Modelling And Inferring On Role-Based Access Control Policies Using Data Dependencies

Romuald Thion and Stéphane Coulondre

LIRIS: Lyon Research Center for Images and Intelligent Information Systems,
Bâtiment Blaise Pascal, 20 av. Albert Einstein, 69621 Villeurbanne Cedex, France
`firstname.surname@liris.cnrs.fr`

Abstract. Role-based access control (RBAC) models are becoming a de facto standard, greatly simplifying management and administration tasks. Organizational constraints were introduced (e.g.: mutually exclusive roles, cardinality, prerequisite roles) to reflect peculiarities of organizations. Thus, the number of rules is increasing and policies are becoming more and more complex: understanding and analyzing large policies in which several security officers are involved can be a tough job. There is a serious need for administration tools allowing analysis and inference on access control policies. Such tools should help security officers to avoid defining conflicting constraints and inconsistent policies.

This paper shows that theoretical tools from relational databases are suitable for expressing and inferring on RBAC policies and their related constraints. We focused on using Constrained Tuple-Generating Dependencies (CTGDs), a class of dependencies which includes traditional other ones. We show that their great expressive power is suitable for all practical relevant aspects of RBAC. Moreover, proof procedures have been developed for CTGDs: they permit to reason on policies. For example, to check their consistency, to verify a new rule is not already implied or to check satisfaction of security properties. A prototype of RBAC policies management tool has been implemented, using CTGDs dedicated proof procedures as the underlying inference engine.

1   Introduction

Databases management systems (DBMS) are cornerstones of information systems: they provide mechanisms to store, modify, retrieve and query information of an organization. In order to enhance security of data, authorization (or access control) mechanisms have been developed to manage users' rights over data store in the DBMS. In its broader sense, access control, denotes the fact of determining whether a *subject* (process, computer . . .) is able to perform an *operation* (read, write . . .) on an *object* (a tuple, a table, . . .). An operation right on an object is called *permission*. Access control policies define the subjects' permissions.

Applications developed using a DBMS can contain large amount of data with highly differentiated access for different users, depending upon their function or role within the organization [1]. Role-based access control (RBAC) received considerable attention as an alternative to traditional mandatory and discretionary

access control policies in databases. The RBAC models constitute a family in which permissions are associated with roles. A role is a job function or job title within the organization. Users are made members of appropriate roles. Permissions are not directly assigned to users (roles can be seen as collections of permissions) [2]. RBAC provides a powerful mechanism for reducing the complexity, cost, and potential for error in assigning permissions to users within the organization. RBAC was found to be among the most attractive solutions for providing access control in e-commerce, e-government or e-health [1, 3].
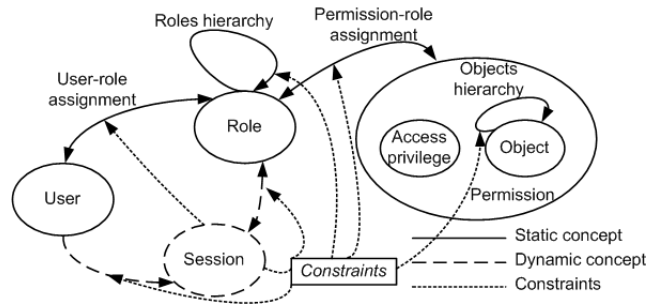

Fig. 1. RBAC Model

Nevertheless number of users in RBAC policies is increasing and rules are more and more complex: diverse constraint [1] types introduced to reflect peculiarities of organizations. RBAC constraints specify conditions that cannot be violated by the components of the system.

Policies engineering is considered to be of high practical importance [4] : a large part of flaws in information systems are due to administration mistakes or security misconceptions. There is a need for tools facilitating design and maintenance of RBAC policies. Accroding to [5], such tools need to be able to capture access control model mechanisms and peculiarities (e.g. RBAC constraints). These tools need to be able to check consistency of policies and to answer queries for particular permissions or relation holdings in the policies. Last requirement is a comprehensible inference mechanism, even by non-logicians. Our goal is to provide a formal framework satisfying these requirements.

