure access condition, we make the following assumptions:

- a) The user  $usr$  attempts to read the information  $inf$ .
- b) Roles played by the user  $usr$  constitute the role set  $RLE_{usr}$ . The role set can be identified from user-role assignments.
- c) Read permissions on  $inf$  assigned to the roles in  $RLE_{usr}$  constitute the read permission set  $RPER_{RLE_{usr}}$ . The read permissions can be identified from role-permission assignments.
- d) The user  $usr$  possesses the read permissions in the set  $RPER_{usr}$ , which is a subset of  $RPER_{RLE_{usr}}$ . A read permission “( $inf, R, ua$ )” in  $RPER_{RLE_{usr}}$  is an element of  $RPER_{usr}$  if  $usr$  is within the user set of  $ua$  (see items *c* and *d* in Definition 1 for the definitions of a user association and a permission, respectively).

According to the above assumptions, the user  $usr$  can read the information  $inf$  if the following condition is true.

**First secure access condition:**  $(RPER_{usr} \neq \emptyset) \wedge (\forall \text{ law permission } “(inf, law_i, R, condition)” , condition \text{ is true}) \wedge (\forall \text{ culture permission } “(inf, cul_j, R, condition)” , condition \text{ is true})$ .

The first secure access condition requires that the user  $usr$  should possess permission to read  $inf$ . Moreover, the condition requires that every law and culture restriction regarding the read access should be obeyed. To define the second secure access condition, we make the following assumptions, which are similar to those for the first secure access condition.

- a) The user  $usr$  attempts to write the information  $inf$ .
- b) Roles played by the user  $usr$  constitute the role set  $RLE_{usr}$ .
- c) Write permissions on  $inf$  assigned to the roles in  $RLE_{usr}$  constitute the write permission set  $WPER_{RLE_{usr}}$ .

- d) The user  $usr$  possesses the write permissions in the set  $WPER_{usr}$ , which is a subset of  $WPER_{RLE_{usr}}$ . A write permission “ $(inf, W, ua)$ ” in  $WPER_{RLE_{usr}}$  is an element of  $WPER_{usr}$  if  $usr$  is within the user set of  $ua$ .

According to the above assumptions, the user  $usr$  can write the information  $inf$  when the following condition is true.

**Second secure access condition:**  $(WPER_{usr} \neq \phi) \wedge (\forall \text{ law permission “}(inf, law_i, W, condition)\text{”}, condition \text{ is true}) \wedge (\forall \text{ culture permission “}(inf, cul_j, W, condition)\text{”}, condition \text{ is true})$ .

The second secure access condition requires that the user  $usr$  should possess permission to write  $inf$ . Moreover, the condition requires that every law and culture restriction regarding the write access should be obeyed. After finishing an information flow, permissions related to the information produced by the information flow should be changed to prevent indirect information leakage. Here we define a *permission related to a piece of information* as the permission that allows a role to access the information. For example, the permissions “ $(inf, R, ua_1)$ ” and “ $(inf, W, ua_2)$ ” are related to the information  $inf$ . YAIFCM uses a join operation [8-11] for the change. To define the join operation, we make the following assumptions:

- The information  $inf$  is derived from the information set “ $\{inf_i \mid inf_i \text{ is a piece of information and } i \text{ is between } 1 \text{ and } n\}$ ”.
- The read permission set related to the information  $inf_i$  is “ $\{(inf_i, R, ua) \mid (inf_i, R, ua) \text{ is a permission, and the permission is granted to a user } usr \text{ that plays the role } rle \text{ only when the permission } (inf_i, R, ua) \text{ is assigned to } rle \text{ and } usr \text{ is within the user set of } ua\}$ ”. We call the read permission set  $RPER_{inf_i}$ .
- The write permission set related to the information  $inf_i$  is “ $\{(inf_i, W, ua) \mid (inf_i, W, ua) \text{ is a permission, and the permission is granted to a user } usr \text{ that plays the role } rle \text{ only when the permission } (inf_i, W, ua) \text{ is assigned to } rle \text{ and } usr \text{ is within the user set of } ua\}$ ”. We call the write permission set  $WPER_{inf_i}$ .
- The user associations in  $RPER_{inf_i}$  constitutes the set “ $\{(Rua\_name_{i,j}, Rusr\_set_{i,j}) \mid j \text{ is between } 1 \text{ and } k \text{ in case that there are } k \text{ elements in } RPER_{inf_i}\}$ ”. We call the set  $RUA_{inf_i}$  (see item *c* of Definition 1 for the contents of a user association).
- The user associations in  $WPER_{inf_i}$  constitute the set “ $\{(Wua\_name_{i,j}, Wusr\_set_{i,j}) \mid j \text{ is between } 1 \text{ and } m \text{ in case that there are } m \text{ elements in } WPER_{inf_i}\}$ ”. We call the set  $WUA_{inf_i}$ .

