Asian Journal of Information Technology 5 (1) : 96-106, 2006

Controlling Information Flows Among Object-Oriented Systems to Prevent Information Leakage

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

Abstract: Many information flow control models were available to prevent information leakage within a system. Since systems may cooperate, it is necessary to prevent information leakage among cooperating systems when they communicate. Our survey shows that no existing model offers the prevention. In the past years, we developed an information flow control model based on RBAC (role-based access control), which is named OORBAC (object-oriented role-based access control). Like other existing models, OORBAC cannot prevent information leakage among systems. To offer the prevention, we extended OORBAC. The extension is based on the consideration: when information is passed from a system 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 receiving the information. This study shows the extended model and its evaluation.

Key words: Information security, information flow control, information leakage, object-oriented system

INTRODUCTION

Access control within a system prevents leakage of sensitive information managed by the system. Information leakage prevention can be achieved by controlling information flows and therefore quite a few information flow control models are available[1-15]. Existing models were generally based on approaches such as discretionary access control (DAC)[1-4], mandatory access control (MAC)[5-8], role-based access control (RBAC)[9-10], and label-based approach [11-15]. Our survey reveals that existing models prevent information leakage within a system. Nevertheless, they fail to prevent information leakage among systems. In our opinion, the latter prevention is necessary because systems may exchange information when they communicate. Important issues of the latter prevention are described below:

- Information passed from a source system to a destination system should not be leaked to unauthorized users in the destination system.
- Information passed from a source system to a destination system should not be leaked to another systems (if this kind of leakage occurs, the destination system becomes a Trojan horse[12-15].

In the past years, we developed an RBAC-based information flow control model for object-oriented systems named OORBAC (object-oriented role-based access control)[9]. Its original design is for controlling information flows within a system. Therefore, it cannot prevent information leakage among systems. Since preventing information leakage among systems is necessary, we extended OORBAC to achieve the prevention. The extended model is called OORBAC^{ar}, in which the super-script ar means that the model prevents both intra-system and inter-system information leakage. Here intra-system information flows refer to information flows within a system. On the other hand, inter-system information flows refer to information flows among systems. This study presents OORBAC^{ar}.

RELATED WORK

The model in[1] is based on DAC. It controls information flows within object-oriented systems. It uses ACLs of objects to compute ACLs of executions (which are composed of methods). A message filter filters out possibly non-secure information flows. Flexibility is added by allowing exceptions during or after method execution[2-3]. More flexibility is provided using versions[4].

The model presents by Bell and LaPadula[5] is an important milestone of MAC, which categorizes the security levels of objects and subjects. Information flows in the model follow the no read up and no write down rules[5]. Bell and LaPadula's model was generalized into the lattice model[6-8]. In the typical lattice model presents in[7-8], a lattice (SC,$\rightarrow$,$\oplus$) is constructed using SC, which is the set of security classes, the symbol $\rightarrow$, which is the can flow relationship, and the symbol $\oplus$, which is the join operator. The can flow relationship controls information flows and the join operator avoids Trojan horses (i.e., indirect information leakage).

RBAC[16] can also be used in information flow control. It is primarily composed of users, roles, sessions, permissions, assignment relationships, and constraints. A role is a collection of permissions[17]. Within a session, a user possesses the permissions of the role he plays. RBAC was proved to be a super set of both DAC and MAC[17-20]. Since DAC and MAC are useful in information flow control[1-8], RBAC can also be used in that control. Nevertheless, we can only identify the models in[10] uses RBAC for the control. The model classifies object methods and derives a flow graph from method invocations. Non-secure information flows can be identified from the graph.

The approach in[11] presents a labeling system in UNIX. Every file, device, pipe, and process is attached with a label. Join operation is used to avoid Trojan horses. The decentralized label approach[12-15] 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. Join operation is used to avoid Trojan horses.

As a summary of the survey, no model controls inter-system information flows. This motivates our research.

OVERVIEW OF OORBAC

OORBAC controls information flows within object-oriented systems[9]. The most importance mechanism in OORBAC is ICRel (Information flow Control Relationship) as defined below:

Definition 1: An ICRel exists among classes if information may directly flow among the instances of the classes. Each ICRel is associated with a security policy for class instances to obey. If multiple security policies must be obeyed by class instances, more than one ICRels should be defined among the classes, in which an ICRel enforces a security policy.

The above definition defines an ICRel as a relationship among classes, which can be instantiated to link objects of the classes. We let an instance of an ICRel be a session, and let objects linked by the relationship be *roles* within the session. For example, if manager1 and operator1 are friends, they are roles within a friend session. In OORBAC, objects within a session should obey the security policy defined by the ICRel from which the session is instantiated. The security policy of an ICRel is exhibited by permissions, roles, access control lists (ACLs) of variables (a variable can be an object attribute or a method variable), and ACLs of method return values. Permissions limit method invocation. For example, a permission (o1.md1, o2.md2) allows the method o1.md1 to invoke the method o2.md2. Since permissions are not enough to control information flows among variables, OORBAC attaches an ACL to every variable and method return value. An ACL is composed of an RACL (read access control list) to control read access and a WACL (write access control list) to control write access. The union of ICRels' security policies constitutes the security policy of a system, which requires that: (a) direct information flows within a session are allowed whereas those among sessions are prohibited and (b) information flows among a session's objects should obey the security policy of the ICRel from which the session is instantiated. Although information cannot directly flow among sessions, it may indirectly flow among sessions. For example, if o1 and o2 are in a session and o2 and o3 in another, information of o1 may indirectly flow to o3 via o2. Both direct and indirect information flows should be secure.

