A Logical Model for Security of Web Services*

Hristo Koshutanski and Fabio Massacci

Dip. di Informatica e Telecomunicazioni - Univ. di Trento
via Sommarive 14 - 38050 Povo di Trento (ITALY)
{hristo,massacci}@dit.unitn.it

Abstract. Business Processes for Web Services are the new paradigm for the lightweight integration of business from different enterprises.
Yet, there is not a comprehensive proposal for a logical framework for access control for business processes though logics for access control policies for basic web services are well studied.
In this paper we propose a logical framework for reasoning (deduction, abduction, consistency checking) about access control for business processes for web services.

1 Introduction

In the past millennium the development of middleware connected the IT efforts to integrate distributed resources of a single enterprise. The new century has seen the rise of a new IT concept: virtual enterprise. Virtual enterprise is born when a business process is no longer closed within the boundary of a single corporation. The business process is thus composed by partners that offer their services on the web and integrate each other efforts into one (hopefully) coherent process.

The scenario offered by business processes for web services is particularly challenging for the definition of its security features. Indeed it has some aspects of trust management systems and some aspects of workflow security management.

From the trust management systems (see e.g. [13, 7]) it takes the credential-based view: a (web) service is offered on its own and the decision to grant or deny access can only be made on the basis of the credentials sent by the client.

From workflow authorization systems (see e.g. [2, 9]) we borrow all classical problems such as dynamic assignment of roles to users, dynamic separation of duties, and assignment of permission to users according to the least privilege principles. In contrast with workflow security management schemes a business process for web services crosses organizational boundaries and is provided by entities that sees each other as just partners and nothing else. We have something even more loosely coupled than federated databases.

So, it is not a surprise that there is not a comprehensive proposal for a logical framework that tackles these aspects, though logics for access control policies for basic web services, workflows, and distributed systems are well studied.

* This work is partially funded by the IST programme of the EU Commission, FET under the IST-2001-37004 WASP project and by the FIRB programme of MIUR under the RBNE0195K5 ASTRO Project.

We identify a number of differences w.r.t. traditional access control:

- credential vs user based access control,
- interactive vs one-off evaluation of credentials (i.e., controlled disclosure of information vs all-or-nothing decision),
- on-line vs off-line analysis of consistency of roles and users assignments (e.g., for separation of duties),

In this paper we propose a logical framework for reasoning about access control for business processes for web services. We identify the different reasoning tasks (deduction, abduction, consistency checking) that characterize the problem and clarify the problems of temporal evolution of the logical model (addition and revocation of credentials).

2 The Formal Framework

Our formal model for reasoning on access control is based variants of Datalog with the stable model semantics and combines in a novel way a number of features:

- logics for trust management by Li et al. [11];
- logic for workflow access control by Bertino et al. [2];
- logic for release and access control by Bonatti and Samarati [3].

We consider the view of the single partner, as we cannot assume sharing of policies between partners. In [10] it is explained how the entire process can be orchestrated by using "mobile" business processes, while keeping each partner policy decision process as a black-box.

In our framework each partner has a *security policy for access control* $\mathcal{P}_{\mathcal{A}}$ and a *security policy for interaction control* $\mathcal{P}_{\mathcal{I}}$, whose syntax will be defined later in §3.

The policy for access control is used for making decision about usage of all web services offered by the partner. The policy for interaction control is used to decide which credentials must be additionally provided or must be revoked by the user if those available are not adequate to obtain the service.

In many workflow authorization schemes, the policy is not sufficient to make an access control decisions and thus we need to identify the *history of the execution* $\mathcal{H}$ of the business process as perceived by the current partner, and a set of *active (unrevoked) credentials* $\mathcal{C}_{\mathcal{A}}$ that have been presented by the agent in past requests to other services comprised in the same business process.

To execute a service of the fragment of business process under the control the partner the user will submit a set of *presented credentials* $\mathcal{C}_{\mathcal{P}}$, a set of *revoked credential* $\mathcal{C}_{\mathcal{R}}$ and a *service request* $\mathcal{R}(.)$ We assume that $\mathcal{C}_{\mathcal{R}}$ and $\mathcal{C}_{\mathcal{P}}$ are disjoint.

