PAPER

An RBAC-Based Access Control Model for Object-Oriented Systems Offering Dynamic Aspect Features

Shih-Chien CHOU[†a)], ***Nonmember***

SUMMARY This paper proposes a model for access control within object-oriented systems. The model is based on RBAC (role-based access control) and is called DRBAC (dynamic RBAC). Although RBAC is powerful in access control, the original design of RBAC required that user-role assignments and role-permission assignments should be handled statically (i.e., the assignments should be handled by human beings). Nevertheless, the following dynamic features are necessary in access control within a software system: (a) managing dynamic role switching, (b) avoiding Trojan horses, (c) managing role associations, and (d) handling dynamic role creation and deletion. DRBAC offers the dynamic features. This paper proposes DRBAC.

key words: *access control, security, role-based access control (RBAC), object-oriented system*

1. Introduction

Access control within a system prevents information leakage during system execution. The prevention can be achieved through information flow control. Many information flow control models are available [1]–[10], in which RBAC [11] is frequently applied. The original RBAC required that user-role assignments and role-permission assignments should be handled *statically* (i.e., the assignments should be handled by human beings). Statically handling the assignment increases the workload of human beings and the handling may be error-prone for complicated programs. An alternative approach is dynamic handling. We define *dynamic* handling as using computer programs for the handling. The need for dynamic user-role assignment is exhibited by *dynamic role switching*. The switching changes user-role assignment during program execution. For example, suppose a company will automatically promote a customer to a VIP when the customer's total consumption amount passes a threshold. Then, a customer playing the role "customer" will be dynamically switched to the role "VIP" when his consumption amount passes the threshold.

The need for dynamic role-permission assignment is exhibited by *avoiding Trojan horses* [1] and *managing role associations*. A Trojan horse occurs when an object's information is leaked indirectly. As to role associations, they are relationships among roles. Different role associations may result in different role-permission assignments. Since role associations may dynamically change during the execution of a system, role-permission assignments in a program may be dynamically changed. For example, suppose a customer can get a supplementary item when he orders an item, in which the supplementary item is selected from a list. If the customer and the manager managing the order are not friends, the customer selects the supplementary item from a default list. On the other hand, if they are friends, an extra list is provided. In this example, the permission on the extra list for the customer is decided by the association "friends" between the roles "manager" and "customer". Since friendship may change anytime during the execution of a system, the permission on the extra list for the customer may change dynamically.

In addition to dynamic user-role and role-permission assignments, dynamic role creation and deletion should also be handled. For example, when a customer orders an item from a company, a new "customer" role should be added. When the transaction finishes, the role should be deleted. According to the above description, a model that controls information access within object-oriented systems should offer the following *dynamic features*: (a) managing dynamic role switching, (b) avoiding Trojan horses, (c) managing role associations, and (d) managing dynamic role creation and deletion. We involved in the research of access control for years and designed models for that control [9], [10]. Although our previous work provides partial dynamic features, they are insufficient. For example, they fail to handle dynamic role switching. We thus designed a new model DRBAC (dynamic RBAC) that offers all the dynamic features. This paper presents DRBAC.

2. Related Work

Mandatory access control (MAC) [6]–[8] is useful in access control. An important milestone of MAC is that proposed by Bell&LaPadula [6], which categorizes the security levels of objects and subjects. Access control in the model follows the "no read up" and "no write down" rules [5], [6]. Bell&LaPadula's model was generalized into the lattice model [7], [8]. Although MAC avoids Trojan horses, it does not offer other dynamic features. The model in [2] uses access control lists (ACLs) of objects to compute ACLs of executions. A message filter is used to filter out possibly non-secure information access. Flexibility is added by allowing exceptions during or after method execution [3]. The model fails to offer dynamic features, except avoiding Tro-

Manuscript received July 14, 2004.
Manuscript revised September 21, 2004.

[†]The author is with the Department of Computer Science and Information Engineering, National Dong Hwa University, Hualien 974, Taiwan.
a) E-mail: scchou@mail.ndhu.edu.tw

DOI: 10.1093/ietisy/e88–d.9.2143

jan horses. The decentralized label approach [1] marks the security levels of variables using labels. A label is composed of one or more policies, which should be simultaneously obeyed. Join operation avoids Trojan horses. Nevertheless, the model fails to offer other dynamic features. Role-based access control (RBAC) models [11] can also be used in access control. Since RBAC was not designed for access control within software systems, it fails to offer the dynamic features. The model in [4] applies RBAC for access control. It classifies object methods and derives a flow graph from method invocations. From the graph, non-secure information access can be identified. The model fails to offer the dynamic features.