OORBAC should be embedded in an object-oriented system to ensure secure information flows when the system is being executed. Below we define OORBAC from the perspective of embedding the model in an object-oriented system.

Definition 2: OORBAC = (USR, CLS, ICREL, OBJ, VAR, MD, MSG, SES, PER, RLE, IH, URA, RPA, ACLS, DSOURCES, CNS), in which

- USR is a set of users. A user may be a human being or another system. Users play roles when a system is being executed.
- CLS is a set of classes. A class can be instantiated to create objects.
- ICREL is a set of ICRels defined in Definition 1. An ICRel can be instantiated to create sessions.
- OBJ is a set of objects instantiated from classes when the object-oriented system is being executed. OORBAC regards an object as a role.
- VAR is the union of the set of object attributes, that of private method variables, and that of method return values.
- MD is a set of object method.
- MSG is a set of messages passed among objects. A message corresponds to a method invocation.
- SES is a set of sessions. A session is an instance of an ICRel.
- PER is a set of permissions. A permission has the form (obj1.md1, obj2.md2), which allows the method obj1.md1 to invoke the method obj2.md2.

- RLE is a set of roles. A role is an object. Moreover, a role is composed of a set of permissions.
- URA is a set of user-role assignments, which is defined as USR→ 2^{RLE}.
- RPA is a set of role-permission assignments, which is defined as RLE →2^{PER}
- IH is a set of inheritance relationships among classes. An object instantiated from a class possesses all the permissions of the class's super classes. Since objects in OORBAC are roles, IH facilitates establishing role hierarchies (which are important components in RBAC) when an object-oriented system is being executed.
- ACLS is a set of ACLs. As described before, permissions are not enough to control information flows among variables, OORBAC attaches ACLs to variables for more detailed control. The ACL ACL_{var} of a variable var is defined below:

ACL_{var} = {$RACL_{var}$; $WACL_{var}$}, in which
$RACL_{var} \in 2^{MD}$, in which methods in $RACL_{var}$ are allowed to read var.
$WACL_{var} \in 2^{MD}$, in which methods in $WACL_{var}$ are allowed to write var.

- DSOURCES is a set of data sources (DSOURCEs). Each variable in OORBAC is associated with a DSOURCE to facilitate write access control. Suppose the attribute attName is derived from the variable var1 and var2, and var1 and var2 are respectively written by the methods mdx and mdy. Then, the DSOURCE of attName is the set (mdx, mdy) after the derivation.
- CNS is a set of cardinality and modality constraints.

As mentioned before, direct information flows within a session are allowed whereas those among sessions are prohibited. Moreover, information flows among a session's objects should obey the permissions, roles, and ACLs of the ICRel from which the session is instantiated. Therefore, the security of direct information flows can be enforced by the following security conditions:

First security condition: If information directly flows among objects, the objects should be within a session.

Second security condition: For an information flow caused by a method invocation from obj1.md1 to obj2.md2, obj1 should possess the permission (obj1.md1, obj2.md2).

During method invocation, the ACLs and DSOURCEs of arguments should first be copied to the corresponding parameters. The copying is necessary because a parameter receiving the value of an argument inherits the security level of the argument. Note that if an object is passed as an argument, the copying is bypassed because ACLs and DSOURCEs of the object's variables are already defined. After the copying, every information flow in the invoked method should fulfill the following two secure flow requirements. In defining the requirements, we suppose that: (a) the variable d_var is assigned a value derived from the variables in the set {var_i | $var_i \in$ VAR in Definition 2 and i is between 1 and n}, (b) the assignment appears in the method md1, (c) the original ACL of d_var is {$RACL_{d\_var}$; $WACL_{d\_var}$}, (d) the ACL of the i^{th} variable that derives d_var is {$RACL_{var1}$; $WACL_{var1}$}, and (e) the DSOURCE of var_i is $DSOURCE_{var_i}$.

First secure flow requirement: $(RACL_{d\_var} \subseteq \cap_{i=1}^{n} RACL_{var_i}) \wedge (md1 \in \cap_{i=1}^{n} RACL_{var_i})$

Second secure flow requirement: $WACL_{d\_var} \supseteq \cup_{i=1}^{n} DSOURCE_{var_i} \cup \{md1\}$

The first secure flow requirement controls read access. The condition $RACL_{d\_var} \subseteq \cap_{i=1}^{n} RACL_{var_i}$ requires that d_var must be at least the same restricted as the variables deriving d_var. The condition $md1 \in \cap_{i=1}^{n} RACL_{var_i}$ is necessary because md1 reads the variables deriving d_var. The second secure flow requirement controls write access. It requires that the data sources of the variables deriving d_var should be within $WACL_{d\_var}$, because the data derived from the variables are written into d_var. The requirement also requires that the method md1 must be within $WACL_{d\_var}$ because the method performs the write operation.

After the derived data is assigned to the variable d_var, the ACL of d_var should be changed by the join operation[12-15]. This change prevents indirect information leakage. We use the symbol ⊕ to represent the join operator. With join, ACL_{d_var} will be changed to be $\oplus_{i=1}^{n} ACL_{var_i}$ after the derived data is assigned to the variable d_var.

