Asian Journal of Information Technology, 5 (2) : 145-154, 2006
© Medwell Online, 2006

Coordinating Heterogeneous Information Flow Control Models

Shih-Chien Chou
Department of Computer Science and Information Engineering,
National Dong Hwa University, Hualien 974, Taiwan

Abstract: A complicated software system can be decomposed into subsystems for different teams to develop, in which the teams may distribute geographically and need not belong to the same company. Many issues should be solved when developing software systems in this manner. For example, developing a software system that can prevent information leakage during the execution of the systems is an essential issue. The prevention can be achieved by embedding an information flow control model in the system. Since the teams that develop a software system need not belong to the same company, different teams may be familiar with different information flow control models. To embed information flow control model in a complicated software system decomposed into subsystems, the following approaches can be used: (a) embedding the same information flow control model in every subsystem and (b) allowing different information flow control models to be embedded in different subsystems. If the first approach is used, software developers unfamiliar with the assigned model may be unwilling to use the model. Moreover, if a software system is developed through reuse, the reused components may be embedded with different information flow control models. This causes difficulty in the reuse. We suggest that the second approach should be used and propose a model IFCMC (information flow control model coordinator) to coordinate heterogeneous information flow control models. This study proposes IFCMC and its evaluation.

Key words: Information security, prevent information leakage, information flow, information flow control, coordinate heterogeneous information flow control models

INTRODUCTION

To develop complicated software system, decomposing the system into subsystems and then assigning the subsystems to different software development teams is an acceptable approach. The development teams can distribute geographically and need not belong to the same company. In this development approach, many issues should be solved. This paper discusses information leakage prevention during the execution of a software system developed using the approach. The prevention corresponds to avoiding users in low security levels to access information in high security levels. For example, customers should be kept from accessing sensitive information that can only be accessed by managers. A model for the prevention is a language-based model[1] because the model should be embedded in a language to implement a secure software system.

Preventing information leakage within a software system can be achieved through *information flow control*. Many information flow control models have been developed[2-30]. An information flow control model can be embedded in a software system during software development. The model will prevent information leakage during the execution of the system. Information flow control models are based on the generic concept security levels because the models prevent users in *low security levels* from accessing information in *high security levels*. The generic concept can be implemented using various mechanisms such as access control matrixes[31], labels[2-5], access control lists[11] and lattices[8-9]. In other words, the generic concept security levels is the base of an information flow control model no matter what mechanism implements the concept.

Having clarified the concept of security levels, we describe the concepts of *information flow*, *information leakage* and *information flow control* used in this paper. Describing the concepts is necessary because different models may use the same terms for different meanings.

a. An information flow occurs when the content of a variable or the result of a computation is sent to a variable, an output device, an output file, or communication media among software systems (e.g., shared variables, files and network).
b. Information leakage refers to leaking information to a user that is not allowed to access the information. Here a user may be a person or another system. Information leakage may happen when information

that cannot be accessed by a user is sent to a variable, an output device, an output file, or communication media that can be accessed by the user.

c. Information flow control refers to controlling the flows of information to prevent information leakage. For example, suppose the variable a can be accessed by every customer of a bank and the variable b contains the password of a customer that can only be accessed by the bank's president. Then, blocking the statement a = b; is a kind of information flow control.

As mentioned above, many information flow control models have been developed. Existing models offer more or less attractive features such as purpose-oriented method invocation[15-18], declassification[2-5, 14] and avoiding Trojan horses[2-5]. Nevertheless, the ultimate goal of an information flow control model is to prevent information leakage during the execution of a software system. Let's continue the discussion of the prevention within a complicated software system. As described in the beginning of this section, software development teams that develop a complicated software system may distribute geographically and need not belong to the same company. In this case, different development teams may be familiar with different information flow control models. To embed an information flow control model in a complicated software system decomposed into subsystems, the following approaches can be used.

a. Requiring the software development teams to embed the same information flow control model in the subsystems they develop.
b. Allowing different teams to embed different information flow control models in the subsystems they developed.

If the first approach is applied, software developers unfamiliar with the assigned information flow control model should be trained to use the model. It is possible that the developers are unwilling to use a new model, which results in serious management problems. Moreover, it is possible that a software system is developed through reusing existing software components that are embedded with information flow control models different from the assigned one. If the first approach is used, the reuse is not allowed. According to the above description, we suggest that the second approach should be used. Since different information flow control models are incompatible in general, coordinating heterogeneous information flow control models becomes an essential issue. In the coordination, coordination information should be added to every subsystem. Nevertheless, the information flow control model embedded in a subsystem should not be changed.

We developed a model to coordinate heterogeneous information flow control models for object-oriented systems. It is based on access control list (ACL) and named IFCMC (information flow control model coordinator). This paper presents IFCMC and its evaluation.

RELATED WORK

We have involved in the research of information flow control for years[26-30]. Since we cannot identify a model that coordinates heterogeneous information flow control models, we cannot compare IFCMC with existing models. Nevertheless, we describe some information flow control models below for readers' reference.

Traditional access control is achieved by access control matrix (ACM)[31]. A subject can access an object if the required access right appears in the matrix. ACM allows only static access control[32-33]. On the other hand, DACM (dynamic access control matrix)[32] allows dynamic access right allocation.

