D. Gabbay et al. editors, *Proceedings of the International Joint Conference on Qualitative and Quantitative Practical Reasoning (ECSQARU/FAPR-97)*, pag. 421-435, volume 1244 of *Lecture Notes in Artificial Intelligence*,

Reasoning about Security: a Logic and a Decision Method for Role-Based Access Control

Fabio Massacci*

Computer Laboratory,
University of Cambridge (UK)

Abstract. Role-based access control (RBAC) is one of the most promising techniques for the design and implementation of security policies and its diffusion may be enhanced by the development of formal and automated method of analysis.
This paper presents a logic for practical reasoning about role based access control which simplifies and adapts to RBAC the calculus developed at Digital SRC. Beside a language and a formal semantics, a decision method based on analytic tableaux is also given. Analytic tableaux make it possible to reason about logical consequence, model generation and consistency of a formalised role-based security policy.

1 Introduction

Access control is a key issue for the security of distributed systems (see [28] for an introduction) and plays an important role in the security policies of many organisations such as the military [4], banking [7] or health services [2].

Recently, new formal models for access control have emerged besides the traditional access matrix [21] for the design of more sophisticated policies [5,11,15,25,26]. Logic and formal reasoning techniques are also used for describing and reasoning about security policies and access control [1,8,16,22,29]

Among the new paradigms, Role-Based Access Control (RBAC) [11,12,15,26] has received particular attention. It stems from an extensive field study [13] which pointed out that in practice permissions are assigned to a user according his/her roles in the organisation. Moreover, the explicit representation of roles makes also possible to simplify the system design and to use engineering principles such as least privilege, separation of duties, etc. [11,26].

As languages become more expressive and systems more complex, formal methods of analysis (possibly automated) are necessary: the simple table (or list) lookup of the access matrix [18,27] must be replaced by careful reasoning for complex and highly expressive systems such as [1,5,8,11,16,22,26].

This work aims at practical but formal reasoning methods for RBAC:

1. *define a logic* (a language and a semantics) to express RBAC policies in a simple and (hopefully) natural way;

* Current address: Dip. Informatica e Sistemistica, Univ. di Roma I, "La Sapienza", via Salaria 113, I-00198 Roma, `massacci@dis.uniroma1.it`.

2. *find a decision method* (a terminating inference procedure) which can be used to verify in a direct and automatic way the consistency or the logical consequences of an RBAC policy.

The logic we propose is a simplified and enhanced version of the logic for access control developed in [1,22]. The language is tailored to express directly role hierarchies, privileges attribution and user attributions as requested in many formal models of RBAC [11,26]. We also developed a model theoretic semantics — the relevance for security analysis of formal semantics has been pointed out in [29] — and have improved the semantics proposed in [1].

A decision method is equally important: suppose we formalised the key RBAC features of a health care system, we also want to "do something" with this formalisation. For instance checking whether it is consistent, or whether a particular (un)desired property is a logical consequence of our model or not. To this extent axiom systems, which are often used to define formal models based on logics [8,16,19,20], are inadequate: axioms systems require people to be good logicians and cannot be automated at all. The exclusive focus on axiomatisation, without calculi for automated reasoning, may condemn logics for practical reasoning to be unused in practice and therefore fail in their very first aim.

Therefore we developed a decision method based on analytic tableaux for modal logics [14] and in particular on Simple Step Tableaux [23,9]. Tableaux have many advantages: they can be used for both logical consequence and satisfiability; deduction steps follows the semantics of the logic and the calculus can be easily automated along the lines of "lean theorem proving" [3].

We view our calculus as a building block of a more general framework such as authentication logics for cryptographic protocols [6,30], where the issue of jurisdiction of agents is a key problem. In this perspective access control can be seen as a problem of jurisdiction in complex and distributed systems.

Notice that we *do not* advocate the use of logic and semantic tableaux for actually implementing RBAC systems. In the same way that a civil engineer use mathematical instruments to design a bridge but concrete and iron to build it, we believe that logic and decision methods based on logic should be used in the *design and verification phase* of RBAC systems.

In this paper we present a logical language for RBAC and describe how RBAC features can be captured (§2), together with the desired reasoning capabilities (§3). Next we present the formal semantics (§4) and the tableaux-based decision method (§5). The discussion on related works (§6) concludes the paper.

2 A Logic for RBAC: Language and Features

The main actors of RBAC (see [11,26] for an introduction) are *users* and *roles*, denoted respectively by U and R. We refer to either of them as *atomic principals* A. We use also composed principals (denoted by P) which are user under a certain roles i.e. "$U \,\mathtt{as}\, R$". Intuitively by $U \,\mathtt{as}\, R$ we denote a principal U who has restricted his privileges to those allowed by the role R; for instance

$fm205\,\texttt{as}\,WebMaster$. Notice that U may also be claiming a role R which has not been assigned to him by the organisation.

For *role hierarchies* [11,26] we use "$R_1\,\texttt{isa}\,R_2$", that is role R_1 has at least all privileges of R_2; for instance *SwAdm* isa *User*. For *users assignments* to roles we use the construct "$U\,\texttt{has}\,R$"; for instance $maj\,\texttt{has}\,SysAdm$.

The role of operations over objects is played by *atomic requests* which are uninterpreted operations [11,26], e.g. *login*(*ftp*). The intuition is that they are requests from a user and can be granted or not. Therefore we interpret them as propositional letters which can be true in a particular state of the system (granted) or false (not granted). We admit more complicated requests with boolean connectives $\wedge$, $\vee$, $\supset$: for instance *login*(*telnet*) $\supset$ *login*(*ftp*) whose intuitive (and formal) interpretation is "if telnet has been granted so has ftp". Requests are denoted with φ, ψ.

User requests of the form "$P\,\texttt{req}\,\varphi$" corresponds to the intuition that P requests φ to be executed (or tries to execute φ);e.g. *maj* req (*fm205* has *User*). For the meaning of $R\,\texttt{req}\,\varphi$, where R is a role, we follow [1] and interpret it as "somebody, who has role R, requests φ" (see [1,22] for discussion). Note that "$P\,\texttt{req}\,\varphi$" is itself a request and that an user may claim that a request comes from somebody else e.g. $fm205\,\texttt{req}\,(kw10009\,\texttt{req}\,exec(emacs))$.

