Int. Computer Symposium, Dec. 15-17, 2004, Taipei, Taiwan.

Preventing Information Leakage within Object-Oriented Systems Using RBAC-Based Model

Shih-Chien Chou and Chien-Jung Wu
Department of Computer Science and Information Engineering
National Dong Hwa University, Hualien 974, Taiwan
E-Mail: scchou@mail.ndhu.edu.tw

Abstract - *This paper proposes a role-based access control (RBAC) model to prevent information leakage within object-oriented systems. It is named MRBAC (modified role-based access control) because it is a modification of RBAC96.*

Keywords: Information security, access control, information flow control, prevent information leakage

1. Introduction

Access control has been recognized as important to prevent information leakage within a system. The control within a system is a language-based security model [1]. That is, an access control model should be embedded in a language to prevent information leakage within a program implemented by the language. The control can be achieved through information flow control [2-25]. We involved in the research of controlling information flows within object-oriented systems for years and developed an RBAC-based models for information flow control called MRBAC (modified RBAC), because it is a modification of RBAC96 [26]. This paper proposes MRBAC, which can adapt to dynamic object state change and dynamic role change as described below.

a. Adapt to dynamic object state change. An object state is *a snapshot of objects and object relationships at a certain time point*. Therefore, object state changes when objects or object relationships change. We use an example to describe the need of the adaptation. Suppose an employee may be a manager or a worker. A manager can read the personal information of a worker assigned to him but cannot read that information of other workers. Assume that initially the worker “w1” is assigned to the manager “m1”. With this object state, “m1” is allowed to read the personal information of “w1”. Suppose after a period of time, “w1” is re-assigned to the manager “m2”. With this object state, “m1” is no longer allowed to read the personal information of “w1”. The example reveals that *changing object state results in changing access rights*. To adapt to dynamic object state change, access rights should be allowed to change dynamically.

b. Adapt to dynamic role change. In an object-oriented system, an object or object method plays a role. Dynamic role change refers to changing object or method’s role during program execution. For example, a worker “w1” may be promoted to be a manager (i.e., the object “w1” changes role from “worker” to “manager”). When role changes, access rights will also be changed. For example, access rights of “w1” will change when he changes role from “worker” to “manager”. To adapt to dynamic role change, access rights should be allowed to change dynamically.

2. Related Work

Traditional access control is achieved by access control matrix (ACM) [27]. A subject can access an object if the required access right appears in the matrix. ACM allows only static access control [28-29]. On the other hand, DACM (dynamic access control matrix) [28] dynamically grants access rights, which allows dynamic access right allocation.

MAC is useful in access control. An important milestone of MAC is that proposed by Bell&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&LaPadula’s model has been generalized into the lattice model [8-10] (see [30] for a survey of lattice models). In the typical lattice model proposed by Denning [8-9], the “can flow” relationship controls information flows and the join operator avoids Trojan horses.

The model in [11] controls information flows in object-oriented systems. It uses ACLs of objects to compute ACLs of executions (which may consist of one or more methods). 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. 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 access 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 [5]. 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.

Section 1 states that RBAC is useful in information flow control because it is a super set of DAC and MAC. Since the original design of RBAC is not for access control within object-oriented systems, most features mentioned in section 1 are not offered by the general cases of RBAC. 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.

3. The Model

We first describe RBAC96, which is the basis of MRBAC and then describe MRBAC in this section.

3.1 RBAC96

RBAC96 [26] is composed of a set of permissions, a set of roles, a set of users, a set of sessions, a set of permission to role assignment, a set of user to role assignment, a function that maps sessions to users, a function that maps sessions to roles, a role hierarchy, and a set of constraints. Among the components, a role is composed of a set of permissions. Roles are structured using the "$\geq$" relationship. If a relationship "x $\geq$ y" exists, "x" possesses all the permissions of "y". A user is a human being or an agent. Users can create sessions, during which a user playing a role possesses the permissions of the role. The permissions will be revoked when the session ends or the user does not play the role. Since permissions are not assigned to users, adjustment of user permissions can be achieved through role assignment.

