INTERACTIVE ACCESS CONTROL FOR 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 (BPEL4WS) are the new paradigms for lightweight enterprise integration. They cross organizational boundaries and are provided by entities that see each other just as business partners. Web services require shift in the access control mechanism: from identity-based access control to trust management and negotiation, but this is not enough for cross organizational business processes. For many businesses no partner may guess a priori what kind of credentials will be sent by clients and clients may not know a priori which credentials are required for completing a business process.

We propose a logical framework for reasoning about access control for BPEL4WS and a BPEL4WS based implementation using Collaxa server. Our model is based on interaction and exchange of requests for supplying or declining missing credentials. We identify the formal reasoning services (deduction, abduction, consistency checking) that characterise the problem and discuss their implementation.

Keywords: Web Services; Business Processes; Credential-Based Systems; Interactive Access Control; Internet Computing; Logics for Access Control

1. Introduction

In the past millennium the development of middleware marked influenced the IT sector efforts to integrate distributed resources of a corporation. The new century has seen the rise of a new concept: virtual enterprises, the result of the outsourcing trend of the last 10 years in the IT sector.

*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 and RBAU01P5SS Project.

Conceptually, a virtual enterprise is born when a business process is not longer closed within the boundary of a single corporation. It is composed by partners that offer their services on the web and lightly integrate their efforts into one (hopefully coherent) process.

To support the process of lightweight integration of partners' resources, a number of specifications and standards have emerged. SOAP and Web Services Description Language[1] (WSDL) help organisations in exposing their basic functionalities as Web Services. Business Process Execution Language[2] (BPEL4WS) and Electronic Business XML initiative[3] (ebXML) describe the behavior of complex business processes.

Intuitively, business processes are hierarchical graphs where each composite node represents an orchestration activity and primitive nodes are Web Services interfaces described in WSDL.

Considering the nature of a virtual enterprise – orchestration and choreography of WS, global and local business processes, complex business transactions – the picture gets complicated. Distributed processes, in a virtual enterprise, become more dynamic, allowing new partners and services to be selected at runtime.

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

From the trust management systems (see e.g. [18, 8, 15]) 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. In contrast with trust management system, we have a process and thus a notion of assignment of permissions to credentials that requires to look beyond the single access decision.

From workflow access control systems (see e.g. [2, 3, 10, 11]) we borrow all classical problems such as dynamic assignment of roles to users, dynamic separation of duties, and assignment of permissions to users according 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 see each other as partners and nothing else. We have something even more loosely coupled than federated databases.

Also, we can no longer assume that an enterprise will assign tasks and roles to users (its employees) in a way that makes the completion of

[1] WSDL–http://www.w3.org/TR/wsdl
[2] BPEL4WS–http://www-106.ibm.com/developerworks/webservices/library/ws-bpel
[3] ebXML Business Process Spec. – www.ebxml.org/specs/ebBPSS.pdf

the workflow possible w.r.t. its security constraints. The reason is that such enterprise no longer exists. So, it must be possible for a user to communicate missing credentials.

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.

2. System Architecture

In this section we sketch the architecture of a system for distributed access control for Web and Business Processes that we have implemented. We refer to [12] for additional information on the rationale behind the architecture. At the time of writing we have done an initial prototype including the main entities of the system given below.

PolicyEvaluator makes endpoint decisions on access control. All partners involved in a business process are likely to be as different entities, each represented by a PolicyEvaluator. It encapsulates the partner's specific authorization policy, and presents it as a service using standardized WS interface (e.g., WSDL).

PolicyOrchestrator is an entity responsible for the workflow level access and release control. It decides which are the partners that are involved in the requested service and on the basis of some orchestration security policies combines the corresponding PolicyEvaluators in a form of a business process that is suitable for execution by the AuthorizationServer.

AuthorizationServer is responsible for *locating*, *executing*, and *managing* all needed PolicyEvaluators, and returning an appropriate result to the ApplicationServer. Also it is responsible for managing all the *interactions* with the Client.

At the application level, the architecture does not envisage the typical exchange of messages in access control system: "data" level (credentials, policies, requests, objects, etc.) that must be interpreted by the recipients. We can exchange messages at "source code" level and in particular at the level of business process description. Partners exchanges "mobile" processes (namely BPEL files) passing from one entity to another indicating themselves what the recipient has to do.