Definition 3: $\oplus_{i=1}^{n} ACL_{var_i} = (\cap_{i=1}^{n} RACL_{var_i}, \cup_{i=1}^{n} WACL_{var_i})$ The join operation trusts less or the same set of readers. Therefore, join will not lower down security level. On the other hand, the operation trusts more writers. This is reasonable because a writer that can write a variable should be regarded as a trusted data source for the data derived from the variable. If a variable is assigned a constant, no join operation will be executed and the

ACL of the variable remains unchanged[9]. for the proof that the join operation prevents indirect information leakage. In addition to joining ACLs, the DSOURCE of d_var will be adjusted as follows:

$$DSOURCE_{d\_var} = \cup_{i=1}^{n} DSOURCE_{var_i} \cup \{md1\}$$

$DSOURCE_{d\_var}$ is set the union of the DSOURCEs of the variables deriving d_var and the method {md1}. The union of the DSOURCEs is obvious because all data sources deriving the computation result should be considered data sources of the result. The method md1 is also a data source because the computation result is written by md1 to d_var.

We embedded OORBAC in JAVA to obtain the language OORBACL. Please refer to[9] for the use of OORBAC and the details features offered by OORBAC.

THE OORBAC^{ar} MODEL

OORBAC^{ar} controls information flows among cooperating systems. It is organized as cooperating OORBACs. We give the following assumptions for OORBAC^{ar}:

- The network is secure. Since cooperating systems may exchange information through the network, the network should be secure.
- The cooperating systems are well-known. This assumption allows programmers to know every object of the cooperating systems. With this, programmers can correctly determine the security level of information.
- The objects and methods for communication among cooperating systems are well-known by the systems. We use JAVA RMI (remote method invocation)[21] to communicate systems. As long as an object in a system is registered, the object's methods can be known and invoked by other systems.

Oorbac^{ar}: OORBAC^{ar} achieves inter-system information flow control through the assistance of RMI objects and methods. The model is designed based on the consideration: when inter-system communication (inter-system information flow) occurs, the security level of the information being passed should be the same as or lower than the security level of the variable receiving the information. Under this consideration, the passing of arguments and return values in RMIs should be controlled to ensure secure inter-system information flows. We define the following secure inter-system information flow protocols, which control RMIs between two systems. In describing the protocols we assume that: (a) the method md1 of the object obj1 in the system ap1 (i.e., ap1.obj1.md1) invokes ap2.obj2.md2, (b) ap1.obj1.md1 passes the arguments arg1, arg2, . . ., argn to ap2.obj2.md2, in which the latter method receives the arguments using the parameters par1, par2, . . , parn, and (c) ap2.obj2.md2 returns the value ap2.obj2.md2.rtv and ap1.obj1.md1 receives the value using ap1.obj1.md1.var1.

First secure inter-system information flow protocol: Every parameter pari in the method ap2.obj2.md2 is associated with an ACL ACL_{pari} and a DSOURCE $DSOURCE_{pari}$. The ACL and DSOURCE of pari define the lowest security level of a variable that can receive the value of pari. The ACL and DSOURCE of a parameter requires that the variable obtaining the parameter's value should possess a security level at least the same as that of the parameter. This prevents low security level variables from obtaining values from high security level arguments passed by ap1.obj1.md1. For example, if a variable var1 in the system ap1 contains a value that can be access by managers or higher rank employees, then employees with a rank lower than a manager should not read var1. If var1 is passed to ap2.obj2.md2 as an argument and the argument is received by the parameter par1, then a variable in the system ap2 that obtains the value from par1 should possess a security level at least the same as a manager's security level.

Second secure inter-system information flow protocol: Pseudo parameters are associated with the method ap1.obj1.md1. They are s_par1, s_par2, . . , s_parn. Every pseudo parameter s_pari in the method ap1.obj1.md1 is associated with an ACL ACL_{s_pari} and a DSOURCE $DSOURCE_{s\_pari}$. The ACL and DSOURCE define that can be passed as arguments to the method ap2.obj2.md2. The ACLs and DSOURCEs of pseudo parameters require that the arguments passed to ap2.obj2.md2 possess security levels at most the same as those of the pseudo parameters. This prevents high sensitive information from being leaked to low sensitive variables. For example, if the ACL and DSOURCE of a parameter par1 in ap2.obj2.md2 state that par1 can be accessed by managers and lower rank employees in the system ap2, then an argument arg1 passed to par1' by the system ap1 should not possess a security level higher than that of a manager. In this case, the pseudo parameter s_par1 of ap1.obj1.md1 limits the security level of arg1.