3.2 MRBAC

A difficult job to solve by MRBAC is adapting to dynamic object state change. We found that class relationships [31-32] are useful in the adaptation. In using class relationships for the adaptation, every relationship should be associated with an *access control policy*. Objects linked by a relationship should obey the relationship's access control policy. When the relationship linking two or more objects changes, the access control policy to obey by the objects changes. This corresponds to adapting to dynamic object state change. For example, in the manager/worker example mentioned in section 1, we can define two reflexive relationships for the class "employee", which are "assigned" and "not_assigned". When the worker employee "w1" is assigned to the manager employee "m1", they are linked by an "assigned" relationship. In this relationship, the access control policy allows "m1" to read the personal information of "w1". When "w1" is re-assigned to another manager, the "assigned" relationship instance between "w1" and "m1" should be removed and a "not_assigned" relationship should be established to link them. In this relationship, the access control policy disallows "m1" to read the personal information of "w1".

MRBAC defines an *instance of a class relationship* as a *session*. When a class relationship is instantiated to link objects, a session is established among the objects. With this definition, changing object state corresponds to changing sessions. When objects change session, the access control policy for the objects to obey changes. This change accomplishes the adaptation of dynamic object state change.

As described above, every class relationship is associated with an access control policy. The access control policies of all class relationships in a system constitute the access control policy of the system. In MRBAC, information flows within a session is allowed whereas those among sessions are prohibited. Moreover, information flows among objects within a session should obey the policy of the class relationship from which the session is instantiated.

After defining sessions and their access control policies, permissions and roles should be defined. A permission is composed of a variable and its access rights. The access right of a variable may be "R" (for "read"), "W" (for "write"), or "RW" (for both "read" and "write"). A role is a set of permissions. *It is played by an object method* because methods manipulate variables. An object is a *composite role* because objects consist of methods. Since roles in MRBAC are object methods, dynamically changing a role within an object causes other roles (i.e., methods) in the same object to change. For example, when a worker becomes a manager, every method in the worker object change role. Therefore, *changing role in MRBAC corresponds to changing composite role*.

In addition to the components mentioned above, MRBAC associates two more components with each variable to facilitate read and write access control. They are the senders (SENDER) and data sources (DSOURCE) of a variable. SENDER records the methods that pass a variable as an argument. DSOURCE records the methods from which a variable's data are derived. For example, 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. MRBAC is formally defined below:

Definition 1. *MBRAC* = (*RELATIONSHIP*, *SESSION*, *SESSION_OBJECT_MAPPING*, *CONSTRAINT*, *DSOURCE*, *SENDER*), in which

a. *RELATIONSHIP* is a set of class relationships. A relationship can be instantiated to create sessions. Definition 2 defines a class relationship.
b. *SESSION* is a set of sessions. Each session is an instance of a class relationship.
c. *SESSION_OBJECT_MAPPING* is a set of functions, each of which maps a session to the objects that are within the session.
d. *CONSTRAINT* is the set of constraints.
e. *DSOURCE* is the set of data sources.
f. *SENDER* is the set of senders.

Definition 2. The *RELATIONSHIP* component in MRBAC is the set of class relationships. A class relationship rel_i is defined below:

rel_i = (*NAME*, *CLASS*, *METHOD*, *VARIABLE*, *PERMISSION*, *ROLE*, *METHOD_ROLE_MAPPING*, *COMPOSITE_ROLE*, *DECLASSIFICATION*), in which

a. *NAME* is the name of the relationship.
b. *CLASS* is the set of classes linked by the relationship. A composite role name is associated with a class. Instances of the class play the composite role.
c. *METHOD* is the set of class methods. A method belongs to a class.
d. *VARIABLE* is the set containing attributes, method variables, and method return values. A variable belongs to a class.
e. *PERMISSION* is the set of permissions in the relationship.
f. *ROLE* is the set of roles in the relationship. *A role is played by a method* and is composed of a set of permissions.
g. *MEDTOH_ROLE_MAPPING* is a set of functions, each of which maps a method to a role (which means that the method plays the role).
h. *COMPOSITE_ROLE* is the set of composite roles. A composite role is composed of roles (i.e., methods). Composite roles are used in role change.
i. *DECLASSIFICATION* is a set of special variables for declassification.

3.3 Secure information flows in MRBAC

Information flows in a system includes direct flows and indirect flows. Indirect flows refer to accessing information via the third one. For example, after the method "md1" reads the information of "var1" into "var2", a method that read "var2" corresponds to indirectly reading "var1" via "md1". Both direct and indirect flows should be secure.