To specify how the access control decision is made we now assume the usual inference capabilities, that is, for any set of formulae (Datalog rules and facts) $\mathcal{F}$ and any formula f:

deduction determines whether f is a logical consequence of $\mathcal{F}$, $\mathcal{F} \models f$;
consistency determines whether $\mathcal{F}$ is consistent, $\mathcal{F} \not\models \bot$;
abduction given an additional set of atoms $\mathcal{A}$ called the abducible atoms, and a partial order relation $\prec$ between subsets of A determine a set of atoms $\mathcal{E} \subset \mathcal{A}$ such that (i) f is a logical consequence of $\mathcal{F}$ and $\mathcal{E}$, namely $\mathcal{F} \cup \mathcal{E} \models f$, (ii) adding $\mathcal{E}$ to $\mathcal{F}$ does not generate an inconsistency, namely $\mathcal{F} \cup \mathcal{E} \not\models \bot$, and finally (iii) $\mathcal{E}$ is a $\prec$-minimal subset of $\mathcal{A}$ having this property (See further Def. 1).

For an introduction to abduction see Shanahan [12], for a survey of complexity results - Eiter et al. [6]. We shall see later on in section 5 how abduction is used.

3 Logical Syntax

For the syntax we build upon [2, 3, 11]. We have four disjoint sets of constants, one for users identifiers denoted by $\mathsf{User}:U$, one for roles $\mathsf{Role}:R$, one for services $\mathsf{WebServ}:S$, and finally one for keys that are used to certify credentials $\mathsf{Key}:K$.

We assume that we have the following security predicates:

$\mathsf{Role}:R_i \succ \mathsf{Role}:R_j$ when role $\mathsf{Role}:R_i$ dominates in the global role hierarchy role $\mathsf{Role}:R_j$.
$\mathsf{Role}:R_i \succ_{\mathsf{WebServ}:S} \mathsf{Role}:R_j$ when role $\mathsf{Role}:R_i$ dominates in the local role hierarchy for service $\mathsf{WebServ}:S$ role $\mathsf{Role}:R_j$.
$\mathsf{assign}\,(P, \mathsf{WebServ}:S)$ when an access to the service $\mathsf{WebServ}:S$ is granted (assigned) to P. Where P can be either a $\mathsf{Role}:R$ or $\mathsf{User}:U$.
$\mathsf{forced}\,(P, \mathsf{WebServ}:S)$ if the predicate is true then an access right to access the service $\mathsf{WebServ}:S$ must be given (forced) to P. Where P can be either a $\mathsf{Role}:R$ or $\mathsf{User}:U$.

We have three type of predicates for credentials:

$\mathsf{declaration}\,(\mathsf{User}:U)$ it is a statement declared by the $\mathsf{User}:U$ for its identity.
$\mathsf{credentialID}\,(\mathsf{Key}:K, \mathsf{User}:U, \mathsf{Role}:R)$ it is a statement declared and signed by $\mathsf{Key}:K$ corresponding to some trusted authority that $\mathsf{User}:U$ has activated $\mathsf{Role}:R$.
$\mathsf{credentialTask}\,(\mathsf{Key}:K, \mathsf{User}:U, \mathsf{WebServ}:S)$ it is a statement declared and signed by $\mathsf{Key}:K$ corresponding to some trusted authority that $\mathsf{User}:U$ has the right to access $\mathsf{WebServ}:S$.

Three type of predicates describing the current status of each service:

$\mathsf{running}\,(P, \mathsf{WebServ}:S, \mathsf{number}:N)$ if it is true then the $\mathsf{number}:N$th activation of $\mathsf{WebServ}:S$ is executed by P.
$\mathsf{abort}\,(P, \mathsf{WebServ}:S, \mathsf{number}:N)$ if it is true then the $\mathsf{number}:N$th activation of $\mathsf{WebServ}:S$ within a workflow aborts.
$\mathsf{success}\,(P, \mathsf{WebServ}:S, \mathsf{number}:N)$ if it is true then the $\mathsf{number}:N$th activation of $\mathsf{WebServ}:S$ within a workflow successfully executes.

