Front. Electr. Electron. Eng. China (2006) 3: 283–286
DOI 10.1007/s11460-006-0033-0

RESEARCH ARTICLE

HONG Fan, DUAN Su-juan, LI Cheng-bing

A delegable model based on graph

Abstract Trust management was discussed according to the security requirements in distributed environment. A graph-based delegable model and its formal description were given. Permissions propagating to control, conflict and cyclic in delegable model were analyzed and resolved. Finally, the scheme of determining access request of subject in distributed environment and future work was proposed.

Keywords access control, trust management, delegable, graph, model

1 Introduction

Computer systems are moving to the distributed pattern from the stand-alone and centralized mode gradually with the development of computer and telecommunication technology. Recently, trust management, which has a prominent characteristic that supports access to unknown subjects, was researched extensively as a technology of access control for supporting distributed environment [1], and adopts transmissible authorization to enhance the expansibility. In Ref. [2], keynote was defined, supporting the delegation of permission and forming the chain of credential. Reference [3] utilized SPKI to solve authorization in distributed environment. In [4], Li Ning-hui et al. proposed trust management framework based role (RT0). To support the expansibility in distributed environment, decentralized, and transmissible authorization is a necessary mechanism, in which many problems exist:

1) Difficulties in controlling permissions propagating. x can trust y, but x cannot assure that y does not abuse permission. Namely, trust cannot be transmitted.

Translated from *Journal of Beijing University of Posts and Telecommunications*, 2005, 28(6): 5–7 (in Chinese)

HONG Fan, DUAN Su-juan (✉), LI Cheng-bing
Department of Computer,
Huazhong University of Science and Technology,
Wuhan 430074, China
E-mail: dsj8837@126.com

2) Compared to centralized authorization, the decentralized method has preferable decentralization, but easily brings in many problems, such as circle and conflict in authorization.

To trust management, obviously, trust cannot be transmitted: x trusts y and y trusts z, but x also trusting z cannot be deduced. In the past research, the requirement of decentralized authorization can be satisfied, but permissions propagating controlling, the problem of circle and conflict, cannot be solved effectively. To deal with those problems, an effective model of delegation is proposed in this paper.

2 A delegable model based on graph

2.1 Basic elements in model

2.1.1 Regulation

S is the set of subjects (users). O is the set of objects (resource). P is the set of permissions. T is the set of time, $I = \{[b, e] \mid b, e \in T$, denotes the start-time and the halt-time, respectively, and $b \leqslant e\}$ is the useful-life of authorization.

2.1.2 Definition

Definition 1 Depth of delegation. $D = \{d \mid d \in N \bigcup \{0, 3\}\}$, $\forall d \in D$, where d denotes the depth of delegation, which means the degree to which the grantor of permission trusts the receptor, i.e. the degree to which the permission can be delegated, and can be used to control propagating permission. If $d = 0$ holds good, the permission cannot be delegated continuously, and if $d \in N$ holds good, then the permission can be delegated continuously in not more than n steps, equation $d = *$ denotes that the permission can be delegated any number of times. Because the depth of delegation is finite in practical system, we regulate that the maximum value of depth is D and define

$[0 \cdots *]_D = [0 \cdots D] \bigcup \{*\}$ and $(\forall d \in [0, D]$ and $d < *)$.

Rules for computing depth of delegation: $\forall d_1, d_2 \in [0 \cdots *]_D$

$$d_1 + d_2 = \begin{cases} *, & \text{if } d_1 = * \text{ or } d_2 = * \text{ or } d_1 + d_2 > D \\ d_1 + d_2 & \text{else} \end{cases}$$

$$d_1 - d_2 = \begin{cases} 0, & \text{if } d_1 < d_2 \\ *, & \text{if } d_1 = * \text{ or } d_1 - d_2 > D \\ d_1 - d_2, & \text{else} \end{cases}$$

Definition 2 Authorization. An authorization is a 6-tuple, $\text{Grant}(g,s,o,p,d,i) \in S \times S \times O \times P \times D \times I$, where $g \in S$, $s \in S$, $o \in O, p \in P, d \in D, i \in I$, denotes grantor g grants the permission p, $p = \{r \mid r \in P\}$ and p is the subset of P, to user s in time limit i with less than d steps, where depth of delegation d will descend by 1 there is once a delegation from g to s. $\text{Grant}(g,s,o,p,d,i)$ is an authorization relevant to time, and is constrained by time limit; permission over useful-life will be revoked automatically. An authorization is a 6-tuple $a = \text{Grant}(g,s,o,p,d,i)$, where each element can be shown as following: $a.g, a.s, a.o, a.p, a.d, a.i$.