To visualize the intuition behind the formulae we may compare a request such as $fm205\,\texttt{req}\,rm(test.c)$ to a command typed by the user on a terminal: `fm205{ely}: rm test.c`. As we press enter, it is evaluated according our access privileges and either executed or not (i.e. $rm(test.c)$ may be true or false).

Notice the difference with the logics of knowledge and obligations for computer security (see for instance [8,16,20]) whose main concern is usually confidentiality, characterized by the following axiom schema:

$$\text{“}A \text{ knows } \varphi\text{”} \ \supset\ \text{“}A \text{ is allowed to know } \varphi\text{”} \tag{1}$$

This work on access control follows [1] and focuses more on a theory of actions, to verify that the actions executed by a system respect its security policy. In this sense the modal operator $A\,\texttt{req}\,\varphi$ is closer to the *Do* operator of Kanger in [19] or the "A sees to it that X" in [20], where X is a action. The key observation, from the viewpoint of access control, is that we should rather use an operator for "A *tries* to see to it that X", since A may not be allowed to do X. This is the underlying intuition behind this work and also [1] and is crucial to understand the subsequent properties that we may find desirable.

The link between principals and requests (or privileges) is given by *privileges attributions* [26]. The construct "$P\,\texttt{controls}\,\varphi$" denotes that principal P has access control over φ. Loosely speaking, P can (is allowed to) make φ happen. In the literature on authentication this is usually referred as *jurisdiction* of a principal [6,30] and it is characterised by the axiom

$$\text{“}A \text{ says } \varphi\text{”} \wedge \text{“}A \text{ has jurisdiction on } \varphi\text{”} \ \supset\ \varphi \tag{2}$$

Our aim is to to replace it by following one (more complex yet more realistic):

$$A\,\texttt{req}\,\varphi \wedge B\,\texttt{controls}\,\varphi \wedge\ \text{“some relation between } A \text{ and } B\text{”} \supset \varphi \tag{3}$$

$$RH = \begin{cases} SysAdm\ \mathtt{isa}\ SwAdm \\ SysAdm\ \mathtt{isa}\ PostMaster, \\ SwAdm\ \mathtt{isa}\ User, \\ PostMaster\ \mathtt{isa}\ User, \\ User\ \mathtt{isa}\ RemUser, \end{cases} \qquad UA = \begin{cases} maj\ \mathtt{has}\ SysAdm \\ pb\ \mathtt{has}\ PostMaster \\ gt\ \mathtt{has}\ SwAdm \\ gt\ \mathtt{has}\ DepPostMaster \\ kw10009\ \mathtt{has}\ User \\ fm205\ \mathtt{has}\ RemUser \end{cases}$$

$$PA = \begin{cases} SysAdm\ \mathtt{controls}\ (fm205\ \mathtt{has}\ User) \\ SwAdm\ \mathtt{controls}\ write(emacs - source) \\ User\ \mathtt{controls}\ login(telnet) \\ User\ \mathtt{controls}\ run(emacs) \\ RemUser\ \mathtt{controls}\ login(ftp) \\ (fm205\ \mathtt{as}\ User)\ \mathtt{controls}\ rm(test.c) \end{cases}$$

Fig. 1. A Logical Formalisation of an RBAC

where the relation between A and B is specified according RBAC policies. Again this interpretation makes sense in our framework where φ is a "potential action" and not a state of affairs. This can be rephrased as follows: if A tried to execute φ and A is allowed to execute φ then φ is indeed executed.

In "pure" RBAC only roles are connected to privileges [26] so we should only have formulae like $R\,\mathtt{controls}\,s$. We don't want to force this commitment on the logic because (i) it is not necessary and (ii) there are applications where privileges may be attached to individual users or to particular users in particular roles. For instance in the `UNIX` file system $fm205$ can write his own files (when he logs in as a legitimate user) and those permission are linked to him as an individual user rather than a role (which may be assigned to many users) e.g. $(fm205\ \mathtt{as}\ User)\ \mathtt{controls}\ rm(test.c)$.

Roles hierarchies, users and privileges attribution are the major requirements for RBAC [12,11,26]. An *RBAC Security Policy* is characterised as follows:

Role Hierarchy (RH) $\bigwedge_{i,j}\{R_i\ \mathtt{isa}\ R_j\}$,
Privileges Attribution (PA) $\bigwedge_{i,j}\{P_i\ \mathtt{controls}\ \varphi_j\}$,
Users Attribution (UA) $\bigwedge_{i,j}\{U_i\ \mathtt{has}\ R_j\}$.

For a particular *realization* or run of the system we simply add the following:

User Requests (UR) $\bigwedge_{i,j}\{P_i\ \mathtt{req}\ \varphi_j\}$

For sake of simplicity, we freely exchange a set of formulae with the conjunction of its elements. An example is shown in Fig. 1.

Example 1. With the RBAC model in Fig. 1 when $maj\ \mathtt{req}\ (fm205\ \mathtt{has}\ User)$ together with $fm205\ \mathtt{req}\ (login(telnet) \wedge run(emacs))$ should the system grant $login(telnet) \wedge run(emacs)$?

Finally we introduce cryptographic keys and the standard notation $\{s\}_K$ to denote a request s encrypted (or signed) with the key K [6]. The construct $K\ \mathtt{speaks\text{-}for}\ P$ indicates that K is associated to a principal P or, more intuitively, that "a key speaks for a principal". We use composed principals because one key may be used by maj and another by $maj\ \mathtt{as}\ User$.

$$UA = \begin{cases} Alice\ \texttt{has}\ Account \\ Bob\ \texttt{has}\ Marketing \\ Charlie\ \texttt{has}\ Store \\ Sam\ \texttt{has}\ Server \end{cases} \qquad PA = \begin{cases} Marketing\ \texttt{controls}\ order \\ Account\ \texttt{controls}\ pay \\ Store\ \texttt{controls}\ recvd \\ Alice\ \texttt{controls}\ (Bob\ \texttt{req}\ order) \\ Server\ \texttt{controls}\ (K_A\ \texttt{speaks-for}\ Alice) \\ Server\ \texttt{controls}\ (K_B\ \texttt{speaks-for}\ Bob) \end{cases}$$

$$UR = \begin{cases} \{Bob\ \texttt{req}\ order\}_{K_A} \\ \{recvd\}_{K_C} \\ \{K_A\ \texttt{speaks-for}\ Alice\}_{K_S} \\ \{K_C\ \texttt{speaks-for}\ Charlie\}_{K_S} \end{cases} \qquad \begin{array}{l} KA = \left\{ K_S\ \texttt{speaks-for}\ Sam \right. \\ \\ Constr = \left\{ Account\ \texttt{req}\ (order \wedge recvd \supset pay) \right. \end{array}$$

Fig. 2. RBAC Models with Keys for Example 2

Since we are interested in access control and not in authentication or confidentiality we assume that the problems of associating keys with principals or finding whether keys are good etc. has been tackled already (see [6,22,30]). It is possible to associate a key to more than one principal (for shared key encryption). In this case the logic has been tailored to assume that the request came from the principal with most privileges (to avoid a denial of service).