Furthermore, for some additional workflow constraints we need to have some meta-level predicates that specify how many statements are true. We use here a notation borrowed from Niemelä smodels system, but we are substantially using the count predicates defined by Das [4]:

$n \leq \{X.\, Pr\}$ where n is a positive integer, X is a set of variables, and Pr is a predicate, so that intuitively $n \leq \{X.\, Pr\}$ is true in a model if at least n instances of the grounding of X variables in Pr are satisfied by the model. The $\{X.\, Pr\} \leq n$ is the dual predicate.

4 Formal Rules and Semantics

Normal logic programs [1] are sets of *rules* of the form:

$$A \leftarrow B_1, \ldots, B_n,\ not\, C_1, \ldots,\ not\, C_m \tag{1}$$

where A, B_i and C_i are (possibly ground) predicates among those listed in Sec.3 A is called the *head* of the rule, each B_i is called a *positive literal* and each $not\, C_j$ is a *negative literal*, whereas the conjunction of the B_i and $not\, C_j$ is called the *body* of the rule. A normal logic program is a set of rules.
In our framework, we also need *constraints* that are rules with an empty head.

$$\leftarrow B_1, \ldots, B_n,\ not\, C_1, \ldots,\ not\, C_m \tag{2}$$

One of the most prominent semantics for normal logic programs is the *stable model semantics* proposed by Gelfond and Lifschitz [8] (see also [1] for an introduction). The intuition is to interpret the rules of a program P as constraints on a solution set S (a set of ground atoms) for the program itself. So, if S is a set of atoms, a rule 1 is a constraint on S stating that if all B_i are in S and none of C_j are in it, then A must be in S. A constraint 2 is used to rule out from the set of acceptable models the situation in which B_i are true and all C_j are false is not acceptable.

Definition 1 (Abduction). *Let P be a logic program, H be a set of predicates (called hypothesis, or abducibles), L be a (positive or negative) ground literal (sometimes called the manifestation), and $\prec$ a p.o. over subsets of H, the* cautious solution *of the abduction problem is a set of ground atoms E such that*

1. *E is a set ground instances of predicates in H,*
2. *$P \cup E \models L$*
3. *$P \cup E \not\models \bot$*
4. *any set $E' \prec E$ does not satisfy all three conditions above*

Remark 1. The choice of the partial order has a major impact in presence of complex role hierarchies. The "intuitive" behavior of the abduction algorithm for what regards the extraction of the minimal set of security credentials is not guaranteed by the straightforward interpretation of H as the set of credentials and by the set cardinality or set containment as minimality orderings.

To understand the problem consider the following logic program:
$\mathsf{assign}\,(\mathsf{User}:U, \mathsf{WebServ}:ws) \leftarrow \mathsf{credentialID}\,(\mathsf{Key}:k, \mathsf{User}:U, \mathsf{Role}:R),$
$\mathsf{Role}:R \succ \mathsf{Role}:r_1$
$\mathsf{Role}:r_2 \succ \mathsf{Role}:r_1 \leftarrow$
The request $\mathsf{assign}\,(\mathsf{User}:fm, \mathsf{WebServ}:ws)$ has two $\subseteq$-minimal solutions:
$\{\mathsf{credentialID}\,(\mathsf{Key}:k, \mathsf{User}:fm, \mathsf{Role}:r_1)\}, \{\mathsf{credentialID}\,(\mathsf{Key}:k, \mathsf{User}:fm, \mathsf{Role}:r_2)\}$
Yet, our intuition is that the first should be the minimal one.

So, we need a more sophisticated partial order. For example, if $E \preceq E'$ is such that for all credentials $c \in E$ there is a credential $c' \in E'$ where $c = c'$. We can revise it so that $E \prec E'$ if $c \in E$ there is a credential $c' \in E'$ where c' is identical to c except that it contains a role R' that dominates the corresponding role R in c. This p.o. generates the "intuitive" behavior of the abduction algorithm.

Definition 2. *An access control policy $\mathcal{P}_\mathcal{A}$ is a logic program over the predicates defined in Sec. 3 in which (i) no credential, no role hierarchy atom, and no execution atom can occur in the head of a rule and (ii) for every rule containing and head A which is the (possibly ground instance of) predicate* $\mathsf{forced}\,(P, \mathsf{WebServ}:S)$ *there is the (possibly ground instance of) rule* $\mathsf{assign}\,(P, \mathsf{WebServ}:S) \leftarrow$ $\mathsf{forced}\,(P, \mathsf{WebServ}:S)$.