3. The Model

When an object-oriented system is executing, objects are instantiated from classes and messages are passed among objects. During system execution, the system's variables are protected independently because variables may be of different sensitivity [15].

3.1 DRBAC

DRBAC grants variable access rights to object methods. It extends RBAC by adding a special component DSTA (dynamic statement set) to offer dynamic features. DRBAC can be embedded in object-oriented systems. An executing object-oriented system embedded with DRBAC is composed of the component *eoosys*, which is the original object-oriented system and *acp*, which is the access control policy of DRBAC. The components *eoosys* and *acp* are defined below.

Definition 1: $eoosys = (OBJ, MSG)$, in which

a. MSG is a set of message passed among object methods.
b. OBJ is a set of objects. DRBAC regards objects as roles. An object is composed of attributes and methods. The code *mdcd* of a method in a system embedded with DRBAC is defined as follows:
$mdcd = LANGSTA \cup DSTA$, in which $LANGSTA$ contains normal language statements and $DSTA$ contains statements to offer dynamic features as described below:
   b.1. *Role instantiation* and *deletion* operators. These operators normally correspond to constructors and destructors of programming languages.
   b.2. *setRole*(*object*, *role*). It assigns a role to an object. This statement and the next one handle dynamic role switching.
   b.3. *isRole*(*object*, *role*). It checks the role of an object.
   b.4. *setAssociation*(*association*, *role_list*). It sets associations among roles. This statement and the next two manage role associations.
   b.5. *breakAssociation*(*association*, *role_list*). It breaks associations among roles.
   b.6. *withinAssociation*(*association*, *role_list*). It checks whether the roles are within a specific association.

Definition 2: $acp = (ROLE, ASO, RPER, APER, CNS)$, in which

a. $ROLE$ is the set of roles involved in a system.
b. ASO is a set of role associations. An association contains roles.
c. $RPER$ is a set of regular role permissions. A permission is a capability list for a method.
d. $APER$ is a set of association permissions. $RPER$ is used when a role is not within a role association whereas $APER$ is used when a role is within a role association.
e. CNS is a set of constraints constraining the DRBAC components.

3.2 Access Control and Features of DRBAC

When executing a program embedded with DRBAC, the access control policy ensures secure access of variables. The following conditions should be true for the ensuring. In defining the conditions, we assume that: (1) the value derived from the variables "$var1$", "$var2$", ..., "$varn$" is assigned to the variable "d_var", (2) the derivation is performed in the method "$obj1.md1$," and (3) the permission of "$obj1.md1$" is "per_{md1}."

First access control condition: $\{\{var1, R\}, \{var2, R\}, \ldots, \{varn, R\}, \{d\_var, R\}\} \subseteq per_{md1}$

Second access control condition: $\{d\_var, W\} \in per_{md1}$

The notation "$\{var1, R\}$" and "$\{d\_var, W\}$" respectively state that "$var1$" is allowed to read and write. The first condition requires that the method "$obj1.md1$" should be allowed to read the variable "d_var" and the variables deriving "d_var" because "$obj1.md1$" reads the variables. The second condition requires that the method "$obj1.md1$" should be allowed to write "d_var" because the variable is written within the method. After the variable "d_var" gets the derived information, the access rights of "d_var" should be changed to avoid Trojan horses using join operation [1].

Definition 3: If "d_var" is derived from "$var1$", "$var2$", ..., "$varn$", the join operation will set "Rd_var", "Wd_var" as follows (in the following formulas, "$Rvar$" and "$Wvar$" are respectively the sets of methods that can read and write the variable "var"):

$$R_{d\_var} = R_{var1} \cap R_{var2} \cap \ldots \cap R_{varn}$$
$$W_{d\_var} = W_{var1} \cup W_{var2} \cup \ldots \cup W_{varn}$$

The resulted R_{d_var} and W_{d_var} should be used to update the permissions containing the variable "*d_var*".

