Complexity of Credential Processing

Krzysztof Sacha
Warsaw University of Technology
Warszawa, Poland
k.sacha@ia.pw.edu.pl

Abstract—**Role-based Trust management (RT) languages are an effective means for describing security policies in decentralized and open environments. The statements of a RT language are credentials that describe entities and roles, which the entities can play in the system. A set of credentials can define the security policy and allow deciding on who is authorized to access a resource, and who is not. A credential graph is a graphical representation of the semantics of a given set of credentials. Making a decision on the membership of an entity in a role is equivalent to checking whether an appropriate path exists in the graph. The core part of this paper is a definition of an algorithm for building a credential graph for a set of RT^T credentials, and an evaluation of the complexity of the graph building process.**

Keywords-software security; trust management; role-based trust management languages; credential graph

I. INTRODUCTION

Access control is a mechanism for making decisions on whether to allow or to deny access to a system resource by an entity. The traditional approach to this problem is such that the system knows the identity of all users and the decision is based on a verification of the identity of the requester. One possible mechanism for such verification is a login window. When the system grows bigger and bigger, the management of privileges granted to particular users becomes more and more difficult. An attempt to overcome these difficulties is the introduction of roles [1]. A role is a set of users, which have the same rights with respect to the system resources. The privileges are no longer assigned to individual users, but to the roles. However, the system must still know the members of roles and the access control is still based on a verification of identity.

Quite a new problem arises in distributed open systems, in which the identity of users is not known in advance and can change over time. If this is the case, a new approach to access control is needed. For example, consider an Internet Bank, which offers special rates for the employees of public universities. The Bank does not know all the employees of the universities. If I send a copy of my identity card to the bank, this will not help in deciding whether I am eligible for a special rate or not. What can help are two credentials: The first one reads that I am employed at a university, and the second proves that the university is a public university. Such credentials can in practice be implemented as signed electronic documents and processed automatically.

Credentials are a means of expressing security policies and answering security queries. To resolve a query over a set of credentials, one can build a credential graph and search through the paths within the graph. The goal of this paper is to define an algorithm for building a credential graph, and to evaluate the efficiency of the graph building process.

The rest of this paper is organized as follows. An overview of languages to express security credentials is given in Section 2. A Role-based Trust management language RT^T is presented in Section 3. Credential graph is introduced in Section 4, and the complexity of graph building is assessed in Section 5. Final remarks are gathered in Conclusions.

II. RELATED WORK

Several trust management systems were developed and described in the literature. The early work on PolicyMaker [2] and KeyNote [3] identified entities by public keys and used a language to define credentials that allowed assigning privileges to entities and delegating permissions from one entity to another entity or to a group of enumerated entities. This way, complex security policies, like threshold and separation of duties, which required a number of entities to agree on some fact, could be expressed. A missing feature was the possibility of defining roles, role membership, and delegation of permissions based not on the identity of entities but on the membership of roles.

This gap has been filled by SPKI/SDSI [4] and a family of RT languages [5], which use roles to represent sets of entities that have the same rights and privileges within the system. Particular languages of RT family differ with respect to the expressive power and complexity. The basic language RT_0 allows describing roles, delegation of authority over roles and role intersections. Other languages add new features to RT_0. RT^T provides manifold roles, i.e. roles that can be satisfied only by groups (sets) of cooperating entities, to express threshold and separation of duties policies.

All the RT languages have well defined syntax and intuitive meaning [5,6]. A set-theoretic semantics has been formally defined for RT_0 in [7,8] and for RT^T in [9]. The semantics of a set of credentials can be represented graphically in the form of a credential graph. A subset of this graph, called a credential chain, is an effective means for resolving security queries. An algorithm for constructing a credential graph for a set of RT_0 credentials, and a proof of correctness, has been defined in [7]. An algorithm for constructing a credential graph for a set of RT^T credentials, and a proof of correctness, has been defined in [10]. However, no evaluation of the computational complexity of this algorithm has been published as yet.