An access control request is a ground instance of an $\mathsf{assign}\,(i, \mathsf{WebServ}:k)$ *predicate.*

The request r is a security consequence of a policy $\mathcal{P}_\mathcal{A}$ if (i) the policy is logically consistent and (ii) the request is a logical consequence of the policy.

Definition 3. *An interaction policy $\mathcal{P}_\mathcal{I}$ is a logic program in which no credential and no role hierarchy atom can occur in the head of a rule.*

Definition 4 (Fair Access). *Let $\mathcal{P}_\mathcal{A}$ be an access control policy, let $\mathcal{C}_\mathcal{D}$ be the set of ground instances of credentials occurring in $\mathcal{P}_\mathcal{A}$, and let $\prec$ be a p.o. over subsets of $\mathcal{C}_\mathcal{D}$. The access control policy guarantees* $\prec$-fair access *if for any ground request r that is an instance of a head of a rule in $\mathcal{P}_\mathcal{A}$ there exists a set $\mathcal{C}_\mathcal{M} \subseteq \mathcal{C}_\mathcal{D}$ that is a solution of the abduction problem.*

Definition 5 (Fair Interaction). *Let $\mathcal{P}_\mathcal{A}$, and $\mathcal{P}_\mathcal{I}$ be, respectively, an access control policy and an interactive policy, and let $\mathcal{C}_\mathcal{D}$ be the set of ground instances of credentials occurring in $\mathcal{P}_\mathcal{A}$, and let $\prec$ be a p.o. over subsets of $\mathcal{C}_\mathcal{D}$. The policies guarantee* $\prec$-fair interaction *w.r.t. a set of credentials $\mathcal{C}_\mathcal{I}$ if (i) $\mathcal{P}_\mathcal{A}$ guarantees $\prec$-fair access and (ii) for any solution of the abduction problem $\mathcal{C}_\mathcal{M} \subseteq \mathcal{C}_\mathcal{D}$ and any credential $c \in \mathcal{C}_\mathcal{M}$ if it $\mathcal{P}_\mathcal{I} \cup \mathcal{C}_\mathcal{I} \models c$. If the set of initial credentials $\mathcal{C}_\mathcal{I}$ only contains declarations then the access control is* unlimited.

5 The Formal Framework: Reasoning

To allow for an easier grasp of the problem we start with a basic framework. **Traditional Access Control.** This approach is the cornerstone of most logical formalization [5].

1. verify that the request is a logical consequence of the credentials, namely $\mathcal{P}_{\mathcal{A}} \cup \mathcal{C}_{\mathcal{P}} \models r$
2. if the check succeeds then grant access else deny access

A number of works has deemed such blunt denial unsatisfactory and therefore it has been proposed by Bonatti and Samarati [3] and Yu et al. [14] to send back to the client some of the rules that are necessary to gain additional access. **Disclosable Access Control.** It is revised to allow for the flow of rules and information to users:

1. verify that the request is a logical consequence of the credentials, namely $\mathcal{P}_{\mathcal{A}} \cup \mathcal{C}_{\mathcal{P}} \models r$
2. if the check succeeds then access is granted, otherwise select some rule $r \leftarrow p \in \mathcal{P}_{\mathcal{A}}$ and send the rule back to the client

In many cases, this solution is neither sufficient nor desirable. For instance if the policy is not flat, it has constraints on the credentials that can be presented at the same time (e.g., separation of duties) or a more complex role structure is used, these systems would not be complete. Also repeated queries allow for the disclosure of the entire policy, which might well be undesirable. In this case we need the interactive access control solution for Web Services proposed by Koshutanski and Massacci [10] that is described below.

Interactive Access Control for Stateless WS.

1. verify that the request is a logical consequence of the credentials, namely $\mathcal{P}_{\mathcal{A}} \cup \mathcal{C}_{\mathcal{P}} \models r$
2. if the check succeeds then access is granted, otherwise
   (a) compute the set of *disclosable credentials* as $\mathcal{C}_{\mathcal{D}} = \{c | c \text{ credential and } \mathcal{P}_{\mathcal{I}} \cup \mathcal{C}_{\mathcal{P}} \models c\}$
   (b) use abduction to find a minimal set of missing credentials $\mathcal{C}_{\mathcal{M}}$ such that both $\mathcal{P}_{\mathcal{A}} \cup \mathcal{C}_{\mathcal{M}} \models r$ and $\mathcal{P}_{\mathcal{A}} \cup \mathcal{C}_{\mathcal{M}} \not\models \bot$
   (c) if no such set exists then $\bot$ is sent back to the user,
   (d) otherwise the set of missing credentials $\mathcal{C}_{\mathcal{M}}$ is send back to the client and the process re-iterates.

