A Time and Mutable Attribute-Based Access Control Model

Ran Yang
Department of Computer Science and Technology, Tsinghua University, Beijing, P.R. China
yangran07@csnet1.cs.tsinghua.edu.cn

Chuang Lin and Fujun Feng
Department of Computer Science and Technology, Tsinghua University, Beijing, P.R. China
{clin, fjfeng}@csnet1.cs.tsinghua.edu.cn

Abstract—**Access control is one of the most important technologies to guarantee computer security. A new model with support for valid time and usage constraint is described based on full analysis of flaws in existing models. In the new model, authorization rules can express access control policies completely and access constraints are necessary conditions to prevent authorization abuse. To solve the problems in implementation of the model, a sound scheme for administration of authorizations is proposed and some access decision algorithms are developed.**

Index Terms—**mutable attribute, access constraint, time, access control**

I. Introduction

In recent years, many new applications are emerging with the rapid development of the Internet. These applications need to access resources through the Internet. For example, some online media accessing and reading services require accounting based on usage of customers, and digital right management and content protection classification systems need to keep user online behaviors under control. These applications put forward a lot of new requirements on online resource sharing system. It is a great challenge for us to devise the systems that meet the needs. First, there may have huge number of potential users in online applications, which increase difficulty in realizing administration of authorizations. Second, the users in open network environment are trustless. Administrators have to handle complex security incident. Finally, the access control systems need to carry some of the new characteristics of these applications.

The existing access control model research mainly focus on validity of authorization, that is, all the authorizations only can be revoked by the operations of administrators. They can not make dynamic access decision according to session. Furthermore, the effect of resource usage on authorization can not be expressed in these models. So they can't meet the needs of these new applications.

Some researchers propose the concept of usage control, and give the models and specifications recently. Usage control models have the abilities of expressing attribute preconditions for authorization and continuous access control on user session, which meet the need of new applications in some extent. But the complexity restricts the effort to make use of them.

In this paper, we propose a new model that is easier to operate according to the ideas of usage control. The TMAAC model supports quantitative authorization and mutable attribute. The model provides more strict constraint to prevent abuse of authorization, which makes it safer than the other models. At the same time, the concept of TMAAC is succinct and very easy to realize in real applications.

A. Related Work

Access control is one of the main mechanism for computer security, and some models such as DAC and MAC has been used for a long time. Sandhu et al. propose RBAC to solve the resource protection in organizations and it has been widely used now.

Some researchers introduce temporal attribute in RBAC firstly to enhance authorization management [1,2,3,4,7]. Niezette and Stevenne [7] devise an effective way to express periodic conception in symbolic expression. Bertino et al. [2,3] applies this representation to express temporal constraint of authorization in access control model. In line with this idea, Bertino et al. [4] extend restrictions to periodic time and introduce reasoning of authorization validity in RBAC based on time interval.

Though these models take into account temporal constraint in time interval, their authorizations are still static. That is, user can use system authorization without limit until explicit revocation. These researches can not keep the ongoing access behaviors under control. So it is clear that they can't meet the need of these new applications.

Supported by the Scientific and Technological Innovation Nurture Funds Major Projects of MOE., P.R.C under Grant No.707005.

Sandhu et al. attempt to unify access control model at a deeper level and propose the concept of usage control [5,6,8,9,10]. A series of models that can express time and consumptive attribute emerge in the UCON model [5]. Zhang et al. [10] give the specifications for these models. Sandhu [8] summarizes the principals for next-generation access control. Although the UCON model has features for future access control, it is complex in authorization management and hard to implement. So the TUCON model proposed in [11] suggest applying valid time and usage time constraints to simplify the implement of usage control. But we argue that there is defect in authorization in TUCON too. For example, the assumption of usage time constraint is unreasonable for many applications.

So we propose TMAAC model in this paper which supports valid time constraint of temporal access control and quantitative authorization. The latter makes users with authorizations can only have limited access permission to resources. And by continuous decision the system can control the whole usage process of authorization.

The rest of this paper is organized as follows. Section 2 analyzes the application requirements and gives the TMAAC authorization features and state transition. Section 3 defines the basic components of TMAAC. Then we analyze model capability and prove the properties of authorization rules and constraints. Section 4 discusses implementation of model and gives algorithms for access control decision. In section 5, we conclude the paper.

II. System State and Mutable Attribute

A. Mutable Attribute

In order to establish a more reasonable model, we first consider actual application scenarios. We conclude that many applications need to exert more thorough control upon usage of authorization. In the media access service, for example, users need to pay in advance to obtain certain authorizations. Then users can access resources and system will deduct a certain amount of money based on accounting about the user. Another example is the online reading, the privilege consumption occurred when each chapter have been read by the user. In addition, Enterprise applications need to protect sensitive documents from been accessed too much. A reasonable approach is restrictions on the authorized user's possession of sensitive information by setup the number of available times of authorization.