Encrypted Requests (ER) $\bigwedge_{i,j}\{\varphi_j\}_{K_i}$
Key Assignments (KA) $\bigwedge_{i,j}\{K_i\ \texttt{speaks-for}\ P_j\}$

Example 2. *Alice* is responsible for the accounts. *Bob* for marketing and ordering of goods and *Charlie* is the storekeeper, in charge of receiving and shipping of goods. *Alice* is trusted in forwarding *Bob* orders. *Sam* is the security office responsible for the authentication server. The account department pays only after the store has certified goods' arrival. Messages and certificates are shown in Fig. 2. Should the department *pay*?

In logical term we are asking whether

$$UA \wedge PA \wedge KA \wedge UR \wedge Constraints \models^{?} pay$$

where the symbol $\models$ denotes traditional logical consequence.

In full fledged RBAC models [11,15,26] there are also other *constraints*. Most static constraints, such as separation of duties, can be easily captured:

$$\neg\,(U\ \texttt{has}\ R_j \wedge U\ \texttt{has}\ R_i) \quad \textit{for mutually exclusive } R_i, R_j$$

We can also express the fact that roles hierarchies must not yield cycles with $\neg(R_1\ \texttt{isa}\ R_2 \wedge R_2\ \texttt{isa}\ R_1)$ for distinct R_1 and R_2.

Once we have defined the language we can briefly sketch the properties we are interested to model. The basic properties are *transitivity and cumulativity* for role hierarchies and user attributions:

$$\begin{array}{ll} (Tr) & R_1\ \texttt{isa}\ R_2 \wedge R_2\ \texttt{isa}\ R_3 \supset R_1\ \texttt{isa}\ R_3 \\ (Cm) & U\ \texttt{has}\ R_1 \wedge R_1\ \texttt{isa}\ R_2 \supset U\ \texttt{has}\ R_2 \end{array}$$

Other desired properties concern the relation between users and roles which follow from the meaning of $R\,\texttt{req}\,\varphi$ as "somebody with role R requested φ":

$$\begin{array}{ll}(R) & U\,\texttt{has}\,R \wedge U\,\texttt{req}\,\varphi \supset R\,\texttt{req}\,\varphi \\ (U/R) & U\,\texttt{has}\,R \wedge (U\,\texttt{as}\,R)\,\texttt{req}\,\varphi \supset R\,\texttt{req}\,\varphi\end{array}$$

Some simplification properties for users and roles are also useful, in particular *idempotence*[1]: it does not make sense to have "logically stammering" principals for whom $U\,\texttt{req}\,(U\,\texttt{req}\,\varphi)$ is logically different from $U\,\texttt{req}\,\varphi$. *Awareness* is also important: principals should be aware of the consequences of what they requested (or didn't request):

$$\begin{array}{ll}(Id) & U\,\texttt{req}\,(U\,\texttt{req}\,\varphi) \supset U\,\texttt{req}\,\varphi \\ (K) & U\,\texttt{req}\,(\varphi \supset \psi) \wedge U\,\texttt{req}\,\varphi \supset U\,\texttt{req}\,\psi \\ (4) & U\,\texttt{req}\,\varphi \supset U\,\texttt{req}\,(U\,\texttt{req}\,\varphi) \\ (5) & \neg U\,\texttt{req}\,\varphi \supset U\,\texttt{req}\,\neg(U\,\texttt{req}\,\varphi)\end{array}$$

and equally for roles. In this way we have simplifications such as $U\,\texttt{req}\,(\varphi \wedge U\,\texttt{req}\,\psi)$ implies $U\,\texttt{req}\,(\varphi \wedge \psi)$.

As in other system for reasoning about security [1,8,16] we avoid reflexivity, i.e. $A\,\texttt{req}\,\varphi \supset \varphi$, since the request of an agent may not be granted. Indeed the whole point of using `controls` is exactly to denote explictly the case where reflexivity holds, for a particular principal and a particular request. This idea is also used in [1] for their access control system and in [30].

$$(Ctrl)\ P\,\texttt{controls}\,\varphi \;\equiv\; [(P\,\texttt{req}\,\varphi) \supset \varphi]$$

The intuition may become clearer by rephrasing it as "P controls φ means that if P tries to execute φ then φ will indeed be executed by the system".

In further development of the logic it may be desirable to replace material implication with an intuitionistic or a relevant implication. With $(Ctrl)$ a principal may pretend to control whatever she does not explicitly request i.e. $\neg P\,\texttt{req}\,\varphi \supset P\,\texttt{controls}\,\varphi$. Such situation may also be a feature if we are interested in modelling "bluffing" principals. In any case this is still sufficient for modelling a wide range of situations.

Simplification properties are not imposed on cryptographic keys since it is open whether it would make sense to logically flatten multiple encryption:

$$\{p \wedge \{q\}_K\}_K \equiv^? \{p \wedge q\}_K$$

Therefore keys have different logical properties from users (in contrast with [1] where keys share the same properties of other principals). We only require that

$$\begin{array}{ll}(KKey) & \{\varphi \supset \psi\}_K \wedge \{\varphi\}_K \supset \{\psi\}_K \\ (SF) & K\,\texttt{speaks-for}\,P \wedge \{\varphi\}_K \supset P\,\texttt{req}\,\varphi\end{array}$$

Notice that even if we don't flatten encryption, we still want to derive from $K\,\texttt{speaks-for}\,U$ and $\{p \wedge \{q\}_K\}_K$ the request $U\,\texttt{req}\,(p \wedge q)$.