This work was partially supported by the Polish Ministry of Science and Higher Education under the grant number 5321/B/T02/2010/39.

III. ROLE-BASED TRUST MANAGEMENT LANGUAGE RT^T

All the RT languages are built upon a set of three basic notions. **An entity** is someone who issues certificates, makes requests to access a resource, or both. In a computer system, an entity can be identified by a user account or by a public key. **A role name** represents privileges that can be assigned to entities, like Admin, Guest or Simple User in Windows operating system. **A role** is a set of entities or a set of groups of entities that have privileges related to a certain role name.

RT^T language allows manifold roles. It defines six types of credentials that are used for describing membership of roles, delegation of privileges from one role to another, and security policies. Writing credentials we use capital letters to denote groups of entities, which can be role issuers or role members, and small letters to denote role names. Roles are written using a dot notation: Role issuer – dot – role name. Using this notation we can describe RT^T credentials as shown in Table I.

If the owner of resources defines privileges assigned to role names (this is not part of RT^T), then he or she can specify security policy related to the resources using RT^T credentials. The semantics of a set of RT^T credentials defines groups of entities that can play the roles. Formally, the semantics of RT^T defines the meaning of a set of credentials as a relation over a set of roles and the power set of entities. Thus, the meaning of a role is defined as a set of all the groups of entities that can fulfill the role. This way, manifold roles are supported. If the role is not manifold, then the group of entities is a singleton set.

Let E be a set of entities, R a set of role names and P a set of RT^T credentials. The semantics S_P of the set P is a relation:

$$S_P \subseteq 2^E \times R \times 2^E$$

An instance of this relation, e.g. (A, r, B) consists of a group A of entities that issue a role, the role name r and a group B of entities that fulfill the role $A.r$. If the cardinality of B in (A, r, B) is greater than one, then $A.r$ is a manifold role.

Let A, B, C, X, Y be arbitrary groups of entities (may be singletons) and r, s, t arbitrary role names. The semantics of RT^T is defined in the following way [10].

Definition 1 (Semantics of RT^T). The semantics of a set P of RT^T credentials is the smallest relation $S_P \subseteq 2^E \times R \times 2^E$, which is closed with respect to the following six properties:

TABLE I. RT^T CREDENTIALS

Syntax	Meaning
$A.r \leftarrow B$	group B of entities satisfies role $A.r$
$A.r \leftarrow B.s$	role $A.r$ includes all members of $B.s$ (delegation of authority over r from A to B)
$A.r \leftarrow B.s.t$	role $A.r$ includes all members of $C.t$, where C is a member of $B.s$ (delegation of authority over r from A to all the members of $B.s$)
$A.r \leftarrow B.s \cap C.t$	role $A.r$ includes those who are members of both roles $B.s$ and $C.t$
$A.r \leftarrow B.s \oplus C.t$	role $A.r$ can be satisfied by one member of role $B.s$ and one member of role $C.t$ (separation of duties)
$A.r \leftarrow B.s \otimes C.t$	role $A.r$ can be satisfied by one member of role $B.s$ and one member of role $C.t$, where both members are disjoint (threshold policy)

- If $A.r \leftarrow X \in P$, then $(A, r, X) \in S_P$
- If $A.r \leftarrow B.s \in P$ and $(B, s, X) \in S_P$, then $(A, r, X) \in S_P$
- If $A.r \leftarrow B.s.t \in P$ and $(B, s, C) \in S_P$ and $(C, t, X) \in S_P$, then $(A, r, X) \in S_P$
- If $A.r \leftarrow B.s \cap C.t \in P$ and $(B, s, X) \in S_P$, $(C, t, X) \in S_P$, then $(A, r, X) \in S_P$
- If $A.r \leftarrow B.s \oplus C.t \in P$ and $(B, s, X) \in S_P$, $(C, t, Y) \in S_P$, then $(A, r, X \cup Y) \in S_P$
- If $A.r \leftarrow B.s \otimes C.t \in P$ and $(B, s, X) \in S_P$, $(C, t, Y) \in S_P$ and $X \cap Y = \phi$, then $(A, r, X \cup Y) \in S_P$ □