Mandatory access control (MAC) is useful in access control. An important milestone of MAC is that proposed by Bell and LaPadula[7]. It categorizes the security levels of objects and subjects. Access control follows the no read up and no write down rules[7, 23]. Bell and LaPadula's model has been generalized into the lattice model[8-10] (see[34] for a survey of lattice models). As described in section 1, the typical lattice model proposed in[8-9] uses the can flow relationship to control information flows and the join operator to avoid Trojan horses.

The model in[11] controls information flows in object-oriented systems. It uses ACLs of objects to compute ACLs of execution. A message filter is used to filter out possibly non-secure information flows. Since the computation of an execution's ACL takes information propagation into consideration, Trojan horses are avoided. The model uses different modes of information flow control to loosen the restriction MAC. Flexibility is added to the model by allowing exceptions during or after method execution[12-13]. More flexibility is added using versions[25].

The purpose-oriented model[16-18] proposes that invoking a method may be allowed for some methods but disallowed for others, even when the invokers belong to the same object. This consideration is correct, because the security levels of an object's methods may be different[23]. Different methods can thus access information in different security levels. The model uses object methods to create a flow graph, from which non-secure information flows can be identified.

The decentralized label approach[2-5] marks the security levels of variables using labels. A label is composed of 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. Join operation is used to avoid Trojan horses. Declassification is allowed. Write access is controlled[2]. The approach in[14] also applies the label approach. Every file, device, pipe and process in a UNIX system is attached with a label to control the access. Join operation is used to avoid Trojan horses. Declassification is allowed.

RBAC was also used in information flow control[15, 26-28] because it is a super set of DAC and MAC[35-38]. In RBAC, a *role* is a collection of permissions[39]. A role can establish one or more sessions. During a session, a user playing a role possesses the permissions of the role. Users can change role to enforce the need-to-know principle[40-42]. Since the original design of RBAC was not for information flow control, the general cases of RBAC is not suitable for controlling information flow in object-oriented systems. The model in[15] applies RBAC for access control within object-based systems. It classifies object methods and derives a flow graph from method invocations. From the graph, non-secure information flows can be identified.

DESIGN PHILOSOPHY OF IFCMC

In an executing object-oriented system, objects are instantiated from classes and messages are passed among objects (a messages corresponds to a method invocation and messages are actually passed among object methods). According to information hiding, message passing is the only way for an object to access the internals of another object. If an object-oriented system is decomposed into subsystems, a subsystem can be arranged to consist of classes and public interface, in which the interface is composed of class methods for accessing the subsystem's internals. In other words, a subsystem can encapsulate classes and offer interface for subsystem communication. A subsystem can encapsulate not only classes but also information flow control model, in which different subsystems may embed different models. Since subsystems offer interfaces for communication, information can exchange among subsystems. Suppose information exchanges between

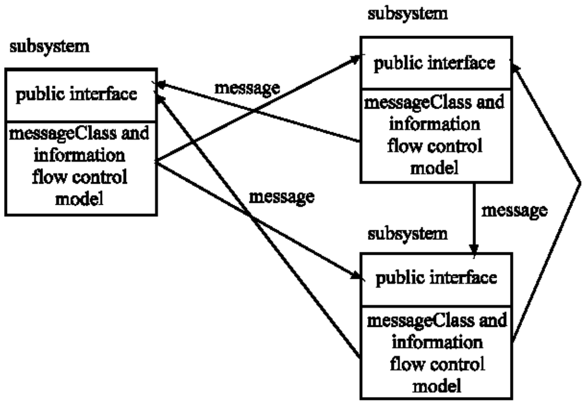

subsystem
public interface
messageClass and information flow control model
message
subsystem
public interface
messageClass and information flow control model
message
message
subsystem
public interface
messageClass and information flow control model

Fig. 1: Subsystems and their communications

two subsystems and the information flow control models embedded in the subsystems are different. Then, preventing leakage of the information exchanged between the subsystems becomes an issue to solve. IFCMC solves this issue. In other words, IFCMC coordinates heterogeneous information flow control models.

As mentioned above, message passing is the only way for an object to access the internal of another object. Nevertheless, object-oriented languages such as JAVA and C++ allow an object's attributes to be accessed outside the object such as using the statements obj.att = expression; and variable = obj.att; outside the object obj. The statements violate information hiding. In IFCMC, if a subsystem accesses an attribute of an object in another subsystem, we explicitly define methods in the latter subsystem's interface for accessing the attribute. The method names are the attribute name attached with an R (for read) and a W (for write), respectively. For example, if the attribute obj.att within a subsystem is accessed by another subsystem, the interface of the former subsystem will contain the method obj.attW and obj.attR. The former method is for writing obj.att using a statement like obj.att = expression; and the latter for reading obj.att using a statement like variable = obj.att;. According to this arrangement, the statements obj.att = expression; and variable = obj.att can respectively be transferred into a statement that invokes the method obj.attW and obj.attR. Therefore, we will not discuss statements like obj.att = expression; and variable = obj.att in the rest of the paper.