The restrictions on authorization of the three applications above can be summed up as limited usage time, amount and times available by the user. So, the permissions granted to users will be revoked through the following conditions:

- Revoked by authorization attribute: authorization attributes is mutable, that is, it can be consumed by users. User's action reduces the attribute value.
- Revoked by valid time interval: Authorization is valid in a certain time interval and will be revoked immediately when expired.
- Mandatory revocation: System revokes user authorization by explicit operation.

The model must have the ability to describe the three conditions above. In order to control user's behavior, the following three policies are needed:

- Authorization attributes: Users should have greater authorization attributes than resources' to access.
- Access request: The point when access occurred is valid based on authorization.
- Temporal conditions: The events occurred in the process of access support access action continue.

B. Authorization Features

In order to study the characteristics of mutable attributes introduced in model, we need to classify authorization features firstly. The TMAAC model features with time and mutable attribute.

1) Time Features

Based on different semantics of time constraints, time features are classified as valid time interval and valid time pattern.

a) Valid Time Interval

Valid time interval means that system authorizations are always valid in certain time interval and will be revoked after expired.

b) Valid Time Pattern

Valid time pattern means that although authorization is valid, user can not access resources if it is not in permitted time period.

2) Mutable Attribute Feature

Based on different owners of mutable attributes, authorization features are classified as authorization attributes and resource attributes.

a) Authorization attributes

Authorization attributes are access constraints set by system. So, users can only get a certain amount of authorization. These constraints can prevent authorizations from being abused.

b) Resource Attributes

Resource attributes are access constraints made by system before user accessing. In order to access resource, user's authorization attribute must not less than resource attribute.

3) System State Transition

We give states of the systems with mutable authorization attribute and state transitions in Fig. 1 based on analysis above. The meaning of states and transitions in Fig. 1 are given below:

- Initial: Initial state of the system, which is the starting point of all user access and transfer actions.
- trigger: A set of conditions that cause the system grant access to users.

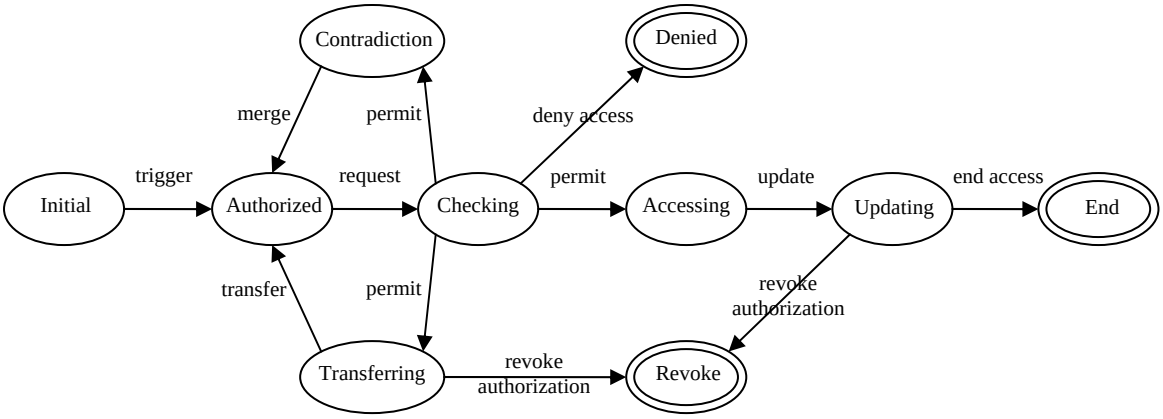

Figure 1. The state transition

- Authorized: The state that the users have been authorized to some resources.
- request: User behaviors of applying for resources. It may be access request or transfer request. Users must wait for validity check implemented by system before accessing or transferring.
- Checking: The system implements validity check to decide whether or not the request should be allowed.
- permit: After validity check, user is permitted to access resource.
- Contradiction: The set of user authorizations are in inconsistency state. There exist different authorizations about a user.
- transfer and Transferring: The system transfers amount of authorization attribute from one user to another. It is equivalent to giving amount of authorization to others.
- merge: The system merges user's inconsistency authorizations and authorization state regain consistency.
- Accessing: The user is accessing resources. The system audits user behavior during accessing and decides whether to terminate access or not.
- update: It is the transition that user stops accessing resources. Audit result records the user's usage accounting.
- Updating: The system updates authorization attribute according to usage. Then the system needs to check to ensure that the value of attribute is not less than zero. Otherwise authorization should be revoked.
- end access: If attribute value is greater than zero, then the system will enter into the End state after updating attribute.
- Denied, Revoke and End: The final state of system. If the user requests for resources do not have enough privilege, the system will enter into Refuse state. If attribute value equals to zero after accessing or transferring, the system will enter into Revoke state.