The mobility of authorization processes has a number of advantages. First of all, a server simply needs an off-the-shelf interpreter for business

processes for a quick implementation. Second, we have more flexibility for describing the process leading to an access control decision.

To say few words on the implementation, Collaxa[5] is used as a main BPEL manager (on the AuthorizationServer side) for executing and managing all policy composition processes returned by the PolicyOrchestrator, as well as, for the implementation of the of the AuthorizationServer itself.

The AuthorizationServer itself is a BPEL process deployed under Collaxa that internally deploys the policy process returned by the PolicyOrchestrator as an internal web service and then also internally executes it. The advantage in this case is that if the AuthorizationServer is requested to get an access decision for a service that has already been asked for it and there is no change in the workflow policy then the AuthorizationServer *does not* deploy the service's policy process again but just (internally) executes it. In that way we speed up the access decision time.

PolicyOrchestrator in the current prototype is just a mapping between a service resource and its workflow policy process. We assume that the the process is already created by some GUI (e.g., could be used any BPEL visual tool generator that actually connects all involved partners' PEs in a BPEL process) and is available to the orchestrator.

PolicyEvaluator is another key point in our system. In its core, it is a Java module that acts as a wrapper for the DLV[6] system and implements our interactive algorithm for stateless WS described in Section 6.

3. 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

- logic for trust management by Li et al. [15];
- logic for workflow access control by Bertino et al. [3];
- logic for release and access control by Bonatti and Samarati [4].

We consider the view of a single partner since we cannot assume sharing of policies between partners. In [12] 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.

[5]Collaxa BPEL Server – www.collaxa.com
[6]DLV System – www.dlvsystem.com

In our framework each partner has a *security policy for access control* $\mathcal{P}_{\mathcal{A}}$ and a *security policy for disclosure control* $\mathcal{P}_{\mathcal{D}}$, whose syntax will be defined later in section 5.

The policy for access control is used for making decision about usage of all web services offered by the partner. We will use abduction to infer the missing credentials from the access policy and the credentials already presented by the user. The disclosure policy is used (as the name implies) for controlling disclosure of credentials. Basically, we ask the client only the missing credentials that are disclosable according to $\mathcal{P}_{\mathcal{D}}$.

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

For the syntax we build upon [3, 4, 15]. We have three disjoint sets of constants: one for users identifiers denoted by $\mathsf{User}:U$; one for roles denoted by $\mathsf{Role}:R$; and one for services denoted by $\mathsf{WebServ}:S$.

The predicates can be divided into three classes: predicates for assignments of users to roles and services (Fig. 1a), predicates for credentials (Fig. 1b), and predicates describing the current status of the system. The last class of predicates keeps track on the main activities done by users and services, such as: a predicate specifying successful activation of services by users; a predicate for successful completion of services; its dual one for abortion; predicates indicating granting a service to a user and, the opposite one, denial user's access to a service.

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 Niemela smodels system, but we are substantially using the count predicates defined by Das [6]:

$n \leq \{X.\, Pr\}$ 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.

We assume additional comparison predicates (for instance for equality or inequalities) or some additional monadic predicates for instance to qualify service, users, or keys for credentials.

4. Logic Programming Background

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

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

$\mathsf{Role}:R_i \succ \mathsf{Role}:R_j$ when role $\mathsf{Role}:R_i$ dominates role $\mathsf{Role}:R_j$.

$\mathsf{Role}:R_i \succ_{\mathsf{WebServ}:S} \mathsf{Role}:R_j$ when role $\mathsf{Role}:R_i$ dominates, just for service $\mathsf{WebServ}:S$, the role $\mathsf{Role}:R_j$.

$\mathsf{assign}\,(P, \mathsf{WebServ}:S)$ when an access to the service $\mathsf{WebServ}:S$ is granted to P. P can be either a $\mathsf{Role}:R$ or $\mathsf{User}:U$.

$\mathsf{forced}\,(P, \mathsf{WebServ}:S)$ when access the service $\mathsf{WebServ}:S$ must be forced to P. Principal P can be either a $\mathsf{Role}:R$ or $\mathsf{User}:U$.

(a) Predicates for assignments to Roles and Services

$\mathsf{declaration}\,(\mathsf{User}:U)$ it is a statement by the $\mathsf{User}:U$ for its identity.