Requiring strict information hiding as described above is necessary because IFCMC uses subsystem interfaces to coordinate heterogeneous information flow control models. The coordination is based on the rule: when information is passed from a subsystem to another one, the security level of the information being passed should be the same as or lower than the security level of the variable that receives the information. If low security level variables receive high security level information, the information may be leaked to the users that can access the variables. The coordination rule mentioned above can be expended as follows:

a. If a parameter in a subsystem's interface receives an argument from another subsystem, the security level of the parameter should be the same as or higher than that of the argument.
b. If a variable in a subsystem receives the return value of a method in another subsystem, the security level of the variable should be the same as or higher than that of the return value.

The first rule ensures that information passed to a subsystem via arguments will not be leaked within the subsystem. The second rule ensures that information passed to a subsystem via method return values will not be leaked within the subsystem. The rules use subsystem interfaces to coordinate heterogeneous information flow control models. In using IFCMC, we require the following conditions to be simultaneously true:

a. The control granularity of the models coordinated by IFCMC should detail to variables. This condition is necessary because information managed by a program is generally stored in variables. Since different variables may be in different security levels[23], variables should be protected independently. In other words, the control granularity should detail to variables.
b. Models coordinated by IFCMC should follow subsystem interfaces to exchange information. This condition corresponds to using protocols to communicate cooperating software systems. Therefore, the condition is reasonable.
c. The same programming language should be used to program the subsystems. This is reasonable because using different languages in a software system may result in unexpected errors.

The second and third conditions do not limit information flow control models coordinated by IFCMC. Instead, the second condition requires IFCMC users to follow subsystem interfaces to exchange information and the third condition can be fulfilled when consensus is reached. In this regard, as long as the control granularities of several information flow control models are detailed to variables, they can be coordinated by IFCMC. Since most existing models detail the control granularity to variables, IFCMC can be applied widely. IFCMC uses ACL to define security levels of the information exchanged in the interfaces, as described below:

a. Every parameter in a subsystem's interface is associated with an ACL to depict the parameter's security level.
b. Every argument in a message is associated with an ACL to depict the argument's security level when the message is passed to another subsystem.
c. Every return value of a method in a subsystem's interface is associated with an ACL to depict the return value's security level.
d. Every variable that receives the return value of another subsystem's method is associated with an ACL to depict the variable's security level.

With the ACLs associated with parameters, arguments, method return values and variables that receive method return values, coordination among different information flow control models can be implemented through ACL comparison. If two ACLs are incomparable, the passing of argument to parameter or the receiving of method return value by a variable is not allowed. That is, the corresponding statement is non-secure. As a summary, the design philosophy of IFCMC is listed below:

a. Explicitly describe the interface of every subsystem and attach an ACL to every parameter and method return value to ensure the security of information flows across subsystems.
b. If a statement in a subsystem invokes a method in another subsystem, attach an ACL to every argument and the variable to receive the method return value. The ACLs ensure information flow security across subsystems.

IFCMC

This section describes IFCMC and its coordination operation. IFCMC is composed of two parts. One is in the message senders (i.e., the subsystems that invoke methods in other subsystems) and the other in the message receivers (i.e., the subsystems that offer methods for other subsystems to invoke).

Definitions: We give definitions related to IFCMC below. Definitions 2 and 3 define the IFCMC part of message receivers and the description after Definition 3 constitutes the IFCMC part of message senders.

Definition 1: A subsystem subsys is defined below:

subsys = (*INTF*, *ENC*), in which

INTF is the interface of the subsystem for other subsystems to invoke. It is composed of methods offered by the subsystem (Definition 2 defines interface methods).
ENC is composed of those encapsulated in the subsystem (see the description after Definition 3).

Definition 2: A method *ifmd* in a subsystem's interface is defined below:

ifmd = (*name*, *par and ACL*, *retACL*), in which

name is the method's name.
par and ACL is a sequence, in which each element is composed of a parameter and the ACL associated with the parameter (definition 3 defines ACL).
retACL is the ACL of the method return value if the method returns a value.

Definition 3: An ACL is composed of a read access control list (RACL) to control read access and a write access control list (WACL) to control write access. An ACL ACL_{var} attached to a variable *var* is defined below:

ACL_{var} = ($RACL_{var}$, $WACL_{var}$), in which
$RACL_{var}$ = {*md* | *md* is a method that is allowed to read the variable *var*}.
$WACL_{var}$ = {*md* | *md* is a method that is allowed to write the variable *var*}.

Those encapsulated in a subsystem include classes and an information flow control model. IFCMC cares neither classes nor the information flow control model. IFCMC cares only the statements that invoke methods in other subsystems because the statements may result in information leakage among subsystems. IFCMC attaches an ACL to every argument of a method invocation statement across subsystems. It also attaches an ACL to the variable that receives the method return value if the method returns a value. The ACLs prevent information leakage among subsystems, as described in section 4.2.

IFCMC coordination (ensuring the security of method invocations across subsystems): If a software system is decomposed into subsystems, information flows within a subsystem are controlled by the information flow control model embedded in the subsystem. If a message is passed across subsystems, IFCMC checks the information flows induced by the message to ensure that the message passing will not result in information leakage. Checking an information flow corresponds to comparing ACLs related to the flow. To explain ACL comparison, we suppose that the argument arg is passed to the parameter par of the method obj.md. We also suppose that the ACLs associated with arg and par are respectively ($RACL_{arg}$, $WACL_{arg}$) and ($RACL_{par}$, $WACL_{par}$). Then, the following *secure flow conditions* should be simultaneously true for the information flow induced by passing arg to par to be secure.