The relational semantics of RT^T can easily be represented in a functional form. Denote the power set of entities by $F = 2^E$. An element in F is a group of entities from E. An element in 2^F is a set of groups of entities from E (e.g. a set of role members). The semantics of P can now be described as a function:

$$\hat{S}_P : 2^E \times R \rightarrow 2^F$$

which maps each role from $2^E \times R$ to a set of all such groups of entities, which are members of this role. Knowing the relation S_P, one can define the function $\hat{S}_P$ as follows:

$$\hat{S}_P(A.r) = \{ X \in 2^E : (A, r, X) \in S_P \}$$

Let EX_P be the set of role expressions that appear at left or right side of symbol $\leftarrow$ in credentials of set P. The function $\hat{S}_P$ can be extended to the domain of role expressions EX_P:

$$\hat{S}_P : EX_P \rightarrow 2^F$$

by adding the following six definitions, related to six types of RT^T credentials ($X \subseteq E$ is a set of entities, may be a singleton):

$$\hat{S}_P(X) = \{ X \}$$

$$\hat{S}_P(A.r) = \{ X : (A, r, X) \in S_P \}$$

$$\hat{S}_P(B.s.t) = \bigcup_{C:\,(B,s,C) \in S_P} \{ X \in 2^E : (C, t, X) \in S_P \}$$

$$\hat{S}_P(B.s \cap C.t) = \{ X : (B, s, X) \in S_P \wedge (C, t, X) \in S_P \}$$

$$\hat{S}_P(B.s \oplus C.t) = \{ X \cup Y : (B, s, X) \in S_P \wedge (C, t, Y) \in S_P \}$$

$$\hat{S}_P(B.s \otimes C.t) = \{ X \cup Y : (B, s, X) \in S_P \wedge (C, t, Y) \in S_P \wedge X \cap Y = \phi \}$$

IV. CREDENTIAL GRAPH

The privileges in a system are assigned to roles, such as $A.r$ or $B.s$, which members are groups (may be singletons) of entities. When a group X of entities tries to access a resource, then the access mediator needs to decide whether X is a member of a role, say $A.r$, which has a privilege to access the resource. The decision is based on a set of credentials. One way to make this decision is to build and check a credential graph.

Credential graph represents the semantics of a set P of credentials that define the security policy within the system. The nodes of the graph are role expressions, which appear within the credentials, and the directed edges reflect inclusion of sets that are the meaning of those expressions. Making a decision on the membership of a group X of entities in the role

$A.r$ is equivalent to checking whether a path from X to $A.r$ exists in the graph. A construction of the credential graph of a set of RT_0 credentials was introduced in [7]. An extended credential graph of a set of RT^T credentials is presented below.

Let P be a set of RT^T credentials over a set E of entities and a set R of role names. The credential graph that represents the semantics of P is defined in the following way.

Definition 2 (RT^T Credential Graph). Credential graph of a set P of RT^T credentials is a pair $G_P = (N_P, E_P)$ comprising a set N_P of nodes, which are role expressions that appear in credentials from P and groups of entities from E, and a set E_P of directed edges, which are ordered pairs of nodes from N_P. The sets N_P and E_P are the smallest sets that are closed with respect to the following properties:

1) If $A.r \leftarrow e$, where e is a role expression, belongs to P, then the nodes $A.r$ and e belong to N_P and a *credential edge* $(e, A.r)$ belongs to E_P.

2) If role expressions $B.s.t$ and $B.s$ belong to N_P, then for each $X \subseteq E$, such that $X.t$ belongs to N_P and a path from X to $B.s$ exists in G_P, a *derived edge* $(X.t, B.s.t)$ belongs to E_P.

3) If role expressions $B.s \cap C.t$, $B.s$, and $C.t$ belong to N_P, then for each $X \subseteq E$, such that paths from X to $B.s$ and from X to $C.t$ exist in G_P, a *derived edge* $(X, B.s \cap C.t)$ belongs to E_P.