Third secure inter-system information flow protocol: A pseudo return variable ap2.obj2.md2.s_rtv is associated

with the method ap2.obj2.md2. The pseudo return variable is associated with an ACL $ACL_{ap2.obj2.md2.s\_rtv}$ and a DSOURCE $DSOURCE_{ap2.obj2.md2.s\_rtv}$. The ACL and DSOURCE of ap2.obj2.md2.s_rtv state that the security level of the return variable ap2.obj2.md2.rtv should not be higher than that of ap2.obj2.md2.s_rtv. The ACL and DSOURCE of a pseudo return variable requires that the variable being returned should possess a security level at most the same as the variable receiving the return value. This prevents information leakage similar to the description in the second secure inter-system information flow protocol

Fourth secure inter-system information flow protocol: A pseudo variable ap1.obj1.md1.r_rtv is associated with the method ap1.obj1.md1 to receive the return value of the method ap2.obj2.md2. The pseudo variable is associated with an ACL $ACL_{ap1.obj1.md1.r\_rtv}$ and a DSOURCE $DSOURCE_{ap1.obj1.md1.r\_rtv}$. The ACL and DSOURCE of ap1.obj1.md1.r_rtv state that the variable ap1.obj1.md1.var1 should not possess a security level lower than that of ap1.obj1.md1.r_rtv. The ACL and DSOURCE of a pseudo variable requires that the variable receiving the return value of ap2.obj2.md2 should possess a security level at least the same as that of the pseudo variable. This prevents information leakage similar to the description in the first secure inter-system information flow protocol.

Fifth secure inter-system information flow protocol: Arguments passed to a system should not be passed to another systems. This protocol avoids Trojan horses mentioned in section 1.

The first two secure inter-system information flow protocols ensure that inter-system parameter/argument passing is secure. The third and fourth secure inter-system information flow protocols ensure that inter-system return value passing is secure. And, the fifth secure inter-system information flow protocol ensures that no system will become a Trojan horse. We extended OORBAC according to the secure inter-system information flow protocols above. The extension is described below:

- Add a component IAP (inter-system security protocol) to OORBAC for controlling inter-system information flows.
- Disallow arguments passed to a system to be passed to other systems.
- Extend the security policy (i.e., security conditions and secure flow requirements) mentioned in section 3 as follows.

First extended security condition: If no RMI is invoked in an information flow, the security conditions and secure flow requirements mentioned in section 3 should be fulfilled to make the flow secure.

Second extended security condition: If an RMI appears in an information flow, the following RMI secure flow requirements should be fulfilled.

First RMI secure flow requirement: If a variable vari in the method ap2.objm.mdn reads the value of the parameter pari, the conditions below should be true.

- $(RACL_{vari} \subseteq RACL_{pari}) \wedge (ap2.objm.mdn \in RACL_{pari})$
- $WACL_{vari} \supseteq (DSOURCE_{pari} \cup \{ap2.objm.mdn\})$

This RMI secure flow requirement implements the first secure inter-system information flow protocol. The two conditions above are actually the two secure flow requirements mentioned in section 3. They respectively control read and write access of the parameter pari, which requires that the variables receiving the arguments passed by the remote method should be at least the same restricted as the arguments.

Second RMI secure flow requirement: If a variable argi in the system ap1 is passed as the i^{th} argument when invoking the method ap2.obj2.md2, the conditions below should be true (remember that the remote invocation is performed by the method ap1.obj1.md1).

- $(RACL_{s\_pari} \subseteq RACL_{argi}) \wedge (ap1.obj1.md1 \in RACL_{argi})$
- $WACL_{s\_pari} \supseteq (DSOURCE_{argi} \cup \{ap1.obj1.md1\})$

This RMI secure flow requirement implements the second secure inter-system information flow protocol. The two conditions respectively control read and write access of the argument argi, which requires that the arguments passed to the remote method should be at most the same restricted as the pseudo parameters of the method ap1.obj1.md1.

Third RMI secure flow requirement: If a variable vari in the method ap2.obj2.md2 is returned, the conditions below should be true.

- $(RACL_{ap2.obj2.md2.s\_rtv} \subseteq RACL_{vari}) \wedge (ap2.obj2.md2 \in RACL_{vari})$
- $WACL_{ap2.obj2.md2.s\_rtv} \supseteq (DSOURCE_{vari} \cup \{ap2.obj2.md2\})$

This RMI secure flow requirement implements the third secure inter-system information flow protocol. The

two conditions above respectively control read and write access of the return variable vari, which requires that the return value of a remote method should be at most the same restricted as the pseudo return variable ap2.obj2.md2.s_rtv.

Fourth RMI secure flow condition: If a variable vari in the method ap1.obj1.md1 receives the return value, the conditions below should be true.

- $(RACL_{vari} \subseteq RACL_{ap1.obj1.md1.r\_rtv}) \wedge (ap1.obj1.md1 \in RACL_{ap1.obj1.md1.r\_rtv})$
- $WACL_{vari} \supseteq (DSOURCE_{ap1.obj1.md1.r\_rtv} \cup \{ap1.obj1.md1\})$

This RMI secure flow requirement implements the fourth secure inter-system information flow protocol. The two conditions above respectively control read and write access of the variable vari that receives the return value of the method ap2.obj2.md2, which requires that the variable receiving the return value from a remote method should be at least the same restricted as the pseudo variable ap1.obj1.md1.r_rtv.

Fifth RMI secure flow condition: If a variable argi in the system ap1 is passed as an argument to a remote method, the DSOURCE of argi should not contain any method belonging to the systems other than ap1. This RMI secure flow requirement implements the fifth secure inter-system information flow protocol.

Using OORBAC^{ar}: We use two cooperating systems to depict the use of OORBAC^{ar}. They are the case history management system and the doctor management system. The case history management system manages patients' case histories. In the system, a patient is assigned to a doctor. A doctor can read and change the case histories of the patients assigned to him. On the other hand, a doctor can only read the case histories of the patients not assigned to him. The system offers remote methods for the doctor management system to retrieve the case histories of patients assigned to a doctor.