First secure flow condition:
$(RACL_{par} \subseteq RACL_{arg}) \wedge (\{obj.md\} \subseteq RACL_{arg})$

Second secure flow condition:
$(WACL_{par} \supseteq WACL_{arg}) \wedge (WACL_{par} \supseteq \{obj.md\})$

The first secure flow condition controls read access. The condition $RACL_{par} \subseteq RACL_{arg}$ requires that the security level of par should be the same as or higher than that of arg because the methods that can read par can also read arg. The condition $\{obj.md\} \subseteq RACL_{arg}$ is necessary because the argument will be read by the method obj.md. The second secure flow condition controls write access. It requires the methods that can write the argument arg can also write the parameter par. This means that the methods trusted by arg should also be trusted by par. The condition also requires the method obj.md to be within $WACL_{par}$ because the write operation (i.e., write arg to par) is performed in obj.md.

With the secure flow conditions, IFCMC ensures inter-subsystem information flow security using the following two *IFCMC coordination rules* (the rules are a formal representation of the coordination rules mentioned in section 3). In defining the rules, we suppose: (a) a statement in the subsystem subsys1 invokes the method obj.md in the subsystem subsys2, (b) the statement in subsys1 passes the arguments (arg1, arg2,..., argn) to the method obj.md, (c) obj.md receives the arguments using the parameters (par1, par2,..., parn), (d) the method obj.md returns a value retval and (e) retval is received by the variable var in subsys1.

First IFCMC coordination rule: For every argument argi and the parameter pari receiving the value of argi, their ACLs ($RACL_{argi}$, $WACL_{argi}$) and ($RACL_{pari}$, $WACL_{pai}$) should make the following *secure argument passing conditions* simultaneously true. Note that the conditions are directly derived from the two secure flow conditions.

First secure argument passing condition:
$(RACL_{pari} \subseteq RACL_{argi}) \wedge (\{obj.md\} \subseteq RACL_{argi})$

Second secure argument passing condition:
$(WACL_{pari} \supseteq WACL_{argi}) \wedge (WACL_{pai}\ \{obj.md\})$

Second IFCMC coordination rule: For the return value retval and the variable var that receives the return value, their ACLs ($RACL_{var}$, $WACL_{var}$) and ($RACL_{retval}$, $WACL_{retval}$) should make the following secure value returning conditions simultaneously true. Note that the conditions are directly derived from the two secure flow conditions.

First secure value returning condition:
$(RACL_{var} \subseteq RACL_{retval}) \wedge (\{obj.md\} \subseteq RACL_{retval})$

Second secure value returning condition:
$(WACL_{var} \supseteq WACL_{retval}) \wedge (WACL_{var} \supseteq \{obj.md\})$

Proof of correctness Without loss of generality, we use a two-subsystem case for the proof. We call the subsystems subsys1 and subsys2 and make the following assumptions:

a. The subsystem subsys2 offers the interface method *obj.md* for subsys1 to invoke.
b. When subsys1 invoke the interface method *obj.md*, it passes the arguments *arg1*, *arg2*,..., *argn* and the ACLs of the arguments are respectively ACL_{arg1}, ACL_{arg2},... and ACL_{argn}.
c. The subsystem subsys1 uses variable *var* to receive return value and the ACL of *var* is ACL_{var}.
d. The interface method *obj.md* uses the parameters *par1*, *par2*,.., *parn* to receive the arguments passed to the method and the ACLs of the parameters are respectively ACL_{par1}, ACL_{par2} ,... and ACL_{parn} .
e. The interface method *obj.md* returns the value *retn* and the ACL of *retn* is ACL_{retn}.

When the subsystems *subsys1* invokes *obj.md*, both IFCMC coordination rules mentioned in the previous section should be true. According to the first IFCMC coordination rule, both *secure argument passing conditions* are true. That is, for every argument *arg* with an ACL ($RACL_{arg}$, $WACL_{arg}$) and the parameter *par* receiving the value of *arg*, which is associated with an ACL ($RACL_{par}$, $WACL_{par}$), the following conditions will be simultaneously true:

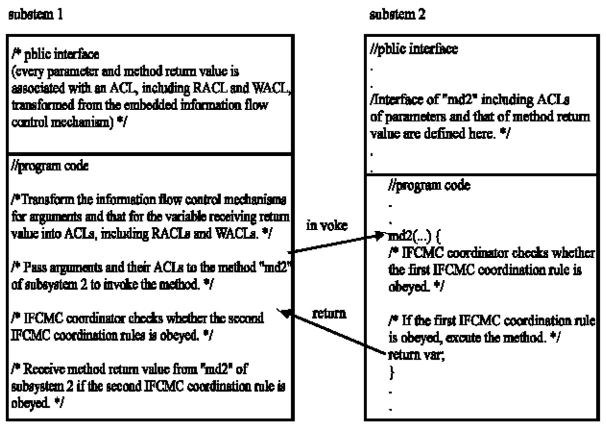
$(RACL_{par} \subseteq RACL_{arg}) \wedge (\{obj.md\} \subseteq RACL_{arg})$
$(WACL_{par} \supseteq WACL_{arg}) \wedge (WACL_{par} \supseteq \{obj.md\})$

According to the first condition above, the security level of a parameter is the same as or higher than that of an argument. As long as the read access control of the information flow control model embedded in *subsys2* is correct, information passed to susbsys2 via parameters will not be leaked by the parameters of subsys2. The rationale is that a variable intending to obtain the information carried by a parameter should possess a security level that is the same as or higher than that of the parameter. According to the second condition above, the writers (which are methods) trusted by an argument are also trusted by the parameter receiving the argument. As long as the write access control of the information flow control model embedded in subsys2 is correct, information passed to susbsys2 via arguments will not corrupt the information carried by the variables in subsys2.