**Definition 2** Suppose the read permission set related to  $inf$  is  $RPER_{inf}$  and the write permission set related to  $inf$  is  $WPER_{inf}$ . Then, the join operation changes  $RPER_{inf}$  and  $WPER_{inf}$  as follows:

$$RPER_{inf} = \{(inf, R, ua) \mid ua = (Rua\_name_{i,j}, \bigcap_{i=1}^n \bigcap_j Rusr\_set_{i,j}),$$

in which  $Rua\_name_{i,j}$  is a user association name appears in every  $RUA_{inf_i}$ ,  $(Rua\_name_{i,j}, Rusr\_set_{i,j})$  is an element of  $RUA_{inf_i}$ ,  $\bigcap_{i=1}^n \bigcap_j Rusr\_set_{i,j}$  is non-empty, and  $i$  is between 1 and  $n$ .

$$WPER_{inf} = \{(inf, W, ua) \mid ua = (Wua\_name_{i,j}, \bigcup_{i=1}^n \bigcup_j Wusr\_set_{i,j}),$$

in which  $Wua\_name_{i,j}$  is a user association name appears in at least one  $WUA_{inf}$ , ( $Wua\_name_{i,j}, Wusr\_set_{i,j}$ ) is an element of  $WUA_{inf}$ , and  $i$  is between 1 and  $n$ ).

The join operation limits readers to read the derived information. Therefore, join will not lower the information's security level. On the other hand, the operation trusts more writers because union instead of intersection is used. We use an example to explain the rationale. Suppose  $inf$  is derived from the information  $inf1$  and  $inf2$ . Also suppose that the user  $usr1$  can write  $inf1$  and the user  $usr2$  can write  $inf2$ . Under this assumption,  $usr1$  is a trusted source of  $inf1$  and  $usr2$  is a trusted source of  $inf2$ . Suppose the current value of  $inf1$  was written by  $usr1$  and that of  $inf2$  was written by  $usr2$ . When  $inf$  is derived from  $inf1$  and  $inf2$ ,  $usr1$  and  $usr2$  become data sources of  $inf$ . That is,  $usr1$  and  $usr2$  should be regarded as trusted sources of  $inf$ . The above example explains why union instead of intersection is used to change write permissions related to  $inf$ .

### 3.2 Redoing Join Operations

As described in section 1, permissions may be invalidated by user association change. Therefore, if user associations change before accessing a piece of information, the invalidated permissions should be corrected before checking the secure access conditions. Otherwise, information leakage may occur. The correction is a join redoing process as described below.

Suppose the information to be accessed is derived from the information in the set  $WFI_1$ . Since a piece of information may be derived from other information, we suppose that the information deriving the information in  $WFI_1$  constitute the set  $WFI_2$ , the information deriving the information in  $WFI_2$  constitute the set  $WFI_3$ , and so on. The above derivation process results in ripple effects. The effects end when  $WFI_m$  is empty. We use Fig. 1 as an example to describe the effects. Suppose  $d\_inf$  is the information to be accessed. Then,  $WFI_1$  is “{ $inf1, inf2$ }”,  $WFI_2$  is “{ $inf3$ }”,  $WFI_3$  is “{ $inf4, inf5$ }”,  $WFI_4$  is “{ $inf6$ }”, and  $WFI_5$  is empty. The ripple effect ended at  $WFI_5$ .