$\mathsf{credential}\,(\mathsf{User}:U, \mathsf{Role}:R)$ when $\mathsf{User}:U$ has a credential activating $\mathsf{Role}:R$.

$\mathsf{credentialTask}\,(\mathsf{User}:U, \mathsf{WebServ}:S)$ when $\mathsf{User}:U$ has the right to access $\mathsf{WebServ}:S$.

(b) Predicates for Credentials

Figure 1. Predicates used in the model

where A, B_i and C_i are (possibly ground) predicates among those described in Section 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. If the body is empty the rule is called a *fact*. A normal logic program is a set of rules.

In our framework, we also need *constraints* that are rules with an empty head.

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

One of the most prominent semantics for normal logic programs is the *stable model semantics* proposed by Gelfond and Lifschitz [9] (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, 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.

We now consider ground rules, i.e. rules where atoms do not contain variables.

DEFINITION 1 *The* reduct P^S *of a ground logic program P with respect to a set of atoms S is the definite program obtained from P by deleting:*

1 each rule that has a negative literal $not\, C$ *in its body with* $C \in S$*;*

2 each negative literal in the bodies of the remaining rules.

The reduct P^S is a definite logic program. Let $M(P^S) = M_{PS}$ be the semantics of the definite logic program P^S, i.e. its minimal model.

DEFINITION 2 *A set of atoms* S *is a stable model of a normal logic program* P *iff* $S = M(P^S)$.

A program can have none, one or many stable models. The definition of stable models captures the two key properties of solution sets of logic programs.

1 Stable models are minimal: a proper subset of a stable model is not a stable model.

2 Stable models are grounded: each atom in a stable model has a justification in terms of the program, i.e. it is derivable from the reduct of the program with respect to the model.

Though this definition of stable models in terms of fix points is non-constructive there are constructive definitions [1] and systems [17, 14] that can cope with ground programs having tens of thousands of rules.

Logic programs with variables can be given a semantics in terms of stable models.

DEFINITION 3 *The stable models of a normal logic program* P *with variables are those of its ground instantiation* P_H *with respect to its Herbrand universe*[8].

If logic programs are function free, then an upper bound on the number of instantiations is rc^v, where r is the number of rules, c the number of the constants, and v the upper bound on the number of distinct variables in each rule.

DEFINITION 4 (LOGICAL CONSEQUENCE AND CONSISTENCY) *Let* P *be a logic program and* L *be a (positive or negative) ground literal.* L *is a* logical consequence *of* P *(*$P \models L$*) if* L *is true in every stable model of* P*.* P *is* consistent *(*$P \not\models \bot$*) if there is a stable model for* P*.*

DEFINITION 5 (ABDUCTION) *Let* P *be a logic program,* H *a set of predicates (called hypothesis, or abducibles),* L *a (positive or negative) ground*

[8] Essentially, we take all constants and functions appearing in the program and combine them in all possible ways. This yields the Herbrand universe. Those terms are then used to replace variables in all possible ways thus building its ground instantiation.

literal, 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 conditions above*

Traditional p.o.s are subset containment or set cardinality. Other solutions are possible with orderings over predicates.

5. The Logical Model

In this section we define the semantics of our logical model and give formal definitions of the security policies introduced in Section 3.

DEFINITION 6 *An* access control policy $\mathcal{P}_\mathcal{A}$ *is a logic program over the predicates defined in Section 3 in which (i) no credential and no execution atom can occur in the head of a rule, (ii) role hierarchy atoms occur as facts, (iii) for every rule containing a 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 request is a ground instance of an $\mathsf{assign}(\mathsf{User}:U, \mathsf{WebServ}:S)$ *predicate.*

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

In contrast to the proposal by Bertino et al. [3] for workflows we don't need any special rule for determining which services cannot be executed and which services must be executed by a specific user or role. The $\mathsf{forced}(,)$ predicate and the constraints guarantee the same result.