4) If role expressions $B.s \oplus C.t$, $B.s$ and $C.t$ belong to N_P, then for each X, $Y \subseteq E$, such that paths from X to $B.s$ and from Y to $C.t$ exist in G_P, a *derived node* $X \cup Y$ belongs to N_P and a *derived edge* $(X \cup Y, B.s \oplus C.t)$ belongs to E_P.

5) If role expressions $B.s \otimes C.t$, $B.s$ and $C.t$ belong to N_P, then for each X, $Y \subseteq E$, such that $X \cap Y = \phi$ and paths from X to $B.s$ and from Y to $C.t$ exist in G_P, a *derived node* $X \cup Y$ belongs to N_P and a *derived edge* $(X \cup Y, B.s \otimes C.t)$ belongs to E_P. □

Example. Consider a computational grid, which allows a scientific instrument to be used remotely by several research teams from different universities. Local owner (A) of the instrument may define a role $A.use$ and decide that only the members of this role are allowed to use the instrument. Then, A can add members (B, C) of own research team to the role:

1. $A.use \leftarrow B$
2. $A.use \leftarrow C$

A knows the leaders (X, ...) of other teams but does not know the members of these teams. So, A can decide to delegate responsibility to the team leaders, and to allow all members of their teams to use the instrument. To do this, A can issue a role *leader*, and allow the members of this role (e.g. X) to designate members of their teams as members of role $A.use$. These new members are not known to A and their membership to roles is administered by others (team leaders in this example).

3. $A.leader \leftarrow X$
4. $A.use \leftarrow A.leader.team$
5. $X.team \leftarrow Y$

To decide whether Y is allowed to use the instrument or not, one can build a credential graph (Figure 1) and check for a path in the graph from Y to role $A.use$.

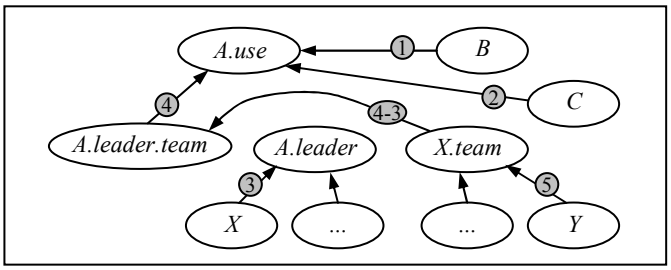

Figure 1. Credential graph of a grid access control system

V. Complexity of the Credential Graph

Credential graph G_P of a set P of credentials consists of nodes and edges. Part of the nodes can be defined by a static analysis of credentials from P. These nodes, called static nodes, are roles that stand at the left hand side of symbol $\leftarrow$ and role expressions that appear in credentials at the right hand side of symbol $\leftarrow$. Other nodes are created dynamically in the process of building the graph, by repetitive scanning through the set of credentials and executing role expressions with the operators $\oplus$ and $\otimes$. These additional nodes are union sets of entities, added to the graph according to properties 4 and 5 in Definition 2. Credential edges (property 1) are defined statically, while derived edges (properties 2 through 5) are added dynamically.

Looking at Definition 2, one can note that dynamically added nodes can be connected directly only to role expressions of type $B.s \oplus C.t$, $B.s \otimes C.t$ and $B.s \cap C.t$. In order to enhance the efficiency of the graph building algorithm, the necessary search for paths within the graph will be restricted to a subgraph composed of static nodes and edges between these nodes. If a path from a source node of type $B.s \oplus C.t$, $B.s \otimes C.t$ or $B.s \cap C.t$ to a certain node w is found, then paths from all groups of entities that are direct predecessors of the source node to w are also considered.