We can conclude from the above analysis that user access action is a continuous process that the system audits user behaviors to update authorization. This requires the system grant authorization quantitatively. It is clear that existing access control models do not concern these problems. So, we propose a new model as the solution to such information system.

III. TMAAC MODEL

Existing access control models support checking validity of authorization when the user requests for accessing resources. But these models can not keep user behavior under control during session. Users can access resources freely when conditions are met. Administrators have to revoke relevant privilege from users by explicit operation. This kind of mechanism is prone to authorization abuse.

If the application needs to control user's access behavior, new model must be designed to address these problems. We attach mutable attributes to authorization in order to control users' actions during session. Then the system can manage the whole life cycle of authorization.

A. TMAAC Concepts

TMAAC model has two main components, that is, authorizations with mutable attribute and access constraints. Fig. 2 illustrates the relations among them.

1) Components of TMAAC

The interpretations for basic components in Fig. 2 are given below.

- *S* is a set composed of subject, which can apply for access to resources and transfer authorization to others.
- *O* is a set composed of object, which are resources been protected by security system. *ATT*(*O*) represents object attribute.
- *R* is a set composed of permission, which means access rights been authorized to subject. *ATT*(*R*) represents its attributes.

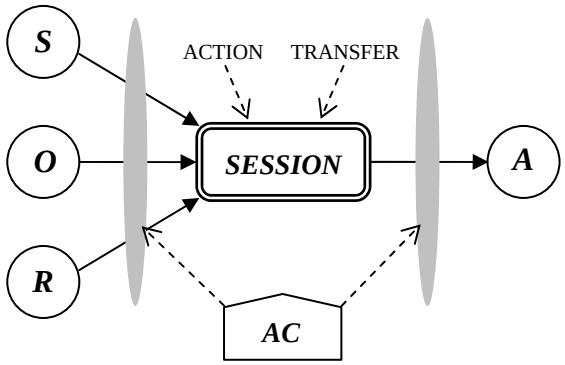

Figure 2. TMAAC Components

- *A* is the authorization set with mutable attribute, which includes all authorizations in system.
- *AC* is a set composed of access constraints, which represents constraint on subject behavior when activating authorization.
- *ACTION* is a set composed of access behavior, which represents the process of using authorization. We define *ACTION* as $\forall a \in ACTION,\ a \in S \times O \times R$.
- *TRANSFER* is a set composed of transfer behavior, which represents the transformation process between subjects. We define *TRANSFER* as $\forall a \in TRANSFER,\ a \in S \times S \times R$.
- *SESSION* is a set composed of system session, which represents the state of system. It is the set of all active subject behaviors.

Authorization and access constraints will be discussed later.

2) Authorization Attributes and Update

The resources and authorizations of TMAAC model are associated with attributes. System makes access decision according to these attributes. The authorization attribute represents quantitative authorization to subject. These attributes have the same class. The subject access actions consume a certain amount of authorization attribute when finish accessing resources. So, no one can abuse their authorizations.

Example: In some applications, sensitive documents only can be accessed by users with special authorization. But the system still needs to guarantee against too frequent use of them. In order to meet this requirement, the administrator may associates maximum access counts to these documents and special authorizations. Then system reduces authorization attribute by 1 automatically after every access. So access frequency to sensitive documents can be limited.

Object attributes are set by administrator beforehand. The authorization attributes are mutable and can be set at different opportunity:

- Object attribute set: Administrator sets object attributes according to application requirements.
- Set before accessing: The system grants appropriate authorizations to subject and sets its attributes before subject accessing.
- Decide before accessing: The system makes decision when receives request from subject. It checks attributes to decide whether it is a valid request or not. Then, system allows or denies the request.
- Decide during accessing: The system judges through audit results and decides whether it is necessary to terminate user access.
- Update after accessing: The system calls consumption function to calculate loss value of authorization. This function will be defined later.

B. Authorization of TMAAC

In this section, the formal definitions and authorization rules are discussed in detail, and a brief proof for completeness of rule set is given. To begin with, we first introduce a virtual time system to establish relevant conception about authorization.

1) Definition of Time System

Definition 1. time system.

$Tr = \{t_i \mid i \in N\}$ is a series of time points when events in real system occur, $Tv = \{t_i \mid i \in N\}$ is a sequence of time points which represent corresponding virtual time point of event occurred in real system. $RTime: Tv \to Tr$ is the mapping function. For every virtual time point in Tv, there exist only one time point in Tr.