[1] Idempotence has been advocated also in [1] and corresponds to K4 axioms of modal logic and a weak form of the axiom T [10,17].

3 The Reasoning Services

Once we have formalised a RBAC system with a set G of role hierarchies, users attributions etc. we expect a number of *reasoning services*:

Consistency Check: check whether G has a model. Typically verify the compatibility of role hierarchies, users attributions etc. with the constraints.

Logical Consequence: decide whether a property φ logically follows from G. It could be used to decide whether a certain request should be granted or whether $R_1 \mathtt{isa} R_2$ is entailed even if is not explicitly written in G.

Model Generation: in case a property φ does not follows from G (it is not a logical consequence) derive a counter model for it.

Satisfiability: if logical consequence is too strong, decide whether ψ is compatible with G i.e. there is a model for G where a state satisfies ψ.

Of course we also expect *soundness, completeness and termination.*

The *proof search* of tableaux methods is based on *satisfiability* [14]. Suppose we have a G and we want to prove that a request ψ is satisfiable, then we try to construct a model for the initial formula by assuming that ψ holds in the actual world and breaking connectives or adding formulae from G and breaking them too. New states may also be introduced. If we always end up with contradictions then the initial formula is unsatisfiable.

To prove *logical consequence* i.e. $G \models \varphi$, we apply refutational theorem proving: start with the negation of φ and try to derive a countermodel. If we fail then the formula is valid otherwise we have directly the counterexample. *Consistency* of G can be proved by starting with $True$ and verifying that it is satisfiable. *Model generation* comes for free in this approach.

Notice that if we fail an axiomatic proof we don't know whether we (or our programs) are just bad logician or a proof does not actually exists. Tableaux methods yield a proof (no counter model found) or a counter model (a proof does not exists).

4 Formal Syntax and Semantics

The language, already described in §2, is the following:

$$P ::= U \mid R \mid P \,\mathtt{as}\, R \qquad \varphi, \psi ::= r \mid \neg\varphi \mid \varphi \wedge \psi \mid P \,\mathtt{req}\, \varphi \mid \{\varphi\}_K \mid$$
$$R_1 \,\mathtt{isa}\, R_2 \mid U \,\mathtt{has}\, R \mid K \,\mathtt{speaks\text{-}for}\, P$$

Other connectives are abbreviations, such as material implication $\varphi \supset \psi \equiv \neg(\varphi \wedge \neg\psi)$. We also use $P \,\mathtt{controls}\, \varphi$ as a shortcut (see axiom (*Ctrl*)).

The semantics is based on Kripke models (see [10,17] for an introduction). A *model* is a pair $\langle W, \mathcal{I}\rangle$ where W is a non empty set of states and $\mathcal{I}$ an interpretation such that for every atomic principal A it is $A^{\mathcal{I}} \subseteq W \times W$ and for every atomic request r it is $r^{\mathcal{I}} \subseteq W$.

Intuitively $r^{\mathcal{I}}$ represents the states where request r has been granted while the relation $A^{\mathcal{I}}$ models the compatibility between states: if $\langle w, w^*\rangle \in U^{\mathcal{I}}$ this

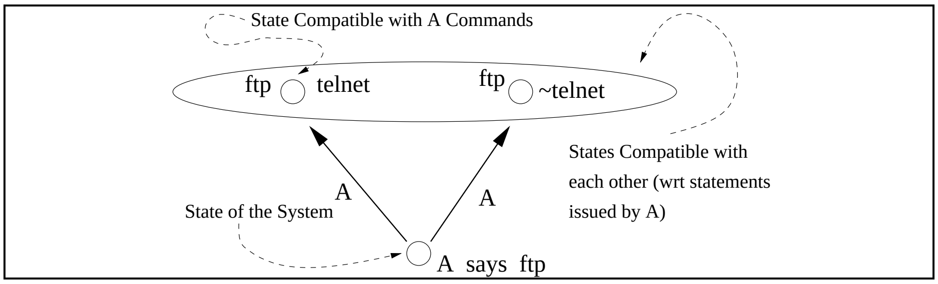

Fig. 3. A Simple Kripke Model

means that all requests made by user U in state w has been granted in state w^*. An example is shown in Fig. 3

There are *further requirements on roles and users* i.e. the corresponding relations $U^{\mathcal{I}}$ and $R^{\mathcal{I}}$ must be K45 relations [10,17] i.e. transitive and euclidean[2] Intuitively a K45 is an "almost equivalence relation" which is not reflexive and thus we have $A^{\mathcal{I}} = (\{w_0\} \cup W^*) \times W^*$ for $\{w_0\} \cup W^* \subseteq W$ and $w_0 \notin W^*$. as in Fig. 3. A K45 relation satisfies the properties of idempotence and awareness we have required in §2.

Keys K are also such that $K^{\mathcal{I}} \subseteq W \times W$ but we do not impose any constraint on the compatibility relation. However, the semantics of the `speaks-for` operator has been crafted to satisfy the properties described in §2.

The interpretation $\mathcal{I}$ is extended as follows:

$$
\begin{array}{ll}
(\varphi \wedge \psi)^{\mathcal{I}} & = \varphi^{\mathcal{I}} \cap \psi^{\mathcal{I}} \\
(\neg\varphi)^{\mathcal{I}} & = W - \varphi^{\mathcal{I}} \\
(P \,\texttt{req}\, \varphi)^{\mathcal{I}} & = \{w \mid \forall w' \in W \text{ if } \langle w, w'\rangle \in P^{\mathcal{I}} \text{ then } w' \in \varphi^{\mathcal{I}}\} \\
(R_1 \,\texttt{isa}\, R_2)^{\mathcal{I}} & = \{w \mid \forall w' \in W \text{ if } \langle w, w'\rangle \in R_2^{\mathcal{I}} \text{ then } \langle w, w'\rangle \in R_1^{\mathcal{I}}\} \\
(U \,\texttt{as}\, R)^{\mathcal{I}} & = \{\langle w, w'\rangle \mid \exists w''\langle w, w''\rangle \in U^{\mathcal{I}} \text{ and } \langle w'', w'\rangle \in R^{\mathcal{I}}\} \\
(U \,\texttt{has}\, R)^{\mathcal{I}} & = \{w \mid \forall w' \in W \text{ if } \langle w, w'\rangle \in R^{\mathcal{I}} \text{ then } \langle w, w'\rangle \in U^{\mathcal{I}}\} \\
(\{s\}_K)^{\mathcal{I}} & = \{w \mid \forall w' \in W \text{ if } \langle w, w'\rangle \in K^{\mathcal{I}} \text{ then } w' \in \varphi^{\mathcal{I}}\} \\
(K \,\texttt{speaks-for}\, P)^{\mathcal{I}} & = \{w \mid \forall w' \in W \text{ if } \langle w, w'\rangle \in P^{\mathcal{I}} \text{ then } \langle w, w'\rangle \in K^{\mathcal{I}}\}
\end{array}
$$

In the sequel we write $w \Vdash \varphi$ for $w \in \varphi^{\mathcal{I}}$.

