SECURE KNOWLEDGE EXCHANGE BY POLICY ALGEBRA AND ERML

Steve Barker
Dept Computer Science, King's College, London, U.K.

Paul Douglas
Westminster University, London, U.K

Keywords: Database, Security, Logic Programming.

Abstract: In this paper, we demonstrate how role-based access control policies may be used for secure forms of knowledge module exchange in an open, distributed environment. For that, we define an algebra that a security administrator may use for defining compositions and decompositions of shared information sources, and we describe a markup language for facilitating secure information exchange amongst heterogeneous information systems. We also describe an implementation of our approach and we give some performance measures, which offer evidence of the feasibility of our proposal.

1 INTRODUCTION

We address the problem of securely exchanging knowledge modules in the context of the Semantic Web. The approach that we propose involves using a form of *Role-based Access Control (RBAC)* (Barker, 2003), a module algebra, and a new form of markup language. RBAC provides us with a way of defining access control requirements to help to ensure the security of information sources that are represented as identifiable modules, which can be composed using algebraic operators; the markup language that we introduce provides a way of securely exchanging accessible information modules.

In related work, Bonatti et al (2002) describes an approach for composing sets of *authorization facts*. However, the advocated approach is applicable only to "simple Horn clause" programs where composition of the least fixed point of programs is possible. The work of Wijesekera and Jajodia (2001) is similar to Bonatti et al's but Wijesekera and Jajodia use a state transformation-based approach on (s,a,o) triples, rather than using Horn clauses, to derive authorization atoms. In both approaches, policy algebras for discretionary access control policies are considered.

In our algebra, non-Horn specifications of policies are possible and rather than viewing policy composition at the level of authorizations, composition operators are defined on shareable information sources. Moreover, our emphasis is on the use of RBAC policies represented by using rule-based specifications with a meta-program used to process access requests. The use of rule-based representations of access policies is well known in the security literature (see, for example, Jajodia (2001), Barker (2000), and Barker (2003)). Our work is also related to work on secure XML (see Bhatti (2003), and Bertino (2004)), and to work on RuleML (Boley, 2001). However, our approach is concerned with secure information sharing amongst heterogeneous information systems, rather than XML documents; for this, we propose an extension of RuleML.

The approach that we describe is based on a concept that we call security filtering. That is, when a user requests to access a set of information sources or a subset of a information source then only the fragment of knowledge that the user is authorized to see is made available to them. The different information systems that we consider in this paper are stratified normal deductive databases (Baral, 1994), constraint databases (Marriott, 1998), and null-free and aggregate-free SQL databases (Date, 2003).

The rest of the paper is organized thus. In Section 2, preliminary notions are described. In Section 3, we present an algebra for policy composition.

Barker S. and Douglas P. (2007).
SECURE KNOWLEDGE EXCHANGE BY POLICY ALGEBRA AND ERML.
In *Proceedings of the Ninth International Conference on Enterprise Information Systems - DISI*, pages 212-217
DOI: 10.5220/0002358402120217

In Section 4, we describe our markup language. In Section 5 we describe a candidate implementation and performance measures. Finally, in Section 6, conclusions are drawn, and further work is suggested.

2 PRELIMINARIES

In this section, we briefly describe background material to help to make the paper self-contained. Further details on the databases that we consider may be found in, for example, Abiteboul (1995) and Kuper (2000).

Definition 2.1 *A normal database is a finite set of formulas of the form:*

$$A \leftarrow A_1, A_2, \ldots, A_m, not\ A_{m+1}, not\ A_{m+2}, \ldots, not\ A_{m+n}.$$

The head A of the clause in Definition 2.1 is a single atom. In the *body* of a clause

$$A_1, A_2, \ldots, A_m, not\ A_{m+1}, not\ A_{m+2}, \ldots, not\ A_{m+n}$$

$A_1, A_2, \ldots, A_m$ is a conjunction of atoms, and $not\ A_{m+1}, not\ A_{m+2}, \ldots, not\ A_{m+n}$ is a conjunction of atoms negated by *not* where *not* is *negation-as-failure* (Baral, 1994). A *literal* is an atom or its negation. A clause with an empty body (i.e., $A \leftarrow \emptyset$) is a *fact*. In the discussion that follows, we represent the fact $A \leftarrow \emptyset$ by A, and we use the term *rule* to refer to a fact or a clause with a non-empty body.