In an object-oriented system, direct information flows includes the flows *among methods* and those *within a method*. Information flows among methods are induced by statements that involve messages. Other information flows are flows within a method. In the following discussion, we use the term "method invocation" to replace the term "message".

When a method invocation from "obj1.md1" to "obj2.md2" appears, "obj1" and "obj2" should be within a session. Otherwise, the invocation is not allowed. Suppose "obj1" and "obj2" are within a session and "obj1.md1" passes the argument list "(arg1, arg2, . . ., argn)" to the parameter list "(par1, par2, . . ., parn)" of "obj2.md2". Then, the access rights, DSOURCE, and SENDER of every argument in the argument list should be copied to the corresponding parameter in the parameter list. This copying is necessary because a parameter receiving an argument inherits the security level of the argument. After the copying, the invoked method is executed and every information flow within the method should be secure. To ensure secure information flows within a method, the following *secure information flow conditions* should be true when the value derived from the variables "var1", "var2", . . ., "varn" is assigned to the variable "d_var" (suppose the derivation appears in the method "mdx" playing the role "role_{mdx}").

First secure information flow condition: $(\{\{var1, R\}, \{var2, R\}, \ldots, \{varn, R\}\} \subseteq role_{mdx}) \wedge (\{var1, R\} \in (\cap_i role_{sender\_var1(i)}) \wedge (\{var2, R\} \in (\cap_i role_{sender\_var2(i)}) \wedge \ldots \wedge (\{varn, R\} \in (\cap_i role_{sender\_varn(i)})$

Second secure information flow condition: $(\{d\_var, W\} \in role_{mdx}) \wedge (\{d\_var, W\} \in (\cap_{i,j}$

$role_{dsource\_var(i,j)})) \wedge (\{d\_var, W\} \in (\cap_{i,j} role_{sender\_var(i,j)}))$

The permission "{var1,R}" means "var1" is allowed to be read whereas "{d_var,W}" means "d_var" is allowed to be written. The notation "$role_{sender\_var1(i)}$" is the role played by the ith method in the SENDER of the variable "var1". The notation "$\cap_i role_{sender\_var1(i)}$" is the intersection of roles' permissions, in which the roles are played by the methods in the SENDER of the variable "var1". The notation "$role_{dsource\_var(i,j)}$" is the role played by the jth method in the DSOURCE of the ith variable that derives "d_var". The notation "$role_{sender\_var(i,j)}$" is the role played by the jth method in the SENDER of the ith variable that derives "d_var". The notation "$\cap_{i,j} role_{dsource\_var(i,j)}$" is the intersection of roles' permissions, in which the roles are played by the methods in the DSOURCEs of the variables deriving "d_var". The notation "$\cap_{i,j} role_{sender\_var(i,j)}$" is the intersection of roles' permissions, in which the roles are played by the methods in the SENDERs of the variables deriving "d_var".

The first secure information flow condition controls read access. It requires that the method "mdx" should be allowed to read the variables deriving "d_var" because "mdx" directly read the variables. It also requires that the senders of a variable should be allowed to read the variable because the senders indirectly read the variable. The second secure information flow condition controls write access. It requires that the method "mdx" as well as every method in the DSOURCEs and SENDERs of the variables deriving "d_var" must possess a permission to write "d_var".

The two secure information flow conditions ensure secure *direct* information flows. As mentioned above, the security of *indirect* information flows should also be ensured. Ensuring this security corresponds to avoiding Trojan horses. We use the join operation [2-5] (the symbol is "$\oplus$") to avoid Trojan horses. If the value of the variable "var3" is derived from the variables "var1" and "var2", the access rights of "var3" will be changed by the join operation.

To explain the join operation, we let "R_{var1}" and "R_{var2}" be respectively the sets of methods allowed to read "var1" and "var2"; "W_{var1}" and "W_{var2}" be respectively the sets of methods allowed to write "var1" and "var2"; "$DSOURCE_{var1}$" and "$DSOURCE_{var2}$" be respectively the DSOURCEs of "var1" and "var2"; and "$SENDER_{var1}$" and "$SENDER_{var2}$" be respectively the SENDERs of "var1" and "var2". When "var3" is derived from "var1" and "var2", then "R_{var3}", "W_{var3}", "$DSOURCE_{var3}$", and "$SENDER_{var3}$" will be set by the result of "var1 $\oplus$ var2" as defined in Definition 3. Here "R_{var1}/R_{var2}" and "W_{var1}/W_{var2}" can be extracted from the permissions containing "var1/var2". After the join, the resulting "R_{var3}" and "W_{var3}" should be used to change the permissions containing "var3".

