Chapter VII
Interactive Access Control and Trust Negotiation for Autonomic Communication

Hristo Koshutanski
University of Trento, Italy

Fabio Massacci
University of Trento, Italy

ABSTRACT

Autonomic communication and computing is the new paradigm for dynamic service integration over a network. In an autonomic network, clients may have the right credentials to access a service but may not know it; equally, it is unrealistic to assume that service providers would publish their policies on the Web so that clients can do policy evaluation themselves. To solve this problem, the chapter proposes a novel interactive access control model: Servers should be able to interact with clients asking for missing or excessing credentials, whereas clients my decided to comply or not with the requested credentials. The process iterates until a final agreement is reached or denied. Further, the chapter shows how to model a trust negotiation protocol that allows two entities in a network to automatically negotiate requirements needed to access a service. A practical implementation of the access control model is given using X.509 and SAML standards.

INTRODUCTION

Recent advances of Internet technologies and globalization of peer-to-peer communications offer for organizations and individuals an open environment for rapid and dynamic resource integration. In such an environment, federations of heterogeneous systems are formed with no central authority and no unified security infrastructure. Considering this level of openness, each server is

responsible for the management and enforcement of its own security policies with a high degree of autonomy.

Controlling access to services is a key aspect of networking and the last few years have seen the domination of policy-based access control. Indeed, the paradigm is broader than simple access control, and one may speak of policy-based self-management networks (see, e.g., IEEE Policy Workshop series; Lymberopoulos, Lupu & Sloman, 2003; Sloman & Lupu, 1999). The intuition is that actions of nodes controlling access to services are automatically derived from policies. The nodes look at events, requested actions and credentials presented to them, evaluate the policy rules according to those new facts and derive the actions (Sloman & Lupu, 1999; Smirnov, 2003). Policies can be "simple" iptables configuration rules for Linux firewalls (see *http://www.netfilter.org*) or complex logical policies expressed in languages such as Ponder (Damianou, Dulay, Lupu, & Sloman, 2001) or a combination of policies across heterogeneous systems as in OASIS XACML framework (XACML).

Dynamic coalitions and autonomic communication add new challenges: A truly autonomic network is born when nodes are no longer within the boundary of a single enterprise, which could deploy its policies on each and every node and guarantee interoperability. An autonomic network is characterized by properties of self-awareness, self-management and self-configuration of its constituent nodes. In an autonomic network nodes are like partners that offer services and lightly integrate their efforts into one (hopefully coherent) network. This cross enterprise scenario poses novel security challenges with aspects of both trust management and workflow security.

From trust management systems (Ellison et al., 1999; Li, Grosof, & Feigenbaum, 2003; Weeks, 2001) it takes the credential-based view. Since access to network services is offered by autonomic nodes and to potentially unknown clients, the decision of grant or deny access can only be made on the basis of credentials sent by a client.

From workflow access control systems (Atluri, Chun, & Mazzoleni, 2001; Bertino, Ferrari, & Atluri, 1999; Georgakopoulos, Hornick, & Sheth, 1995; Kang, Park, & Froscher, 2001) we borrow all classical problems such as dynamic assignment of roles to users, dynamic separation of duties, and assignment of permissions to users according to the least privilege principles.

In an autonomic communication scenario a client might have all the necessary credentials to access a service but may simply not know it. Equally, it is unrealistic to assume that servers will publish their security policies on the Web so that clients can do a policy combination and evaluation themselves. So, it should be possible for a server to ask a client on-the-fly for additional credentials whereas the client may disclose or decline to provide them. Next, the server reevaluates the client's request, considering the newly submitted credentials and computes an access decision. The process iterates between the server and the client until a final decision of grant or deny is taken. We call this modality "interactive access control."

Part of these challenges can be solved by using policy-based self-management of networks, but not all of them. Indeed, if we abstract away the details on the policy implementation, one can observe that the only reasoning service actually used by nowadays policy-based approaches is *deduction*: given a policy and a set of additional facts, find out all consequences (actions or obligations) from the policy according to the facts. We simply look whether granting the request can be deduced from the policy and the current facts. Policies could be different (Bertino et al., 2001; Bertino, Ferrari, & Atluri, 1999; Bonatti & Samarati, 2002; Li, Grosof & Feigenbaum, 2003), but the kernel reasoning service is the same.