This type of decision is characteristic of most logical approaches to access control [11, 2, 3]: we only look at the policy, the request and the set of credentials.

If the authorization decisions of business processes are stateful, and the the corresponding workflow of the partners has constraints on the execution of future services on the basis of past services, then even this solution is not adequate enough. As we already noted, the problems are the following:

- the request may be inconsistent with some role that the user has taken up in the past;
- the new set of credential may be inconsistent with requirements such as separation of duties;

So, this means that we must have some roll-back procedure by which, if the user has by chance sent the "wrong" credentials, he has some revocation mechanism to drop them.

Access Control for Stateful Business Processes. At this stage, we need all the policy and set of credentials that we have envisaged and indeed the partner expects from the client the set of current credentials $\mathcal{C}_{\mathcal{P}}$ plus the set of revoked $\mathcal{C}_{\mathcal{R}}$. The (logical) access control decision takes the following steps:

1. remove the revoked credentials from the set of active credentials, namely $\mathcal{C}_{\mathcal{A}} \leftarrow \mathcal{C}_{\mathcal{P}} \cup \mathcal{C}_{\mathcal{A}} \setminus \mathcal{C}_{\mathcal{R}}$,
2. verify the consistency of the request with the active set of credentials and the history of execution, namely $\mathcal{P}_{\mathcal{A}} \cup \mathcal{H} \cup \mathcal{C}_{\mathcal{A}} \cup \{r\} \not\models \bot$
3. If this check succeeds goes to the next step, otherwise
   (a) derive a subset of *excessing credentials* that must be revoked by the user $\mathcal{C}_{\mathcal{E}} \subseteq \mathcal{C}_{\mathcal{A}}$ such that the set $\mathcal{C}_{\mathcal{E}}$ is minimal w.r.t. the $\prec$ partial order and that by removing it from $\mathcal{C}_{\mathcal{A}}$ the consistency check would succeed
   (b) if no such set exists then $\bot$ is sent back to the user
   (c) if it exists, this set is send back to the user and the process is re-iterated.
4. verify that the request is a logical consequence of the credentials, namely $\mathcal{P}_{\mathcal{A}} \cup \mathcal{H} \cup \mathcal{C}_{\mathcal{A}} \models r$,
5. if this check succeeds then access is granted
6. if the step fails
   (a) compute the set of *disclosable credentials* as $\mathcal{C}_{\mathcal{D}} = \{c | c \text{ credential and } \mathcal{P}_{\mathcal{I}} \cup \mathcal{H} \cup \mathcal{C}_{\mathcal{A}} \models c\}$
   (b) use abduction to find a minimal set of missing credentials $\mathcal{C}_{\mathcal{M}}$ such that both $\mathcal{P}_{\mathcal{A}} \cup \mathcal{H} \cup \mathcal{C}_{\mathcal{A}} \cup \mathcal{C}_{\mathcal{M}} \models r$ and $\mathcal{P}_{\mathcal{A}} \cup \mathcal{H} \cup \mathcal{C}_{\mathcal{A}} \cup \mathcal{C}_{\mathcal{M}} \not\models \bot$
   (c) if this set exists then $\mathcal{C}_{\mathcal{M}}$ is send back to the client and the process re-iterates.
   (d) if it does not exist then
      i. generalize the set of disclosable credentials to all credentials occurring in $\mathcal{P}_{\mathcal{A}}$
      ii. use adbuction to find a minimal set of missing credentials $\mathcal{C}_{\mathcal{M}}$ such that both $\mathcal{P}_{\mathcal{A}} \cup \mathcal{H} \cup \mathcal{C}_{\mathcal{M}} \models r$ and $\mathcal{P}_{\mathcal{A}} \cup \mathcal{H} \cup \mathcal{C}_{\mathcal{M}} \not\models \bot$
      iii. if no such set does exist then $\bot$ is sent back to the user,
      iv. if such set do exists then compute the set of revocable credentials $\mathcal{C}_{\mathcal{E}}$ as the set $\mathcal{C}_{\mathcal{A}} \setminus \mathcal{C}_{\mathcal{M}}$, return this set to the client and re-iterate the process

When the request is granted the appropriate grounding of suitable history predicates are added to $\mathcal{H}$.