Definition 3: If "var3" is derived from "var1" and "var2" within the method "mdx", then "var1 $\oplus$ var2" will set "R_{var3}", "W_{var3}", "$DSOURCE_{var3}$", and "$SENDER_{var3}$" as follows:

$$R_{var3} = R_{var1} \cap R_{var2}$$
$$W_{var3} = W_{var1} \cup W_{var2}$$
$$DSOURCE_{var3} = DSOURCE_{var1} \cup DSOURCE_{var2} \cup \{mdx\}$$
$$SENDER_{var3} = SENDER_{var1} \cup SENDER_{var2}$$

The join operation trusts less or the same set of readers, which will not lower down security level of the derived data. The operation trusts more writers. It is reasonable because a writer that can write a variable should be considered a trusted data source for the data derived from the variable. Below we prove that the join operation avoids Trojan horses.

Lemma 1: The join operation avoids Trojan horses.

Proof: A Trojan horse results when a method "md2" leaks the information retrieved from "md1" to "md3" in which "md2" is allowed to read the information of "md1" whereas "md3" is not. To prove that Trojan horses are avoided, we let "var1" be a variable in "md1" which can be read by the methods in the set "R_{var1}". According to the above assumption, "md2" is in the set "R_{var1}" but "md3" is not. We also let "var2" be a variable in "md2" whose value is derived from "var1" and other variables. After the derivation, "R_{var2}" is modified by the join operation.

Suppose that a Trojan horse exists among "md1", "md2", and "md3". Without loss of generality, we assume that "md3" can read "var2". If this assumption is true, "md3" is within "R_{var2}". However, according to the join operation in Definition 3, "R_{var2}" is the intersection of "R_{var1}" and other sets of methods because "var2" is derived from "var1" and other variables. Since "md3" is not in "R_{var1}", "md3" is not in "R_{var2}". This contradicts the assumption. #

4. Conclusions

This paper proposes a role-based information flow control model for object-oriented systems. It is a modification of RBAC96, which is named MRBAC (modified RBAC). It uses secure information flow conditions to ensure information flows security. Moreover, it can adapt to dynamic object state change and dynamic role change.

References

[1] A. Sabelfeld, and A. C. Myers, "Language-Based Information-Flow Security. IEEE Journal on Selected Areas in Communications", vol. 21, no. 1, pp. 5-19, 2003

[2] A. C. Myers, "JFlow: Practical Mostly-Static Information Flow Control", *Proc. 26'th ACM Symp. Principles of Programming Language*, pp. 228-241, 1999.

[3] A. C. Myers and B. Liskov, "A Decentralized Model for Information Flow Control", *Proc. 17'th ACM Symp. Operating Systems Principles*, pp. 129-142, 1997.

[4] A. Myers and B. Liskov, "Complete, Safe Information Flow with Decentralized Labels", *Proc. 14'th IEEE Symp. Security and Privacy*, pp. 186-197, 1998.

[5] A. Myers and B. Liskov, "Protecting Privacy using the Decentralized Label Model", *ACM Trans. Software Eng. Methodology*, vol. 9, no. 4, pp. 410-442, 2000.

[6] C. J. McCollum, J. R. Messing, and L. Notargiacomo, "Beyond the Pale of MAC and DAC - Defining New Forms of Access Control", *Proc. 6'th IEEE Symp. Security and Privacy*, pp. 190-200, 1990.

[7] D. E. Bell and L. J. LaPadula, "Secure Computer Systems: Unified Exposition and Multics Interpretation", *technique report, Mitre Corp.*, Mar. 1976. http://csrc.nist.gov/publications/history/bell76.pdf

[8] D. E. Denning, "A Lattice Model of Secure Information Flow", *Comm. ACM*, vol. 19, no. 5, pp. 236-243, 1976.

[9] D. E. Denning and P. J. Denning, "Certification of Program for Secure Information Flow", *Comm. ACM*, vol. 20, no. 7, pp. 504-513, 1977.

[10] D. F. C. Brewer, and M. J. Nash, "The Chinese Wall Access control policy", *Proc. 5'th IEEE Symp. Security and Privacy*, pp. 206-214, 1989.