The properties of virtual time system are as follows:

- Irrelevance: The only relation between *Tr* and *Tv* is the mapping function of *RTime*.
- Correspondence: $RTime(t_i) \neq RTime(t_j)$ and $\forall t_k \in Tv, \exists RTime(t_k) \in Tr$.
- Comparability: $\forall i \neq j$, $RTime(t_i) < RTime(t_j)$ or $RTime(t_i) > RTime(t_j)$.
- Incremental: $\forall i < j, RTime(t_i) < RTime(t_j)$.

Based on the properties of time system, we give the definition of time interval.

Definition 2. time interval.

Time interval is defined as $\{(t_i, t_j) \mid i, j \in N \text{ and } i < j\}$ and represented by TRange. $P\_TRange = 2^{TRange}$ is the power set of time interval.

Time intervals have a basic property:

$\forall range_1, range_2 \in TRange$, $range_1 \subset range_2$ if and only if $(t_{i2} < t_{i1}) \wedge (t_{j2} > t_{j1})$.

2) TMAAC Authorization and Rules

Definition 3. TMAAC authorization.

TMAAC authorization is defined as a triple and represented as M_auth=(vt,tp,auth), where auth=(s,o,r,n), $s \in S$, $o \in O$, $r \in R$, $n = ATT(R)$ *or* $n = -1$, $vt \in TRange$. *tp is a periodic expression that is a set composed of time intervals and is defined as* $\forall i \in N, tp_i \in tp$ *and* $tp_i \subset TRange$.

The time interval *vt* in Definition 3 is the valid time interval of authorization. *tp* is a periodic expression that represents time constraints for activating authorization in *vt*. Please turn to [2,3] for detailed descriptions about periodic expression. The definition about *vt* and *tp* correspond to time properties described previous section. The 4-tuple *auth*=(*s*,*o*,*r*,*n*) means that the system grants permission *r* with attribute *n* on resource *o* to subject *s*.

We would like to give some explanations about periodic expression. This kind of expression is a way to express time pattern in concept. For example, an expression can be written as $Weeks + \{1,3,5\}.Days$, where *Weeks* and *Days* are calendars. The expression means that this time pattern includes Monday, Wednesday and Friday every week. We only use the expressions to validate user authorizations periodically and any further discussion about them is not given in this paper.

Sometimes the policy needs to represent authorization of administrator. Attribute of this kind of authorization is immutable. Then administrator authorization is represented by *auth*=(*s*,*o*,*r*,*-1*).

Definition 4. Extensions.

$CF: R \times ACTION \to \Delta ATT(R)$. *The function calculates consumption of authorization based on subject actions.*

$currenttime: \Phi \to Tv$. *The function returns the current time point in virtual time system.*

$start: TRange \to Tv$. *The function returns the start point of arbitrary time interval.*

$end: TRange \to Tv$. *The function returns the end point of arbitrary time interval.*

$session: S \to ACTION$. *The function returns all the active actions of a subject in session.*

$inrange: Tv \times TRange \to \{true, false\}$. *The function calculates containment relationship between a time point and time interval;*

$inranges: Tv \times P\_TRange \to \{true, false\}$. *The function calculates containment relationship between a time point and a set of time intervals.*

The function *CF* should be established according to application. But it must meet several basic properties about return value. First, return value should not less than zero because every access will consume privilege. Second, return value should no more than attribute value before access because the access process should be terminated by system before privilege consumption exceeds maximum limit.

After the statements of authorization and extensions, the rules changing authorizations are given below. We will focus on the *auth* 4-tuple only. Detailed constraints will be discussed later.

Definitions 5. Authorization rule set

The TMAAC authorization rule set is composed by five types of rules:

Grant rule: $Grant(s,o,r,n) \leftarrow L_1 \& \ldots \& L_n$. *Grant rule is used to grant permission r with attribute n on object o to subject s.*

Access rule: $Access(s,o,r,n) \leftarrow L_1 \& \ldots \& L_n$. *Access rule is used to change attribute of authorization to n after subject s has finished accessing object o.*

Transfer rule: $Transfer(s,o,r,n) \leftarrow L_1 \& \ldots \& L_n$. *Transfer rule is used to change attribute of authorization to n after subject s has finished transferring operation.*

Merge rule: $Merge(s,o,r,n) \leftarrow L_1 \& \ldots \& L_n$. *Merge rule is used to merge multiple authorizations of subject s into one authorization.*

Revoke rule: $Revoke(s,o,r,n) \leftarrow L_1 \& \ldots \& L_n$. *Revoke rule is used to revoke authorization of subject s.*

In the above rules, $L_i, i \in N$ represents the conditions triggering execution of rules. It is the preconditions that the system grant authorization to users in grant rule. Although it has different meanings, the original authorization for subject must be included in these preconditions. After subjects finish their actions, the consumption of authorizations are calculated by consumption function defined in Definition 4 and corresponding authorization rules are used to update the attributes.