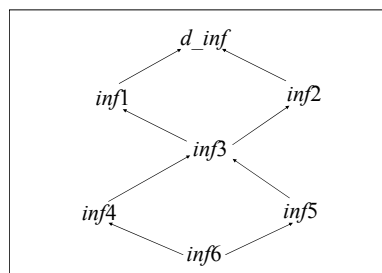


Fig. 1. Derivation relationships among information.

Suppose  $UWFI$  is the set “ $\bigcup_{i=1}^n WFI_i \cup \{info\}$ ”, in which  $info$  is the information to be accessed and the ripple effects end at  $WFI_{n+1}$ . Also suppose that the earliest time the

information  $wfi_i$  being a derived information is  $t_{wfi_i}$ , in which  $wfi_i \in UWFI$ . For each information  $wfi_i$  in  $UWFI$ , every join operation in which  $wfi_i$  is a derived information should be redone from  $t_{wfi_i}$  down to the current time. When redoing join operations, the current user associations should be used as a reference because permissions should be corrected under the current user associations (see section 1). Below we define the component  $JH$  in Definition 1 and then offer an algorithm to redo join operations.

**Definition 3** An element  $jh$  of  $JH$  is defined below:

$jh = (t, d\_wfi, \{(wfi, PER_{wfi})\}, tag)$ , in which

- (1)  $t$  is the time that a join operation occurs.
- (2)  $d\_wfi$  is the derived information.
- (3)  $\{(wfi, PER_{wfi})\}$  is the set of information that derive  $d\_wfi$  and the permissions related to the information.
- (4) If  $tag$  is true,  $t$  is the earliest time that  $d\_wfi$  is a derived information.

According to Definitions 3 and the description in this subsection, we conclude the redoing of join operations using Algorithm 1.

**Algorithm 1** Redoing Join Operations

1. Input data:
  - 1.1.  $info$ : the information to be accessed
  - 1.2.  $WFI_1 = \{wfi \mid wfi \text{ is a piece of information that derives } info\}$
2. Algorithm:
  - 2.1. Backtrack  $JH$  to identify  $WFI_1$  through  $WFI_n$  following the procedure described in the second paragraph of this subsection.
  - 2.2. Let  $UWFI$  be the set " $\cup_{i=1}^n WFI_i \cup \{info\}$ ".
  - 2.3. For each  $wfi_i \in UWFI$ , do
    - 2.3.1. Backtrack  $JH$  to identify the earliest time  $t_{wfi_i}$  in which  $wfi_i$  is a piece of derived information. The tags in  $JH$  (see Definition 3) facilitate the identification.
    - 2.3.2. From the time  $t_{wfi_i}$  down to the current time, mark every join operation in  $JH$  in which  $wfi_i$  is a derived information.
  - 2.4. End do
  - 2.5. Redo the marked join operations recorded in  $JH$  from the earliest time a join operation is marked down to the current time. During the redoing, the current user associations are used as a reference.

#### 4. PROOF OF CORRECTNESS

We prove that YAIFCM prevents information leakage and offers the two functions mentioned in section 1 as follows.

**Lemma 1** YAIFCM prevents information leakage.

**Proof:** To prove the lemma, we suppose that a user  $usr$  that has no permission to read the information  $inf$  can obtain the content of  $inf$ . Since  $usr$  has no permission to read  $inf$ ,  $usr$  will not appear in the user association  $ua$  of any permission “( $inf, R, ua$ )”. According to the assumption of the first secure access condition, the set  $RPER_{usr}$  will be empty. An empty  $RPER_{usr}$  set will cause the first secure access condition to be false, which disables the read access. This contradicts the assumption. The above proof can be applied to write access. We do not duplicate the proof.

The above proof shows that direct information leakage is prevented. We still have to prove that indirect information leakage is prevented. Indirect leakage occurs when a piece of information  $inf_2$  leaks the information  $inf_1$  to the user  $usr_1$  under the following assumption: (a)  $usr_1$  is initially allowed to read  $inf_2$ , (b)  $inf_2$  is derived from  $inf_1$ , and (c)  $usr_1$  is not allowed to read  $inf_1$ . Since  $usr_1$  is not allowed to read  $inf_1$ ,  $usr_1$  does not exist in any permission related to  $inf_1$ . That is, for each permission ( $inf_1, R, ua$ ),  $usr_1$  is not within  $ua$ 's user set. Suppose indirect information leakage occurs among  $inf_1$ ,  $inf_2$ , and  $usr_1$ . Without loss of generality, we assume that  $usr_1$  can read  $inf_2$  after  $inf_2$  is derived from  $inf_1$ . If this assumption is true, then there exists a permission ( $inf_2, R, ua$ ) in which  $usr_1$  is within  $ua$ 's user set. Nevertheless, the assumption will never be true because of the intersection operation in Definition 2.  $\square$