Access control for autonomic communication needs another less-known reasoning service, taken from AI domain, called "abduction" (Sha-

nahan, 1989). Loosely speaking, we could say that abduction is deduction in reverse: Given a policy and a request to access a network service, we want to know what are the credentials (facts) that would grant access. Logically, we want to know whether there is a (possibly minimal) set of facts that added to the policy would entail (deduce) the request.

If we look again at our intuitive description of the interactive access control it is immediate to realize that abduction is the core service needed by the policy-based autonomic servers to reason for missing credentials.

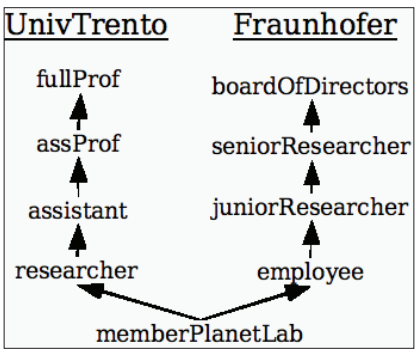
We can also use abduction on a client side so that whenever a client is requested for missing credentials it can perform evaluation on its policy and counter-request the server for some evidences in order to establish confidence (trust) to disclose the originally requested credentials.

Chapter Scope

This chapter targets readers who want to put into a practical framework security policies for access control. As a chapter outcome, the readers will be able to understand the logical reasoning services of deduction and abduction, and how to use them to model a practical access control framework. Furthermore, the readers will be able to model interactive access control between two entities, each of them running its own deduction and abduction algorithms, thus allowing a bilateral exchange of access requirements until an agreement is reached or denied.

For those readers with practical background, the chapter presents how to implement and integrate the interactive access control model with the security standards such as X.509 and SAML. Readers should be familiar with either logic programming or answer set programming or datalog, as a prerequisite to the chapter's content.

Figure 1. Joint hierarchy model



A PRIMER ON INTERACTIVE ACCESS CONTROL

Motivation by Example

Let us consider a shared overlay network PlanetLab between the University of Trento and Fraunhofer institute in Berlin in the context of the E-NEXT project. For the sake of simplicity assume that there are three main access types to resources: *disk* – read access to data residing on the Planet-Lab machines; *run* – execute access to data and possibility to run processes on the machines; and *configure* – including the previous two types of access plus the possibility of configuring network services on the machines.

Members of the two labs are classified in a hierarchy that is shown in Figure 1. The figure shows the joint hierarchy model of the roles at both institutions. The partial order of roles is indicated by arcs where higher the role in the hierarchy is more powerful it is. A role *dominates* another role if it is higher in the hierarchy and there is a direct path between them.

The access policy of the Planet-Lab network specifies that:

- *Disk* access is allowed for any request coming from the two institutions.
- *Run* access is allowed for any request coming either from specific machines at the two institutions or from the two institutions ac-

companied with a membership certificate.

- *Configure* access is allowed to anybody that has run access to the network resources and is at least researcher at University of Trento or junior researcher at Fraunhofer institute. Configure access is also granted to associate professors or senior researchers with the requirement of accessing the Planet-Lab network from the respective country domains of Italy or Germany. The least restrictive access is granted to full professors or members of board of directors obliging them to provide the appropriate credential attesting their positions.

Let us have the scenario where Alice is a senior researcher at Fraunhofer and daily she needs to get run access to resources at Planet-Lab network. So, whenever she is at her office and she wants to execute some services she sends her employee certificate to the system. According to the access policy, run access is granted to Alice because as an employee she is a member of the Planet-Lab hierarchy model (see Figure 1).

Now, examine the case in which Alice wants to have access to the system from his home place (deciding to work at home) presenting her employee certificate assuming that it is potentially enough to get run access to certain services. But, according to the policy rules the system should deny the request because run access requests coming from domains different than University of Trento or Fraunhofer institute are allowed only to associate professors or senior researchers or higher role positions.

So, the natural question is, "*is it the behavior we want from the system?*" Shall we leave Alice with only "access denied" decision and being idle for the whole day simply because she did not know or just has forgotten that access to the system outside Fraunhofer needs another certificate?

An answer like "*sorry, we also need a credential for being at least a senior researcher*" would be more than welcomed by most employees. At the same time, the server wants to be sure to ask this additional credential only to employees.