As describe in Sect. 1, DRBAC offer the following dynamic features:

a. Managing dynamic role switching. This feature is primarily achieved by the statement "setRole", which dynamically switches role. For example, a regular customer can be dynamically switched to be a VIP using the statement.
b. Avoiding Trojan horses. We formally prove that DRBAC offers these features as follows.
   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 "*var*1" be a variable in "md1" which can be read by the methods in the set "R_{var1}". Here, "*var*1" can be read by "md2" but not "md3". That is, "md2" is in the set "R_{var1}" but "md3" is not. We also let "*var*2" be a variable in "md2" whose value is derived from "*var*1" and other variables. After the derivation, the read set of "*var*2" is modified by the join operation to "R_{var2}". Suppose that a Trojan horse exists among "md1", "md2", and "md3". Without loss of generality, we assume that "md3" can read "*var*2". 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 "*var*2" is derived from "*var*1" and other variables. Since "md3" is not in "R_{var1}", "md3" is not in "R_{var2}". This contradicts the assumption.
c. Managing role associations. This feature is primarily achieved by "setAssociation", "breakAssociation", and the "associationPermissions" declaration. The statements dynamically establishes/breaks associations among roles. The "associationPermissions" declaration decides the permissions of a method whose object is within an association.
d. Managing dynamic role creation and deletion. This feature is offered by constructors and destructors of classes. Although the feature is inherently offered by every object-oriented programming language, no existing flow control model mentioned the concept of managing dynamic role creation and deletion.

4. Implementation and Evaluation

We embedded DRBAC in JAVA to produce the language DRBACL. The implementation of DRBACL temporarily excludes the issues of multiple threads [12], exceptions, timing channels [13], read channels [14], and covert channels [15]. An application written in DRBACL should first be processed by the DRBACL preprocessor. The output of the preprocessor is a JAVA program, which is composed of two parts. One is the original JAVA program and the other a *security checker*. During program execution, the checker ensures secure information access within the JAVA program. To ensure security, the security checker records the information of DRBAC including roles, role-permission assignments, user-role assignments, associations among roles, and so on. Primary functions of DRBACL preprocessor are listed below:

a. Record initial DRBAC information in the DRBAC database.
b. Replace the DSTA statements "setRole", "setAssociation", "breakAssociation", "isRole", and "withinAssociation" with method invocation statements. Every DSTA statement is implemented as a method of the class "DSTAC" created by the DRBACL preprocessor.
c. Add program code to check access control. Note that every information access should fulfill the two access control conditions.
d. Add program code to do the join operation and then changes the permissions affected by the join.

Currently the security checker does most checking dynamically. This contradicts the static checking proposal [2], [15]. Nevertheless, dynamic checking is impossible to be totally replaced by static checking because of the dynamic features offered by DRBAC. We used many examples to evaluate DRBAC, such as inventory management systems, employee management systems, and library management systems. We selected twenty students to program the examples and then execute their programs. We collected the following metrics data: (a) execution time of the programs without DRBAC embedded, (b) execution time of the programs with DRBAC embedded, and (c) number of statements that violate the access control policy (per 100 LOC) in the programs embedded with DRBAC.

We averaged the metrics data collected. The experiment result is shown in Fig. 1, which depicts that the averaged execution time of the programs with DRBAC embedded is about four times that without DRBAC embedded. This runtime overhead cannot be avoided because access control is mostly checked dynamically. Moreover, about six statements per 100 LOCs that violate the access control policy were identified. Since statements that violate access control policy can be identified, we believe that DRBAC is valuable.

Currently the DRBACL preprocessor does not solve quite a few difficult problems such as those raised by timing channels [13] and covert channels [15]. The rationales are described below.

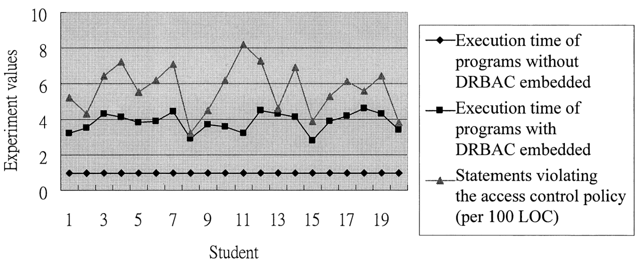

Fig. 1 Experiment result.

a. Problems raised by timing channels
Stealing information through timing channel is achieved by tracking the execution time of program segments. We use the following C program segment to explain this:

```
for (i = 0, c = 0; i < 1000; i++)
    if (a == 1)
        c += 1;
```