The doctor management system manages doctors' ranks. A manager of the hospital can read and write (change) the rank of a doctor. To change a doctor's rank, the case histories of the patients assigned to the doctor should be taken as a reference. The case histories are retrieved from the case history management system through RMI. The retrieved case histories can only be read by the manager. We disallow doctors to access the case histories in the doctor management system. Nevertheless, a doctor can access the case histories in the case history management system.

The cooperating systems are implemented in APPENDIX 1 using the language of OORBAC^{ar}, which is named OORBACL^{ar}. The first system in the appendix is the case history management system. The second is the doctor management system. The first system offers a remote methodcase_history_service. get_case_history_doctor for the second one to invoke. The remote method is defined within the class case_history_service (lines 16 through 21 in APPENDIX 1). The object HSOBJ in main offering the remote method is defined and registered (lines 20.3 through 20.4). The second system invokes the remote method offered by the object HSOBJ in the method doctor_rank_mng.change_doctor_rank (lines 33.5 through 33.6).

To ensure inter-system information flow security, both systems include an IAP. The IAP in the first system (lines 7 through 8) defines ACLs to fulfill the first and third RMI secure flow requirements mentioned in section 4.1. The IAP in the second system (lines 24 through 25) defines ACLs to fulfill the second and fourth RMI secure flow requirements. We trace the IAPs to explain the RMI secure flow requirements.

Line 7.1.1 defines the ACL of the parameter of the remote methodcase_history_service. get_case_history_doctor offered by the first system. A variable receiving the parameter should be at least the same restricted as the parameter (i.e., the first RMI secure flow requirement should be fulfilled). For example, using the attribute case_history_service.patient_case_history to receive the parameter is considered non-secure because the first RMI secure flow requirement is false (please compare the ACLs in lines 5.3.1 and 7.1.1). Line 7.3.1 defines the ACL of the pseudo return value of the remote method offered by the first system. A variable being returned should be at most the same restricted as the pseudo return value (i.e., the third RMI secure flow requirement should be fulfilled). For example, returning the variable doctor_mng.doctor is considered non-secure because the third RMI secure flow requirement is false (please compare the ACLs in lines 5.3.2 and 7.3.1).

Line 24.1.1 defines the referenced name of the pseudo parameter for the second system. Line 24.3.1 defines the ACL of the pseudo parameter when invoking the remote method rmiObj.get_case_history_doctor offered by the first system. An argument sending to the pseudo parameter should be at most the same restricted as the parameter (i.e., the second RMI secure flow requirement should be fulfilled). For example, using the attribute doctor_mng.doctor as an argument is considered non-secure because the second RMI secure flow requirement is false (please compare the ACLs in lines 22.3.1 and

24.3.1). Line 24.5.1 defines the ACL of the pseudo variable that can receive the remote method's return value. A variable receiving the return value should be at least the same restricted as the pseudo variable (i.e., the fourth RMI secure flow requirement should be fulfilled). For example, using the variable manager.doctor_name to receive the return value is considered non-secure because the fourth RMI secure flow requirement is false (please compare the ACLs in lines 22.3.2 and 24.5.1).

EVALUATION

OORBAC^{ar} was embedded in the JAVA language to produce the language OORBACL^{ar}. The syntax of OORBACL^{ar} can be informally understood by tracing APPENDIX 1. We implemented an environment for OORBACL^{ar}, which is primarily composed of an OORBACL^{ar} preprocessor. The preprocessor translates OORBACL^{ar} program into a pure JAVA program, which is composed of the original JAVA program and a security checker. During program execution, the security checker checks the security of every information flow in the original JAVA program. The security checker for OORBACL^{ar} and that for OORBACL[9] are nearly the same. The only difference is the management of pseudo parameters, pseudo return variables, and pseudo variables to receive return values. We evaluated OORBAC^{ar} using the following cooperating systems:

- An employee management system, which handles the hiring, promoting, downgrading, salary computing, firing, and retiring of employees. It also manages the responsibility of employees in a company.
- An inventory management system, which manages the current levels and re-order levels of items. When a problem occurs (e.g., an item's current level is under its re-order level but no re-order is initiated), the inventory management system communicates with the employee management system to identify the employees that are responsible for the problem.
- A report generator that periodically produces a report showing the sold amounts of items. Clearly, the sold amounts of items should be retrieved from the inventory management system (i.e., the two systems will communicate).

In evaluating OORBAC^{ar}, we collected non-secure information flows induced by inter-system communications. We selected ten students to program the above systems. We then required them to execute their programs using five different sets of input data and collected the number of non-secure inter-system information flows. To ensure that OORBAC^{ar} is capable of identifying non-secure inter-system information flows, we required the students to inject five non-secure inter-system information flows in each program. The experiment result is shown in Fig. 1. When we checked the experiment results, every injected non-secure ones were identified. Note that the identified non-secure inter-system information flows were more then those injected because students usually committed errors. According to Fig. 1, we conclude that OORBAC^{ar} is capable of identifying non-secure inter-system information flows.

