AN INTERACTIVE TRUST MANAGEMENT AND NEGOTIATION SCHEME*

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 Interactive access control allows a server to compute on the fly missing credentials needed to grant access and to adapt its responses on the basis of client's presented and declined credentials. Yet, it may disclose too much information on *what credentials a client needs*. Automated trust negotiation allows for a controlled disclosure on *what credentials a client has* during a mutual disclosure process. Yet, it requires pre-arranged policies and sophisticated strategies. How do we bootstrap from simple security policies a comprehensive interactive trust management and negotiation scheme that combines the best of both worlds without their limitations? This is the subject of the paper.

Keywords: Trust Management; Trust Negotiation; Interactive Trust Management; Interactive Access Control; Credential-Based Systems; Internet Computing; Logics for Access Control;

1. Introduction

The new business hype of the moment – virtual organizations based on Web Services [1] – is particularly challenging for security research in access control. In a nutshell, the idea is to orchestrate into a coherent business process the Web Services (WS for short) offered by different partners. The functional orchestration is not trivial but the orchestration of security policies of partners even less, even if we take for granted the usage of Trust Management systems [3, 2].

First, the client may have no idea on the right set of credentials that have to be presented to each partner and the process may bring different

*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.

partners on the forefront depending on the actual business execution path. So, business partners must have a way to find out what credentials are required (missing) for clients to get access to their resources. Second, the client, once asked for the missing credentials, may be unwilling to disclose them unless the server discloses some of its credentials first, i.e. negotiates the need to disclose his own credentials.

Solution for the first problem has been proposed by Koshutanski and Massacci [7, 8]: interactive access control. Assuming a logical formalization (actually a rule-based policy is enough) and using some advanced inference services, it is possible for the server to compute the missing credentials on the fly. Credentials that may not be straightforwardly deduced from the security policy, as approached by the trust negotitation paradigm, but may require a more sophisticated reasoning service.

Solution for the second problem is trust negotiation, for instance as advocated by Winslett et al. [11]. Here, we can structure a security policy to specify what credentials a client must have already shown to get access to our own credentials, i.e. we specify the sequence of disclosable credentials that gradually establish trust.

Notice that the two problems are related but different. For sake of example consider the view point of a server. In the first one, we help the server to compute the *missing set* of foreign credentials that a client needs to get access to a service. The second approach helps the server, in response to some counter requests, to control the disclosure of its own credentials by stipulating what *foreign credentials* a client must supply to get access to the server's local ones.

Both approaches in their core have limitations: the first approach does not allow for a piecemeal disclosure to the clients of what they eventually need. The second one requires a sophisticated and rigid structuring of policies to work.

1.1 The Contribution of this Paper

If we merge the two frameworks we have the following problems:

1 Alice wants to access some service of Bob

2 Alice does not know exactly what credentials Bob needs, so

   (a) Bob must compute what is missing and ask Alice,
   (b) Alice must send to Bob all credentials he requested.

3 In response to 2b, Alice may want to have some credentials from Bob before sending hers, so

   (a) She must tell Bob what he needs to provide,

(b) Bob must have a policy to decide how access to his credentials is granted.

4 In response to 2a, Bob may not want to disclose all that is missing at once but may want to ask Alice first some of the less sensitive credentials, so

(a) Bob must have a way to request in a piecewise fashion the missing credentials.

Here we combine the best of the both worlds under the limited assumptions that we have just three policies:

(i) a policy for determining the credentials needed by a client to get access to a service,

(ii) a policy for determining the credentials needed by a client to access (see) server's credentials,

(iii) a policy for specifying what credentials are disclosable whose need can be potentially demanded from a client.