According to the second IFCMC coordination rule, both *secure value returning conditions* are true. That is, for the return value of the method *obj.md* with an ACL ($RACL_{obj.md}$, $WACL_{obj.md}$) and the variable *var* receiving the value of the return value, which is associated with an ACL ($RACL_{var}$, $WACL_{var}$), the following conditions will be simultaneously true:

$(RACL_{var} \subseteq RACL_{obj.md}) \wedge (\{obj.md\} \subseteq RACL_{obj.md})$
$(WACL_{var} \supseteq WACL_{obj.md}) \wedge (WACL_{var} \supseteq \{obj.md\})$

According to the first condition above, the security level of the variable receiving the return value of the method obj.md is the same as or higher than that of the return value. As long as the read access control of the information flow control model embedded in *subsys1* is correct, information passed to susbsys1 via method return values will not be leaked by the variables receiving the return values. The rationale is that a variable intending to obtain the information carried by a variable that receives a method return value should possess a security level that is the same as or higher than that of the variable. According to the second condition above, the writers (which are methods) trusted by the variable receiving the return value of the method obj.md are also trusted by return value of the method obj.md. As long as the write access control of the information flow control model



```
/* pblic interface
(every parameter and method return value is
associated with an ACL, including RACL and WACL,
transformed from the embedded information flow
control mechanism) */

//program code

/*Transform the information flow control mechanisms
for arguments and that for the variable receiving return
value into ACLs, including RACLs and WACLs. */

/* Pass arguments and their ACLs to the method "md2"
of subsystem 2 to invoke the method. */

/* IFCMC coordinator checks whether the second
IFCMC coordination rules is obeyed. */

/* Receive method return value from "md2" of
subsystem 2 if the second IFCMC coordination rule is
obeyed. */
```

```
//pblic interface
.
.
/Interface of "md2" including ACLs
of parameters and that of method return
value are defined here. */
.
.
    //program code
    .
    .
    md2(...) {
    /* IFCMC coordinator checks whether
    the first IFCMC coordination rule is
    obeyed. */

    /* If the first IFCMC coordination rule
    is obeyed, excute the method. */
    return var;
    }
    .
    .
```

Fig. 2: Transformation of ACLs from information flow control mechanisms

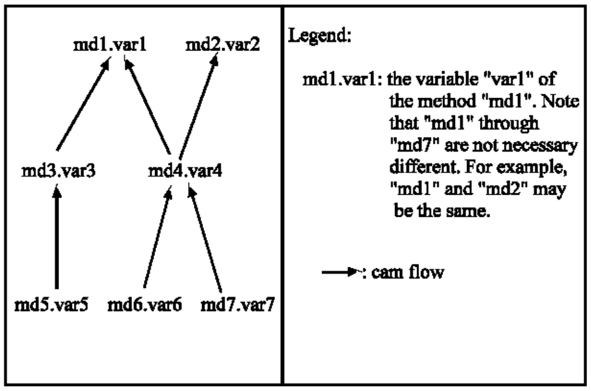

md1.var1
md2.var2
md3.var3
md4.var4
md5.var5
md6.var6
md7.var7
Legend:
md1.var1: the variable "var1" of the method "md1". Note that "md1" through "md7" are not necessary different. For example, "md1" and "md2" may be the same.
: cam flow

Fig. 3: A lattice

embedded in subsys1 is correct, information passed to *susbsys1* via method return values will not corrupt the information carried by the variables in subsys1.

USING IFCMC

To coordinate subsystems embedded with heterogeneous information flow control models using FCMC, programmers should attach ACLs to arguments, parameters, method return values and variables that receives method return values. If a subsystem is embedded with an information flow control model that is nothing to do with ACLs, transformation is necessary. We use Fig. 2 to explain this concept.

Figure 2 depicts the communication of two subsystems. In the public interface of a subsystem, methods for other subsystems to invoke are declared. In the declaration, every parameter and every method return value is associated with an ACL for information flow control. An ACL is composed of an RACL and WACL, which are transformed from the information flow control mechanism embedded in the subsystem. For example, suppose the lattice model is used in subsystem 1. Then, the RACL and WACL of md4.var4 transformed from the lattice in Figure 3 are {md1, md2, md4} and {md6, md7, md4}, respectively. The transformation follows the no read up and no write down rules[7].

As shown in Fig. 2, before a method in subsystem 1 invokes the method md2 in subsystem 2, every argument passed to md2 and the variable to receive the return value of md2 should be associated with an ACL. When md2 is invoked, the IFCMC coordinator checks whether the first IFCMC coordination rule is obeyed. If the rule is obeyed, the method is executed. Otherwise, the invocation is blocked. Suppose the rule is obeyed and the method returns a value to subsystem 1, the IFCMC coordinator checks whether the second IFCMC coordination rule is obeyed. If the rule is obeyed, the variable in subsystem 1 receives the return value. Otherwise, the return operation is blocked.