For $\forall x = [b_x, e_x], y = [b_y, e_y] \in I$, several relationships are defined: $x >_t y \Leftrightarrow b_x < b_y \wedge e_x > e_y$, $x \geq_t y \Leftrightarrow x >_t y \vee (b_x = b_y \wedge e_x = e_y)$. $x \bigcap_t y = (b,e)$, where $b = \max(b_x, b_y)$ and $e = \min(e_x, e_y)$ hold. $x \bigcup_t y = (b,e)$, where $b = \min(b_x, b_y)$ and $e = \max(e_x, e_y)$ hold.

Definition 3 Boolean function for time interval. $\text{in} : T \times I \rightarrow \{\text{true}, \text{false}\}$. $\forall t \in T, i \in I, \text{in}(t,i) = \text{true}$, we can say t is in the time interval of i.

Definition 4 Status of authorization and the set of authorizations. $A_t = \{\text{Grant}(g,s,o,p,d,i) \mid \text{in}(t,i) = \text{true}\}$ denotes all the authorizations at t, $t \in T$, and depicts the status of authorization. The set of authorizations, $A_S = \bigcup_{\forall t \in T} A_t$, denotes the set of all the authorizations. $\forall o \in O$, A_{So} denotes all the authorizations relevant to o.

Definition 5 Original grantor. Any permission for accessing the resource must be granted by the owner of resource. The owner may be a common user or a security manager in an enterprise or department. By definition S_o is the owner of o, that is to say, S_o is the original grantor.

Definition 6 Directed graph of delegation: $G = \{G_o \mid o \in O\}$, where G_o is the directed graph with a tag relevant to the object o, $G_o = (V,E,l)$, and denotes all the authorizations relevant to o. V is the set of nodes, and each node corresponds to a subject which holds the authorization relevant to the object o. E is the set of edges, and each edge is an order binary, grantor and receptor, $\forall e = (g,s) \in E \Leftrightarrow \text{Grant}(g,s,o,p,d,i) \in A_{S_o}$; l denotes function: $E \rightarrow P$: $l((g,s)) = \{p \mid \text{Grant}(g,s,o,p, \quad d,i) \in A_{S_o}\}$.

l represents the permission that the grantor assigns to the receptor. If $l((g,s)) \neq \varnothing$, there exists a directed edge with tag $l(e)$ from the node g to the node s in the graph G_o.

G_o is the subgraph of G. From the node a to the node b, there is a directed path(a,b,n) with length n: (a,x_1) (x_1,x_2) $\cdots$ (x_{i-1},x_i) $(x_i,x_{i+1}) \cdots (x_{n-1},b)$, where $x_i \in V \wedge i \in [1, n-1]$, $a = x_0$, $b = x_n$ holds good. In the path(a,b,n), $\text{predecessor}(x_i) = \{x_j \mid x_j \in V \wedge j \in [0, i-1]\}$ denotes the set of predecessors, where x_{i-1} is the immediate predecessor of x_i, and $\text{successor}(x_i) = \{x_k \mid x_k \in V \wedge k \in [i+1, n]\}$ denotes the set of successors, where x_{i+1} is the immediate successor of x_i. Nodes a and b are called the start-point and the end-point respectively in the directed path.

2.2 Policy on controlling propagating permission

$\forall o \in O$, if $\exists a = \text{Grant}(g,s,o,p,d,i) \in A_{S_o}$, following problems need to be considered.

1) Whether or not $a.g$ can assign $a.p$, accessing object $a.o$, to $a.s$ in $a.i$.

2) If $a.d \neq 0$ holds good, then how does $a.s$ delegate $a.p$ ulteriorly, accessing object $a.o$.

3) How to control the time limit in the ulterior delegation process?