Definition 2.2 *A constraint database (CDB) consists of a finite set of rules of the following form (an extended form of normal deductive rules):*

$$A \leftarrow C_1, C_2, \ldots, C_m \mid L_1, L_2, \ldots, L_n \qquad (m \geq 0, n \geq 0).$$

The elements of the set $\{L_1, L_2, \ldots, L_n\}$ are literals, A is an atom, and C_1,C_2, ... ,C_m is a conjunction of constraints.[1]

Definition 2.3 *A primitive constraint c has the form $p(t_1, \ldots, t_n)$ where p is a (predefined) constraint relation of arity n and $t_1, \ldots, t_n$ are terms.*

Definition 2.4 *A constraint C is a conjunction of primitive constraints $c_1 \wedge \cdots \wedge c_k$ where $\wedge$ is the logical 'and' operation.*

In the normal databases that we use later, variables appear in the upper case and using characters from the end of the alphabet; constants appear in the lower case and using characters from the start of the alphabet.

[1] The | symbol is simply used to separate the conjunction of constraints from the conjunction of literals L_1,L_2, ... ,L_n; the translators we describe later may be modified for use with specific CDB systems.

We assume that the reader is familiar with basic SQL, and recognises that a normal clause and a normal rule in the CDBs that we consider may be equivalently represented in SQL and conversely.

In this paper, we restrict attention to the $RBAC_F$ model that is formally defined in Barker (2003) (i.e., flat RBAC as described by Sandhu (2001)).[2]

From Barker (2003), we recall that, in formulations of $RBAC_F$ policies, users are assigned to a role by using definitions of a 2-place *ura* predicate (where *ura* is short for "user role assignment"). The assignment of an access privilege on an object to a role is expressed by using definitions of a 3-place *pra* predicate in $RBAC_F$ policy specifications (where *pra* is short for "permission role assignment").

Example 2.5 *Suppose that the users u_1 and u_2 are assigned to the roles r_2 and r_1 respectively, and that write (w) permission on object o_1 is assigned to r_1 and read (r) permission on o_1 is assigned to r_1 and r_2. Then, the following set of facts may be used to represent the $RBAC_F$ policy in force:*

$$\{ura(u_1, r_2), ura(u_2, r_1), pra(w, o_1, r_1), pra(r, o_1, r_1), pra(r, o_1, r_2)\}.$$

A user u has the a access privilege on an object o (i.e., the *authorization* (u, a, o) holds) from an $RBAC_F$ policy specification if a user u is assigned to a role r, and r has been assigned the a access privilege on o. In terms of normal clauses, authorization triples are defined thus:

$$access(U, A, O) \leftarrow ura(U, R), pra(A, O, R).$$

Example 2.6 *By inspection of the user-role and permission-role assignments that are specified in Example 2.5, it follows that the following set of authorizations apply (the extension of the predicate access/3):*

$$\{access(u_1, w, o_1), access(u_1, r, o_1), access(u_2, r, o_2)\}.$$

As a final point on preliminaries, in our proposal each information source to be protected is uniquely identifiable. We use the notation, $\upsilon \equiv I$ (where υ may be subscripted) to denote that the information source I is identified by υ (e.g., in a Web context, υ is a uniform resource identifier).

3 A POLICY ALGEBRA

In this section, we describe an algebra for information source definition and manipulation. Our algebra is indicative of the type of algebra that security administrators may use to define access control requirements.

[2] Extending our approach to allow for richer RBAC models is a straightforward matter.

The algebra may be extended to permit other forms of policy to be defined.

3.1 Grammar

The grammar that we propose for the algebra of information sources that are identified by υ_i and υ_j may be expressed thus:

$$\mid \upsilon_i \mid \upsilon_i \cup \upsilon_j \mid \upsilon_i \cap \upsilon_j \mid \upsilon_i^* \mid \upsilon_i \circ \upsilon_j \mid$$

The operators in our algebra are defined next. In these definitions, B (possibly subscripted) denotes the body of a rule in normal database, a CDB, or the SELECT clause in a view definition.