[11] P. Samarati, E. Bertino, A. Ciampichetti, and S. Jajodia, "Information Flow Control in Object-Oriented Systems", *IEEE Trans. Knowledge Data Eng.*, vol. 9, no. 4, pp.524-538, Jul./Aug. 1997.

[12] E. Bertino, Sabrina de Capitani di Vimercati, E. Ferrari, and P. Samarati, "Exception-based Information Flow Control in Object-Oriented Systems", *ACM Trans. Information System Security*, vol. 1, no. 1, pp. 26-65, 1998.

[13] E. Ferrari, P. Samarati, E. Bertino, and S. Jajodia, "Providing Flexibility in Information flow control for Object-Oriented Systems", *Proc. 13'th IEEE Symp. Security and Privacy*, pp. 130-140, 1997.

[14] M. D. McIlroy and J. A. Reeds, "Multilevel Security in the UNIX Tradition", *Software - Practice and Experience*, vol. 22, no. 8, pp. 673-694, 1992.

[15] K. Izaki, K. Tanaka, and M. Takizawa, "Information Flow Control in Role-Based Model for Distributed Objects", *Proc. 8'th International Conf. Parallel and Distributed Systems*, pp. 363-370, 2001.

[16] M. Yasuda, T. Tachikawa, and M. Takizawa, "Information Flow in a Purpose-Oriented Access Control Model", *Proc. 1997 International Conf. Parallel and Distributed Systems*, pp. 244-249, 1997.

[17] M. Yasuda, T. Tachikawa, and M. Takizawa, "A Purpose-Oriented Access Control Model", *Proc. 12'th International Conf. Information Networking*, pp. 168-173, 1998.

[18] T. Tachikawa, M. Yasuda, and M. Takizawa, "A Purposed-Oriented Access Control Model in Object-Based Systems", *Trans. Information Processing Society of Japan*, vol. 38, no. 11, pp. 2362-2369, 1997.

[19] R. Graubart, "On the Need for a Third Form of Access Control", *Proc. 12'th Nat'l Computer Security Conf.*, pp. 296-303, 1989.

[20] S. Jajodia and B. Kogan, "Integrating an Object-Oriented Data Model with Multilevel Security", *Proc. 6'th IEEE Symp. Security and Privacy*, pp. 76-85, 1990.

[21] S. N. Foley, "A Model for Secure Information Flow", *Proc. 5'th IEEE Symp. Security and Privacy*, pp. 248-258, 1989.

[22] S. Zdancewic, L. Zheng, N. Nystrom, and A. C. Myers, "Untrusted Hosts and Confidentiality: Secure Program Partitioning", *Proc. 18th ACM Symp. Operating Systems Principles*, 2001.

[23] V. Varadharajan and S. Black, "A Multilevel Security Model for a Distributed Object-Oriented System", *Proc. 6'th IEEE Symp. Security and Privacy*, pp. 68-78, 1990.

[24] Z. Tari and S.-W. Chan, "A Role-Based Access Control for Intranet Security", *IEEE Internet Computing*, vol. 1, no. 5, pp. 24-34, 1997.

[25] A. Maamir and A. Fellah, "Adding Flexibility in Information Flow Control for Object-Oriented Systems Using Versions", *International Journal of Software Engineering and Knowledge Engineering*, vol. 13, no. 3, 313-326, 2003.

[26] R. Sandhu, "Role Hierarchies and Constraints for Lattice-Based Access Controls", *Proc. Fourth European Symposium on Research in Computer Security*, pp. 65-79, 1996.

[27] M. H. Harrison, W. L. Ruzzo, and J. D. Ullman, "Protection in Operating Systems", *Communications of the ACM*, vol. 19, no. 8, pp. 461-471, 1976.

[28] M. S. Olivier, R. P van de Riet, and E. Gudes, "Specifying Application-level Security in

Workflow Systems", in *Proceeding of the 9'th International Workshop on Database and Expert Systems Applications*, pp 346-351, 1998.

[29] R. K. Thomas and R. S. Sandhu, "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*, 1997.

[30] R. S. Sandhu, "Lattice-Based Access Control Models", *IEEE Computer*, vol. 26, no. 11, pp. 9-19, Nov. 1993.

[31] G. Booch, *Object-Oriented Analysis and Design with Application*, second edition, The Benjamin/Cummings Publishing Company, 1994.

[32] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-Oriented Modeling and Design*, Prentice-Hall, 1991.