**Lemma 2** YAIFCM corrects the permissions invalidated by user association change (Algorithm 1 is correct).

**Proof:** We prove the correctness by induction.

- a) Assume that only one element  $info$  is within  $UWFI$ . Then,  $info$  never be a derived information. In this case, the permissions related to  $info$  are unchanged during application execution. Unchanged permissions are correct. The rationale is that permissions may be invalidated only when the following two conditions are simultaneously true: (a) one or more user associations change and (b) the permissions are changed by join operation. If permissions are unchanged, the second condition is false. Therefore, unchanged permissions are always correct.
- b) Assume Algorithm 1 is correct when there are  $(k - 1)$  elements in the set  $UWFI$ , in which  $(k - 1) > 1$ . Let's add a piece of information  $wfi_k$  to the original  $UWFI$  (we let  $NewUWFI = UWFI \cup \{wfi_k\}$ ). We prove that Algorithm 1 is correct for a  $k$ -element  $NewUWFI$  as follows. First, according to the assumption above, the permissions related to the information in  $NewUWFI$  excluding  $wfi_k$  are correct after join redoing. Second, the permissions related to  $wfi_k$  is correct because: (1) if the permissions have not been changed, the permissions are correct and (2) if the permissions has been changed by the join operation, lines 2.3 through 2.5 of Algorithm 1 corrects the permissions.  $\square$

**Lemma 3** YAIFCM adapts to different laws and cultures in different locations.

**Proof:** To prove the lemma, we assume that: (a) a user  $usr$  fails to pass the restriction of the law  $law_i$  when he/she intend to read the information  $inf$  and (b)  $usr$  can read or write  $inf$ . According to assumption a, the attributes and roles of  $usr$  at least cause the condition of one law permission “( $inf, law_i, R, condition$ )” to be false. In this case, neither the first

secure access condition nor the second one mentioned in section 3.1 will be true (which means that *usr* cannot read or write *inf*). This contradicts the assumption. The above proof can be applied to culture restrictions. We do not duplicate the proof.  $\square$

## 5. CONCLUSIONS

We developed a series of models for information flow control, in which new features are added to newer models. This paper presents the newest model named YAIFCM we developed, which offers the following important functions:

- (1) Correcting permissions invalidated by user association change. Permissions generated by previous join operations may be invalidated by user association change. We correct the invalidated permission using the history information of join operations.
- (2) Adapting to different laws and cultures in different locations. Users in different locations may be under the restrictions of different laws and/or cultures. We embed sub-conditions in the secure access conditions to ensure that the requirements of laws and cultures are obeyed.

A prototype system has been implemented. Nevertheless, since the runtime overhead is about 5 times the original application in average, we do not show the experiment result. We are trying to lower down the overhead currently.

## REFERENCES

1. A. Sabelfeld and A. C. Myers, "Language-based information-flow security," *IEEE Journal on Selected Areas in Communications*, Vol. 21, 2003, pp. 5-19.
2. S. C. Chou, "Embedding role-based access control model in object-oriented systems to protect privacy," *Journal of Systems and Software*, Vol. 71, 2004, pp. 143-161.
3. S. C. Chou, "Providing flexible access control to an information flow control model," *Journal of Systems and Software*, Vol. 73, 2004, pp. 425-439.
4. S. C. Chou, "Information flow control among objects: taking foreign objects into control," in *Proceedings of the 36th Hawaii International Conference on System Sciences*, 2003, pp. 335-344.
5. S. C. Chou, "An RBAC-based access control model for object-oriented systems offering dynamic aspect features," *IEICE Transactions on Information and Systems*, Vol. E88-D, 2005, pp. 2143-2147.
6. S. C. Chou, "An agent-based inter-application information flow control model," *Journal of Systems and Software*, Vol. 75, 2005, pp. 179-187.
7. S. C. Chou and Y. C. Chen, "Managing role relationships in an information flow control model," *Journal of Systems and Software*, Vol. 79, 2006, pp. 507-522.
8. A. C. Myers, "JFlow: practical mostly-static information flow control," in *Proceedings of the 26th ACM Symposiums on Principles of Programming Language*, 1999, pp. 228-241.
9. A. C. Myers and B. Liskov, "A decentralized model for information flow control," in *Proceedings of the 17th ACM Symposiums on Operating Systems Principles*, 1997,