EXAMPLE 7 *Consider a security policy in which having a credential for the role accountant is incompatible with the assignment of any role manager, and that the execution of a service phoneCall from user billG requires that the service answer must be executed by anybody having the role headOfStaff. The following rules guarantees the desired behavior:*

$$\leftarrow \mathsf{credential}(\mathsf{User}:U, \mathsf{Role}:accountant), \mathsf{assign}(\mathsf{User}:U, \mathsf{Role}:manager).$$
$$\mathsf{forced}(\mathsf{Role}:headOfStaff, \mathsf{WebServ}:answer) \leftarrow$$
$$\mathsf{running}(\mathsf{User}:billG, \mathsf{WebServ}:call, \mathsf{number}:N). \quad \blacksquare$$

EXAMPLE 8 *Consider an e-stock portal where we have roles associated to services as follows: role* eSeller *– for selling shares and bonds on the*

floor; role eBuyer *– for buying shares and bonds; role* eAdvisor *– used by accredited consultants to sell their advice to other customers of the portal. Then examine the case where one could send the* eAdvisor *credential to the service publishing advisories and suggest to sell shares, and at the same time the* eBuyer *credential to the service hosting bids.*
In such situations we can define separation of duty rules:

$customer(\mathsf{eSeller}) \leftarrow .$
$customer(\mathsf{eBuyer}) \leftarrow .$
$\leftarrow \mathsf{assign}\,(\mathsf{User}:U, \mathsf{Role}:R_1), customer(R_1), \mathsf{assign}\,(\mathsf{User}:U, \mathsf{Role}:\mathsf{eAdvisor}).$

The access control rule on reviewing selling bids is the following:

$$\mathsf{assign}\,(\mathsf{User}:U, \mathsf{WebServ}:S) \leftarrow \mathsf{credential}\,(\mathsf{User}:U, \mathsf{Role}:R), \mathsf{assign}\,(\mathsf{Role}:R, \mathsf{WebServ}:S).$$
$$\mathsf{assign}\,(\mathsf{Role}:R, \mathsf{WebServ}:reviewSell) \leftarrow \mathsf{Role}:R \succ \mathsf{Role}:\mathsf{eSeller}.\blacksquare$$

As mentioned, we will use the disclosure policy $\mathcal{P}_\mathcal{D}$ to decide which missing credentials are to be asked from the client.

DEFINITION 9 *A* disclosure policy $\mathcal{P}_\mathcal{D}$ *is a logic program in which no role hierarchy atom and no execution atom can occur in the head of a rule.*

DEFINITION 10 *A credential c is disclosable if it is a logical consequence of the disclosure policy* $\mathcal{P}_\mathcal{D}$ *and presented credentials* $\mathcal{C}_\mathcal{P}$ *(*$\mathcal{P}_\mathcal{D} \cup \mathcal{C}_\mathcal{P} \models c$*).*

EXAMPLE 11 *Considering again the access policy in Example 8. A possible (part of) the disclosure policy* $\mathcal{P}_\mathcal{D}$ *could be:*

$\mathsf{credential}\,(\mathsf{User}:U, \mathsf{Role}:eUser) \leftarrow \mathsf{declaration}\,(\mathsf{User}:U).$
$\mathsf{credential}\,(\mathsf{User}:U, \mathsf{Role}:eSeller) \leftarrow \mathsf{credential}\,(\mathsf{User}:U, \mathsf{Role}:eUser).$
$\mathsf{credential}\,(\mathsf{User}:U, \mathsf{Role}:eSellerVIP) \leftarrow \mathsf{credential}\,(\mathsf{User}:U, \mathsf{Role}:eSeller).$

The second rule says: to reveal the need for a eSeller *credential there should be already a credential attesting the client as a valid user of the system together with a declaration of its identity.* ■

So, the request $\mathsf{assign}\,(\mathsf{User}:fm, \mathsf{WebServ}:reviewSell)$ together with $\mathsf{credential}\,(\mathsf{User}:fm, \mathsf{Role}:eUser)$ and $\mathsf{declaration}\,(\mathsf{User}:fm)$ will yield a counter request – $\mathsf{credential}\,(\mathsf{User}:fm, \mathsf{Role}:eSeller)$ – specifying the need for additional privileges necessitated to get the service.

Note that the need for a credential attesting the role *eSellerVIP*, disclosed together with *eSeller*, should not be considered as a potential output by the system because the "intuition" says that *eSeller* is enough.