The policies can be arbitrarily complex with almost everything that is on the (Datalog for) Access Control market (say with negation as failure, constraints on separation of duties, or other fancy credentials such as those by Li et al. [9]). We only need deduction and its sister abduction[1]

Out of these two services we have constructed an algorithm that first evaluates a client's request by checking whether the client can access the requested service – using policy (i). If the client is not enough trusted (i.e. he does not have enough credentials), the algorithm computes a (minimal, trusted enough) set of missing credentials, from policies (i) and (iii), that unlocks the desired resource. Then it starts a negotiation process in which needed credentials are disclosed in a piecewise manner according to policy (iii) and requested credentials are disclosed according to policy (ii). The process continues until enough trust is established and the service is granted. In a negotiation process a client, itself, may also run the algorithm to control access to its own credentials.

2. Interactive Access Control for Web Services

In the framework introduced by Koshutanski and Massacci [7, 8] each partner has a *security policy for access control* $\mathcal{P}_{\mathcal{A}}$ and a *security policy for disclosure control* $\mathcal{P}_{\mathcal{D}}$. The policy for access control is used for making decision about usage of all web services offered by a partner. The

[1]Note that if the former is decidable within complexity class $\mathcal{C}$, the latter is decidable within complexity class $\Sigma^{\mathcal{C}}$ or at worst $\Pi^{\mathcal{C}}$ if minimality of abductive solutions is requested.

Role: $R_i \succ$ Role: R_j when role Role: R_i dominates role Role: R_j.

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

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

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

(a) Predicates for assignments to Roles and Services

declaration (User: U) it is a statement by the User: U for its identity.

credential (User: U, Role: R) when User: U has a credential activating Role: R.

credentialTask (User: U, WebServ: S) when User: U has the right to access WebServ: S.

(b) Predicates for Credentials

Figure 1. Predicates used in the model

policy for disclosure control is used to decide the credentials whose need can be potentially disclosed to a client.

To execute a service, under the control of a partner, a user will submit a *service request* r and a set of *credentials* $\mathcal{C}_r$. When the user sends the request r the server starts a negotiation session and creates a client's profile. The client's profile consists of two sets – the set of *presented credentials* $\mathcal{C}_\mathcal{P}$ and the set of *declined credentials* $\mathcal{C}_\mathcal{N}$. Both sets are kept up-to-date by the server as at each interaction step, $\mathcal{C}_\mathcal{P}$ is updated with the credentials the client currently sends, while $\mathcal{C}_\mathcal{N}$ is updated as a difference between the *missing credentials* $\mathcal{C}_\mathcal{M}$, the client was asked in the previous interaction, and the ones presented in the current step.

For the syntax we have three disjoint sets of constants: one for users identifiers denoted by User: U; one for roles denoted by Role: R; and one for services denoted by 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: successful activation of services by users; successful completion of services; abortion; etc. We refer to [8] for additional details on the model.

We note here that the model, presented in the this section, can be adapted to *any* generic policy framework. Since the information we

need from the underlying policy model, for our basic reasoning services, is shown in Figure 1 and that infromation can be found in (extracted from) most policy languages.

Below are the definitions of the basic reasoning services used in our formal framework.

DEFINITION 1 (LOGICAL CONSEQUENCE AND CONSISTENCY) *We use the symbol $P \models L$, where P is a policy and L is either a credential or a service request, to specify that L is a logical consequence of a policy P. P is* consistent *($P \not\models \bot$) if there is a model for P.*

DEFINITION 2 (1-STEP DEDUCTION) *We use the symbol $P \models_1 A$, where P is a policy with a predefined set of ground atoms $\mathscr{A}$ and A is a positive literal, if for some literals $L_1, \ldots, L_n$ holds the following:*
(i) $A \leftarrow L_1, \ldots, L_n$ *is in ground(P),*
(ii) $\mathscr{A} \models L_1, \wedge \ldots \wedge, L_n$.

DEFINITION 3 (ABDUCTION) *Abduction solution (see Fig. 2(b)) over a policy P, a set of predicates H (with a defined p.o. over subsets of H) and a ground literal L is a set of ground atoms E such that:*
(i) $E \subseteq H$,
(ii) $P \cup E \models L$,
(iii) $P \cup E \not\models \bot$,
(iv) any set $E' \prec E$ does not satisfy all conditions above.