RESULTS AND DISCUSSIONS

We developed two RBAC-based information flow control models namely OORBAC[26-27] and L^{n}RBAC[28] whose permissions are based on ACLs. We also developed an association-based information flow control model[43] whose access rights are also based on ACLs. Moreover, we developed a label-based information flow control model[29-30] whose access rights are nothing to do with ACLs. We use IFCMC to coordinate the four models. Conceptually, Figure 4 depicts a software system embedding the four models coordinated by IFCMC (we suppose that the system is decomposed into four subsystems). In the figure, the bottom level of a subsystem is the subsystem implementation, the middle level is the information flow control model embedded in the subsystem and the top level is the IFCMC model. Before executing the software system, every subsystem should be processed by the preprocessor of the information flow control model embedded in the subsystem. For example (see Figure 4), subsystem 1 should be processed by the preprocessor of OORBAC[26] before execution. The preprocessing produces a security monitor of the model and the JAVA program for the subsystem.

After every subsystem is preprocessed, the software system should be preprocessed the second time by the IFCMC preprocessor. The product produced by the IFCMC preprocessor is a software system represented in a programming language. Currently, OORBAC, L^{n}RBAC,

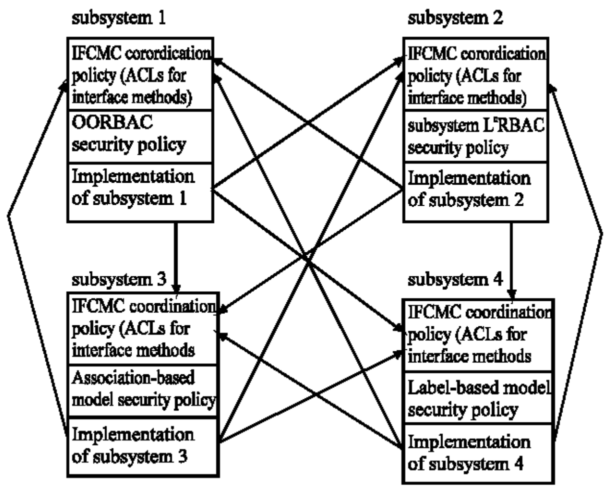

subsystem 1
IFCMC corordication policty (ACLs for interface methods)
OORBAC security policy
Implementation of subsystem 1
subsystem 2
IFCMC corordication policty (ACLs for interface methods)
subsystem L^{n}RBAC security policy
Implementation of subsystem 2
subsystem 3
IFCMC coordination policy (ACLs for interface methods
Association-based model security policy
Implementation of subsystem 3
subsystem 4
IFCMC coordination policy (ACLs for interface methods
Label-based model security policy
Implementation of subsystem 4

Fig. 4: IFCMC coordination in which an arrow is a set of message associated with ACLs

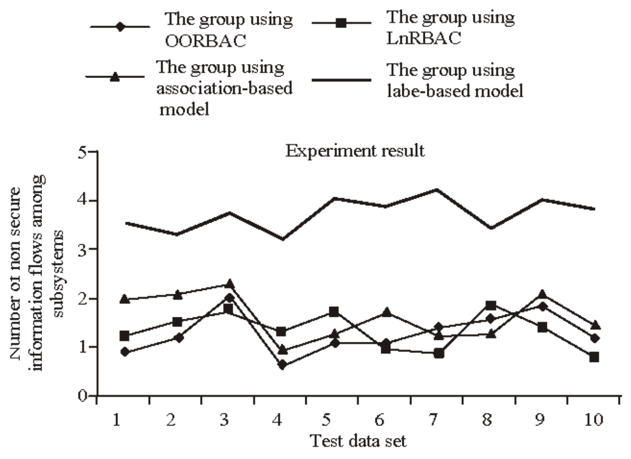


Fig. 5: Experiment result

the association-based information flow control model and the label-based information flow control model are embedded in JAVA. Therefore, the output of the IFCMC preprocessor is a pure JAVA program, which is composed of an IFCMC coordinator, the security monitors for various information flow control models and the original JAVA programs. The IFCMC coordinator ensures that the two IFCMC coordination rules are obeyed during program execution. The security monitor embedded in a subsystem such as the OORBAC security monitor[26] ensures information flows security within the subsystem. With the cooperation of the IFCMC coordinator and the security monitors of the information flow control models embedded in subsystems, information flow security within a subsystem and that among subsystems will be ensured.

We evaluated IFCMC using the system model in Figure 4. Many examples were used in our evaluation. Here we offer an experiment result using the example of a simple student management system. The system was decomposed into for subsystems, in which the first subsystem primarily manages the basic information such as student names and IDs, the second primarily manages the examinations taken by students and the scores obtained by the students, the third primarily manages the inter-relation between students and the library (e.g., manages the borrowing and returning of books) and the last primarily manages students' good records (e.g., a student donates money to the school) and bad records (e.g., a student does not return a book he borrowed from the library on time). We trained twelve students to use the models shown in Figure 4, in which three of them formed a group to use an information flow control model. We used ten sets of test data to run the system and collected the averaged non-secure information flows per 10 statements across subsystems. The experiment result is shown in Figure 5. From the figure, we identify that students in the group using the label-based model committed more errors than those in other groups. This might be a consequence that the group of students is unfamiliar with ACLs because the label-based model is nothing to do with ACLs.