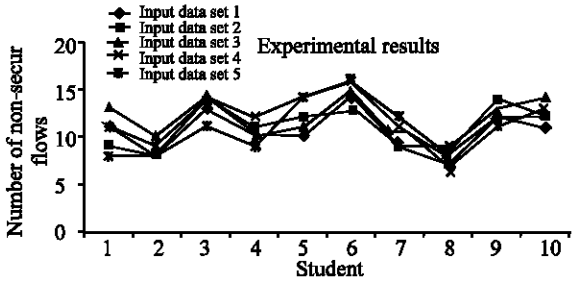

Fig.1: Experiment result

CONCLUSIONS

Our survey reveals that no existing information flow control model controls information flows among systems. In our opinion, controlling inter-system information flows (i.e., controlling information flows among systems) is essential because systems may exchange information during execution. We thus extended our previous model OORBAC (object-oriented role-based access control) to offer the control. The extension is based on the consideration: when inter-system information flow occurs, the security level of the information being passed should be the same as or lower than the security level of the variable receiving the information. We named the extended model OORBAC^{ar}, in which the super-script ar means that the model prevents both intra-system and inter-system information leakage. OORBAC^{ar} uses the mechanisms provided by OORBAC to control intra-system information flows and uses JAVA RMI (remote method invocation) and the following protocols to control inter-system information flows.

- Every parameter in the invoked remote method is associated with an ACL (access control list) and DSOURCE (data source). The ACL and DSOURCE define the lowest security level of a variable that can receive the value of the parameter.

- Pseudo parameters are associated with the method invoking the remote method. Every pseudo parameter is associated with an ACL and a DSOURCE, which define the most sensitive argument that can be passed to the remote method.
- A pseudo return variable is associated with the remote method being invoked. The pseudo return variable is associated with an ACL and a DSOURCE, which define the most sensitive information that can be returned by the remote method.
- A pseudo variable is associated with the method invoking the remote method. The pseudo variable is associated with an ACL and a DSOURCE, which define the lowest security level of a variable that can receive the return value of the remote method.
- Arguments passed to a system cannot be passed to another systems. This avoids Trojan horses among systems.

We evaluated OORBAC^{ar} and found that it did identify non-secure inter-system information flows.