Protecting Sensitive Policies

Practical access control policies like those protecting companies' resources, EU project sensitive documents etc, may leak valuable business information when exposed to public. Furthermore, an access control policy sometimes may disclose the entire business strategy of a company or an institution. Consider the following examples:

Example 1 *(Seamons, Winslett, & Yu, 2001) Suppose a Web page's access control policy states that in order to access documents of a project in the site, a requester should present an employee ID issued either by Microsoft or by IBM. If such a policy can be shown to any requester, then one can infer with high confidence that this project is a cooperative effort of the two companies.*

Example 2 *(Yu & Winslett, 2003) [Access Policy] McKinley clinic makes its patient records available for online access. Let r be Alice's record. To gain access to r a requester must either present Alice's patient ID for McKinley clinic (CAliceID), or present a California social worker license (CCSWL) and a release-of-information credential (CRoI) issued to the requester by Alice.*

[Sensitive Policy Protection] Knowing that Alice's record specifically allows access by social workers will help people infer that Alice may have a mental or emotional problem. Alice will probably want to keep the latter constraint inaccessible to strangers. However, employees of McKinley clinic (CMcKinleyEmployee) should be allowed to see the contents of the policy.

To conclude so far, we have identified the following two issues:

- Provide additional information on missing credentials back to clients in case they do not have enough access rights.
- Protect access policies and their requirements from unnecessary disclosure.

How we approach the above cases is the subject of the next section.

Interactive Access Control vs Current Approaches

In this section we introduce step-by-step the novel contribution of interactive access control model by "evolving" the existing access control frameworks.

Let us start with the traditional access control. A server has a *security policy for access control* P_A that is used when taking access decisions about usage of services offered by a service provider. A user submits a set of credentials C_p and a service request r in order to execute a service. We say that policy P_A and credentials C_p entail r (informally for the moment, $P_A \cup C_p \models r$) meaning that request r should be granted by the policy P_A and the presented credentials C_p.

Figure 2 shows the "traditional" access control decision process. Whether the decision process uses RBAC (Sandhu et al, 1996), SDSI/SPKI (SPKI), RT (Li & Mitchell, 2003) or any other trust management framework it is immaterial at this stage: they can be captured by suitably defining P_A, C_p and the entailment operator ($\models$). This approach is the cornerstone of most logical formalizations (De Capitani di Vimercati & Samarati, 2001): If the request r is a consequence of the policy and the credentials, then access is granted; otherwise it is denied.

A number of works has deemed such blunt denials unsatisfactory. Bonatti and Samarati (2002) and Yu, Winslett and Seamons (2003) proposed to send back to clients some of the rules that are necessary to gain additional access. Figure 3 shows the essence of the approaches.

Both works have limitations because they impose several syntactical restrictions on the format of the policy and essentially merge two different security issues: the policy for governing access to server's own resources and the policy for governing the disclosure of foreign credentials.

The first and foremost limitation is that both approaches require policies to be flat: A policy protecting a resource must contain all credentials needed to allow access to that resource. As a result, it calls for structuring of policy rules that is counter-intuitive from the access control point of view. For instance, a policy rule may say that for access to the full text of an online journal article a requester must satisfy the requirements for browsing the journal's table of contents plus some additional credentials. A rule detailing access to the table of contents could then specify another

Figure 2. Traditional access control

1. check whether P_A and C_p entail r,
2. if the check succeeds then *grant* access
3. else *deny* access.

Figure 3. Disclosable access control

1. check whether P_A and C_p entail r,
2. if the check succeeds then *grant* access
3. else
   (a) find a rule $r \leftarrow p \in PartialEvaluation(P_A \cup C_p)$, where p is a (partial) policy protecting r,
   (b) if such a rule exists then *send* it back to the client else *deny* access.

set of credentials. Even this simple scenario is not allowed in either formalisms.

Constraints that would make policy reasoning non-monotone (such as separation of duties) are also ruled out as they require to look at more than one rule at a time. So, if the policy is not flat and 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.

Bonatti and Samarati's (2002) approach has further limitations on the granularity level of disclosure of information. In their work governing access to a service is composed in two parts: a prerequisite rule and a requisite rule. Prerequisite rules specify the requirements that a client should satisfy before being considered for the requirements stated by the requisite rules, which in turn grant access to services. Thus, prerequisite rules play the role of controlling the disclosure of the service requisite rules. In this way their approach does not decouple policy disclosure from policy satisfaction, as already noted by Yu and Winslett (2003), which becomes a limitation when information disclosure plays crucial role.