Definition 3.1 *Let υ_i and υ_j be the identifiers for two information sources, then:*

$$\upsilon_i \cup \upsilon_j \stackrel{def}{=} \{L \leftarrow B : L \leftarrow B \in \upsilon_i \vee L \leftarrow B \in \upsilon_j\}.$$

The $\cup$ operator is used to combine the rules in different databases. For example, given $\upsilon_{10} \equiv p(X) \leftarrow q(X)$ and $\upsilon_{11} \equiv p(X) \leftarrow r(X)$, for $\upsilon_{10} \cup \upsilon_{11}$ we have:

$$p(X) \leftarrow q(X).$$
$$p(X) \leftarrow r(X).$$

Definition 3.2 *Let υ_i and υ_j be the identifiers for two information sources in a universe, then:*

$$\upsilon_i \cap \upsilon_j \stackrel{def}{=} \{L \leftarrow B : L_i \leftarrow B_1 \in \upsilon_i \wedge L_j \leftarrow B_2 \in \upsilon_j \wedge \theta = mgu(L_i, L_j) \wedge L = L_i\theta \wedge B = \{B_1, B_2\}\theta\}$$

where $mgu(L_i, L_j)$ is the most general unifier of L_i and L_j (see Apt, 1997).

The $\cap$ operator is used when rules that are common to a information source need to be included in a composite information source. As a simple example, $\{p(a,b), q(a,b)\} \cap \{p(a,b)\}$ generates $\{p(a,b)\}$.

Definition 3.3 *Let υ_i denote an information source in a universe and let $\beta(\upsilon_i)$ be the closure of the information source identified by υ_i under a consequences operator $\models$ (i.e., $\beta(\upsilon_i)$ is the set of all literal consequences of the information source identified by υ_i) then:*

$$\upsilon_i^* \stackrel{def}{=} \{L : L \in HBASE(\upsilon_i) \wedge L \in \beta(\upsilon_i)\}$$

where $HBASE(\upsilon_i)$ is the Herbrand Base of υ_i (see Abiteboul, 1995).

The $*$ operator is useful from a security viewpoint for restricting a information source $\mathcal{K}$ to just the atomic consequences of $\mathcal{K}$.

Definition 3.4 *Let υ_i and υ_j be the identifiers for two information sources, in a universe, let $LIT(L)$ denote the predicate symbols of literals in $HBASE(\upsilon_i) \cup HBASE(\upsilon_j)$, and let $\gamma(\upsilon_i)$ be the literals defined in the information source identified by υ_i, then:*

$$L\,\upsilon_i \circ \upsilon_j \stackrel{def}{=} \upsilon_i \cup \{L \leftarrow B : L \leftarrow B \in \upsilon_j \wedge LIT(L) \notin \gamma(\upsilon_i)\}.$$

The algebraic expression $\upsilon_i \circ \upsilon_j$ is used to specify that the definitions in υ_i override the definitions in υ_j when common predicates are defined. For example, for $\upsilon_{10} \equiv p(X) \leftarrow q(X) \circ \upsilon_{11} \equiv p(X) \leftarrow r(X)$, we have $\upsilon_{10} \equiv p(X) \leftarrow q(X)$.

To understand the use of our algebra, we present some examples of its use. For the examples, we consider the following normal database

$$\upsilon_1 \equiv \begin{cases} p(X,Y) \leftarrow q(X,Y). \\ q(a,2). \end{cases}$$

the constraint database

$$\upsilon_2 \equiv \begin{cases} p(X,Y) \leftarrow Y < 2 \mid r(X,Y). \\ r(b,2). \end{cases}$$

and the SQL database, identified by υ_3, that includes the view definition

```
create view s as select * from v
where v.A ≠ b;
```

together with the SQL table, called *v*, in Figure 1.

A	B
a	1
b	2

Figure 1: The SQL table v in υ_3.

Example 3.5 *Consider υ_1 and υ_2 from the set of databases above. Then, $\upsilon_1 \cup \upsilon_2$ defines the following information source:*

$$p(X,Y) \leftarrow q(X,Y).$$
$$p(X,Y) \leftarrow Y < 2 \mid q(X,Y).$$
$$q(a,2).$$
$$r(b,2).$$

Example 3.6 *The algebraic expression*

$$\cup(\upsilon_3^*, \circ(\upsilon_1, \upsilon_2))$$

defines the following information source:

$$p(X,Y) \leftarrow q(X,Y).$$
$$q(a,2).$$
$$r(b,2).$$
$$s(a,1).$$

where $\upsilon_3^* = s(a,1)$.