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 the extraction of the minimal set of security credentials is not guaranteed by the straightforward interpretation of H (abducibles) as the set of credentials and by the set cardinality or set containment orderings.*

Consider the following program:

$$\begin{aligned}&\mathsf{Role}:r_2 \succ \mathsf{Role}:r_1 \leftarrow .\\ &\mathsf{assign}\,(\mathsf{User}:U, \mathsf{WebServ}:ws) \leftarrow \mathsf{credential}\,(\mathsf{User}:U, \mathsf{Role}:R),\\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{Role}:R \succ \mathsf{Role}:r_1.\end{aligned}$$

Request $\mathsf{assign}\,(\mathsf{User}:fm, \mathsf{WebServ}:ws)$ has two $\subseteq$-minimal solutions:

$$\{\mathsf{credential}\,(\mathsf{User}:fm, \mathsf{Role}:r_1)\}, \{\mathsf{credential}\,(\mathsf{User}:fm, \mathsf{Role}:r_2)\}$$

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.

Another alternative, currently implemented in out prototype, is to include extra information to credentials in the hypotheses (abducibles), specifying the position of a role in the role-lattice hierarchy. Then it is easy to select the set(s) with the lowest role-position values. After having obtained the missing credentials, we drop this extra information from the set that is to be sent back to the client.

DEFINITION 12 (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 policy $\mathcal{P}_\mathcal{A}$ 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 13 (FAIR INTERACTION) *Let $\mathcal{P}_\mathcal{A}$ and $\mathcal{P}_\mathcal{D}$ be, respectively, an access and disclosure control policies, 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 initial 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 $\mathcal{P}_\mathcal{D} \cup \mathcal{C}_\mathcal{I} \models c$. If the set $\mathcal{C}_\mathcal{I}$ only contains declarations then the disclosure is* unlimited.

The above process does not take into account the progressive disclosure of credentials in the interactive process.

REMARK 2 *It is possible to define a process of trust negotiation along the lines of Yu et al. [19] if at each interaction step we ask only for the credentials that are entailed by a 1-step deduction over $\mathcal{P}_\mathcal{D}$ and $\mathcal{C}_\mathcal{P}$. In this case, the interaction policy must be a* monotonic *logic program.*

6. Reasoning

In this section we show how the various notions that we have seen so far can be combined into a complete authorization mechanism. PolicyEvaluator receives the request r, processes it according to the access control algorithm and eventually takes a decision. A decision may have involved interactions and so we also keep track of the current set of active credentials and the history of the requests made by the client.

Since the client must collect all relevant credentials (if required) for getting access to a service, one could borrow mechanisms for discovering distributed credentials' chains from [16, 5].

Once again it is worth noting that this view is partial as we only focus on the knowledge of one single partner: there is no authorization domain crossing partnerships.

To allow for an easier grasp of the problem, we start with a basic framework shown in Figure 2. This approach is the cornerstone of most logical formalizations [7].

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

Figure 2. Traditional Access Control

A number of works has deemed such blunt denials unsatisfactory and therefore it has been proposed by Bonatti and Samarati [4] and Yu et al. [19] to send back to the client some of the rules that are necessary to gain additional access (see Figure 3). In their work it is revised to allow for the flow of rules and information to users.

Since the systems proposed by both Bonatti and Samarati [4] and Yu et al. [19] are flat, in p the client will find all missing credentials to continue the process until r is granted.

In many cases, this is neither sufficient nor desirable. For instance, if the policy is not flat, it has constraints on the credentials that can

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 $r \leftarrow p \in \texttt{PartialEvaluation}(\mathcal{P}_{\mathcal{A}} \cup \mathcal{C}_{\mathcal{P}})$ and send the rule back to the client

Figure 3. Disclosable Access Control

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[11].

Another point in our formal model, worth discussing here, is the way we address the disjunctive information in the partner's disclosure policy (e.g., "present either a VISA or a Mastercard or an American Express card"). In presence of such disjunctive information an arbitrary disjunct will be selected and on the next interaction step (if the client has declined the credential) the abduction algorithm is informed that the previous solution was not accepted. We approach this by discarding the set of declined credentials from the set of newly computed disclosable credentials. In this case the abduction algorithm does not consider the declined credentials, from the last step, in the next interaction step.

Our interactive access control solution for Web Services is shown in Figure 4.