3) Completeness

It is easy to prove that the authorization rule set is the minimal set necessary to express TMAAC policies.

Theorem 1. *These authorization rules of TMAAC can express the policies completely.*

Proof:

When a transition occurs, the corresponding authorization rule is used to make a decision. So, we can do model checking based on the system state graph to prove the completeness.

The system state transition is illustrated in Fig. 3. We label the states using symbols and do model checking on the state graph.

- In S1, authorization conditions are triggered and the system grant authorization to subject. The system state enters into S2.
- In S2, the subject applies for accessing object and the system state enters into S3.
- In S3, after the system makes decision, the system state enters into S4, S6 or S7 if the request has been permitted, otherwise the system state enters into S8.
- In S4, subject accesses resource being protected. The system audits user behavior continuously.

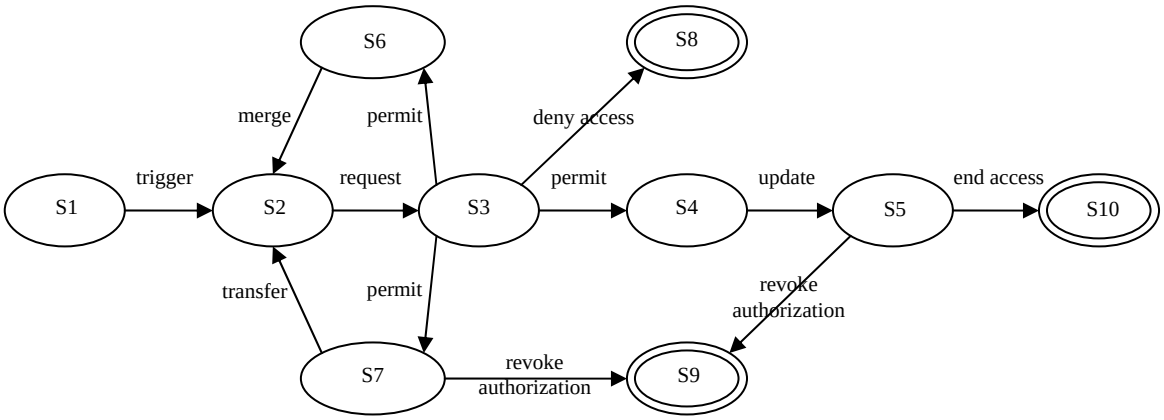

Figure 3. The state transition

The state enters into S5 after the subject finishes accessing.

- In S5, the system calculates consumption of subject authorization and updates the attribute.
- In S6, the system executes grant or transfer operation by request. Then the authorization for subject is in inconsistency and the system state enters into S2 after inconsistent authorizations have been merged.
- In S7, the request of transfer operation is executed and the authorization will be updated through transfer rule. After that, system state enters into S2.
- S8, S9 and S10 are final states and need no rules.

After model checking, we conclude that the five rules solve all problems in the process of transformation. So the rule set is complete.

Q.E.D.

C. Access Constraints

Access constraints are conditions that must be met during the accessing. There are two types of these conditions, that is, the conditions set by administrator and basic constraint. The former is set by administrator according to application needs and we don't plan to discuss this issue.

1) Basic Constraint

Subject access actions can be considered as processes with authorization attribute decreasing continuously. So, the parallel implementations of some operations may lead to privilege leakage. Basic constraints prevent that from happening.

a) Constraint 1.

$$\forall a_1, a_2 \in ACTION \ \ if \ \ a_1 = (s, o_1, r) \ \ a_2 = (s, o_2, r) \ ,$$
$$then \ \ (start(a_1), end(a_1)) \cap (start(a_2), end(a_2)) = \Phi \ .$$

Constraint 1 illustrated in Fig. 4 guarantees that subjects can not access two objects at the same time. In the figure, *a* and *b* represent resources being accessed by one subject and the starting and ending time of access actions are marked with *accessing*() and *accessed*().