3.2 Access Request Evaluation

To process a user's access request, we use the following meta-program (in which *a* is short for "access"):

$a(U,read,X) \leftarrow ura(U,R),pla(read,X,R).$
$a(U,read,\cup(X,Y)) \leftarrow a(U,read,X),a(U,read,Y).$
$a(U,read,\cap(X,Y)) \leftarrow a(U,read,X),a(U,read,Y).$
$a(U,read,\circ(X,Y)) \leftarrow a(U,read,X),a(U,read,Y).$

The following points should be noted about the meta-program:

- We restrict attention to retrieval/read requests. However, any number of access privileges may be supported in our approach.
- Request evaluation is always performed with respect to propositional theories because the meta-program processes module identifiers.
- The meta-program may be used to process access requests in the case where an agent requests to either retrieve a knowledge module that a security administrator has predefined, by using the security algebra, or to retrieve a specific knowledge module that the agent wishes to retrieve.
- We adopt a file-oriented semantics for our algebra (not a consequence-oriented semantics). This is because we do not restrict attention to information sources that are categorical; our approach is file-based rather than rule-based because the latter raises a number of semantic and practical issues relating to rule identification. Although, at first sight, it may be argued that file level access is overly blunt, it must be noted that the security administrator can use the algebra to define very fine-grained information sources.

Example 3.7 *Consider the user-role and permission-role assignments from Example 2.5 together with the following expression:*

$$access(bob,read,\cup(\upsilon_1,\cap(\upsilon_2,\circ(\upsilon_3^*,\upsilon_4)))).$$

The `access/3` *expression represents a request by an agent Bob to access the information source that is specified by:*

$$\cup(\upsilon_1,\cap(\upsilon_2,\circ(\upsilon_3^*,\upsilon_4))).$$

Conversely, suppose that a security administrator has included the following expressions in an access policy specification:

$$\upsilon_8 \equiv \cup(\upsilon_1,\cap(\upsilon_2,\circ(\upsilon_3^*,\upsilon_4))).$$
$$pla(read,\upsilon_8,r_1).$$

Then, $access(bob,read,\upsilon_8)$ *may be used to generate (for Bob) the information source defined by* $\cup(\upsilon_1,\cap(\upsilon_2,\circ(\upsilon_3^*,\upsilon_4)))$.

4 ERML

In this section, we describe the markup language that we propose for use for secure knowledge exchange between heterogeneous information sources. Our markup language, called ERML (see Barker, 2004), is an extended form of RuleML (version 0.9) that enables, amongst other things, the set of operators $\{+,-,\div,\times,<,\leq,=,\neq,\geq,>\}$ to be represented in: the body of a normal clause, the specification of a WHERE clause in an SQL view definition, and for constraint formulation in a CDB. In the ensuing discussion, we use Φ to denote the set of comparison operators $\{<,\leq,=,\neq,\geq,>\}$, and Γ to denote the set of arithmetic operators $\{+,-,\div,\times\}$.

The general format for a rule in ERML is as follows:

```
<imp>
    <_head> conc </_head>
      <_body>
        <_constraint>
          <and>
          <builtin>
              <cop> ϕ </cop>
          </builtin>
          <builtin>
              <aop> γ </aop>
          </builtin>
          </and>
        </_constraint>
          <and>
            <atom>
              ...
            </atom>
          <builtin>
              <cop> ϕ </cop>
          </builtin>
          <builtin>
              <aop> γ </aop>
          </builtin>
          </and>
      </_body>
</imp>
```

Here, $< imp >$ is a tag that is used to denote an implication i.e., a normal rule, a rule in a CDB, or a view definition in SQL. The $< head >$ tag is used to represent the head of a normal rule, the head of a rule in a CDB (with arguments if any), or the name of the view definition v_{def} together with the attributes of v_{def}. The $< body >$ tag, as the name suggest, is used to represent the body of a normal rule, a CDB rule, or the SELECT statement that defines a view. The

tag $< cop >$ is used to refer to a comparison operator $\phi \in \Phi$, and $< aop >$ is a tag for markup that refers to an arithmetic operator $\gamma \in \Gamma$. We use strings of characters to denote the operators in $\Phi \cup \Gamma$ when these operators are used in ERML documents. For example, *gteq* is used for $\geq$; all of the operators in $\Phi \cup \Gamma$ can be assigned a literal value that may be used in ERML documents.