Traditional p.o.s are subset containment or set cardinality.

The core of our interactive trust management protocol, introduced in the next section, is shown in Figure 2. The basic computations of deduction (Def. 1) and abduction (Def. 3) are shown in Figure 2(b). The global variables $\mathcal{C}_\mathcal{N}$ and $\mathcal{C}_\mathcal{P}$ represent the client's profile (as described earlier). The protocol takes as input the request r and the partner's policies for access and disclosure control – $\mathcal{P}_\mathcal{A}$, $\mathcal{P}_\mathcal{D}$. The output is either *grant* r, or *deny* r, or *ask*($\mathcal{C}_\mathcal{M}$) – the set of missing credentials that the client needs to provide in order to get r.

The intuition behind the algorithm is the following. First (in step 1) it is checked whether the client's credentials $\mathcal{C}_\mathcal{P}$ are enough to get access to service r according to policy $\mathcal{P}_\mathcal{A}$. In the case of failure, the algorithm runs the abduction process (step 3) to compute what is missing (complement) to $\mathcal{C}_\mathcal{P}$ that unlocks r. A preliminary step to abduction is computing the set of disclosable and not declined credentials $\mathcal{C}_\mathcal{D}$ (step 2). The set $\mathcal{C}_\mathcal{D}$ stores those credentials that are disclosable by $\mathcal{P}_\mathcal{D}$ and $\mathcal{C}_\mathcal{P}$ and does not contain any credential of $\mathcal{C}_\mathcal{N}$. Then the abduction process computes all possible subsets of $\mathcal{C}_\mathcal{D}$ that are consistent with the access policy $\mathcal{P}_\mathcal{A}$ and, at the same time, grant r. Out of all these sets

Global vars: $\mathcal{C}_{\mathcal{N}}$, $\mathcal{C}_{\mathcal{P}}$;
Protocol input: r, $\mathcal{P}_{\mathcal{A}}$, $\mathcal{P}_{\mathcal{D}}$;
Protocol output: grant/deny/ask($\mathcal{C}_{\mathcal{M}}$);

`InteractiveAccessControl(`r, $\mathcal{P}_{\mathcal{A}}$, $\mathcal{P}_{\mathcal{D}}$`){`

1: if `doDeduction(`$r, \mathcal{P}_{\mathcal{A}} \cup \mathcal{C}_{\mathcal{P}}$`)` then return *grant*
2: else compute the set of *disclosable credentials* $\mathcal{C}_{\mathcal{D}} = \{c \mid \mathcal{P}_{\mathcal{D}} \cup \mathcal{C}_{\mathcal{P}} \models c\} \backslash \mathcal{C}_{\mathcal{N}}$;
3: *result* = `doAbduction(`r, $\mathcal{C}_{\mathcal{D}}$, $\mathcal{P}_{\mathcal{A}} \cup \mathcal{C}_{\mathcal{P}}$`)`;
4: if *result* $== \emptyset$ then return *deny*
5: else $\mathcal{C}_{\mathcal{M}}$ = *result* and return *ask*($\mathcal{C}_{\mathcal{M}}$);

`}`

(a) Interactive Access Control Algorithm

`doDeduction(`R: `Query,` P: `LogProgram){` // check for $P \models R$?

1: run DLV* in deduction mode with input: P, R? ;
2: check output: if R is deducible then return *true* else return *false*;

`}`
`doAbduction(`R: `Observation,` H: `Hypotheses,` P: `LogProgram){`

1: run DLV in abduction diagnosis mode with input: R, H, P ;
2: DLV output: all sets C_i that (i) $C_i \subseteq H$, (ii) $P \cup C_i \models R$, (iii) $P \cup C_i \not\models \bot$;
3: if no C_i exists then return $\bot$
4: else select a minimal C_{min} and return C_{min};

`}`

*DLV System - www.dlvsystem.com

(b) Basic Functionalities of Deduction and Abduction

Figure 2. Basic Trust Management Protocol

(if any) the algorithm selects the minimal one. Here we point out that the minimality criterion could be different for different contexts. We have identified two criteria: minimal set cardinality and role minimality (least privilege).