Thus our contribution is twofold:

---

[1] *constraint* may be a confusing word in this paper: it may either designate relations between variables (e.g., $X \leq Y, X \geq 2 \times Z + 1, 3 = T, 2 \neq 3$, etc.), restrictions on RBAC model's concepts (e.g. nobody is allowed to assume simultaneously roles r_1 and r_2) and even data dependencies (integrity constraints). In this paper we do not use the term *integrity constraints*, *constraints* refers to semantic relations between variables and *RBAC constraints* or *organizational constraints* refers to restriction among elements in RBAC policies.

– identification of a theoretical tool from the databases field suitable for homogenous modeling of RBAC principles and its related constraints right into the relational model,
– use tools built (e.g. proof procedures) on top of the underlying theoretical model to provide a set of tools facilitating design and management of RBAC policies in order to detect and correct administration mistakes or misconceptions.

The class of data dependencies we focused on is Constrained Tuple Generating Dependencies (CTGDs). In the literature Tuple-Generating Dependencies are also known as *Generalized Dependencies*, [6, 7]. CTGDs actually subsume many other known class of dependencies [8] and because the operational nature of its dedicated proof procedures lead to a more comprehensible trace of inference than generic ones. We will show that a framework based on CTGDs (Constrained Tuple-Generating Dependencies) is an appropriate formal tool for representing and checking RBAC policies.

In the next section we will introduce Constrained Tuple-Generating Dependencies and proof procedure related in. Section 3 will show how CTGDs can be used to model RBAC concepts, constraints and assignments, of which implements will be shown in section 4. Section 5 will summarize some papers related to this work. Finally, last section will discuss our work and presents perspectives using databases dependencies for security purposes.

2 Background

2.1 Constrained tuple-generating dependencies (CTGDs)

In [8], the authors expose a kind of data dependencies upon the most expressive existing: CTGDs, subsuming any other ones. CTGDs extend tuple-generating dependencies (TGDs) with a *constraint domain* (e.g. linear arithmetic over integers, rationals or real). These kinds of data dependencies which are interesting for geographical modeling refers to deductive and constraint databases [9, 10]. CTGDs can be represented formally in first order logic by formulae [6]:

$$\forall X p_1(X_1) \wedge p_2(X_2) \wedge p_i(X_i) \wedge c \rightarrow \exists Y q_1(X \cup Y_1) \wedge q_2(X \cup Y_2) \wedge p_j(X \cup Y_j) \wedge c'$$

Where p_i and q_j are predicates symbols, X is the set of all terms (no functions symbols) in the left hand side. Terms of X are universally quantified. Y does not designate all terms in the right hand side, but only those that are not bound by the universal quantifier on the left hand side. Terms of Y are existentially quantified. Finally, c and c' are conjunctions of linear constraints (e.g. $<, >, \leq, \geq, \neq, =$) over terms (terms of X for c and terms of $Y \cup X$ for c').

2.2 CTGDs dedicated proof procedures

In [8], the authors propose two bottom-up chase over CTGDs. Their paper addresses the implication problem, that is, given a collection of CTGDs F, and a single CTGD g, determine whether in every database state where F is satisfied,

it is also the case that g is satisfied. The chase proves if F logically implies g, stated briefly as $F \vDash g$. The operational nature of these proof procedures is based on the concept of tuple (a grounded atom, with no variables). Basic outline of such procedures is based on [11] with the adjunction of constraints: hypothesize the existence of some tuples in the relations such that the antecedent l of g is satisfied, treat F as defining a closure operator generating tuples $F(l)$. On each computation step of $F(l)$, the following condition is tested:
- if $F(l)$ contains a copy of r, infer $F \vDash g$ (*termination case one*),
- if $F(l)$ contains an inconsistency produced by constraints, infer $F \vDash g$ vacuously (*termination case two*),
- if $F(l)$ does not contain a copy of r, infer $F \nvDash g$ (*termination case three*).

The CTGDs implication problem is semi-decidable: the procedures may run forever. As each basic step is producing new facts through implication, we can practically bound up the number of successively applied CTGDs (e.g. to avoid circular generating facts leading to infinite loop), but it is unsound and must be reserved for implementation purpose. Constraints are quite interesting when used jointly with existential quantifiers because they permit a more precise definition of such *partially known* facts (e.g., *if a role is granted read-access, then* **another** *role is granted write-access.*

3 A framework for expressing and checking RBAC policies

According to the authors of [12] we use the following predicates to model core concepts of RBAC policies:
- $ura(User, Role)$, to define User Role Assignments,
- $pra(Access, Object, Role)$, to define Permission Role Assignment
- $permitted(User, Access, Object)$, to specify that user $User$ is granted $Access$ access privilege on object $Object$.