The graphical representation of the semantics of "U `has` R" is given in Fig. 4. The basic intuition is that R `req` φ is interpreted as "somebody who `has` R said φ". So if U `has` R then all possible commands by U are also valid for R since that "somebody" could be indeed U. Therefore the set of possible states for R is contained in that of U.

Definition 3. A request φ is *satisfiable* iff there is a model $\langle W, \mathcal{I}\rangle$ where $(\varphi)^{\mathcal{I}}$ is not empty; it is *valid* iff for every model $\langle W, \mathcal{I}\rangle$ it is $(\varphi)^{\mathcal{I}} = W$.

[2] A relation R is euclidean if wRw' and wRw'' implies $w'Rw''$.

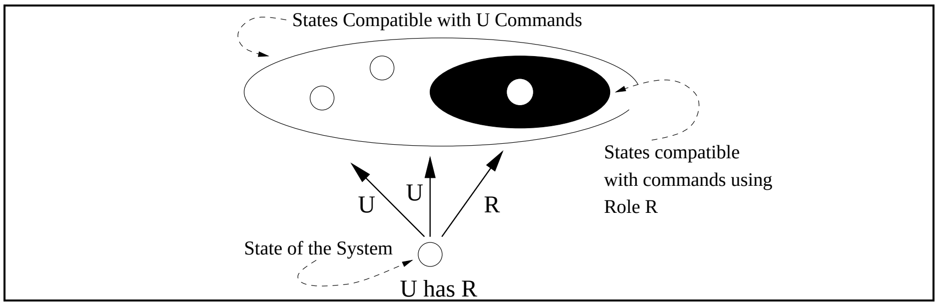

Fig. 4. A User with a Role

$$RH = \begin{cases} TS \ \mathtt{isa}\ S \\ S \ \mathtt{isa}\ C \\ C \ \mathtt{isa}\ U \end{cases} \qquad PA = \begin{cases} TS \ \mathtt{controls}\ distrib(file) \\ TS \ \mathtt{controls}\ read(file) \\ (Bob \ \mathtt{as}\ S) \ \mathtt{controls}\ read(file) \\ Charlie \ \mathtt{controls}\ write(file) \\ Server \ \mathtt{controls}\ (K_A \ \mathtt{speaks\text{-}for}\ Alice) \\ Server \ \mathtt{controls}\ (K_B \ \mathtt{speaks\text{-}for}\ Bob) \\ Server \ \mathtt{controls}\ (K_C \ \mathtt{speaks\text{-}for}\ Charlie) \end{cases}$$

$$UA = \begin{cases} Alice \ \mathtt{has}\ TS \\ Bob \ \mathtt{has}\ S \\ Charlie \ \mathtt{has}\ U \end{cases}$$

$$KA = \begin{cases} K_S \ \mathtt{speaks\text{-}for}\ Server \\ K_A \ \mathtt{speaks\text{-}for}\ Alice \end{cases} \qquad UR = \begin{cases} \{K_B \ \mathtt{speaks\text{-}for}\ Bob\}_{K_S} \\ \{Alice \ \mathtt{req}\ (read(file) \wedge distrib(file))\}_{K_B} \end{cases}$$

Fig. 5. Global Description of an RBAC Model

To define logical consequence we use the set of *Global axioms* G which holds in every possible state [14,23]. Global axioms are used in this framework for role hierarchies, user attributions etc.

$$G = RH \wedge PA \wedge UA \wedge KA \wedge UR \wedge Contraints$$

Definition 4. A request φ is a *logical consequence* of global axioms G iff for every model $\langle W, \mathcal{I} \rangle$ if $\forall v \in W : v \Vdash G$, then $\forall w \in W : w \Vdash \varphi$.

Example 5. Consider the hierarchy top-secret TS, secret S, classified C and unclassified U. The users are $Alice$ with TS privileges, Bob with S and $Charlie$ who has none. The system received a message and some delegation certificates (Fig. 5). Should $read(file) \wedge distrib(file)$ be granted with a write-up policy [28]?

In other terms is $read(file) \wedge distrib(file)$ a logical consequence of the RBAC in Fig. 5? Even in this simple case is not easy to see that it is *not*. Hence the need of a decision method.

5 A Decision Method Based on Tableaux

Prefixed tableaux use *prefixed requests*, i.e.pairs $\langle \sigma : \varphi \rangle$ where φ is a requests and σ is prefix, formally defined as $\sigma ::= 1 \mid \sigma.U.n \mid \sigma.R.n$ where n is an integer.

$$\alpha : \frac{\sigma : \varphi \wedge \psi}{\begin{array}{c}\sigma : \varphi \\ \sigma : \psi\end{array}} \quad \beta : \frac{\sigma : \neg(\varphi \wedge \psi)}{\sigma : \neg\varphi \mid \sigma : \neg\psi} \quad dn : \frac{\sigma : \neg\neg\varphi}{\sigma : \varphi} \quad \mathrm{G} : \frac{\vdots}{\sigma : \varphi} \text{ with } \varphi \in G \text{ and } \sigma \text{ present}$$

$$K : \frac{\sigma : A\,\mathtt{req}\ \varphi}{\sigma.{\scriptstyle A}.n : \varphi} \quad 4 : \frac{\sigma : A\,\mathtt{req}\ \varphi}{\sigma.{\scriptstyle A}.n : A\,\mathtt{req}\ \varphi} \quad 5 : \frac{\sigma.{\scriptstyle A}.n : A\,\mathtt{req}\ \varphi}{\sigma : A\,\mathtt{req}\ \varphi} \quad \pi : \frac{\sigma : \neg(A\,\mathtt{req}\ \varphi)}{\sigma.{\scriptstyle A}.m : \neg\varphi}$$

with $\sigma.{\scriptstyle A}.n$ present, $\sigma.{\scriptstyle A}.m$ new, and $A \in \{U, R\}$