When the abduction is finished the protocol either returns *ask*($\mathcal{C}_{\mathcal{M}}$) or denies r if no $\mathcal{C}_{\mathcal{M}}$ was computed.

3. Automated Trust Negotiation

The main idea in a trust negotiation process is to gradually disclose sensitive credentials between negotiation participants until sufficient trust is established.

In Winslett et al framework [11, 12] a policy protects a resource, being it access to a service or disclosure of a credential, by stipulating what the requestor should satisfy to be authorized for that resource. They require, first, a policy to be monotonic – if a set of credentials unlocks a service also a superset unlocks it – and, second, for each resource there should be exactly one solution (set of statements) that unlocks it.

One can abstract from any policy language by wrapping it in a *policy compliance checker* module and treat it as a black box, encapsulating a decision engine for the underlying policy language. It accepts as an input a set of credentials and a policy and returns as output the subset of the credentials that satisfy the policy. However, the actually used policy language can be easily casted into a set of negation-free Datalog rules. Each alternative set of credentials that unlock a resource can be casted in a Datalog rule having a predicate corresponding to the resource in the head and the needed credentials in the body.

Winslett et al define a TrustBuilder *negotiation protocol* and, running on top of it, *families of strategies* that address the requirements and needs of each party to negotiate in a way best suited for it. The protocol defines message type and ordering while the strategy controls the content of negotiation messages. Both the negotiation protocol and the families of strategies are located in a *negotiation strategy* module – the TrustBuilder.

So, whenever two parties want to negotiate, they first choose (agree on) negotiation strategies that guarantee a successful interoperation and completion of the process. Once they chose the strategies, they run the TrustBuilder protocol.

Abstracting from the concrete strategy and family, in its essence, the relevant strategy for selecting the next set of credentials (message) is the following: for every credential relevant to the service request, if the credential is disclosable by the client's presented set of credentials it is added to the output else its policy (the part that protects the resource) is added instead. Then in the final output of the current strategy step the client can find the newly unlocked credentials together with policies for the others (not disclosed ones) that the client should satisfy in order to continue the process. The process continues with parties swapping roles until all requirements are satisfied and the resource is granted or a consensus was not reached by one of the parties and terminated.

4. Bootstrapping Trust Negotiation

To combine automated trust negotiation and interactive access control we only assume that both a client and a server have some logical security policies. In particular we assume available:

1 a policy for access to *own* resources $\mathcal{P}_{\mathcal{AR}}$ on the basis of *foreign* credentials,

2 a policy for access to *own* credentials $\mathcal{P}_{\mathcal{AC}}$ on the basis of *foreign* credentials,

3 a policy for disclosure the need of (missing) *foreign credentials* $\mathcal{P}_{\mathcal{D}}$.

Here $\mathcal{P}_{\mathcal{AR}}$ is a logic program over the predicates defined in Section 2 in which no credential and no execution atom can occur in the head of a rule and role hierarchy atoms occur as facts. Respectively, $\mathcal{P}_{\mathcal{AC}}$ and $\mathcal{P}_{\mathcal{D}}$ are logic programs in which no role hierarchy atom and no execution atom can occur in the head of a rule.

Technically speaking we could merge 1 and 2 into a flat policy for protecting sensitive resources. We believe that a structured approach is better because the criteria behind (and likely the administrator of) each policy are different. Resource access is decided by the business logic, whereas credential access is due to security and privacy considerations.