3.1 Capturing axiomatic definition of RBAC model

Once basic elements of the policies are defined, we need to model the "axiomatic of RBAC": the core of this access control model which settles how an access is granted to a user through role assignment, how are defined hierarchies, etc. We model an RBAC axiomatic based on [13]. $dSenior(SeniorRole, JuniorRole)$ to define direct inheritance between roles and $senior(SeniorRole, JuniorRole)$ to define role hierarchy (the transitive closure of $seniorDirect$).
- role inheritance is transitive: $senior(X, Y), dSenior(Y, Z) \rightarrow senior(X, Z)$,
- role inheritance is irreflexive: $senior(X, X) \rightarrow false$,
- a user is access granted to an object if he is assigned to a role which is assigned to this permission: $ura(U, R), pra(A, O, R) \rightarrow permitted(U, A, O)$,
- eventually through inheritance $ura(U, R1), senior(R1, R2), pra(A, O, R2) \rightarrow permitted(U, A, O)$.

3.2 Capturing RBAC constraints

Constraints are an important aspect of role-based access control and are a powerful mechanism for laying out higher-level organizational policy [2]. The best known RBAC constraints are:

Mutually exclusive roles constraints settle that no user should be assigned to two roles which are in conflict with each other. In other words, it means that conflicting roles cannot have common users. $ssd(Role1, Role2)$, specificy that $Role1$ and $Role2$ are in Static Separation of Duties (SSD): they are mutually exclusive. Mutually exclusive roles can produce inconsistency. The authors of [13] describe a set of properties that must hold in any RBAC policy. These properties are described in the example of section 4.

Cardinality constraints settle that a number of assignments is limited. Cardinality constraints of n maximum users assigned to role r can be expressed in CTGDs by $\wedge_{i=1}^{n+1} ura(U, N_i) \; \forall i \in [1..n+1], \forall j \in [i..n+1] \; N_i \neq N_j \rightarrow false$. Mutually exclusion and cardinality constraints are not limited to role and can be used on any element of the policy model (for example with access: no role can be granted both read access and write access on an object o). Our approach can be generalized for maximum number of roles assigned to a users, permissions.

More generally, Nullity Generating Dependencies (NGDs) of the form $p_i(X) \wedge c \rightarrow false$ can be used to model RBAC constraints: an RBAC constraints define that if a certain state (the left hand side of the CTGD) is reached, then the policies are inconsistent (right hand side is $false$).

Prerequisite constraints settle that if a particular relation holds, another holds too. Variables appearing only within the terms of the tail in CTGDs are existentially quantified. Intuitively that does mean *at least one element such as ... exists.* This semantic is used to take into account prerequisite RBAC constraints. E.g. role r_2 is required by role r_1: *for any user assigned to role r_1, at least one another user must be assigned to role r_2*, $ura(U_1, r_1) \rightarrow ura(U_2, r_2) U_1 \neq U_2$. Other prerequisite constraints can be expressed using CTGDs, according to the administrator's need. CTGDs can model other forms of prerequisite constraints on any RBAC concept.

3.3 Inference on policies

Depending on which stage of the RBAC specification one is working on, different needs of verification may exist:

- during the stage of modeling axiomatic (the core policy model), we are likely to check the expected behavior of the model and rules redundancies. E.g. how authorizations are derived from user-role and permission-role assignment, 3.1,
- during the stage of defining the role hierarchy, we are likely to check a set of properties. E.g. there is no cycle in the hierarchy, or no role inherits the administrator role,

- during the stage of defining user-role and permission-role assignment we are likely query the policy and to check a set of properties. E.g there is no two roles which have exactly the same permissions,
- during the stage of defining constraints it is interesting to check whether the policy is consistent, in other words if we settled facts violating constraints.