The negation of an atom is expressed by using a $<not> A </not>$ element where *not* is interpreted as negation-as-failure and *A* is an atom.[3]

The general format for a fact in ERML is as follows:

```
<fact>
    <_head> conc </_head>
        <_body>
            <atom>
                ...
            </atom>
        </_body>
</fact>
```

The *var* value in the tag $< var >$ is short for variable. To represent constants, $< ind >$ is used e.g., $< ind > 10 < /ind >$ denotes the constant 10.

A user's request to access a knowledge source r_k is handled by a 2-phase process:

- In the initial phase, evaluation of an access request is performed using the meta-program and a specification of an RBAC policy to determine whether u_i is permitted to exercise the p_j privilege on r_k.
- In the second phase, the authorized actions involving r_k are performed. If permitted by the access policy in force, r_k (and DTDs or XML Schema declarations, as appropriate) is transmitted to the requester in ERML form.

The information, in ERML form, returned to a requester can be manipulated using the rule engines that are available on the requester's machines. To enable this, we have developed XSLT stylesheets to transform ERML into XSB, SQL, and CDBs. For the translation of SQL view definitions, we adopt a 2-step approach that involves converting SQL to normal clause form and then calling the XSB to ERML translator to generate the ERML form of the SQL view definition. Our conversion software converts facts in a base table $\mathcal{T}$ to ERML by converting each row in $\mathcal{T}$ to an ERML fact (see above). We have chosen the 2-step approach to implementation, rather than using Eberhardt's OntoSQL translator (see Eberhardt, 2002), for a number of reasons. For instance, our approach enables arbitrary *n*-ary relations to be manipulated (not just binary relations) and our approach enables SQL, with arithmetic, to be translated into CDB form for efficient processing. Our two-stage approach imposes no significant overhead relative to the processing costs incurred using a single phase translator like OntoSQL.

[3]We do not consider ERML with a $<neg> A </neg>$ element, for explicit negation, because the information sources that we consider are normal databases, CDBs and SQL databases. However, our DTD includes a $<not> A </not>$ element to accommodate a classical negation operator.

Example 4.1 *Consider the request* $access(bob, read, \cup(\upsilon_{14}, \upsilon_{15}))$ *such that Bob is permitted read access on* υ_{14} *and* υ_{15} *and where* $\upsilon_{14} \equiv \{q(a,2)\}$ *and* $\upsilon_{15} \equiv \{p(X,Y) \leftarrow r(X,Y)\}$. *Then, the following ERML code is what Bob is permitted to have delivered to his machine:*

```
<rulebase>
<fact>
  <atom>
   <_opr><rel>q</rel></_opr>
    <ind>a</ind>
    <ind>2</ind>
  </atom>
</fact>

<imp>
  <_head>
   <atom>
    <_opr><rel>p</rel></_opr>
     <var>X</var>
     <var>Y</var>
   </atom>
  </_head>
  <_body>
   <atom>
    <_opr><rel>r</rel></_opr>
     <var>X</var>
     <var>Y</var>
   </atom>
  </_body>
</imp>
</rulebase>
```

5 PRAGMATICS

In this section, we describe implementation issues and some performance measures.

We note that one of the first declarative translators was GEDCOM (Dean, 2001), which (in alternative versions) uses XSLT to produce output suitable for use with either XSB Prolog or Jess (Grosof, 2002). However, we have tested the XSB version of GEDCOM and found it rather unreliable when used with real-world marked-up Prolog programs. Moreover, as XSLT only works on tags embedded within

an XML document, and as these are necessarily absent from, for example, a Prolog program, a declarative translator in the style of GEDCOM is not possible. SweetJess (Grosof, 2002) offers robust procedural, bi-directional translation from RuleML to XSB Prolog; however, it is significantly more cumbersome to deploy than a declarative system.