So, a client and server do not need to worry about interoperable strategies but can simply run the trust negotiation protocol shown in Fig. 3. The intuition behind the protocol is the following:

- A client, Alice, sends a service request r and (optionally) a set of credentials $\mathcal{C}_r$ to a server, Bob (steps 1 and 2).
- Then Bob looks at r and if it is a request for a service he calls `InteractiveAccessControl` with his policy for access to resources and his policy for disclosure of foreign credentials $< \mathcal{P}_{\mathcal{AR}}, \mathcal{P}_{\mathcal{D}} >$ (step 6) and we fall back in the case of Section 2.
- If r is a request for a credential then he calls `InteractiveAccessControl` with his policy for access to own credentials and again his policy for disclosure of foreign credentials $< \mathcal{P}_{\mathcal{AC}}, \mathcal{P}_{\mathcal{D}} >$ (step 9).
- In the case of computed missing credentials $\mathcal{C}_{\mathcal{M}}$ (in step 11), he transforms that into requests for credentials (askCredentials(...) function in Figure 3) and waits until receives all responses. At this point Bob acts as a client, requesting Alice the set of credentials $\mathcal{C}_{\mathcal{M}}$. Alice will run the same protocol swapping roles.
- When Bob's main process receives all responses it checks whether the missing credentials have been supplied by Alice (step 13).

```
Global vars: C_N, C_P:: initially set up to ∅ when the main process is started;
InteractiveTrustManagement(){ // runs in a new thread.
 1: r = receiveRequest();
 2: C_r = receiveCredentials();
 3: C_P = C_P ∪ C_r;
 4: repeat
 5:    if isService(r) then
 6:       result = InteractiveAccessControl(r, P_AR, P_D);
 8:    else // isCredential(r(c))
 9:       result = InteractiveAccessControl(r, P_AC, P_D);
11:    if result == ask(C_M) then
12:       askCredentials(C_M);
13:       if C_M ⊆ C_P then result = grant;
14: until result == grant or result == deny;
15: if result == grant and isCredential(r) then
16:    sendResponse(cred(r));
17: else
18:    sendResponse(result);
}
askCredentials(C_M){
 1: parfor each c ∈ C_M do
 2:    sendRequest(r(c));
 3:    if receiveResponse() == cred(c) then
 4:       C_P = C_P ∪ {c};
 5:       C_N = C_N \ {c};
 6:    else if c ∉ C_P then
 7:       C_N = C_N ∪ {c};
 8: done
}
```

Figure 3. Interactive Trust Negotiation Protocol

- If $\mathcal{C}_{\mathcal{M}}$ was not reached, Bob restarts the loop and consults the `InteractiveAccessControl` algorithm for a new decision.

- When a final decision is taken, a respective response (steps 16 and 18) is sent back to Alice.

The server initiates the main trust negotiation process when a client initially submits a request for a service. Then each counter request from the client side is run in a different thread that shares the same globally accessible client's profile ($\mathcal{C}_{\mathcal{P}}$, $\mathcal{C}_{\mathcal{N}}$) with other threads running under the same negotiation process.

Technicality in the protocol is in the way the server requests missing credentials back to the client. As indicated in the figure, we use the keyword `parfor` for representing that the body of the loop is run each time in a parallel thread under the thread that has computed $\mathcal{C}_{\mathcal{M}}$. At that point of the protocol, it is important that each of the finished threads updates the presented and declined set of credentials appropriately. So, we avoid the situations where some running parallel threads ask the client already asked credentials or already declined ones computed in other running threads under the same main process.

Also an important point here is to clarify the way we treat decliend and not yet released credentials. In a negotiaition process, declining a credential is when an entity is asked for it and the same entity replies to the same request an empty set (saying no). In the second case, when the entity is asked for a credential and, insted of reply, there is a counter request for more credentials, then the thread, started the original request, awaits the client for an explicit reply and treats the requested credential as not yet released. In any case, at the end of a (sub) negotiation process a client either supplies the originally asked credential or declines it.