Security requirements	Reduction into CTGDs
Security property that must hold in all RBAC policy instances. *no role can be senior to itself*	Use proof procedure to check implication
Security property that must hold in a policy instance. *no role inherits the administrator role*	Check consistency of policy or verify if the CTGDs are satisfied in the database instance
Policy management capabilities: queries and data manipulation *which users are assigned to role student?*	Process query over the database

Table 1. Reduction of security administration needs into CTGDs-dedicated tools

The second termination case (vacuously) of algorithms from [8] is very useful while checking access control policies, it denotes that the policies are inconsistent. This semantic is interesting for security administrators when dealing with constrained access control policies: if there are facts violating constraints, the policy is inconsistent.

4 Experimental validation

This section illustrates how a RBAC policy can be modeled into Constrained Tuple Generating Dependencies. The sample code is separated into four parts: the first one models the core mechanisms of the Role-based access control model and settles a set of properties that must holds in any RBAC policy (from [13]). The second part is a sample role hierarchy used in a virtual organization. Next is a sample definition of User-Role Assignments and Permission-Role Assignments. The last part defines a set of specific organizational constraints that must hold in this particular policy.

```
%axiomatic definition of RBAC policies and generic constraints
%-----------------------------------------------------------
%senior is the transitive closure of dSenior
dSenior(SeniorRole,JuniorRole)->senior(SeniorRole,JuniorRole).
senior(SeniorRole,InterRole), dSenior(InterRole,JuniorRole)-> senior(SeniorRole,JuniorRole).
senior(Role,Role)->false.

%granting access to user through role assignments
ura(User,Role),pra(Access,Object,Role)->permitted(User,Access,Object).
ura(User,SeniorRole),senior(SeniorRole,JuniorRole),
pra(Access,Object,JuniorRole)->permitted(User,Access,Object).
```

```
%Property P1: any two roles assigned for a same user are not in separation of duties
ura(User,Role1),ura(User,Role2),ssd(Role1,Role2)->false.

%Property P2: no role is mutually exclusive with itself
ssd(Role,Role)-> false.

%Property P3: mutual exclusion is symetric
ssd(Role1,Role2)->ssd(Role2,Role1).

%Property P4: any two roles in ssd do not inherits one another
senior(Role1,Role2),ssd(Role1,Role2)->false.

%Property P5: there is no role inheriting to roles in ssd
ssd(Role1,Role2),senior(SeniorRole,Role1),senior(SeniorRole,Role2)->false.

%Property P6: If a role inherits another role and
%that role is in SSD with a third one, then the inheriting
%role is in SSD with the third one.
ssd(Role1,Role2),senior(SeniorRole,Role1)->ssd(SeniorRole,Role2).

%definition of role hierarchy
%--------------------------
%roles and hierarchy (with directly senior predicate) modeling
->role(student),role(researcher),role(teacher),role(phDStudent).
->role(postPhD),role(lecturer),role(seniorLecturer),role(professor).
->dSenior(phDStudent,student), dSenior(phDStudent,researcher).
->dSenior(postPhD,phDStudent), dSenior(postPhD,teacher).
->dSenior(lecturer,teacher), dSenior(lecturer,researcher).
->dSenior(professor,seniorLecturer), dSenior(seniorLecturer,lecturer).

%definition of assignments
%-----------------------
%Permission-Role Assignments
->pra(read,test,student),pra(write,test,teacher),pra(read,finalTest,professor).
->pra(read,smallPaper,lecturer),pra(write,bigPaper,professor).

%User-Role Assignments
->ura(alice,student),ura(bob,phDStudent),ura(charly,professor).

%definition of organizational constraints
%---------------------------------------
%prerequisite on permissions: if one can read and object, another one can write
pra(read,Object,Role1) -> pra(write,Object,Role2) {Role1=\=Role2}.

%uniqueness constraint on manager
ura(User1,manager),ura(User2, manager){User1=\=User2}->false.

%mutually exclusives roles: student and professor
->ssd(student,lecturer).
```

We have described chase procedures as algorithms proving that a set of CT-GDs F implies a single CTGD g: $F \vDash g$. The above ruleset is such an F collection, and g is the security property to check. The table 1 describes how CTGDs-dedicated tools can be used by administrators to design, verify and manage their policies. Six properties ($P1$ to $P6$) are settled in the sample policy, the authors of [13] have manually demonstrated the following theorem : $P2 \wedge P3 \wedge P6 \Rightarrow P4 \wedge P5$.