For our implementation, we have written a parser in C (for the details, see *www.cscs.wmin.ac.uk/* $\sim$ *douglap/translator.c*). Our parser is able to transform Prolog and various CDB forms into ERML, and can generate ERML from XSB Prolog, SQL or ECLIPSE. We have tested our translator on a information source containing 3,000 facts and rules (in normal clause form). The C translator took an average (over 10 runs) of 0.043 seconds to convert the information source into a ERML format. We used the Saxon XSLT processor (Kay, 2001) to convert the ERML file back into normal clause form/Prolog, which took an average of 2.455 seconds.[4] Thus, the bidirectional translator for ERML-SQL conversion adds no appreciable computation costs to access requests. The time taken by our meta-program to evaluate access requests is of the order of a few milleseconds; the size of the access control program has very little affect on the timings. It follows that access request checking and authorized knowledge base generation can be performed in a time of the order of a few seconds (for large information sources).

6 CONCLUSIONS

The principal contributions of our work have been: to specify an algebra that may be used by security administrators to define information sources that may be securely exchanged between agents using heterogeneous systems; to define a small but powerful access control program that determines what a user is permitted to access given an $RBAC_F$ policy specification; and to introduce a markup language that permits secure forms of information sources to be exchanged between heterogeneous information systems.

In future work, we wish to investigate implementations of our approach with more expressive algebras. Although $RBAC_F$ policies are adequate for the work that we have carried out, we intend to experiment with more expressive forms of RBAC policies in future work. The approach that we describe may be applied to problems in secure business rule processing, secure e-commerce, and secure e-contracting. A matter for further work is to investigate these applications.

[4]These results were obtained on a Sun Sparc Ultra 10 machine with a 440MHz CPU and 1Gb RAM running Solaris 10.

REFERENCES

Abiteboul, S., Hull, R. and Vianu, V. 1995. *Foundations of Databases*. Addison-Wesley.

Apt, K. 1997. *From Logic Programming to Prolog*. Prentice Hall.

Baral, C. and Gelfond, M. 1994. *Logic Programming and Knowledge Representation*. JLP, vol 19/20, pp73-148.

Barker, S. 2000. *Data Protection by Logic Programming*. Proc. 1st International Conference on Computational Logic. Springer-Verlag.

Barker, S. and Stuckey, P. 2003. *Flexible Access Control Policy Specification with Constraint Logic Programming*. ACM Trans. on Information and System Security, vol 6, number 4, pp501–546.

Barker, S. 2004. *Labeled Logic Programs*. Springer-Verlag.

Bhatti, R., Joshi, J., Bertino, E. and Ghafoor, A. 2003. *Access Control in Dynamic XML-Based Web-Services with X-RBAC*. In ICWS 2003, pp243-249.

Bonatti, P., Vimercati, S. and Samarati, P. 2002. *An algebra for Composing access control policies*. TISSEC 2002, vol 5, number 1, pp1-35.

Date, C. 2003. *An Introduction to Database Systems*. Addison-Wesley.

Dean, M. 2001. *RuleML Experiments with GEDCOM*. www.daml.org/2001/02/gedcom-ruleml/

Eberhardt, A. 2001. *Prolog2RuleML Parser*. www.i-u.de/schools/eberhart/prolog2ruleml

Eberhardt, A. 2001. *OntoSQL*. www.aifb.uni-karlsruhe.de/WBS/aeb/ontosql/

Grosof, B., Gandhe, M. and Finin, T. 2002. *SweetJess: Translating DAMLRuleML to JESS*. SunSITE.Informatik.RWTH-Aachen.DE/Publications/CEUR-WS/Vol-60/grosof.pdf

Jajodia, S., Samarati, P., Sapino, M. and Subrahmaninan, V. 2001. *Flexible Support for Multiple Access Control Policies*. ACM TODS, vol 26, number 2, pp214-260.

Kay, M. 2001. *The SAXON XSLT and XQuery Processor*. http://saxon.sourceforge.net/

Kuper, G., Libkin, L. and Paredaens, J. 2000. *Constraint Databases*. Springer.

Marriott, K. and Stuckey, P. 1998. *Programming with Constraints: an Introduction*. MIT Press.

Sandhu, R., Ferraiolo, D. and Kuhn, R. 2001. *The NIST Model for Role-Based Access Control: Towards a Unified Standard*. Proc. 4th ACM Workshop on Role-Based Access Control, pp47–61.

Wijesekera, D. and Jajodia, S. 2001. *Policy algebras for access control: the propositional case*. Proc. ACM Conference on Computer and Communications Security pp38-47.