This is all we need for business processes made up by *stateless web services*, in which all decisions are taken on the basis of the current input set of credentials, and which envisaged to be the large majority.

This type of decision is characteristic of most logical approaches to access control [15, 3, 4]: we only look at the policy, the request and the set of credentials. The failure of the access control process at step *3c* (Fig. 4) may be due to the presence of badly designed constraints for separation of duties such that no possible set of credentials can unlock the service r. In same cases this might also be a feature of the systems.

[11] In the negotiation process of Yu et al. [19] rules are only disclosed when all preliminary credentials have been already sent by the client. Still this is unsatisfactory because we may well want to tell a user all credentials we may possibly ask him, but not how we are going to evaluate them.

1 extract from the client's input the set of presented credentials $\mathcal{C}_\mathcal{P}$ and the set of declined credentials $\mathcal{C}_\mathcal{N}$

2 verify that the request is a logical consequence of the credentials, namely $\mathcal{P}_\mathcal{A} \cup \mathcal{C}_\mathcal{P} \models r$

3 if the check succeeds then access is granted, otherwise

(a) compute the set of *disclosable credentials* $\mathcal{C}_\mathcal{D}$ as $\mathcal{C}_\mathcal{D} = \{c \mid c \text{ credential that } \mathcal{P}_\mathcal{D} \cup \mathcal{C}_\mathcal{P} \models c\} \setminus \mathcal{C}_\mathcal{N}$

(b) use abduction to find a minimal set of missing credentials $\mathcal{C}_\mathcal{M} \subseteq \mathcal{C}_\mathcal{D}$ such that both $\mathcal{P}_\mathcal{A} \cup \mathcal{C}_\mathcal{P} \cup \mathcal{C}_\mathcal{M} \models r$ and $\mathcal{P}_\mathcal{A} \cup \mathcal{C}_\mathcal{P} \cup \mathcal{C}_\mathcal{M} \not\models \bot$

(c) if no such set exists then $\bot$ is sent back to the user,

(d) otherwise communicate the set of missing credentials $\mathcal{C}_\mathcal{M}$ back to the client and iterate the process.

Figure 4. Interactive Access Control for Stateless WS

7. Implementation of the Logical Model

For the implementation of the algorithm in Figure 4 we used DLV (a disjunctive datalog system with negations and constraints) as a core engine for the basic functionalities of deduction and abduction. The disjunctive datalog front-end (the default one) is used for deductive computations while the diagnosis front-end is used for abductive computations. We refer to Section 5 for definitions of deduction and abduction.

What follows is a step-by-step description of the implementation employing the DLV system:

1 Extract from the client's input the two sets of credentials $\mathcal{C}_\mathcal{P}$ and $\mathcal{C}_\mathcal{N}$, transform them to predicates suitable for the underlying formal model (ref. Fig. 1) and store them in temporary files;

2 Use the DLV's disjunctive datalog front-end. Specify as input the partner's access policy, the two sets from step 1 and the service request r marked as a query over the models computed by DLV. The output of this step are those models of the access policy in which r is true.

3 If it exists a model in step 2 that satisfies r then grant, otherwise:

(a) use again the DLV's front-end as input partner's disclosure policy $\mathcal{P}_\mathcal{D}$ together with presented credentials $\mathcal{C}_\mathcal{P}$. In this case DLV computes all models of $\mathcal{P}_\mathcal{D}$ that are disclosable by $\mathcal{C}_\mathcal{P}$.

Then from the computed models we remove all credentials that belongs to $\mathcal{C}_{\mathcal{N}}$.

(b) find a model, out of the ones in step 3a, for which it exists a subset satisfying the abductive computation described below. Use the abductive diagnosis (subset minimal) front-end with the following input: $\mathcal{P}_{\mathcal{A}}$, $\mathcal{C}_{\mathcal{P}}$, the set of credentials from the model being checked stored in a temporary file with special extension .hyp (called hypotheses or abducibles) and the service request r also stored in a temporary file with extension .obs (observations). The output of such computation are all possible subsets of the hypotheses that satisfy the observations. In that way we find all possible missing sets of credentials satisfying r. Then we filter them, first against role-minimality criterion, and then against set cardinality criterion. The former filters those sets with lowest possible role-position values and the latter filters the ones with minimal cardinality.

(c) if no such set exists reject otherwise send the missing set back to the client.