We can adopt deep first search (DFS) algorithm because permission $a.p$ must be granted by the owner s_o originally. We attempt to find a directed path from s_o to $a.g$. If the path exists, then $a.g$ can assign $a.p$ to $a.s$. In fact, the process of DFS is that each node searches an adjacent node. Suppose $G_o(V,E,l)$ adopts the adjacent table as the storage structure, the time to find the neighbor is $O(|E|)$, and then the complexity of the algorithm is $O(|V| + |E|)$.

2.2.1 Control the depth of delegation

The depth of delegation denotes the steps that permission can be delegated. Suppose $\exists \text{Grant}(g,s,o,p,d,i) \in A_{So}$, if $a.d = 0$ holds good, $a.s$ cannot delegate $a.p$, and if $a.d \neq 0$ and $\exists a' = \text{Grant}(s,s',o,p',d',i') \in A_{S_o}$ are satisfied, then the inequation $a.d - a'.d' \geq 1$ will hold good. If there is a path of delegation from s_o to s, $a_1, a_2, \cdots, a_n$, then the inequation $a_1.d > a_2.d > \cdots > a_n.d$ will hold, namely the value of depth descends only with the direction of delegation.

2.2.2 Constraints in delegation

1) Constraint of permission. The permissions $a_1.p, a_2.p, \cdots,$

$a_n.p$ relevant to the authorizations $a_1, a_2, \cdots, a_n$, from s_o to s, and the relationship $a_1.p \supseteq a_2.p \supseteq \cdots \supseteq a_n.p$ should be satisfied. Due to the decentralization, the relationship cannot be satisfied when it is delegating ulteriorly. In addition, any permission of accessing the resource must be granted by the owner for the resource, so the permission that the authorization a_m can attain is $a_m.p = a_1.p \bigcap a_2.p \bigcap \cdots \bigcap a_{m-1}.p$, where $m \in [2, n]$.

2) Constraint of useful-life. The inequation, $a_1.i \geq_t a_2.i \geq_t \cdots \geq_t a_i.i, a_m.i = a_1.i \bigcap_t a_2.i \bigcap_t \cdots \bigcap_t a_{m-1}.i$, should be sati-sfied for useful-life $a_1.i, a_2.i, \cdots, a_i.i$, where $m \in [2, n]$.

2.3 Circle and conflict in authorization

The decentralized and transmissible authorization satisfies the security requirement in distributed environment, at the same time decentralization and transmission induce many problems.

1) Circle in authorization. Suppose $a_1, \cdots, a_i, \cdots, a_n \in A_{S_o}$, $a_1.s = a_{i+1}.g (i \in [1, n-1])$, if $a_n.s = a_1.g, a_1, \cdots, a_n$ is called a cyclic authorization.

2) Conflict in authorization. Suppose $a_1, a_2 \in A_{S_o}$, if $a_1.s = a_2.s, a_1.g \neq a_2.g, a_1.p \neq a_2.p$ or $(a_1.p \neq a_2.p$ and $a_1.d \neq a_2.d)$, if the depth of delegation or the permission is different for two different grantors, $a_1.g$ and $a_2.g$, to the same receptor, there may be conflicts.

2.3.1 Detection algorithm about circle

Cyclic delegation induces circle in G_o. When g delegates permission p to s and suppose $a = \text{Grant}(g, s, o, p, d, i)$, then detection algorithm can be designed as following: bool circlecheck(G_o, a)

Input: G_o, a.

Output: true denotes that the circle is found, else false.

We define the set of Head and Tail, at the beginning $\text{Head} = \text{Tail} = \varnothing$ and variable $\text{RT} = 0$.

Step 1 Compute the set of predecessor nodes, predecessor($a.g$), and the set of successor nodes, successor($a.s$).

Step 2 Head = successor($a.s$), Tail = predecessor($a.g$).

Step 3 $\text{RT} = \text{RT} + 1$. If $(\text{RT} = 1$ and $((\text{Head} \bigcup \{a.s\}) \bigcap (\text{Tail} \bigcup \{a.g\}) \neq \varnothing))$ holds good and the circle is detected, then true and halts are returned. If $(\text{RT} \neq 1$ and $(\text{Head} \bigcap \text{Tail}) \neq \varnothing)$ holds good and the circle is detected, then true and halts are returned. If $(\text{RT} > [|V| - 1/2])$ holds good, then false and halts are returned.