Our first example illustrates how chase procedures for CTGDs can be used to automatically proove the same theorem:
- let F be the collection of CTGDs modeling properties $P2$, $P3$ and $P6$,
- let be g_1 the CTGD modeling properties $P4$,
- let be g_2 the CTGD modeling properties $P5$.
The chase procedure prove that $F \vDash g_1$ and $F \vDash g_2$, we can conclude the prop-

erties $P4$ and $P6$ are redundant. Such functionalities are very interesting for security administrators: they can check that security properties ($P4$ and $P6$ in this example) hold in all RBAC policy instances (that satisfy $P2$, $P3$ and $P6$ in the example).

Another example is $g \equiv ura(joe, student), ura(joe, seniorLecturer) \rightarrow$: "is the policy consistent if joe is assigned to both $student$ and $seniorLecturer$?". Clearly, with such assignments to user joe, the policy is inconsistent: roles $student$ and $lecturer$ are in SSD, according to property number six, $student$ and $seniorLecturer$ are in SSD too, thus the policy is inconsistent using property number one. It is very interesting for administrators to conduct such verifications $before$ any assignment: they can ensure the consistency of their policy in the presence of updates.

We argue that dedicated proof procedures are conceptually simpler than more generic ones (e.g. SLD-based) in the case of CTGDs. A dedicated proof procedure avoids a prior transformation step required by classical proof procedure on first order logic (e.g., avoiding Skolem Normal Form) which does not work on whole formulae and had to alter them indeed. CTGDs may need to be transformed into Horn clauses without existential quantifiers for example. The steps of such a transformation are:
- convert each CTGD to an equivalent formula in prenex normal form,
- replace existentially quantified variable by functions,
- split each CTGD into Horn clauses.

We have implemented the chase procedures described in [8] in a prototype. On top of this underlying inference engine, we have built a Microsoft Visio 2003 Template dedicated to RBAC policies design. It is able to determine if a permission is granted to a user through his role assignment, it can check if the set of policies is consistent and can answer queries about the relations holding in the RBAC policy.

5   Related work

Our work has been influenced by [12] which express RBAC models with constraint logic programming and [4] which describes the "Flexible Authorization Framework", that can be analyzed using a variant of Datalog (typically either safe stratified Datalog or Datalog with constraints).

The three main arguments we focused on are providing a framework which:
- is able to capture all relevant concepts of RBAC models,
- can benefit researches (e.g. evolutions, theoretical results, implementations) from a well established community,
- can be easily linked with other components of the information system (e.g. databases).

The authors of [12] describe access control programs able to deal with RBAC models. This very complete work addresses many problems arising with the use of closed policies (access denied as a default action, authorizations are only ever positive), open policies (access granted as a default action) or hybrid policies

(authorizations and denial can be explicitly defined). However, logical programs are not intuitive for non-specialits and the logic used do not integrate existential quantifiers. Moreover, RBAC policy are already widespread, a framework base on databases makes integration of administration tools and security data easier.

The autors of [4] argue"'... extensive research activity has resulted in the definition of a variety of access control ... Thus, the need arises for developing tools for reasoning about the characteristics of these models. These tools should support users in the tasks of model specification, analysis of model properties, and authorization management". Their logical framework is based on the C-Datalog language, whereas our is based on CTGDs, which is a able to deal with a wider class of rules thanks to existential quantifiers and constraints within both head and tail of dependencies.

The authors of [14] describe a fragment of FOL which tractable and sufficiently expressive to capture policies for many applications. This work is really interesting and points out tractability and complexity results on their logic. Constraints in policies are necessary to capture peculiarities of organizations, but modeling such restrictions is not develop in [14]. We do agree the authors' statement about the use of logic programming by non-logicians, but we disagree that a "filling the blank on English sentences" interface is sufficient for security administrators. We think that administrators must have a computer-aided software engineering (CASE) interface to design and check policies and such a CASE should provide a comprehensible trace of reasoning.

6   Conclusions and further work

We are confident that CTGDs can be used to express various kinds of access policies such as Task-BAC, Workflow-BAC, Mandatory-BAC or Organization-BAC. The fragment of first-order logic is really closed to the ones used in [12] or [4], which are able to deal with temporal aspects and at least mandatory and discretionary access control models.