The complexity of the algorithm for building a credential graph of a set P of RT^T credentials will be evaluated with respect to the number of credentials in P (the cardinality of P), which is considered the input size of the problem. The method of evaluation is by assessing the complexity of each step of the algorithm and then counting the number of repetitions of each particular step. We assume that Dijkstra's algorithm [11] is used for finding paths between two nodes in the graph. The complexity of this algorithm is $O(n^2)$, where n is the number of nodes.

The following denotations have been made in the sequel:

n – the number of credentials in P,

m – the number of role expressions other than roles and groups of entities in credentials in P, obviously $m \le n$,

m_1 – the number of role expressions of type $B.s.t$,

m_2 – the number of role expressions of type $B.s \cap C.t$,

m_3 – the number of expressions of type $B.s \oplus C.t$ and $B.s \otimes C.t$, obviously $m_1 + m_2 + m_3 = m \le n$.

Moreover, A, B, C, D, E, X, Y denote groups of entities, may be singletons, from E.

The Algorithm (Creation of the Credential Graph).

1) For each credential $A.r \leftarrow e$ in P, add nodes $A.r$ and e to N_P and add an edge (e, $A.r$) to E_P.

Remark. There are $2n$ nodes in the graph that has been built in step 1. The complexity of building this graph is of order O(n).

Loop through the steps 2, 3 and 4:

2) For each node $B.s.t$ find all the reverse paths (i.e. paths that start in $B.s$ and move along edges in the backward direction) from $B.s$ to the other nodes of the graph.

- If there exists a reverse path from $B.s$ to X and role $X.t$ belongs to N_P, then add an edge ($X.t$, $B.s.t$) to E_P.
- If there exists a reverse path from $B.s$ to e, where e equals $D.u \oplus E.v$, $D.u \otimes E.y$ or $D.u \cap E.v$, then for each group X of entities, such that X is a direct predecessor of e and role $X.t$ belongs to N_P, add an edge ($X.t$, $B.s.t$) to E_P.

Remark. The number of nodes $B.s$, which are the initial nodes in searching for paths, is equal to the number m_1 of nodes $B.s.t$. Hence, the search is repeated m_1 times.

3) For each node $B.s \cap C.t$ find all the reverse paths from $B.s$ and from $C.t$ to the other nodes of the graph.

- If there exist reverse paths from $B.s$ to X and from $C.t$ to X, then add an edge (X, $B.s \cap C.t$) to E_P.
- If there exist reverse paths from $B.s$ and from $C.t$ to e, where e equals $D.u \oplus E.v$, $D.u \otimes E.v$ or $D.u \cap E.v$, then add an edge (e, $B.s \cap C.t$) to E_P.

Remark. The number of nodes $B.s$ and $C.t$, which are the initial nodes in searching for paths, is not greater than $2m_2$, hence, the search is repeated not more than $2m_2$ times.

4) For each node $B.s \oplus C.t$ and $B.s \otimes C.t$ find all the reverse paths from $B.s$ and from $C.t$ to the other nodes of the graph. Select all the pairs of nodes w_1, w_2, such that reverse paths from $B.s$ to w_1 and from $C.t$ to w_2 exist, and w_1 as well as w_2 are groups of entities or role expressions of type $D.u \oplus E.v$, $D.u \otimes E.v$ or $D.u \cap E.v$. For each pair w_1, w_2 do the following (in case of expression $B.s \otimes C.t$ take into account only those pairs X, Y, for which $X \cap Y = \phi$).

- If both nodes w_1 and w_2 are groups X and Y of entities, then add node $X \cup Y$ to N_P and add an edge ($X \cup Y$, $B.s \oplus C.t$) or ($X \cup Y$, $B.s \otimes C.t$), respectively, to E_P.
- If one node, w_1 or w_2, is a group X of entities, while the other node is an expression e, where e equals $D.u \oplus E.v$, $D.u \otimes E.v$ or $D.u \cap E.v$, then select all the direct predecessors of e that are groups of entities, and for each such group Y add node $X \cup Y$ to N_P and add an edge ($X \cup Y$, $B.s \oplus C.t$) or ($X \cup Y$, $B.s \otimes C.t$) to E_P.
- If both nodes w_1 and w_2 are expressions of type $D.u \oplus E.v$, $D.u \otimes E.v$ or $D.u \cap E.v$, then select all pairs X, Y of the direct predecessors: X of w_1 and Y of w_2, which are groups of entities, and for each of such pairs add node $X \cup Y$ to N_P and add an edge ($X \cup Y$, $B.s \oplus C.t$) or ($X \cup Y$, $B.s \otimes C.t$), respectively, to E_P.