- pp. 129-142.
10. A. Myers and B. Liskov, "Complete, safe information flow with decentralized labels," in *Proceedings of the 14th IEEE Symposiums on Security and Privacy*, 1998, pp. 186-197.
  11. A. Myers and B. Liskov, "Protecting privacy using the decentralized label model," *ACM Transactions on Software Engineering and Methodology*, Vol. 9, 2000, pp. 410-442.
  12. M. H. Harrison, W. L. Ruzzo, and J. D. Ullman, "Protection in operating systems," *Communications of the ACM*, Vol. 19, 1976, pp. 461-471.
  13. M. S. Olivier, R. P. van de Riet, and E. Gudes, "Specifying application-level security in workflow systems," in *Proceedings of the 9th International Workshop on Database and Expert Systems Applications*, 1998, pp. 346-351.
  14. P. Samarati, E. Bertino, A. Ciampichetti, and S. Jajodia, "Information flow control in object-oriented systems," *IEEE Transactions on Knowledge Data Engineering*, Vol. 9, 1997, pp. 524-538.
  15. E. Ferrari, P. Samarati, E. Bertino, and S. Jajodia, "Providing flexibility in information flow control for object-oriented systems," in *Proceedings of the 13th IEEE Symposiums on Security and Privacy*, 1997, pp. 130-140.
  16. E. Bertino, S. de C. di Vimercati, E. Ferrari, and P. Samarati, "Exception-based information flow control in object-oriented systems," *ACM Transactions on Information System Security*, Vol. 1, 1998, pp. 26-65.
  17. A. Maamir and A. Fellah, "Adding flexibility in information flow control for object-oriented systems using versions," *International Journal of Software Engineering and Knowledge Engineering*, Vol. 13, 2003, pp. 313-326.
  18. D. E. Bell and L. J. LaPadula, "Secure computer systems: unified exposition and multics interpretation," Technique Report No. MTR-2997 Rev. 1, MITRE Corporation, 1976, <http://csrc.nist.gov/publications/history/bell76.pdf>.
  19. D. E. Denning, "A lattice model of secure information flow," *Communications of the ACM*, Vol. 19, 1976, pp. 236-243.
  20. D. E. Denning and P. J. Denning, "Certification of program for secure information flow," *Communications of the ACM*, Vol. 20, 1977, pp. 504-513.
  21. M. Yasuda, T. Tachikawa, and M. Takizawa, "A purpose-oriented access control model," in *Proceedings of the 12th International Conference on Information Networking*, 1998, pp. 168-173.
  22. T. Tachikawa, M. Yasuda, and M. Takizawa, "A purposed-oriented access control model in object-based systems," *Transactions on Information Processing Society of Japan*, Vol. 38, 1997, pp. 2362-2369.
  23. V. Varadharajan and S. Black, "A multilevel security model for a distributed object-oriented system," in *Proceedings of the 6th IEEE Symposiums on Security and Privacy*, 1990, pp. 68-78.
  24. R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *IEEE Computer*, Vol. 29, 1996, pp. 38-47.
  25. D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed NIST standard for role-based access control," *ACM Transactions on Information and System Security*, Vol. 4, 2001, pp. 224-274.
  26. K. Izaki, K. Tanaka, and M. Takizawa, "Information flow control in role-based

model for distributed objects,” in *Proceedings of the 8th International Conference on Parallel and Distributed Systems*, 2001, pp. 363-370.

**Shih-Chien Chou (周世杰)** received a Ph.D. degree from the Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan. He is currently a professor in the Department of Computer Science and Information Engineering, National Dong Hwa University, Hualien, Taiwan. His research interests include software engineering, software reuse, and information flow control.