For sake of clarity the example exposed in section 4 does not include session. According to [12] sessions and dynamic constraints can be captured easily with CLP, thus with CTGDs. We are investigating the interest of chase procedure to check RBAC policies involving sessions. For example, using chase procedure we might answer queries like *Are the policies consistent for all possible sessions?*. Moreover, incorporating the model for administration of roles exposed in [15] is promising for distributed policies verification purpose.

Integrating of temporal aspects in RBAC models has been investigated in [16]. The authors of [12] use the Constraint Logic Programming framework. We can use the same approach to model Temporal-RBAC models, and according to [17] we extend the inequalities to geographical trigerring of assignments. Integrating temporal or geographical concerns into CTGDs, is mainly related to the choice of a right *constraint domain* [8]. For example, to define that a role is assigned to a user only on [t1, t2] shift, (between the times t1 and t2,): $time(H)t1 \leq H \leq t2 \rightarrow ura(user, role)$.

A promising opening to the use of CTGDs for access control modeling purpose are the results exposed in [18]. Their paper propose a new kind of dependencies subsumming CTGDs : Disjunctive-CTGDs. Their enhanced expressivity can be used to models new king of RBAC constraints involving disjunctions: a new class of organizational constraints which have not been adressed yet but that might be usefull to models desirable property.

References

1. Ramaswamy, C., Sandhu, R.: Role-based access control features in commercial database management systems. In: Proc. 21st NIST-NCSC National Information Systems Security Conference. (1998) 503–511
2. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role-based access control models. IEEE Computer **29**(2) (1996) 38–47
3. CERT/CC, U.S.S., magazine, C.: E-crimewatch survey. Technical report, http://www.cert.org/archive/pdf/ecrimesummary05.pdf (2005)
4. Bertino, E., Catania, B., Ferrari, E., Perlasca, P.: A logical framework for reasoning about access control models. ACM Trans. Inf. Syst. Secur. **6**(1) (2003) 71–127
5. Bonatti, P.A., Samarati, P.: Logics for authorization and security. In Chomicki, J., van der Meyden, R., Saake, G., eds.: Logics for Emerging Applications of Databases, Springer (2003) 277–323
6. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
7. Coulondre, S.: A top-down proof procedure for generalized data dependencies. Acta Inf. **39**(1) (2003) 1–29
8. Maher, M.J., Srivastava, D.: Chasing constrained tuple-generating dependencies. In: PODS, ACM Press (1996) 128–138
9. Ramakrishnan, R., Ullman, J.D.: A survey of research on deductive database systems. Journal of Logic Programming **23**(2) (1993) 125–149
10. Revesz, P.Z.: Constraint databases: A survey. In Libkin, L., Thalheim, B., eds.: Semantics in Databases. Volume 1358 of Lecture Notes in Computer Science., Springer (1995) 209–246
11. Beeri, C., Vardi, M.Y.: A proof procedure for data dependencies. J. ACM **31**(4) (1984) 718–741
12. Barker, S., Stuckey, P.J.: Flexible access control policy specification with constraint logic programming. ACM Trans. Inf. Syst. Secur. **6**(4) (2003) 501–546
13. Gavrila, S.I., Barkley, J.F.: Formal specification for role based access control user/role and role/role relationship management. In: ACM Workshop on Role-Based Access Control. (1998) 81–90
14. Halpern, J.Y., Weissman, V.: Using first-order logic to reason about policies. In: CSFW, IEEE Computer Society (2003) 187–201
15. Sandhu, R.S., Munawer, Q.: The arbac99 model for administration of roles. In: ACSAC, IEEE Computer Society (1999) 229–240
16. Bertino, E., Bonatti, P.A., Ferrari, E.: Trbac: A temporal role-based access control model. ACM Trans. Inf. Syst. Secur. **4**(3) (2001) 191–233
17. Grumbach, S., Rigaux, P., Segoufin, L.: Spatio-temporal data handling with constraints. GeoInformatica **5**(1) (2001) 95–115
18. Wang, J., Topor, R., Maher, M.: Reasoning with disjunctive constrained tuple-generating dependencies. Lecture Notes in Computer Science **2113** (2001) 963–973