5N－5

A Transaction-based Purpose-oriented Access Control Model for Object-based Systems *

Tsunetake Ishida, Masashi Yasuda, Hiroaki Higaki, and Makoto Takizawa †
Tokyo Denki University‡
E-mail {tsune, masa, hig, taki}@takilab.k.dendai.ac.jp

1 Introduction

In order to make a system secure, it is critical to consider what subject s can manipulate what object o by what type t of operation. In the access control model, an access rule is specified in a form $\langle s, o, t \rangle$. A subject s is granted an access right $\langle o, t \rangle$ by an authorizer. A system is *secure* if and only if (*iff*) every object is manipulated according to the access rules. However, the access control model cannot resolve the *confinement* problem where information illegally flows among subjects and objects. The *lattice-based* model aims at protecting against the illegal information flow. One security class is given to each subject and object in the system. A *can-flow* relation among the security classes is defined to denote that information of one class s_1 can *flow into* s_2. In the *mandatory* model, an authorizer specifies every access rule $\langle s, o, t \rangle$ so that the flow relation between a subject s and an object o holds. In the *discretionary* model, the access right of one subject can be granted to other subjects. In the *role-based* model [1], a *role* R shows a job function in the application, which is modeled to be a collection of access rights, i.e. $R = \{\langle o, t \rangle\}$. The access rule is defined to bind a subject s to a role R, i.e. $\langle s, R \rangle$.

Distributed applications are modeled in an object-based model like CORBA. Each object is an encapsulation of data and methods for manipulating the object. A method of the object is invoked on receipt of a request message. The method may invoke methods on other objects, i.e. invocation is *nested*. Yasuda and Takizawa propose a purpose-oriented access control model [2] in the object-based system. It is essential to discuss a *purpose* of a subject s to manipulate an object o. In the *purpose-oriented* model, an access rule shows a *purpose* for which each subject s manipulates an object o by a method t of o. In the object-based system, the purpose is modeled to be a method u of an object s which invokes a method t of an object o. That is, a purpose-oriented access rule is specified in a form $\langle s : u, o : t \rangle$, where u shows the purpose. Even if each purpose-oriented rule between a pair of objects satisfies the information flow relation, some data in one object may illegally flow to another object through the nested invocation.

In this paper, we discuss the purpose concept which shows an invocation sequence of methods in the object-based systems.

2 Object-based model

In the object-based system, each object o_i supports abstract level of data and methods. In addition, o_i is *encapsulated* so that o_i can be manipulated only through the methods supported by o_i. Here, methods are assumed to be unnested. An object s sends a request message q of a method op_i to an object o_i. On receipt of q, o_i computes op_i and sends the response r back to s. q and r carry the input and output of op_i, respectively. The method op_i changes the state of o_i by using the input. Thus, the data in s may flow into o_i if q carries some data in s. If op_i derives data from o_i and then returns the data to s, the data in o_i may flow out to s if r carries the data derived from o_i by op_i. Thus, input and output of op_i have to be discussed to clarify the information flow relation between s and o_i.

Each method op_i of o_i is characterized in terms of input (I_i), output (O_i), and state transition of o_i. The input I_i exists if some data flows from s to o_i. The output O_i exists if some data in o_i flows out to s. Only data stored in o_i can flow out from o_i to s and the data in s can flow to o_i in the computation of the method of o_i.

The methods are classified into four *flow* type [Figure 2]: *non-flow* (NF), *flow-in* (FI), *flow-out* (FO), and *flow-in/out* (FIO). Here, let $\tau(op_i)$ denote a flow type of op_i. An NF method op_i implies no information flow from or to the object o_i. In addition, op_i does not change o_i. Even if the input data I_i exists, no information in s flows to o_i unless op_i changes o_i. Similarly, no data in o_i flows out to s unless the output data O_i is derived from o_i. An FI method op_i changes o_i by using I_i where data in s may flow into o_i. An FO method op_i does not change o_i. Since the output O_i of op_i carries data in o_i to s, data in o_i may flow to s. An FIO method op_i changes o_i by using I_i and sends O_i including data in o_i back to s. The access rule is defined to the object-based system as follows.