Fig. 6. Rules for K45 multi-modal logic

$$\langle as \rangle : \frac{\sigma : \neg(U\ \mathsf{as}\ R)\,\mathtt{req}\ \varphi}{\sigma : \neg(U\,\mathtt{req}\ (R\,\mathtt{req}\ \varphi))} \qquad [as] : \frac{\sigma : (U\ \mathsf{as}\ R)\,\mathtt{req}\ \varphi}{\sigma : U\,\mathtt{req}\ (R\,\mathtt{req}\ \varphi)}$$

$$[Key] : \frac{\sigma : \{s\}_K}{\sigma.{\scriptstyle K}.n : \varphi}\ \sigma.{\scriptstyle K}.n \text{ present} \qquad \langle Key \rangle : \frac{\sigma : \neg\{s\}_K}{\sigma.{\scriptstyle K}.m : \neg\varphi}\ \sigma.{\scriptstyle K}.m \text{ new}$$

$$[sf] : \frac{\sigma : K\ \mathtt{speaks\text{-}for}\ P \qquad \sigma : \{s\}_K}{\sigma : P\,\mathtt{req}\ \varphi} \qquad \langle sf \rangle : \frac{\sigma : \neg(K\ \mathtt{speaks\text{-}for}\ P)}{\begin{array}{c}\sigma : \{c_h\}_K \\ \sigma : \neg(P\,\mathtt{req}\ c_h)\end{array}}\ c_h \text{ new}$$

Fig. 7. Users in Roles and Keys

The labelling follows the intuition behind the model: "1" is the actual (real) state. If σ "names" a state then $\sigma.{\scriptstyle Alice}.n$ (for $n = 2, 3 \ldots$) encodes in the name itself the property that the state "called" $\sigma.{\scriptstyle Alice}.n$ is compatible with state σ according *Alice*'s requests. After "giving names to things" we can speak of their properties. For instance $1.Alice.2 : Bob\ \mathtt{controls}\ login(ftp)$ tells that the privilege attribution *Bob* `controls` *login*(*ftp*) holds in the state $1.{\scriptstyle Alice}.2$.

A *tableau* is a rooted dyadic tree where nodes are labelled with requests, and a *branch* $\mathcal{B}$ is path from the root to a leaf [14,23]. A prefix is *present* in a branch if there is a prefixed formula with that prefix already in the branch, and it is *new* if it is not already present. Intuitively a branch is a (tentative) model for the initial formula. The rules for K45 are standard [14,23] and shown in Fig. 6.

The rules for users under certain roles are simple and recall sequential composition for PDL [9]. They are shown in Fig. 7 together with the rules for keys. The latter require a set of propositions $\mathcal{C}$ to represent unused cleartexts.

Roles hierarchy (RH) require more work due to the K45 properties on roles used to get idempotence. The difficulty — and therefore the need of extra rules w.r.t. [24] — is due to the interaction of K45 requirements with the `isa` requirements for the relation $R^{\mathcal{I}}$. The rules are shown in Fig. 8. Rule (IK) is not

$$IK : \frac{\sigma : R_1\,\mathtt{isa}\,R_2 \qquad \sigma : R_1\,\mathtt{req}\;\varphi}{\sigma : R_2\,\mathtt{req}\;\varphi} \qquad I\pi : \frac{\sigma : \neg(R_1\,\mathtt{isa}\,R_2)}{\begin{array}{c}\sigma : R_1\,\mathtt{req}\;x_i\\ \sigma : \neg(R_2\,\mathtt{req}\;x_i)\end{array}}\; x_i \text{ new}$$

$$IC : \frac{\sigma.{\scriptstyle R_2}.n : R_1\,\mathtt{isa}\,R_2}{\sigma : R_2\,\mathtt{req}\;(R_1\,\mathtt{isa}\,R_2)} \qquad I4 : \frac{\sigma : R_1\,\mathtt{isa}\,R_2}{\sigma.{\scriptstyle R_2}.n : R_1\,\mathtt{isa}\,R_2}$$

$$Tr : \frac{\sigma : R_1\,\mathtt{isa}\,R_2 \qquad \sigma : R_2\,\mathtt{isa}\,R_3}{\sigma : R_1\,\mathtt{isa}\,R_3}$$

$$K^* : \frac{\sigma : R_1\,\mathtt{req}\;\varphi \quad \sigma : R_1\,\mathtt{isa}\,R_2}{\sigma.{\scriptstyle R_2}.n : \varphi} \qquad 4^* : \frac{\sigma : R_1\,\mathtt{req}\;\varphi \quad \sigma : R_1\,\mathtt{isa}\,R_2}{\sigma.{\scriptstyle R_2}.n : R_1\,\mathtt{req}\;\varphi}$$

$$5^* : \frac{\sigma.{\scriptstyle R_2}.n : R_1\,\mathtt{req}\;\varphi \quad \sigma : R_1\,\mathtt{isa}\,R_2}{\sigma : R_1\,\mathtt{req}\;\varphi} \qquad C^* : \frac{\sigma.{\scriptstyle R_2}.n : R_1\,\mathtt{req}\;\varphi \quad \sigma.{\scriptstyle R_2}.n : R_1\,\mathtt{isa}\,R_2}{\sigma : R_2\,\mathtt{req}\;(R_1\,\mathtt{req}\;\varphi)}$$

Fig. 8. Roles Hierarchy

necessary given rule Tr and rules $K^*, 4^*, 5^*$. However its application followed by "traditional" K45 rules (Fig. 6) is sufficient in most cases.

For user attribution it is enough to replace "R_1" with "U" and "R_1 `isa`" with "U `has`" in all rules in Fig. 8.

Notice that if `isa` (or `has`) are not nested with `req` (as it is normally the case for RBAC) then rules (IK) and ($I\pi$) are the only necessary ones.