To further show that IFCMC coordinates heterogeneous information flow control models (i.e., IFCMC can identify non-secure information flows across subsystems embedded with different information flow control models), we required the students to inject no-secure information flows across subsystems into their subsystems. We then re-executed the system and found that every non-secure information flow injected by the students was identified by IFCMC. We thus believed that IFCMC is useful in coordinating heterogeneous information flow control models. Below we give some experiences in using IFCMC:

a. Since programmers are required to add IFCMC information to the subsystems they develop (IFCMC information includes ACLs of arguments, those of parameters, those of method return values and those of the variables receiving the return values), the load of programmers is increased. If a programmer is familiar with ACLs, adding IFCMC information to the subsystem developed by the programmer will not bother him much. For example, programmers using OOBBAC and L^{n}RBAC performed well in adding IFCMC information. However, adding IFCMC information to a subsystem developed by a programmer unfamiliar with ACLs did cause trouble. For example, programmers using the label-based model had difficulty in preparing IFCMC information.

This experience was a frustration for us. We thus interviewed the students that used the label-based model and found that they had poor idea about ACL (because they were undergraduate students). We thus trained them to use ACLs and required them to re-program their subsystem. We then re-evaluated the system and found that non-secure information flows among subsystems caused by the label-based model group were decreased to almost the same level as other groups.

b. As mentioned in sections 4 and 4.1, IFCMC information is composed of two parts, in which one is in the subsystems that invoke methods in another subsystems and the other in the subsystems that offer methods for other subsystems to invoke. IFCMC information in a subsystem that offers methods for other subsystems to invoke is grouped in the subsystem's interface. Therefore, this part of IFCMC information is relatively easy to manage. On the other hand, IFCMC information in a subsystem that invokes methods in another subsystems scatters all over the subsystem (i.e., statements invoking other subsystems' interfaces are scattered). Therefore, this part of IFCMC information is relatively difficult to manage. In fact, most non-secure inter-subsystem information flows are caused by this part of IFCMC model.

CONCLUSIONS

This study proposes a model IFCMC (information flow control model coordinator) to coordinate heterogeneous information flow control models using access control list (ACL). It uses read access control lists (RACLs) to control read access and write access control lists (WACLs) to control write access. IFCMC offers coordination rules using RACLs and WACLs to ensure secure information exchange among subsystems embedded with different information flow control models.

We evaluated IFCMC through experiments. During the experiment, we intentionally injected non-secure information flows across subsystems into the systems being tested. The experiment showed that IFCMC identified every injected non-secure information flow. We thus believe that IFCMC is useful in identifying non-secure information flows across subsystems embedding with different information flow control models. That is, IFCMC can coordinate heterogeneous information flow control models.

REFERENCES

1. Sabelfeld, A. and A.C. Myers, 2003. Language-Based Information-Flow Security. IEEE J. Selected Areas in Commun., 21: 5-19.
2. Myers, A. and B. Liskov, 2000. Protecting Privacy using the Decentralized Label Model. ACM Trans. Software Eng. Methodology, 9: 410-442.
3. Myers, A.C., 1999. Jflow: Practical mostly-static information flow control, Proc. 26'th ACM Symp. Principles of Programming Language, pp: 228-241.
4. Myers, A.C. and B. Liskov, 1997. A Decentralized model for information flow control, proc. 17'th ACM Symp. Operating Systems Principles, pp: 129-142.
5. Myers, A. and B. Liskov, 1998. Complete, Safe information flow with decentralized labels, Proc. 14'th IEEE Symp. Security and Privacy, pp: 186-197.
6. McCollum, C.J., J.R. Messing and L. Notargiacomo, 1990. Beyond the pale of MAC and DAC - Defining New Forms of Access Control, Proc. 6'th IEEE Symp. Security and Privacy, pp. 190-200.
7. Bell D.E. and L.J. LaPadula, 1976. Secure computer systems: Unified exposition and multics interpretation, *technique report, Mitre Corp.*, Mar. http://csrc.nist. gov/publications /history/bell76.pdf
8. Denning, D.E., 1976. A Lattice Model of Secure Information Flow, Comm. ACM, 19: 236-243.
9. Denning D.E. and P.J. Denning, 1977. Certification of Program for Secure Information Flow, Comm. ACM, 20: 504-513.
10. Brewer D.F.C. and M.J. Nash, 1989. The Chinese Wall Access control policy, Proc. 5'th IEEE Symp. Security and Privacy, pp: 206-214.
11. Samarati, P., E. Bertino, A. Ciampichetti and S. Jajodia, 1997. Information Flow Control in Object-Oriented Systems, IEEE Trans. Knowledge Data Eng., 9: 524-538.
12. Bertino, E. Sabrina de Capitani di Vimercati, E. Ferrari and P. Samarati, 1998. Exception-based Information Flow Control in Object-Oriented Systems, ACM Trans. Information System Security, 1: 26-65.
13. Ferrari, E., P. Samarati, E. Bertino and S. Jajodia, 1997. Providing Flexibility in Information flow control for Object-Oriented Systems, Proc. 13'th IEEE Symp. Security and Privacy, pp. 130-140.
14. McIlroy M.D. and J.A. Reeds, 1992. Multilevel Security in the UNIX Tradition, Software-Practice and Experience, 22: 673-694.
15. Izaki, K., K. Tanaka and M. Takizawa, 2001. Information Flow Control in Role-Based Model for Distributed Objects, Proc. 8'th International Conf. Parallel and Distributed Systems, pp: 363-370.
16. Yasuda, M., T. Tachikawa and M. Takizawa, 1997. Information flow in a purpose-oriented access control model. Proc. International Conf. Parallel and Distributed Systems, pp: 244-249.