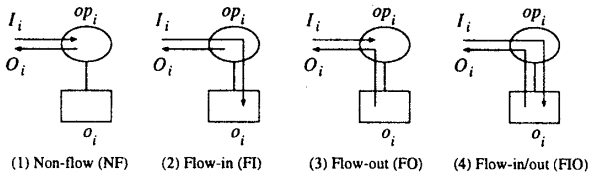

Figure 1: Flow types of methods

[Object-Based access rules] A subject s can manipulate an object o_i by a method op_i according to the following rules.

(1) $\tau(op_i) \in$ {NF, FI} only if $\lambda(s) \preceq \lambda(o_i)$.
(2) $\tau(op_i) \in$ {NF, FO} only if $\lambda(s) \succeq \lambda(o_i)$.
(3) $\tau(op_i) \in$ {NF, FI, FO, FIO} only if $\lambda(s) \equiv \lambda(o_i)$.□

The types of methods and the security class $\lambda(o_i)$ of the object o_i are defined based on the semantics of

*オブジェクトベースシステムのためのトランザクションに基づく目的指向アクセス制御
†石田 常竹　安田 昌史　桧垣 博章　滝沢 誠
‡東京電機大学

o_i. Each time a subject s invokes a method op_i, op_i is accepted to be computed on o_i if $\tau(op_i)$ and $\lambda(o_i)$ satisfy the access rules.

3 Purpose-oriented Model

3.1 Purpose concept

We assume a pair of methods op_1 and op_2 supported by an object o_i can exchange data only through the state of o_i. If data d flowing from an object o_i to another object o_j is neither derived from o_i nor stored in o_j, it is meaningless to consider the information flow from o_i to o_j. If data derived from o_i is stored in o_j, the data may flow out to other objects.

Let us consider a person object p and a bank object b supporting methods *withdraw* and *deposit*. In the access control model, p can derive money from b for any purpose if an access rule $\langle p, b, withdraw \rangle$ is specified. Suppose the person p can withdraw money from the bank object b for the house-keeping. However, p cannot get money from b to go drinking. Thus, it is critical to introduce a *purpose* for which a subject s manipulates an object o by issuing a method op .

[Purpose-oriented (PO) rule] A purpose-oriented access rule $\langle o_i : op_i, o_{ij} : op_{ij} \rangle$ means that an object o_i can manipulate another object o_{ij} through a method op_{ij} invoked by op_i of o_i. □

In the PO rule, op_i shows a *purpose* for which o_i manipulates o_{ij} by op_{ij}.

3.2 Purpose in role

A role is specified in a collection of access rights in the role-based model [1]. We would like to extend the purpose-oriented access control [2] to the role-based model in the object-oriented system. In the object-based system, methods are invoked in a nested manner. Here, suppose that a subject s invokes a method op_1 on an object o_1 and then op_1 invokes another method op_2 on an object o_2. Suppose that the response of op_2 carries some data stored in the object o_2. On receipt of the response, o_1 may store the data carried by the response in the storage while o_1 continues to compute op_1 by using the response. This means, information flows from o_2 to o_1 through the invocation. The data may be brought to other objects by further invocation.

Each method op_i of an object o_i is granted a role $R_i = \{\langle op_{i1}, o_{i1} \rangle, \ldots, \langle op_{ih_i}, o_{ih_i} \rangle\}$. This means, op_i can invoke a method op_{ij} of an object o_{ij} (for $j = 1, \ldots, h_i$). In turn, op_{ij} may be granted a role $R_{ij} = \{\langle op_{ij1}, o_{ij1} \rangle, \ldots, \langle op_{ijh_{ij}}, o_{ijh_{ij}} \rangle\}$. The method op_{ij} can invoke a method op_{ijk} of an object o_{ijk} if op_{ij} is granted the role R_{ij}.