The thread based implementation (with shared $\mathcal{C}_{\mathcal{P}}$ and $\mathcal{C}_{\mathcal{N}}$) is necessary to allow for a *polynomial execution time* of the trust negotiation protocol in the number of queries to the abduction algorithm. Indeed, without a shared memory for received credentials it is possible to structure the policies in a way that a credential will be asked far too many times. In this way the protocol queries to $\mathcal{P}_{\mathcal{AC}}$ are bounded by the number of credentials in the policy.

REMARK 1 *It can be proved that if policies are negation free then the algorithms on the client and server sides interoperate.*

It is possible to run the TrustBuilder by Yu et al. [12] on top of our mechanism so that our framework abstracts away the requirements on policies and strategies that should be imposed on the user's disclosure policy if using TrustBuilder directly.

However, we have not solved the problem of piecewise disclosure of missing foreign credentials yet. This turns out to be also possible as we shall see in the next section.

Global vars: $\mathcal{C}_\mathcal{N}$, $\mathcal{C}_\mathcal{P}$;
`InteractiveTrustManagement(){ ...`
}
`PiecewiseDisclosure(`$\mathcal{C}_\mathcal{M}$`,` $\mathcal{P}_\mathcal{D}$`){`

1: $\mathcal{C}_{\mathcal{D}1} = \{c \mid \mathcal{P}_\mathcal{D} \cup \mathcal{C}_\mathcal{P} \models_1 c\} \setminus \mathcal{C}_\mathcal{N}$;

2: $\mathcal{P}_{\mathcal{D}1} = \{\hat{c} \leftarrow B \mid c \leftarrow B \in \mathcal{P}_\mathcal{D}\} \cup$
$\{c \leftarrow \hat{c} \mid c \notin \mathcal{C}_{\mathcal{D}1}$ and $c \leftarrow B \in \mathcal{P}_\mathcal{D}$ for some $B\}$;

3: $Q = \left\{q \leftarrow \bigwedge_{c \in \mathcal{C}_\mathcal{M}} c\right\}$;

4: $\mathcal{C}_{\mathcal{M}1} =$ `doAbduction(`q`,` $\mathcal{C}_{\mathcal{D}1}$`,` $\mathcal{P}_{\mathcal{D}1} \cup \mathcal{C}_\mathcal{P}$`)`;

5: return $\mathcal{C}_{\mathcal{M}1}$;

}
`askCredentials(`$\mathcal{C}_\mathcal{M}$`,` $\mathcal{P}_\mathcal{D}$`){`

1: repeat

2: $\mathcal{C}_{\mathcal{M}1} =$ `PiecewiseDisclosure(`$\mathcal{C}_\mathcal{M}$`,` $\mathcal{P}_\mathcal{D}$`)`;

3: if $\mathcal{C}_{\mathcal{M}1} == \bot$ then return;

4: parfor each $c \in \mathcal{C}_{\mathcal{M}1}$ do

5: `sendRequest(`$r(c)$`)`;

6: if `receiveResponse() == cred(`c`)` then

7: $\mathcal{C}_\mathcal{P} = \mathcal{C}_\mathcal{P} \cup \{c\}$;

8: $\mathcal{C}_\mathcal{N} = \mathcal{C}_\mathcal{N} \setminus \{c\}$;

9: else if $c \notin \mathcal{C}_\mathcal{P}$ then

10: $\mathcal{C}_\mathcal{N} = \mathcal{C}_\mathcal{N} \cup \{c\}$;

11: done

12: $\mathcal{C}_\mathcal{M} = \mathcal{C}_\mathcal{M} \setminus \mathcal{C}_\mathcal{P}$;

13: until $\mathcal{C}_\mathcal{M} \subseteq \mathcal{C}_\mathcal{N}$;

}

Figure 4. Piecewise Trust Management Protocol

5. Controlled Disclosure of Missing Credentials

The intuition here is that Bob may not want to disclose the missing credentials all at once or directly to Alice. Instead he may want to ask Alice first some less sensitive credentials[6] assuring him that Alice is

[6]Here we point out that the stepwise approach may concern credentials that are not directly related to a specific resource but needed for a finer-grained disclosure control.

enough trustworthy to disclose her other credentials and so on continuing until the missing ones are disclosed.