In the above program segment, tracking the execution time of the "for" loop can guess whether the value of "*a*" is 1 (i.e., a longer execution time implies that "*a*" is 1). To prevent this guessing, an information flow control model should identify every program segment that may leaks information through timing channel. The model should then adjust the program segment to prevent the leakage. For example, the above program segment can be adjusted by adding code as follows to prevent guessing the value of "*a*". The added code may be garbage code.

```
for (i = 0, c = 0; i < 1000; i++)
    if (a == 1)
        c += 1;
    else
        b += 1;
          /* this statement may be garbage code */
```

To do the above adjustment, the entire application should be scanned, sliced, analysed, and then adjusted. The DRBACL preprocessor temporarily excludes the complicated issue. We will solve the issue in the future.

b. Problems raised by covert channels
A covert channel refers to media such as disk file that leaks information. We use an example to explain covert channels. Suppose in a computer system, when an application is abnormally stopped, the operating system dumps the memory space occupied by the application to a disk file. In the normal case, a debugger uses the dumped file for debugging. Nevertheless, if an unauthorized user retrieves the file, he may steal the application's sensitive information by decoding the dumped file. An information flow control model cannot protect the dumped file. In fact, the protection is the responsibility of the operating system. To prevent leakage in the example, the protection system of an operating system should cooperate with an information flow control model. Even if the protection system of an operating system and the information flow control model are compatible, implementing the cooperation is a huge task. If the protection system of an operating system and the information flow control model are incompatible, implementing the cooperation is difficult. Therefore, the DRBACL preprocessor excludes the complicated issue in the current stage.

5. Conclusions

This paper proposes an RBAC-based access control model to ensure secure information access during program execution. It is name DRBAC (dynamic RBAC) because it offers dynamic features as described below:

a. Managing dynamic role switching. DRBAC uses statements in DSTA to achieve this.
b. Avoiding Trojan horses. DRBAC use the join operation to achieve this.
c. Managing role associations. DRBAC uses statements in DSTA to achieve this.
d. Handling dynamic role creation and deletion. DRBAC uses the constructors and destructors of classes to achieve this.

References

[1] A. Myers and B. Liskov, "Protecting privacy using the decentralized label model," ACM Trans. Softw. Eng. Methodol., vol.9, no.4, pp.410–442, 2000.
[2] P. Samarati, E. Bertino, A. Ciampichetti, and S. Jajodia, "Information flow control in object-oriented systems," IEEE Trans. Knowl. Data Eng., vol.9, no.4, pp.524–538, July/Aug. 1997.
[3] E. Bertino, Sabrina de Capitani di Vimercati, E. Ferrari, and P. Samarati, "Exception-based information flow control in object-oriented systems," ACM Trans. Inf. Syst. Secur., vol.1, no.1, pp.26–65, 1998.
[4] 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.
[5] 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.
[6] D.E. Bell and L.J. LaPadula, "Secure computer systems: Unified exposition and multics interpretation," Technique Report, Mitre Corp., March 1976. http://csrc.nist.gov/publications/history/bell76.pdf
[7] D.E. Denning, "A lattice model of secure information flow," Commun. ACM, vol.19, no.5, pp.236–243, 1976.
[8] D.E. Denning and P.J. Denning, "Certification of program for secure information flow," Commun. ACM, vol.20, no.7, pp.504–513, 1977.
[9] S.-C. Chou, "Embedding role-based access control model in object-oriented systems to protect privacy," J. Syst. Softw., vol.71, no.1-2, pp.143–161, April 2004.
[10] S.-C. Chou, "LnRBAC: A multiple-leveled role-based access control model for protecting privacy in object-oriented systems," J. Object Technology, vol.3, no.3, pp.91–120, March/April 2004.
[11] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman, "Role-based access control models," Computer, vol.29, no.2, pp.38–47, 1996.
[12] G. Smith and D. Volpano, "Secure information flow in a multi-thread imperative language," Proc. 25th ACM Symp. on Principles of Programming Languages, pp.355–364, 1998.
[13] J. Agat, "Transforming out timing leaks," Proc. 27th ACM Symp. on Principles of Programming Languages, pp.40–53, 2000.
[14] S. Zdancewic, L. Zheng, N. Nystrom, and A.C. Myers, "Untrusted hosts and confidentiality: Secure program partitioning," Proc. 18th ACM Symp. Operating Systems Principles, pp.1–14, 2001.
[15] R. Focardi and R. Gorrieri, "The compositional security checker: A tool for the verification of information flow security properties," IEEE Trans. Softw. Eng., vol.23, no.9, pp.550–571, 1997.

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