The work by Yu and Winslett (2003) overcomes this latter limitation and proposes to treat policies as fist class resources, i.e., each policy protecting a resource is considered as a sensitive resource itself whose disclosure is recursively protected by another policy. Still they have the same flatness, unicity and monotonicity limitations. These limitations are due to a traditional viewpoint: the only reasoning service one needs for access viewpoint is deduction, i.e., check that the request follows from the policy and the presented credentials.

Intuition 1: We claim that we need another less-known reasoning service, called *abduction*: check which missing credentials are necessary so that the request can follow from the policy and the presented credentials. Thereupon, we present the basic idea of interactive access control in Figure 4.

The "compute a set C_M such that ..." (step 3a) is exactly the operation of abduction. This solution raises a new challenge: how do we decide the potential set of missing credentials? It is clearly undesirable to disclose all credentials occurring in P_A and, therefore, we need a way to define how to control the disclosure of such a set.

As we have already noted, Yu and Winslett (2003) addressed partly this issue by protecting policies within the access policy itself. However, this is not really satisfactory as it does not decouple the decision about access from the decision about disclosure.

So, from a standpoint of a good engineering practice a structured approach of separate access and disclosure policies is better than a flat (merged policy) approach because the criteria behind and 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.

Figure 4. Basic idea of interactive access control

1. check whether P_A and C_p entail r,
2. if the check succeeds then *grant* access
3. else
   (a) compute a set C_M such that:
      - P_A together with C_p and C_M entail r, and
      - P_A together with C_p and C_M preserve consistency.
   (b) if C_M exists then *ask* the client for C_M and iterate
   (c) (c) else *deny* access.

Intuition 2: We claim that we need two policies: one for granting access to one's own resources and one for disclosing the need of foreign (someone else's) credentials. Therefore, we introduce a *security policy for disclosure control* P_D. The policy for disclosure control is used to decide credentials whose need can be potentially disclosed to a client. In other words, P_A protects partner's resources by stipulating what credentials a requestor must satisfy to be authorized for a particular resource while, in contrast, P_D defines which credentials among those occurring in P_A are disclosable so, if needed, can be demanded from the requestor.

The relevant approach with respect to the disclosure policy P_D is the one by Yu and Winslett (2003). It postulates that policies for protecting resources should be themselves treated (protected) as first class sensitive resources. The authors distinguish between policy disclosure and policy satisfaction which allows them to have control on when a policy can be disclosed from when a policy is satisfied.

However, Yu and Winslett policies determine whether a client is authorized to be informed of the need to satisfy a given policy. While, in our case, having a separate disclosure policy P_D allows us to have a finer-grained disclosure control over the information flow back to a client. Instead of controlling the disclosure of (entire) policies as a finest-grained unit we are able to control the disclosure of single credentials composing those policies separately and independently from the disclosure of the policies themselves.

We give a new refined algorithm for interactive access control with controlled disclosure shown in Figure 5.

Now, let us refer to Yu and Winslett's own example (Example 2) formalized as two logic programs:

Example 3

P_D

$C_{AliceID} \leftarrow .$

$C_{CSWL} \leftarrow C_{McKinleyEmployee} .$

$C_{RoI} \leftarrow C_{McKinleyEmployee} .$

P_A

$r \leftarrow C_{AliceID} .$

$r \leftarrow C_{CSWL}, C_{RoI} .$

The disclosure control policy is read as the disclosure of Alice's ID is not protected and potentially released to anybody. The need for disclosing the credentials for California social worker license C_{CSWL} and release-of-information C_{RoI} is released only to users that have already presented their McKinley employee certificates $C_{McKinleyEmployee}$.

The access policy specifies that access to r is granted either to Alice or to California social workers that have a release-of-information credential issued by Alice.