REMARK 3 *The sequence, the two criteria,* set cardinality *and* role minimality *makes sense in different contexts. The sequence* role minimality/set cardinality*, tries to keep the minimal set as lower in the role hierarchy as possible, i.e. selects those sets that have a larger number of not so powerful roles. The other alternative,* set cardinality/role minimality*, selects those sets with fewer roles but with higher privileges.*

The latter may be useful if getting or transmitting credentials is expensive (e.g., in a mobile setting).

8. Stateful Business Processes

If the authorization decisions of business processes are stateful, and the corresponding workflow of the partners has constraints on the execution of future services on the basis of past services this solution is not adequate enough. For instance in the workflow example described by Atluri and Bertino [3, pag.67] a branch manager of a bank clearing a cheque cannot be the same member of staff who has emitted the cheque. 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;

- in contrast to intra-enterprise workflow systems [3], the partner offering the web service has no way to assign to the client the right set of credentials for consistency future request.

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. A preliminary solution has been described in [13].

9. Conclusions

In this paper we proposed a logical framework for reasoning about access control for stateless business processes for web services. Our formal model for reasoning on access control is based on 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. [15]; the logic for workflow access control by Bertino et al. [3]; the logic for controlling the release of information by Bonatti and Samarati [4].

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.

Future work is in the direction of more effective trust negotiation for stateful business processes.

References

[1] Apt, K. Logic programming. In *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed. Elsevier, 1990.

[2] Atluri, V., Chun, S. A., and Mazzoleni, P. A Chinese wall security model for decentralized workflow systems. In *Proceedings of the 8th ACM conference on Computer and Communications Security* (2001), ACM Press, pp. 48–57.

[3] 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.

[4] Bonatti, P., and Samarati, P. A unified framework for regulating access and information release on the web. *Journal of Computer Security 10*, 3 (2002), 241–272.

[5] Clarke, D., Elien, J.-E., Ellison, C., Fredette, M., Morcos, A., and Rivest, R. L. Certificate chain discovery in SPKI/SDSI. *Journal of Computer Security 9*, 4 (2001), 285–322.

[6] Das, S. *Deductive Databases and Logic Programming*. Addison-Wesley, Reading, MA, 1992.

[7] De Capitani di Vimercati, S., and Samarati, P. Access control: Policies, models, and mechanism. In *Foundations of Security Analysis and Design - Tutorial Lectures*, R. Focardi and F. Gorrieri, Eds., vol. 2171 of *LNCS*. Springer Verlag Press, 2001.

[8] Ellison, C., Frantz, B., Lampson, B., Rivest, R., Thomas, B. M., and Ylonen, T. *SPKI Certificate Theory*, September 1999. IETF RFC 2693.

[9] 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.

[10] Georgakopoulos, D., Hornick, M. F., and Sheth, A. P. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases 3*, 2 (April 1995), 119–153.

[11] 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.

[12] Koshutanski, H., and Massacci, F. An access control framework for business processes for Web services. In *Proceedings of the 2003 ACM workshop on XML security* (Fairfax, VA, October 2003), ACM Press.

[13] Koshutanski, H., and Massacci, F. A logical model for security of Web services. Tech. Rep. IIT TR-10/2003, First International Workshop on Formal Aspects of Security and Trust (FAST), Istituto di Informatica e Telematica, Pisa, Italy, September 2003. Editors: Theo Dimitrakos and Fabio Martinelli.

[14] Leone, N., Pfeifer, G., and et al. The DLV system. In *the 8th European Conference on Artificial Intelligence (JELIA)* (September 2002), vol. 2424 of *Lecture Notes in Computer Science*, Springer, pp. 537–540.

[15] 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.

[16] Li, N., Winsborough, W. H., and Mitchell, J. C. Distributed credential chain discovery in trust management. *Journal of Computer Security 11*, 1 (February 2003), 35–86.

[17] Niemelä, I., Simons, P., and Soininen, T. Stable model semantics of weight constraint rules. In *Proceedings of the Fifth International Conference on Logic Programming and Nonmonotonic Reasoning* (December 1999), Springer-Verlag.

[18] Weeks, S. Understanding trust management systems. In *IEEE SS&P-2001* (2001), IEEE Press.

[19] 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.