To define a tableaux proof we follow [9]. If $\mathcal{B}$ is the branch of a tableau, $\mathcal{B}/\sigma$ indicates the set of prefixed requests in $\mathcal{B}$ labelled with the prefix σ, i.e.:

$$\mathcal{B}/\sigma = \{\varphi \mid \langle \sigma : \varphi \rangle \in \mathcal{B}\}.$$

A prefix σ is *reduced* if π-rules are the only rules which have not been applied to requests of $\mathcal{B}/\sigma$. It is *fully reduced* if all rules have been applied. It is a *copy* of a prefix σ' if $\mathcal{B}/\sigma = \mathcal{B}/\sigma'$. In case two x_i, x_j are present in both prefix we assume them equal if they are generated by the same $R_1\,\mathtt{isa}\,R_2$ (similarly for y_k and c_h). The intuition is that we should use π-rule to create a new state only if we have not seen it before.

Definition 6. A branch $\mathcal{B}$ is *π-completed* if (i) all prefixes are reduced, and (ii) for every σ' which is not fully reduced there is a fully reduced copy σ.

Definition 7. A branch $\mathcal{B}$ is *closed* if it contains both $\sigma : \varphi$ and $\sigma : \neg\varphi$, for some request φ and some prefix σ. A branch is *open* if it is π-completed and not closed. A tableau is closed if all branches are closed; it is open if at least one branch is open.

Definition 8. A *validity tableaux proof* for request φ with axioms G is the closed tableau starting with $\langle 1 : \neg\varphi \rangle$.

Theorem 9 ((Strong Soundness)). *If request φ has a validity proof for G then φ is a logical consequence of G.*

Completeness has been obtained with a restriction: an RBAC policy is *stable* iff the `isa`-operator occurs positively[3] only in the Role Hierarchy. This is quite natural since roles are determined by the organisation [13,11,26] and therefore should not be changed by the principals at run time. On the contrary user attributions and privileges can be changed and thus we do not constrain them.

Theorem 10. *Let G with $\{\neg\varphi\}$ be a stable RBAC policy, if φ is a logical consequence of G then it has a validity proof.*

Theorem 11 ((Termination)). *Let G with $\{\neg\varphi\}$ be a stable RBAC policy, the construction of a closed or a π-completed tableau for G and $\neg\varphi$ can be terminated by loop checking if the role hierarchy is acyclic.*

The *computational complexity* of this method needs to be investigated with more details: multi-modal logics with global axioms are EXPTIME complete [17] and therefore we cannot go below this threshold in general. Indeed the termination proof constructs a (worst case) exponential model using a concept similar to the Fisher-Ladner closure of dynamic logic [9,10,17]. Note in comparison that the DEC-SRC calculus is undecidable [1].

Yet, hard cases requires ad hoc constructions and are extremely unlikely to appear in global axioms for RBAC. For instance the simple restriction to user requests of the form $U\,\mathtt{req}\,\varphi$, $(U\,\mathtt{as}\,R)\,\mathtt{req}\,\varphi$ or $\{\varphi\}_K$ where φ does *not* contains `req` (that is no nested `req`) leads to a collapse down to NP since the individual relation for users and roles is a K45 relation [17]. Therefore it is realistic to expect the calculus to be tractable for practical reasoning.

As a practical example we show the deduction for Example 1 (page 424) in Fig. 9. For simplicity we suppose to have direct rules for `controls` rather than transforming it into its definition. We abbreviate *login*(*telnet*) in *telnet* and similarly for *run*(*emacs*).

6 Related Works and Conclusions

To place this work among the various approaches to access control we can loosely classify them in two main streams[4]: transition and policy analysis.

Transition analysis studies the evolution of the system and its *safety properties*: given some privileges and some operations to change them, prove that the system will not evolve in an undesirable state. For instance this problem can be tackled in the form of transition between access matrices [18,27]. *Policy analysis* verifies that security mechanisms do actually fulfill the desired requirements and can be used to define decision procedure for granting access. This operation is a simple task for access matrices but it is more involved for expressive models

[3] Under the scope of an even number of negations.

[4] Of course the very same article may deal with both of them.

$1 : \neg(telnet \wedge emacs)$
$1 : maj\ \mathtt{req}\ (fm205\ \mathtt{has}\ User)$
$1 : maj\ \mathtt{has}\ SysAdm$
$1 : SysAdm\ \mathtt{req}\ (fm205\ \mathtt{has}\ User)$
$1 : SysAdm\ \mathtt{controls}\ (fm205\ \mathtt{has}\ User)$

$\swarrow \quad \searrow$

$1 : \neg SysAdm\ \mathtt{req}\ (fm205\ \mathtt{has}\ User)$
$\perp$

$fm205\ \mathtt{has}\ User$
$fm205\ \mathtt{req}\ (telnet \wedge emacs)$
$User\ \mathtt{req}\ (telnet \wedge emacs)$
$(*)$

$(*)$

$\swarrow \quad \searrow$

$1 : \neg telnet$
$1 : User\ \mathtt{controls}\ telnet$

$\swarrow \quad \searrow$

$1 : \neg User\ \mathtt{req}\ telnet$
$1._{User}.2 : \neg telnet$
$1._{User}.2 : telnet \wedge emacs$
$1._{User}.2 : telnet$
$1._{User}.2 : emacs$
$\perp$

$1 : telnet$
$\perp$

$1 : \neg emacs$
$1 : User\ \mathtt{controls}\ emacs$

$\swarrow \quad \searrow$

$1 : \neg User\ \mathtt{req}\ emacs$
$1._{User}.3 : \neg emacs$
$1._{User}.3 : telnet \wedge emacs$
$1._{User}.3 : telnet$
$1._{User}.3 : emacs$
$\perp$

$: emacs$
$\perp$

Fig. 9. Deduction for the request of Example 1

[1,5,11,22,26]. The approaches based on deontic logic [8,16] can also be placed within this latter form of analysis.

At present, most works on RBAC are concentrated on the aspect of policy analysis and this work is no exception to it. This is also due to the novel features of the model which has been fully characterised only recently [11,26].

Among the features described in [11,26] the practical reasoning mechanism proposed here captures the key points such as role hierarchies, user and privileges attributions. Static constraints (such as mutually exclusive roles) are also tackled. On the contrary dynamic separation of duties is not provided here, because the notion of session and the related dynamic attribution of roles [11,26] have not been considered so far. In comparison with the model proposed by [15] we have decoupled the notions of privileges and sub-roles (through the operators `isa` and `controls`) which are still merged together within and-roles in [15]. Our method should make it possible a more modular construction.