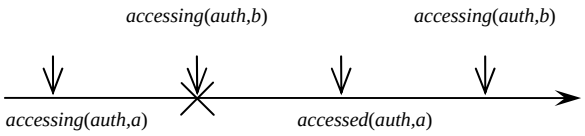

Figure 4. The constraint 1

In Fig. 4, subject can not access *b* before finishing accessing to *a*. Because the system will update authorizations at the end of access, it doesn't know how to change subject authorization attribute until subject finishes the access. Then, consumption function will be called to calculate consumption.

b) Constraint 2.

$$\forall a_1 \in ACTION \ \ , \ \ a_2 \in TRANSFER \ \ if \ \ a_1 = (s_1, o, r)$$
$$a_2 = (s_1, s_2, r) \ , then \ (t < start(a_1)) \vee (t > end(a_1)) \ .$$

Constraint 2 illustrated in Fig. 5 guarantees that subject can not transfer authorization and access resources at the same time. In the figure, the time point when subject transfers authorization is marked with *transfer*().

In Fig. 5, subject can not transfer authorization to others while accessing. Because the system doesn't know the maximum amount of authorization that subject can transfer until the consumption caused by previous action has been calculated.

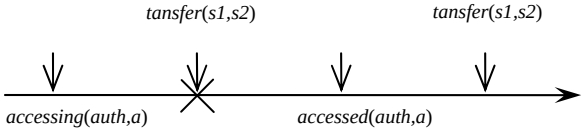

Figure 5. The constraint 2

2) Necessity of basic Constraint

Theorem 2. *The basic constraint is necessary.*

Proof:

Proof of necessity is divided into two parts. We prove one of necessary conditions in one part. Firstly, we assume that the conditions do not hold. Then we prove the correctness of them through proof of contradiction.

Part 1: Assume that constraint 1 does not hold, then

$\exists a_1, a_2 \in ACTION$, if $a_1 = (s, o_1, r)$ $a_2 = (s, o_2, r)$ then $(start(a_1), end(a_1)) \cap (start(a_2), end(a_2)) \neq \Phi$

If the authorization of subject s is *M_Auth*, where *n=ATT*(*M_Auth*), by the assumption we have

$CF(a_1, M\_Auth) < n$ and $CF(a_2, M\_Auth) < n$

$\Rightarrow CF(a_1, M\_Auth) + CF(a_2, M\_Auth) < 2n$

$\Rightarrow \Diamond(n < CF(a_1, M\_Auth) + CF(a_2, M\_Auth) < 2n)$

Contradiction with the definition of *CF* occurred, so

$\exists a_1, a_2 \in ACTION$, if $a_1 = (s, o_1, r)$ $a_2 = (s, o_2, r)$ then $(start(a_1), end(a_1)) \cap (start(a_2), end(a_2)) = \Phi$

Part 2: Assume that constraint 2 does not hold, then

$\exists a_1 \in ACTION$ $a_2 \in TRANSFER$, if $a_1 = (s_1, o, r)$ $a_2 = (s_1, s_2, r)$, then $(t > start(a_1)) \vee (t < end(a_1))$

If the authorization of subject s is *M_Auth*, where *n=ATT*(*M_Auth*), by the assumption we have

$CF(a_1, M\_Auth) < n$ and $CF(a_2, M\_Auth) < n$

$\Rightarrow CF(a_1, M\_Auth) + CF(a_2, M\_Auth) < 2n$

$\Rightarrow \Diamond(n < CF(a_1, M\_Auth) + CF(a_2, M\_Auth) < 2n)$

Contradiction with the definition of *CF* occurred, so

$\forall a_1 \in ACTION$ $a_2 \in TRANSFER$, if $a_1 = (s_1, o, r)$ $a_2 = (s_1, s_2, r)$, then $(t < start(a_1)) \vee (t > end(a_1))$

Q.E.D.

D. Ability of the Model

The ability of TMAAC is improved in many aspects. The following are some examples.

1) Activation Time

The time interval that subject is allowed to access resources must satisfy the valid time interval and time pattern constraint of authorizations.

Authorization: (*vt*,*tp*,*auth*)

Expression:

$session(s) \neq \Phi \Rightarrow$

$inrange(currenttime, vt) \wedge inranges(currenttime, tp)$

2) Usage of Permission

The system makes access decision dynamically based on authorization.

Authorization: *auth*=(*s*,*o*,*r*,*n*)

Expression:

$(s, o, r) \in session(s) \Rightarrow n > ATT(o)$

3) Accessing Resource

The authorization constraint prevents authorization from being abused.

Expression:

$(s, o_1, r_1), (s, o_2, r_2) \in session(s) \Rightarrow (o_1 \neq o_2) \vee (r_1 \neq r_2)$

4) Transferring Privilege

The mutable attribute and Transfer rule guarantee that the subjects must have enough privilege before transferring it to others.

The Auth 4-tuple: (s_1, o, r, n_1) (s_2, o, r, n_2)

Expression:

$(s_1, s_2, r) \in TRANSFER \wedge \Delta ATT(r) = n \Rightarrow n > n_1$

So the model has the abilities for expressing more complex policies and can provide more powerful security protection for resources.