Remark 2. The step 3a looks the opposite of abduction: rather than adding new information to derive more things (the request), we drop information to derive less things (the inconsistency). It is possible to show that by adding a number of rules linear in the number of potentially revocable credentials the two task are equivalent.

Theorem 1. *If an access control policy guarantees $\prec$-fair access and $\mathcal{H} = \emptyset$ the access control algorithm for stateful business processes never returns $\perp$.*

Theorem 2. *If access and interaction control policies guarantee $\prec$-fair interaction w.r.t. a set of credentials $\mathcal{C}_\mathcal{I}$ and $\mathcal{H} = \emptyset$ the access control algorithm for stateful business processes, there exists a sequence of revocable and missing credentials starting with $\mathcal{C}_\mathcal{I}$ such that the access control algorithm for stateful business processes eventually grant r.*

6 Conclusions

In this paper we proposed a logical framework for reasoning about access control for business processes for web services. Our formal model for reasoning on access control is based variants of Datalog with the stable model semantics and combines in a novel way a number of features: the logic for trust management by Li et al. [11]; the logic for workflow access control by Bertino et al. [2]; and the logic for controlling the release of information by Bonatti and Samarati [3]. We identified the different reasoning tasks (deduction, abduction, consistency checking) that characterize the problem and clarify the problems of temporal evolution of the logical model (updates and downdates of credentials).

References

1. Apt, K. Logic programming. In *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed. Elsevier, 1990.
2. Bertino, E., Ferrari, E., and Atluri, V. The specification and enforcement of authorization constraints in workflow management systems. *ACM Transactions on Information and System Security (TISSEC) 2*, 1 (1999), 65–104.
3. Bonatti, P., and Samarati, P. A unified framework for regulating access and information release on the Web. *Journal of Computer Security*. (to appear).
4. Das, S. *Deductive Databases and Logic Programming*. Addison-Wesley, Reading, MA, 1992.
5. di Vimercati, S. D. C., and Samarati, P. Access control in federated systems. In *Proceedings of the 1996 workshop on New security paradigms* (1996), ACM Press, pp. 87–99.
6. Eiter, T., Gottlob, G., and Leone, N. Abduction from logic programs: Semantics and complexity. *Theoretical Computer Science 189*, 1-2 (1997), 129–177.
7. Ellison, C., Frantz, B., Lampson, B., Rivest, R., Thomas, B. M., and Ylonen, T. *SPKI Certificate Theory*, September 1999. IEFT RFC 2693.
8. Gelfond, M., and Lifschitz, V. The stable model semantics for logic programming. In *Proceedings of the Fifth International Conference on Logic Programming (ICLP'88)* (1988), R. Kowalski and K. Bowen, Eds., MIT-Press, pp. 1070–1080.
9. Kang, M. H., Park, J. S., and Froscher, J. N. Access control mechanisms for inter-organizational workflow. In *Proceedings of the Sixth ACM Symposium on Access control models and technologies* (2001), ACM Press, pp. 66–74.
10. Koshutanski, H., and Massacci, F. An access control system for business processes for Web services. Tech. Rep. DIT-02-102, Department of Information and Communication Technology, University of Trento, 2002.

11. Li, N., Grosof, B. N., and Feigenbaum, J. Delegation logic: A logic-based approach to distributed authorization. *ACM Transactions on Information and System Security (TISSEC) 6*, 1 (2003), 128–171.
12. Shanahan, M. Prediction is deduction but explanation is abduction. In *Proceedings of IJCAI '89* (1989), Morgan Kaufmann, pp. 1055–1060.
13. Weeks, S. Understanding trust management systems. In *IEEE SS&P-2001* (2001).
14. Yu, T., Winslett, M., and Seamons, K. E. Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM Transactions on Information and System Security (TISSEC) 6*, 1 (2003), 1–42.