The approach proposed in this paper is closer to the logic for access control developed in [1] of which is at the same time an enhancement and a restriction. In particular we have incorporated into the semantics some features advocated by [1,22] such as idempotence of principals and refined the notions of the "speaks-for" operator of [1], which eliminate its undesired properties. For this logic it has been possible to provide soundness, completeness and decidability results, whereas the DEC-SRC logic is undecidable. However this work is also a restriction wrt [1,22] since we do not consider delegation. Extension to incorporate a full fledged notion of delegation are planned.

Beside the definition of a logic and a semantics for RBAC, one of the contribution of this paper is the development of a decision method which extends to a logic for RBAC the tableaux methods used for the logics of knowledge and belief [14,23] and the DEC-SRC calculus [24].

The combination of logical languages and decision methods based on logic may be a step further towards the use of formal methods for verification and design of RBAC with practical reasoning tools in computer security.

Acknowledgments

I would like to thank L. Paulson and the Computer Laboratory for their hospitality in Cambridge, M. Abadi, the Computer Security group (Cambridge), and the anonymous referees for many suggestions which helped to improve this paper. This research has been partly supported by ASI, CNR and MURST 40% and 60% grants and by EPSRC grant GR/K77051 "Authentication Logics".

References

1. M. Abadi, M. Burrows, B. Lampson, and G. Plotkin. A calculus for access control in distributed systems. *ACM Trans. on Programming Languages and Systems*, 15(4):706–734, 1993.
2. R. Anderson. A security policy model for clinical information systems. In *Proc. of the Symp. on Security and Privacy*. IEEE Press, 1996.
3. B. Beckert and R. Goré. Free variable tableaux for propositional modal logics. In *Proc. of TABLEAUX-97*, LNAI. Springer-Verlag, 1997. To appear.
4. D. Bell and L. La Padula. Secure computer systems: unified exposition and MULTICS. Report ESD-TR-75-306, The MITRE Corporation, March 1976.
5. E. Bertino, S. Jajodia, and P. Samarati. Supporting multiple access control policies in database systems. In *Proc. of the Symp. on Security and Privacy*, pp. 94–109. IEEE Press, 1996.
6. M. Burrows, M. Abadi, and R. Needham. A logic for authentication. *ACM Trans. on Comp. Sys.*, 8(1):18–36, 1990. Also as research report SRC-39, DEC - System Research Center, 1989.
7. D. Clark and D. Wilson. A comparison of commercial and military computer security policies. In *Proc. of the Symp. on Security and Privacy*, pp. 184–194. IEEE Press, 1987.
8. F. Cuppens and R. Demolombe. A deontic logic for reasoning about confidentiality. In *3rd Int. Workshop on Deontic Logic in Computer Science*, Portugal, 1996.
9. G. De Giacomo and F. Massacci. Tableaux and algorithms for propositional dynamic logic with converse. In *Proc. of the 13th Int. Conf. on Automated Deduction (CADE-96)*, LNAI 1104 , pp. 613–628. Springer-Verlag, 1996.
10. R. Fagin, J. Halpern, Y. Moses, and M. Vardi. *Reasoning about Knowledge*. The MIT Press, 1995.
11. D. Ferraiolo, J. Cugini, and K. Richard. Role-based access control (RBAC): Features and motivations. In *Proc. of the Annual Computer Security Applications Conf.*. IEEE Press, 1995.
12. D. Ferraiolo and R. Kuhn. Role based access control. In *Proc. of the NIST-NCSC Nat. (U.S.) Comp. Security Conf.*, pp. 554–563, 1992.

13. D. Ferraiolo, D. Gilbert, and N. Lynch. An examination of federal and commercial access control policy needs. In *Proc. of the NIST-NCSC Nat. (U.S.) Comp. Security Conf.*, pp. 107–116, 1993.
14. M. Fitting. *Proof Methods for Modal and Intuitionistic Logics.* Reidel, 1983.
15. L. Giuri and P. Iglio. A formal model for role based access control with constraints. In *Proc. of the Computer Security Foundations Workshop*, pp. 136–145. IEEE Press, 1996.
16. J. Glasgow, J. MacEwen, and P. Panangaden. A logic for reasoning about security. In *Proc. of the Symp. on Security and Privacy*, pp. 2–13. IEEE Press, 1990.
17. J. Halpern and Y. Moses. A guide to completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence*, 54:319–379, 1992.
18. M. Harrison, W. Ruzzo, and J. Ullman. Protection in operating systems. *Comm. of the ACM*, 19(8):461–471, 1976.
19. S. Kanger. Law and logic. *Theoria*, 38(3):105–132, 1972.
20. C. Krogh. Obligations in multiagent systems. In *Scandinavian Conf. on Artificial Intelligence (SCAI-95)*, pp. 29–31. ISO Press, 1995.
21. B. Lampson. Protection. *ACM Operating Sys. Reviews*, 8(1):18–24, 1974.
22. B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in distributed systems: Theory and practice. *ACM Trans. on Computer Systems*, 10(4):265–310, 1992.
23. F. Massacci. Strongly analytic tableaux for normal modal logics. In *Proc. of the Int. Conf. on Automated Deduction (CADE-94)*, LNAI 814, pp. 723–737. Springer Verlag, 1994.
24. F. Massacci. Tableaux methods for access control in distributed systems. In *Proc. of TABLEAUX-97*, LNAI. Springer-Verlag, 1997. To appear.
25. C. McCollum, J. Messing, and L. Notargiacomo. Beyond the pale of MAC and DAC - defining new forms of access control. In *Proc. of the Symp. on Security and Privacy*, pp. 190–200, IEEE Press, 1990.
26. R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-based access controls models. *IEEE Computer*, 29(2), February 1996.
27. R. Sandhu. The typed access matrix model. In *Proc. of the Symp. on Security and Privacy*, pp. 122–136. IEEE Press, 1992.
28. R. Sandhu and P. Samarati. Access control: Principles and practice. *IEEE Communications Magazine*, pp. 40–48, September 1994.
29. P. Syverson. The use of logic in the analysis of cryptographic protocols. In *Proc. of the Symp. on Security and Privacy*, pp. 156–170. IEEE Press, 1991.
30. P. Syverson and P. van Oorschot. On unifying some cryptographic protocols logics. In *Proc. of the Symp. on Security and Privacy.* IEEE Press, 1994.