Step 4 Compute $\text{Head} = \bigcup_{\forall s \in \text{Head}} \text{successor}(s)$ and $\text{Tail} = \bigcup_{\forall s \in \text{Tail}} \text{predecessor}(s)$ and go to Step 3.

The above algorithm makes the set of predecessors and the set of successors closed middle-ward each other, and simultaneously computes the intersection to deter whether there exists a cyclic path in the graph $G_o(V, E, l)$ or not. Considering that the radix of the set of predecessors and the set of successors does not exceed $|V|$ and that the length of noncyclic path does not exceed $|V| - 1$, then the time complexity of the algorithm is $O(|V|^2)$. To avoid the cyclic delegation, we regulate $\forall a \in A_{S_o}, a.s$ cannot delegate permission $a.p$, delegated to $a.s$ directly or indirectly from $a.g$, to $a.g$ by cyclic delegation. When $a.s$ wants to delegate the subset of $a.g$ to others ulteriorly, the action will be forbidden if $\text{circlecheck}(G_o, a) = \text{true}$ holds good.

2.3.2 Rules for eliminating the cyclic authorization

We distinguish the following for conflict authorization, $\forall a_1, a_2 \in A_{S_o}$.

1) Suppose $a_1.g \in \text{predecessor}(a_2.g)$ or $a_1.g \in \text{successor}(a_2.g)$, where $a_1.g$ is the predecessor or the successor of $a_2.g$. We regulate that the priority of the predecessor is higher than the priority of the successor. If $\text{predecessor}(a_2.g)$ holds good, then a_1 mantles a_2. If $a_1.g \in \text{successor}(a_2.g)$ holds good, then a_2 mantles a_1. The authorization that is mantled need not be deleted from the graph and can be marked with nonactivation. When a_1 or a_2 is revoked, a_2 or a_1 can be activated newly.

2) If $a_1.g \notin \text{predecessor}(a_2.g)$ and $a_2.g \notin \text{predecessor}(a_1.g)$ holds good, then there does not exist a relationship between the predecessor or the successor and $a_1.g$ and $a_2.g$, following rules of revoking can be defined:

a) Intersection ($\bigcap$). S gains the permission to access object o, $p = a_1.p \bigcap a_2.p$, then the depth of delegation is $d = \min(a_1.d, a_2.d)$ and useful-life of permission is $i = a_1.i \bigcap_t a_2 i$.

b) Union ($\bigcup$). S gains the permission to access object o, $p = a_1.p \bigcup a_2.p$, then the depth of delegation is $d = \max(a_1.d, a_2.d)$ and useful-life of permission is $i = a_1.i \bigcup_t a_2 i$.

3 Make decision about access request

In a distributed environment, how to make a decision about the access request to resource o can be boiled down to find a directed path from the node S_o to the node s while eliminating

the circle and conflict in G_o. Due to many existing elimination rules, there may be many different G_o. (G_o, R), where we can choose appropriate rule R according to the specific application.

4 Conclusions

This paper proposes a delegable model-based on graph, and discusses and resolves circle problem and conflict problem due to decentralized and transmissible authorization. In this model, we control the depth of delegation of permission lengthways, and enhance the constrain to restrict the propagation in delegation process. For the conflict in authorization, according to the relationship between predecessors and successors, we define the elimination rules, which can also guarantee the switch among different rules, to improve the flexibility.

However the delegation is based on trust among the principles and hence bears risk. How to evaluate the risk in delegation process and define the reasonable criteria of evaluation need to be researched all the same.

References

1. Chen Yu, Cao Dong-gang, Meng Luo-ming, Access control mechanism applied in telecommunicaction management network, Journal of Beijing University of Posts and Telecommunications, 2000, 23(3): 20–24 (in chinese)
2. Blaze M., Feigenbaum J., Ioannidis J. et al., RFC 2704-1999, The keynote trust-management version 2.
3. Liu Jing, Zhou Ming-tian, Decentralized CORBA authorization service based on SPKI certificate, Journal of Beijing University of Posts and Telecommunications 2003, 26(Suppl): 81–88 (in chinese)
4. Li Ning-hui, Grosof B. N., Feigenbaum J., Delegation logic: a logic based approach to distributed authorization, ACM Transactions on Information and System Security, 2003, 6(1): 128–171