Remark. The number of nodes $B.s$ and $C.t$, which are the initial nodes in searching for paths, is not greater than $2m_3$, hence, the search is repeated not more than $2m_3$ times. □

The number of static nodes is constant, equal to $2n$, in the graph that is searched for paths in steps 2, 3 and 4 of the above algorithm. Therefore the complexity of finding the paths that begin in a given node is of order O(n^2). The total number of nodes, which are the initial nodes in searching for paths in steps 2, 3 and 4 equals $m_1 + 2m_2 + 2m_3 \leq 2m$. Hence, the search is repeated not more than $2m \leq 2n$ times, and the complexity of a single pass through the loop (steps 2, 3, 4) is of order O(n^3).

A single pass through the loop corresponds to a single search through the set of n credentials. Each pass adds edges to the static part of the graph. The possibility of adding an edge depends on the existence of certain paths in the graph, which means that it depends on the sequence in which the credentials are processed. Repeating the loop n times guaranties that all the possible sequences of credentials have been exercised. So, the complexity of the algorithm is of order O(n^4).

VI. CONCLUSIONS

The main contribution of this research is an algorithm for building RT^T credential graph. The complexity of this algorithm has been proved polynomial, of order O(n^4), with respect to the number n of credentials at hand.

REFERENCES

[1] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman, "Role-Based Access Control Models," IEEE Computer, no 2, pp. 38–47, 1996.

[2] M. Blaze, J. Feigenbaum, and J. Lacy, "Decentralized Trust Management," Proc. 17th IEEE Symposium on Security and Privacy, pp. 164–173, IEEE Computer Society Press, 1996.

[3] M. Blaze, J. Feigenbaum, and J. Ioannidis, "The KeyNote Trust Management System Version 2," Internet Society, RFC 2704, 1999.

[4] D. Clarke, J-E. Elien, C. Ellison, M. Fredette, A. Morcos, and R.L. Rivest, "Certificate chain discovery in SPKI/SDSI," J. Computer Security, vol, 9, pp. 285–322, 2001.

[5] N. Li, J. Mitchell, W. Winsborough, "Design of a Role-Based Trust-Management Framework," Proc. IEEE Symposium on Security and Privacy, IEEE Computer Society Press, pp. 114–130, 2002.

[6] N. Li, J. Mitchell, "RT: A Role-Based Trust-Management Framework, Proc. 3rd DARPA Information Survivability Conference and Exposition, IEEE Computer Society Press, pp. 201–212, 2003.

[7] N. Li, W. Winsborough, J. Mitchell, "Distributed Credential Chain Discovery in Trust Management," J. Computer Security, no 1, pp. 35–86, 2003.

[8] D. Gorla, M. Hennessy, V. Sassone, "Inferring Dynamic Credentials for Role-Based Trust Management," Proc. 8th ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, ACM, pp. 213–224, 2006.

[9] A. Felkner, K. Sacha, "The Semantics of Role-Based Trust Management Languages," Proc. 4th IFIP Central and East European Conference on Software Engineering Techniques, pp. 195–206, 2009.

[10] K. Sacha, "Credential Chain Discovery in RT^T Trust Management Language," in: I. Kotenko, V. Skormin (eds) Computer Network Security, LNCS 6258, pp. 195–208 Springer, Berlin Heidelberg, 2010.

[11] T. H. Cormen, C. E. Leiserson, R. L.Rivest, C. Stein, Introduction to Algorithms, MIT Press and McGraw-Hill, 2001.