Appendix 1: The two cooperating systems mentioned in section 4.2 are coded below using OORBACL^{ar}.

```
System 1: Case history management system
1ICRel assigned {
1.1   classes {doctor; patient}
1.2   cardinality {doctor: 1, patient: *} /* A doctor can have many patients but a patient can be assigned to only one doctor */
1.3   modality {doctor: O, patient: M} /* A doctor can have no patient but a patient should be assigned to a doctor */
1.4   permissions {
1.4.1 p1 {doctor.get_patient_case_history, patient.get_case_history};
1.4.2 p2 {doctor.change_patient_case_history, patient.change_case_history};
1.5   }
1.6   roles {
1.6.1 r1 {p1, p2};
1.7   }
1.8   UA { /* User assignment, which assigns objects to roles. */
1.8.1 doctor: r1; /* Patients have no rights on doctors. */
1.9   }
1.10  attributeACLs { /* An ACL is composed of an RACL and a WACL. A semicolon separates them, in which the former is the RACL and the latter the
WACL. */
1.10.1          patient.case_history {patient.get_case_history, doctor.get_patient_case_history; patient.change_case_history,
doctor.change_patient_case_history};
1.10.2          doctor.patient_case_history {doctor.get_patient_case_history; doctor.get_patient_case_history, patient.get_case_history};
1.10.3          doctor.new_patient_case_history {doctor.change_patient_case_history, patient.change_case_history; doctor.change_patient_case_history};
1.11  } /* end of attributeACLs */
2} /* end of ICRel */
3ICRel not_assigned {
3.1   classes {doctor; patient}
3.2   cardinality {doctor: *, patient: *} /* A many-to-many relationship */
3.3   modality {doctor: M, patient: M}
3.4   permissions {
3.4.1 p1 {doctor.get_patient_case_history, patient.get_case_history};
3.5   }
3.6   roles {
3.6.1 r1 {p1};
3.7   }
4}
5ICRel remoteMethodInvocation {
5.1   class {doctor, patient, doctor_mng, case_history_service}
5.2   //. . .
5.3   attributeACLs {
5.3.1 case_history_service.patient_case_history {case_history_service.get_case_history_doctor, patient.get_case_history; patient.get_case_history}; /* This
variable cannot receive the parameter */
5.3.2 doctor_mng.doctor {doctor_mng.search_doctor; NONE}; /* This variable cannot be returned */
5.3.3 // . . .
5.4   }
5.5   // . . .
6} // end of ICRel
7IAP { /* inter-system information flow control protocols are defined below. This system offers remote methods for invocation. Therefore, parameter ACLs
and pseudo return value ALCs should be defined. */
7.1   parameterACLs {
7.1.1 case_history_service.get_case_history_doctor.doctor_name {case_history_service.get_case_history_doctor, doctor_mng.search_doctor; NONE};
7.2   }
7.3   pseudoReturnValueACLs {
7.3.1 case_history_service.get_case_history_doctor {case_history_service.get_case_history_doctor; case_history_service.get_case_history_doctor,
patirnt.get_case_history}
7.4   }
8} // end of IAP
/* ---------- JAVA program below ----------*/
9// import the necessary files
10    class doctor {
10.1  public String patient_case_history, new_patient_case_history;
10.2  public void get_patient_case_history(patient p1){
```

```
10.2.1          patient_case_history = p1.get_case_history();
10.3  }
10.4  public void change_patient_case_history(patient p1){
10.4.1          // set up new_patient_case_history
10.4.2          p1.change_case_history(new_patient_case_history);
10.5  }
11    } /* end of class "doctor" */
12    class patient {
12.1  public String case_history;
12.2  public int checked; // if it is set, the patient's case history has been checked
12.3  public String get_case_history(){
12.3.1          return case_history;
12.4  }
12.5  public void change_case_history(String new_case_history){
12.5.1          case_history = new_case_history;
12.6  }
12.7  public void set_check_mark(){
12.7.1          checked = 1;
12.8  }
13    } // end of class
14    class doctor_mng{// this class offers methods to manage the two doctor arrays
14.1  public String doctor[]; //The array for doctor names
14.2  public doctor doc[]; // The array for doctor objects
14.3  /* there is a one-to-one mapping between the two arrays above. That is, the object of the doctor with a name doctor[i] is stored in doc[i] */
14.4  public doctor search_doctor(String doctor_name){
14.4.1          // search the doctor array and identify the doctor named doctor_name
14.4.2          // return the corresponding doctor object stored in the doc array
14.5  }
15    }

/* ---------- The interface case_history_interface and the class case_history_service are for RMI */
16    public interface case_history_interface extends java.rmi.Remote {
16.1  public String get_case_history_doctor(String doctor_name);
17    }
18    public class case_history_service extends UnicastRemoteObject implements case_history_interface {
18.1  public doctor d1;
18.2  public patient p1;
18.3  public String patient_case_history;
18.4  public String get_case_history_doctor(String doctor_name){
18.4.1          d1 = case_history_management.dmng1.search_dortor(doctor_name);
18.4.2          while (TRUE){
18.4.2.1        p1 = getObject(ICRel assigned, doctor d1); /* getObject is an OORBAC[sr] statement. Here the statement retrieves an object existing in a
an assigned session with the object d1 */
18.4.2.2        if (p1.checked !=1){
18.4.2.2.1      p1.checked = 1;
18.4.2.2.2      patient_case_history = p1.get_case_history();
18.4.2.2.3      return patient_case_history;
18.4.2.3        }
18.4.3          }
18.4.4          return end of patient;
18.5  }
19    } // end of class
// RMI object HSOBJ is defined below
20    class case_history_management {
20.1  public static doctor_mng dmng1;
20.2  public case_history_service HSOBJ; // The object HSOBJ offers RMI
20.3  public void main(){
20.3.1          // instantiate the object HSOBJ and register the object for RMI service
20.3.2          // The registration is achieved by invoking the method Naming.rebind
20.3.3          // instantiate dmng1
20.3.4          /* instantiate doctors to set up the array dmng1.doctor[] and dmng1.doc[] */
20.3.5          //. . .
20.4  }
21    }


System 2. Doctor management system
/* An incomplete ICRel defined below will be used to explain the RMI secure flow requirements */
22    ICRel remoteMethodInvocation {
22.1  class {manager, doctor, doctor_mng, doctor_rank_mng}
22.2  //. . .
22.3  attributeACLs {
22.3.1          doctor_mng.doctor {doctor_mng.search_doctor; NONE}; /* This variable cannot be passed as an argument */
22.3.2          manager.doctor_name {manager.change_doctor_rank, doctor.get_name; manager.change_doctor_rank};/* This variable cannot receive the
return value */
22.3.3          // . . .
22.4  }
```

```
22.5  // . . .
23    } // end of ICRel
24    IAP { /* inter-system information flow control protocols are defined below. This system invokes remote methods. Therefore, pseudo parameter ACLs and ACLs for pseudo variables that receive RMI return values should be defined. */
24.1  pseudoParameters {
24.1.1            rmiObj.get_case_history_doctor(par1);
24.2  }
24.3  pseudoParameterACLs {
24.3.1            rmiObj.get_case_history_doctor.par1        {doctor_rank_mng.change_doctor_rank,        manager.change_doctor_rank; doctor_rank_mng.change_doctor_rank, manager.change_doctor_rank, doctor.get_name}
24.4  }
24.5  pseudoVariableACLs {
24.5.1            rmiObj.get_case_history_doctor {doctor_rank_mng.change_doctor_rank; NONE};
24.6  }
25    } // end of IAP
/* ---------- JAVA program below ---------- */
26    // import necessary files
27    class manager {
27.1  public String doctor_name;
27.2  public void change_doctor_rank(doctor_rank_mng dr1, doctor d1){
27.3  // change the rank of doctor d1
27.3.1            doctor_name = d1.get_name();
27.3.2            dr1.change_doctor_rank(doctor_name);
27.4  }
28    } // end of class manager
29    class doctor {
29.1  int rank;
29.2  String name;
29.3  public String get_name(){
29.3.1            return name;
29.4  }
29.5  public int get_rank(){
29.5.1            return rank;
29.6  }
29.7  public void change_rank(int new_rank){
29.7.1            rank = new_rank;
29.8  }
30    } // end of class
31    class doctor_mng{// this class offers methods to manage the two doctor arrays
31.1  public String doctor[]; //The array for doctor names
31.2  public doctor doc[]; // The array for doctor objects
31.3  public doctor search_doctor(String doctor_name){
31.3.1            // search the doctor array and identify the doctor named doctor_name
31.3.2            // return the corresponding doctor object stored in the doc array
31.4  }
32    }
33    class doctor_rank_mng { /* get case histories of patients assigned to a doctor */
33.1  public String patient_case_history;
33.2  public int doctor_rank, new_doctor_rank;
33.3  public doctor d1;
33.4  public case_history_service rmiObj;
33.5  public void change_doctor_rank(String doctor_name) {
/* the following operations invoke remote methods */
33.5.1            // Find the object HSOBJ offering the RMI service and set the object to rmiObj
33.5.2            // The object can be found using the method Naming.lookup
33.5.3            patient_case_history = rmiObj.get_case_history_doctor(doctor_name);
33.5.4            while (patient_case_history != end of patient){
33.5.4.1          //check and record the patient case history
33.5.4.2          patient_case_history = rmiObj.get_case_history_doctor(doctor_name);
33.5.5             }
33.5.6             d1 = doctor_management.dmng1.search_doctor(doctor_name);
33.5.7            doctor_rank = d1.get_rank();
33.5.8            /* set new rank of doctor to new_doctor_rank according to the case histories and the current rank */
33.5.9            d1.change_rank(new_doctor_rank);
33.6  }
34    } // end of class
35    class doctor_management {
35.1  public static doctor_mng dmng1;
35.2  public void main(){
35.2.1            // instantiate dmng1
35.2.2            /* instantiate doctors and set up the array dmng1.doctor[] and dmng1.doc[] */
35.2.3            //. . .
35.3  }
36    }
```