17. Yasuda, M., T. Tachikawa and M. Takizawa, 1998. A purpose-oriented access control model, Proc. 12'th International Conf. Information Networking, pp: 168-173.
18. Tachikawa, T., M. Yasuda and M. Takizawa, 1997. A purposed-oriented access control model in object-based systems. Trans. Information Processing Society of Japan., 38: 2362-2369.
19. Graubart, R., 1989. On the Need for a Third Form of Access Control, Proc. 12'th Nat'l Computer Security Conf., pp: 296-303.
20. Jajodia S. and B. Kogan, Integrating an object-oriented data model with multilevel security. Proc. 6'th IEEE Symp. Security and Privacy, pp: 76-85.
21. Foley, S.N., 1989. A model for secure information flow, Proc. 5'th IEEE Symp. Security and Privacy, pp: 248-258.
22. Zdancewic, S., L. Zheng, N. Nystrom and A.C. Myers, 2001. Untrusted Hosts and Confidentiality: Secure Program Partitioning, Proc. 18th ACM Symp. Operating Systems Principles.
23. Varadharajan V. and S. Black, 1990. A multilevel security model for a distributed object-oriented system. Proc. 6'th IEEE Symp. Security and Privacy, pp: 68-78.
24. Tari Z. and S.W. Chan, 1997. A role-based access control for intranet security. IEEE Internet Computing, 1: 24-34.
25. Maamir A. and A. Fellah, 2003. Adding flexibility in information flow control for object-oriented systems using versions. Intl. J. Software Engin. Knowledge Engin., 13: 313-326.
26. Chou, S.C., 2004. Embedding role-based access control model in object-oriented systems to protect privacy. J. Sys. Software, 71: 143-161.
27. Chou, S.C., 2004. Providing flexible access control to an information flow control model. J. Sys. Software, 73: 425-439.
28. Chou, S.C., 2004. L^nRBAC: A multiple-leveled role-based access control model for protecting privacy in object-oriented systems. J. Object Technol., 3: 91-120.
29. Chou, S.C., 2003. Information flow control among objects: Taking foreign objects intro control. HICSS-36, Hawaii, pp: 335-344.
30. Chou, S.C., 2002. Information flow control in object-based systems. Intl. Computer Symposium, Hualien, Taiwan.
31. Harrison, M.H., W.L. Ruzzo and J.D. Ullman, 1976. Protection in operating systems. Communications of the ACM, 19: 461-471.
32. Olivier, M.S., R.P. van de Riet and E. Gudes, 1998. Specifying application-level security in workflow systems. In proceeding of the 9'th International Workshop on Database and Expert Systems Applications, pp: 346-351.
33. Thomas R.K. and R.S. Sandhu, 1997. Task-Based Authorization Controls (TBAC): A Family of models for active and enterprise-oriented authorization management. In proceedings of the IFIP WG11.3 Workshop on Database Security.
34. Sandhu, R.S., 1993. Lattice-Based access control models. IEEE Computer, 26: 9-19.
35. Sandhu, R., 1996. Role hierarchies and constraints for lattice-based access controls. Proc. Fourth European Symposium on Research in Computer Security, pp: 65-79.
36. Nyanchama M. and S. Osborn, 1995. Modeling mandatory access control in role-based security systems. Database security IX: Status and Prospects, pp: 129-144.
37. Osborn, S., 1997. Mandatory access control and role-based access control revisited. Proc. Second ACM Workshop on Role-Based Access Control, pp: 31-40.
38. Osborn, S., R. Sandhu and Q. Munawer, 2000. Configuring role-based access control to enforce mandatory and discretionary access control policies. ACM Trans. Info. Sys. Security, 3: 85-106.
39. Sandhu, R.S., E.J. Coyne, H.L. Feinstein and C.E. Youman, 1996. Role-based access control models. IEEE Computer, 29: 38-47.
40. Nyanchama M. and S. Osborn, 1994. Access rights in role-based security systems. Database Security VIII: Status and Prospects, pp: 37-56.
41. Sandhu, R., V. Bhamidipati and Q. Munawer, 1999. The ARBAC97 model for role-based administration of roles. ACM Tran. Info. Sys. Security, 2: 105-135.
42. Thomsen, D.J., 1991. Role-based application design and enforcement. Database Security IV: Status and Prospects, pp: 151-168.
43. Chou S.C. and Y.K. Wen, 2004. Association-based information flow control in object-oriented systems. Intl. J. Software Engin. Knowledge Engin., 14: 291-322. 2004.