IV. Implementation of TMAAC

A. Administration of Authorizations

The system grants or denies the access based on authorization, and permits or terminates ongoing access by checking authorization attribute. The TMAAC model centers on administration of authorizations to achieve these features. All system authorizations are organized into a set named Mutable Attribute-based Authorization Base (MAAB).

Definition 6. Mutable Attribute-based Authorization Base and request

Mutable attribute based authorization base is a set of mutable attribute based authorizations, that is, $\forall a \in MAAB$, $a = (vt, tp, (s, o, r, n))$, *where* $s \in S$, $o \in O$, $r \in R$, *and* $n \in ATT(R)$.

Request is the application of subject for activating authorization. It is divided into two categories, that is, access request (ARequest) which is 3-tuple (s,o,r) and transfer request (TRequest) which is 5-tuple (s_1, o, r, s_2, n).

Given the above definition, the problems to be solved are how to add, delete and update authorization in MAAB and how to make access decision according to authorization. We can deal with the first problem easily with authorization rules. Below are detailed instructions.

- First, the system checks the triggers of grant rule and adds an authorization tuple *M_auth*=(*vt*,*tp*,*auth*) in MAAB if preconditions have been satisfied.
- If users have accessed some resources or transferred their permissions to others, then the attributes of corresponding authorizations are consumed. The access rule and transfer rule can be used to update authorization attributes in MAAB.
- If authorization for a subject is inconsistent, for example, been authorized by system once again. The merge rule can be used to merge different rules into a consistent one.

- Finally, if the value of authorization attribute equals to zero, the revoke rule can be used to delete corresponding authorization tuple *M_auth*=(*vt*,*tp*,*auth*) in MAAB.

Algorithm 1. Maintain the Consistence of MAAB
Method name: *Maintain_MAAB*
Input: *type* (*type of the rule*)
Output: *None*
Steps:
If *type* = GRANT
//the Grant rule has been used
If *it is not the first time the Grant rule is used about some subject s*
Then use the "Merge rule" to merge additional authorization, And execute Maintain_MAAB(MERGE).
Else If *type* = ACCESS
// the Access rule has been used
If *the authorization attribute equal zero,*
Then use the "Revoke rule" to revoke user privilege,
And execute Maintain_MAAB(REVOKE).
Else If *type* = TRANSFER
//the Transfer rule has been used
If *the authorization attribute equal zero,*
Then use the "Revoke rule" to revoke user privilege,
And execute Maintain_MAAB(REVOKE).
Use the "Merge rule" to merge additional authorization,
And execute Maintain_MAAB(MERGE).
Else If *type* = MERGE
//the Merge rule has been used
Merge different authorizations of subject s in MAAB into one.
Else
//the Revoke rule has been used
Delete the authorization 3-tuple from MAAB.

It is clear that the above algorithm may call itself recursively. However, through careful analysis we can find that the algorithm will call itself only when the parameter is GRANT, ACCESS, or TRANSFER and then it will call itself with REVOKE or MERGE. So it may call itself at most once and the algorithm will stop eventually.

B. Decision Function

There are two kinds of time points in time sequence when the system needs to make decisions. One is request time point, the other is scheduling events time point. Before the detailed discussion, we first give some notations to be used in algorithms. Table I lists the data structure of event list entry.

TABLE I. DATA STRUCTURE OF EVENT LIST ENTRY

Item	type
Starting point	Virtual time
Ending point	Virtual time
Subject	*S*

The event list is named as event_list. The system schedules events according to entries of event_list. The entry can be inserted into or deleted from event_list through the following two functions.

Add(s, *start_time*, *end_time*): This function inserts an entry about *s*. The starting point item equals to start_time and the ending time point item equals to end_time.

Delete(s, *end_time*): This function deletes an entry about *s*. The ending point item of deleted entry equals to end_time.

1) Decision for Request

The subject requests to activate authorization for access. The decision for request determines whether the request is valid. The following algorithm implements decision for request.