[Purpose-oriented role-based access (POR) rule] $\langle R : o_i : op_i, R_i \rangle$ where $R_i = \{\langle op_{i1}, o_{i1} \rangle, \ldots, \langle op_{ih_i}, o_{ih_i} \rangle\}$. □

4 Invocation Sequence in Purpose

Suppose that a pair of objects o_1 and o_2 support only *read* and *write* methods to simplify the problem. These methods are not nested. Roles $R_1 = \{\langle read, o_1 \rangle, \langle write, o_1 \rangle, \langle write, o_2 \rangle, \langle read, o_2 \rangle\}$ and $R_2 = \{\langle read, o_2 \rangle\}$ are assumed to be specified. Here, the subject p_1 is granted the role R_1 and p_2 is granted R_2 [Figure 2]. That is, p_1 can manipulate the objects o_1 and o_2 but p_2 cannot write o_1 while reading o_2. p_2 may illegally obtain data in o_1. First, suppose that p_1 writes data in o_2 after reading data from o_1. Here, p_1 can indirectly obtain the data stored in o_1. Next, suppose that p_1 reads data from o_1 after writing data to o_2. Here, no data in o_2 flows to o_1. Therefore, there occurs no illegal information flow from o_1 to o_2 if p_1 writes o_2 before reading o_1. This example shows that whether or not illegal information flow occurs depends on how p_1 issues methods to o_1 and o_2.

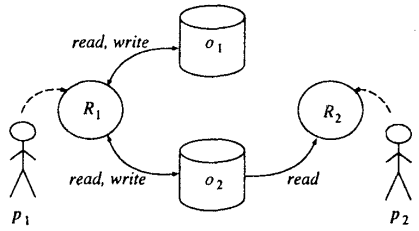

Figure 2: Information flow

We assume that an object o_i has an access list $AL(o_i) = \{\langle s, t \rangle \mid \langle s, o_i, t \rangle$ is authorized$\}$. $AL(o_i)$ indicates what subject can manipulate o_i by what method.

A subject s manipulates a directed graph $H(s)$ to check whether or not illegal information flow to occur. $H(s)$ is composed of two kinds of nodes, *subject* nodes and *object* nodes. Initially, $H(s)$ includes a subject node showing s. If the subject s would manipulate an object o_i by issuing a method t_i, s manipulates the graph $H(s)$ as follows:

[Graph constructions]

1. An object node o_i is created in $H(s)$.
   a. If t is *read*, a directed edge from o_i to s ($o_i \rightarrow s$) is created in $H(s)$.
   b. If t is *write*, a directed edge from s to o_i ($s \rightarrow o_i$) is created in $H(s)$.
2. For each element $\langle s_j, t_j \rangle$ in $AL(o_i)$, a subject node s_j is created in $H(s)$ unless $H(s)$ includes s_j.
   a. If t_j is *read*, a directed edge $o_i \rightarrow s_j$ is created.
   b. If t_j is *write*, a directed edge $s_j \rightarrow o_i$ is created. □

To preserve the order, for any two methods e_i and e_j, if e_i issues before e_j, e_i appears on the left of e_j in the graph.

For every nodes a, b, and c, if $a \rightarrow b$, $b \rightarrow c$, and a appears on the left of c in $H(s)$, an edge $a \rightarrow c$ is created in $H(s)$. If the following condition holds, s is not allowed to manipulate o_i by t_i.

[Access condition] For every subject s_j, $o_j \rightarrow s_i$ in $H(s)$ but $\langle s_i, o_j, read \rangle$ is not authorized. □

5 Concluding Remarks

In this paper, we include the role concepts in the purpose-oriented access control model. The access rules have to satisfy the information flow relation among objects. The occurrence of illegal information flow depends on in which order methods are invoked. In this paper, a *purpose* of the object to manipulate objects is modeled to be a invocation sequence of methods.

References

[1] Sandhu, R. S., Coyne, E. J., Feinstein, H. L., and Youman, C. E., "Role-Based Access Control Models," *IEEE Computer*, Vol. 29, No. 2, 1996, pp. 38–47.

[2] Yasuda, M., Higaki, H., and Takizawa, M., "A Purpose-Oriented Access Control Model for Information Flow Management," *Proceeding of 14th Int'l Information Security Conf. (IFIP/SEC'98)*, 1998, pp. 230–239.