To address this issue we extend the protocol in Section 4 with an algorithm for piecewise disclosure of missing credentials. The basic intuition is that the logical policy structure itself tells us which credentials must be disclosed to obtain the information that other credentials are missing. So, we simply need to extract this information automatically. We perform a step-by-step evaluation on the policy structure. For that purpose we use one step deduction (Def. 2) over the disclosure policy $\mathcal{P}_\mathcal{D}$ to determine the next set of potentially disclosable credentials.

Essentially, the protocol replaces the `askCredentials` function with a new version of it using the piecewise disclosure algorithm and adds the disclosure policy to its arguments, see Figure 3.

With its new version the `askCredentials` function (Figure 4) takes as input the set of missing credentials $\mathcal{C}_\mathcal{M}$ (as the old one) together with the policy for disclosure control $\mathcal{P}_\mathcal{D}$ that $\mathcal{C}_\mathcal{M}$ was computed from. In a nutshell, the algorithm requests the client all missing credentials supplied in the input, but with the difference of stepwise awaiting for each of the computed steps by the `PiecewiseDisclosure` algorithm. In other words, when a next step of missing credentials is computed (step 2) the algorithm waits until the client responds to all current requests. Again here the client's profile is updated on each request/response to facilitate other threads' access decisions. Then the check in step 3 for $\mathcal{C}_{\mathcal{M}1}$ comprises two cases: either the set of presented credentials $\mathcal{C}_\mathcal{P}$ has been updated (indirectly) by other running threads such that now $\mathcal{C}_\mathcal{M}$ is satisfied and there is no next step or the client has declined some credentials that stop his way further to $\mathcal{C}_\mathcal{M}$.

The task of the `PiecewiseDisclosure` is to determine at each interaction step exactly the relevant credentials that are needed to reach at the end the set $\mathcal{C}_\mathcal{M}$.

Basically, we compute the set of abducible credentials in one step as $\mathcal{C}_{\mathcal{D}1}$ (compare with the corresponding step 2 in Figure 2(a)). Out of those, we extract only the minimal set of credentials that is actually necessary to derive $\mathcal{C}_\mathcal{M}$. To this extent, we modify policy $\mathcal{P}_\mathcal{D}$ by adding a new atom q that can be derived if all (and only) $\mathcal{C}_\mathcal{M}$ credentials are derived. Additionally, we also change syntactically the structure of $\mathcal{P}_\mathcal{D}$ rule so that relevant credentials in $\mathcal{C}_{\mathcal{D}1}$ must be abduced and can no longer be derived from chaining other rules.

We do that by changing a rule of the from $c \leftarrow c_1, \ldots, c_n$ into a pair of rules $\hat{c} \leftarrow c_1, \ldots, c_n$ and $c \leftarrow \hat{c}$, where $\hat{c}$ is a new symbol. The informal meaning of the first rule is that c is disclosable if all c_i are. So, we now say that the need for the fictitious $\hat{c}$ is disclosable if the need for all c_i

is disclosable and that the need for credential c is disclosable if the need for c_i is.

Then if we remove the $c \leftarrow \hat{c}$ for all c in $\mathcal{C}_{\mathcal{D}_1}$ there will be no rule to infer that the need for c is disclosable so we must abduce c as a primitive atom (if it is actually needed to derive q, i.e. some of the $\mathcal{C}_{\mathcal{M}}$).

6. Implementation

For the implementation of the framework we have chosen Collaxa[8] manager. Collaxa server supports many standards as BPEL4WS, WSDL, SOAP, etc. and interoperates with platforms as BEA's WebLogic and Microsoft .NET. So, this makes it well-suited for the purposes of the framework. The main idea of the work is that using BPEL4WS specification [5] we can orchestrate the requirements and communications between client and partners in an automatic and transparent way via a main authorization server.