REFERENCES

1. Samarati, P., E. Bertino, A. Ciampichetti and S. Jajodia, 1997. Information Flow control in object-oriented systems. IEEE Transactions on Knowledge and Data Engineering, 9: 524-538.
2. Bertino, E., Sabrina de Capitani di Vimercati, E. Ferrari, P. Samarati, 1998. Exception-based information flow control in object-oriented systems. ACM Transactions on Information and System Security, 1: 26-65.
3. Ferrari, E., P. Samarati, E. Bertino and S. Jajodia, 1997. Providing flexibility in information control for object-oriented systems. In: Proceedings of the 13th IEEE Symposium on Security and Privacy, pp: 130-140.
4. Maamir A. and A. Fellah, 2003. Adding Flexibility in Information Flow Control for Object-Oriented Systems Using Versions, Intl. J. Software Engineering and Knowledge Engineering, 13: 313-326.
5. Bell, D.E.and L.J. LaPadula, 1976. Secure computer systems: unified exposition and multics interpretation. Technique Report, Mitre Corporation, In:http://csrc.nist.gov/publications/history/bell76.pdf
6. Brewer, D.F.C. and M.J. Nash, 1989. The chinese wall security policy. In: Proceedings of the 5'th IEEE Symposium on Security and Privacy, pp: 206-214.
7. Denning, D.E., 1976. A lattice model of secure information flow. Communications of the ACM, 19: 236-243.
8. Denning, D.E. and P.J. Denning, 1977. Certification of program for secure information flow. Communications of the ACM, 20: 504-513.
9. Chou, S.C., Embedding role-based access control model in object-oriented systems to protect privacy, J. Sys. Software.
10. Izaki, K., K. Tanaka, M. Takizawa, 2001. Information flow control in role-based model for distributed objects. In: Proceedings of the 8'th International Conference on Parallel and Distributed Systems, pp: 363-370.
11. McIlroy M.D., J.A. Reeds, 1992. Multilevel Security in the UNIX Tradition. Software - Practice and Experience., 22: 673-694.
12. Myers, A.C., 1999. JFlow: Practical mostly-static information flow control. In: Proceedings of the 26'th ACM Symposium on Principles of Programming Language, pp: 228-241.
13. Myers, A.C. and B. Liskov, 1997. A Decentralized model for information flow control. In: Proceedings of the 17th ACM Symposium on Operating Systems Principles, pp: 129-142.
14. Myers, A. and B. Liskov, 1998. Complete, safe information flow with decentralized labels. In: Proceedings of the 14th IEEE Symposium on Security and Privacy, pp: 186-197.
15. Myers, A. and B. Liskov, 2000. Protecting privacy using the decentralized label model. ACM transactions on software engineering and methodology, 9 : 410-442.
16. Sandhu, R.S., E.J. Coyne, H.L. Feinstein and C.E. Youman, 1996. Role-based access control models. IEEE Computer, 29: 38-47.
17. Sandhu, R., 1996. Role hierarchies and constraints for lattice-based access controls. In: Proceedings of the Fourth European Symposium on Research in Computer Security, pp: 65-79.
18. Nyanchama, M. And S. Osborn, 1995. Modeling mandatory access control in role-based security systems. Database Security IX: Status and Prospects, pp: 129-144.
19. Osborn, S., 1997. Mandatory access control and role-based access control revisited. In: Proceedings of the Second ACM Workshop on Role-Based Access Control, 31-40.
20. Osborn, S., R. Sandhu and Q. Munawer, 2000. Configuring role-based access control to enforce mandatory and discretionary access control policies. ACM transactions on information and system security, 3: 85-106.
21. Niemeyer P. and J. Knudsen, 2000. Learning JAVA, O'REILLY.