Algorithm 2. Decision for Request
Method name: *Access_Decision*
Input: *Request*
Output: *permit* or *deny*
Steps:

If Request = Arequest
//make decision for access request
If $\exists a \in MAAB, a = (vt, tp, (s, o, r, n))$
//check the grant preconditions
$\neg\exists action \in SESSION, action_s = s$
$(cuttenttime \in vt) \wedge (\exists i \in N, currenttime \in tp_i)$
$ATT(r) \geq ATT(o)$
If the above conditions are satisfied, then
Add(s, *currenttime*, *endtime*)
Return *permit*
Else
Return *deny*
Else
//make decision for transfer request
If $\exists a_1 a_2 \in MAAB, a_1 = (vt_1, tp_1, (s_1, o, r, n_1))$
$a_2 = (vt_2, tp_2, (s_2, o, r, n_2))$
//check the transfer preconditions
$\neg\exists action \in SESSION, action_s = s_1$
$(cuttenttime \in vt) \wedge (\exists i \in N, currenttime \in tp_i)$
$ATT(r) \geq n$
$(vt_2 \subseteq vt_1) \wedge (tp_2 \subseteq tp_1)$
If the above conditions are satisfied, then
Return *permit*
Else
Return *deny*

2) Decision for Scheduling Event

The system maintains an event list named event_list. Decision function is used to make decision when events in this list occurred. The following algorithm implements decision for scheduling event.

Algorithm 3. Decision Function for Scheduling Event
Method name: *Schedule_Decision*
Input: *the subject s*
Output: *None*
Steps:

If *scheduling event about s occurred*
//calculate the consumption
$\exists action \in SESSION, action = (s, o, r)$
$\Delta ATT(r) = CF(r, action)$
Terminate the user's access
Delete(s, *endtime*)
Update authorization.
Else
//calculate the consumption
$\exists action \in SESSION, action = (s, o, r)$
$\Delta ATT(r) = CF(r, action)$
Terminate the user's access
Update authorization.

V. Conclusion

To meet the new requirements in recent information system, this paper presents a new access control model named TMAAC. The TMAAC model mainly introduces extensions in time and mutable attribute compared with traditional access control. All authorizations in the model only can exist in a period of time. User access behavior must conform to the time pattern and can not exceed the maximum amount of permission. The discussion about authorization administration and access decision functions solve the problems in implementation. So, the TMAAC model is more powerful than traditional access control models in policy expression. Since the model is very simple, it also can be easily used.

The future research includes enforcement of TMAAC policies in real system. Continuous control of user behaviors should be realized discretely in information system. It is an approximation to policies. How to guarantee the correctness of the realized counterparts of policies is a challenge work. Some researchers put forward some methods to ensure the dependable development process. But these methods are still unable to confirm the degree of approximation. So it is necessary to find a way to evaluate how well the system realizes policies.

References

[1] A. Gal, V. Atluri, "An authorization model for temporal data," In Proceedings of the Seventh ACM Conference on Computer and Communication Security, 2000.
[2] E. Bertino. "An access control model supporting periodicity constraints and temporal reasoning," 1998.
[3] E. Bertino, C. Bettini, P. Samarati, "A temporal authorization model," CCS '94, 1994.
[4] E. Bertino, A. Bonatti, E. Ferrari, "TRBAC: A temporal role-based access control model," ACM Transactions on Information and System Security, 2001.
[5] J. Park and R. Sandhu, "The UCONABC usage control model," ACM Transactions on Information and Systems Security, Feb., 2004.
[6] J. Park, X. Zhang, and R. Sandhu, "Attribute mutability in usage control," IFIP WG 11.3, 2004.
[7] M. Niezette, J. Stevenne, "An efficient symbolic representation of periodic time," In Proceedings of the First International Conference on Information and Knowledge Management, 1992.
[8] R. Sandhu, "The ASCAA principles for next-generation role-based access control," 2008.
[9] R. Sandhu and J. Park, "Usage control: a vision for next generation access control," The Second International Workshop on Mathematical Methods, Models and Architectures for Computer Networks Security, 2003.
[10] X. Zhang, J. Park, F. Parisi-Presicce, "A logical specification for usage control," SACMAT'04, 2004.
[11] Z. Baoxian, R. Sandhu, X. Zhang, X. Qin, "Towards a times-based usage control model," IFIP Data and Applications Security, 2007.

YANG Ran was born in 1981. In 2005, he received his master degree from Institute of Chemical Defence, Peking, P.R.C. He served as a Ph.D. candidate of Department of Computer Science and Technology, Tsinghua University. His research interests include formal method and access control model and technology.

LIN Chuang, born in 1948, Ph.D. professor and Ph.D. supervisor. He is now the head of the Institute of Network Technology of Department of Computer Science, Tsinghua University. He received the Ph.D. degree in Computer Science from Tsinghua University in 1994. His current research interests include computer networks, performance evaluation, network security, Petri net theory, trustworthy networks and trustworthy computing. He has published more than 180 papers in research journals and IEEE conference proceedings in these areas and has published four books.

Feng Fujun was born in 1978. She received her master degree from Engineering College of Missile Force PLA, Xi'an, Shanxi Province, P.R.C. She served as a Ph.D. candidate of Department of Computer Science and Technology, Tsinghua University. Her research interests include network security, Petri net and access control.