For the implementation of the algorithms and protocols, presented in the paper, we need at a lower level a suitable engine for the basic reasoning services of deduction and abductiom. For that purpose we have done a wrapper (a set of interfaces) to the DLV system that manages all internal computations, queries and transformations to and from the DLV's defined front-ends.

For the actual crypto infrastructure we decided to use PERMIS[9] [4]. We chose PERMIS because it implements RBAC using entirely X.509 Identity and Attribute Certificates [6]. It allows for creating, verifying and validating attribute certificates and for storing and allocating them using LDAP directories [10]. For the integration with PERMIS, we extend the PERMIS's Access Decision Function (ADF) with the functionality of our model such that PERMIS validates and gathers client's credentials on its own and then asks our algorithm for an access decision (next possible step) presenting the newly collected credentials.

7. Conclusions

In this paper we have proposed a framework for leveraging trust management and negotiation scheme between a client and a service provider in the WS world. We proposed a basic access control algorithm that evaluates a client's request with respect to a partner's policies and in the case of failure it computes what is necessary for the client to get the desired resource. Then we devised an interactive trust management

[8] www.collaxa.com | www.oracle.com/technology/bpel
[9] www.permis.org

protocol that communicates and negotiates the missing credentials in a piecewise manner until enough trust is established and the service is granted or the negotiation failed and the process was terminated.

The protocol can be run on both the client and server side so that they understand each other and automatically interoperate until a desired solution is reached or denied.

It is also possible to run the TrustBuilder by Yu et al. [12] on top of the protocol with the only requirement of transforming each time the protocol input/output to a syntax understandable by TrustBuilder.

One of the advantages in our approach is that we do not pose any restrictions on partner's policies since the basic computations performed on the policies are deduction and abduction, which do not require any specific policy structure.

References

[1] BENATALLAH, B., CASATI, F., AND TOUMANI, F. Web service conversation modeling: a cornerstone for e-business automation. *IEEE Internet Computing 8*, 1 (Jan/Feb 2004), 46–54.

[2] BLAZE, M., FEIGENBAUM, J., IOANNIDIS, J., AND KEROMYTIS, A. *The KeyNote Trust-Management System Version 2*, 1999. RFC 2704.

[3] BLAZE, M., FEIGENBAUM, J., AND LACY, J. Decentralized trust management. In *Proc. of IEEE Symposium on Security and Privacy* (1996), pp. 164–173.

[4] CHADWICK, D. W., AND OTENKO, A. The PERMIS X.509 role-based privilege management infrastructure. In *7th ACM SACMAT* (2002), pp. 135–140.

[5] FRANCISCO CURBERA, ET AL. *Business Process Execution Language for Web Services (BPEL4WS)*. BEA, IBM, Microsoft, May 2003. http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/.

[6] ITU-T RECOMMENDATION X.509:2000(E) | ISO/IEC 9594-8:2001(E). The directory: Public-key and attribute certificate frameworks.

[7] KOSHUTANSKI, H., AND MASSACCI, F. Interactive access control for Web Services. In *19th IFIP Information Security Conference (SEC 2004)*, pp. 150–166.

[8] KOSHUTANSKI, H., AND MASSACCI, F. A logical model for security of Web services. Tech. rep., 1st International Workshop on Formal Aspects of Security and Trust (FAST), Pisa, Italy, September 2003.

[9] LI, N., AND MITCHELL, J. C. RT: A role-based trust-management framework. In *Proc. of 3rd DARPA Information Survivability Conference and Exposition (DISCEX III)* (Los Alamitos, California, April 2003), pp. 201–212.

[10] WAHL, M., HOWES, T., AND KILLE, S. Lightweight Directory Access Protocol (v3), December 1997. RFC 2251.

[11] WINSLETT M, ET AL. Negotiating trust in the Web. *IEEE Internet Computing 6*, 6 (Nov/Dec 2002), 30–37.

[12] YU, T., WINSLETT, M., AND SEAMONS, K. E. Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM TISSEC 6*, 1 (2003), 1–42.