We note that the disclosure requirement for $C_{McKinleyEmployee}$ cannot be captured via the service accessibility scheme by Bonatti and Samarati (2002) and refer to Yu and Winslett (2003) for details. We also point out (as in Yu & Winslett, 2003) that having $C_{McKinleyEmployee}$ does not allow access

Figure 5. Interactive access control with controlled disclosure

1. check whether P_A and C_p entail r,
2. if the check succeeds then *grant* access
3. else
   (a) compute the set of disclosable credentials C_D entailed by P_D and C_p,
   (b) compute a set C_M out of the disclosable ones ($C_M \subseteq C_D$) such that:
      - P_A together with C_p and C_M entail r, and
      - P_A together with C_p and C_M preserve consistency.
   (c) if C_M exists then *ask* the client for C_M and iterate
   (d) else *deny* access.

to *r* but rather is used to unlock more information on how to access *r*. We also emphasize that the disclosure control on *r*'s policy $\{C_{CSWL}, C_{RoI}\}$ can be further split down on controlling the disclosure of the single credentials constituting it.

There are still tricky questions to be answered such as:

- How do we know that the algorithm terminates? In other words, can a client waste the server's time forever?
- How do we know that a cooperative client, starting with a wrong set of credentials, can actually arrive to a grant? For example, can we assure that the server will not keep asking Alice for a UNITN full professor credential which she does not have, while never asking for a FOKUS senior researcher credential, which she has?

We will show how to fix the details of the algorithm in a later section so that all answers are positive.

So far, we have considered the access control process taking part on a server side. Then one would ask what about protecting clients from unauthorized disclosure of missing credentials. One can use the interactive access control algorithm also on the client side so that the client can do policy evaluation itself to determine whether the requested credentials can be disclosed to (granted, to be seen by) servers. And, alternatively, what additional information the servers should provide in order to see the requested credentials. In this way the interactive access control model can be used on client and server sides allowing them to automatically negotiate missing credentials until an agreement is reached or denied. The full evolvement of the negotiation model is described later in the chapter.

This is enough to cover stateless systems. We still have a major challenge ahead: How do we cope with stateful systems? Stateful systems are systems where the access decisions change depending on past interactions or past presented credentials. Such systems can easily become inconsistent with respect to the client's set of presented credentials mainly because access policies may forbid the presentation of credential if another currently active credential has been presented in the past.

Past requests or services usage may deny access to future services as in Bertino, Ferrari and Atluri (1999) centralized access control model for workflows. Separation of duties means that we cannot extend privileges by supplying more credentials. For instance a branch manager of a bank clearing a cheque cannot be the same member of staff who has emitted the cheque (Bertino, Ferrari & Atluri, 1999, p. 67). If we have no memory of past credentials then it is impossible to enforce any security policy for separation of duties on the application workflow. The problems that could cause a process to get stuck are the following:

- The request may be inconsistent with some role, action or event from the client in the past.
- The new set of credentials may be inconsistent with requirements such as separation of duties.

To address the problem of inconsistency, we extend the stateless algorithm in a way that it allows a service provider to reason of not only what missing credentials are needed to get a service, but also to find out what excessing (conflicting) is among the client's set of credentials that makes the policy state inconsistent.

Intuition 3: We claim that in the stateful systems' domain we need to reason of not only on what missing credentials allow access but also on what excessing (conflicting) credentials make the policy state inconsistent. We need a procedure by which if a user has exceeded his privileges he has the chance to revoke them.

The basic intuitive algorithm for interactive access control for stateful systems is shown in Figure 6. Steps 1 to 3d are essentially the basic interactive access control algorithm (see Figure 5).

The part for stateful systems comes when we are not able to find a set of missing credentials among the disclosable ones (step 3d).

In this case there are two reasons which may cause the abduction failure when computing C_M. The first one could be that in C_D there are not enough disclosed credentials to grant r – case in which we should deny access (step 3(d)iii), or there might be credentials in the client's set of presented credentials C_p that make the policy state inconsistent – case in which any solution among the disclosable credentials cannot be found by the abduction.

The latter reason motivates step 3(d)i. In this step, first, we want to find a set of conflicting credentials C_E, called excessing, among the presented ones C_p such that removing them from C_p preserves the access policy consistent, step 3(d)iA. Second, on top of the not conflicting credentials it must exist a solution set that entails the service request, step 3(d)iB. The second requirement assures that there is a potential solution for the client to get access to the requested service.

We refer the reader to (Koshutanski, 2005) for full details on the stateful model.

Interactive Access Control and Current Policy-Based Approaches

Having introduced the core logical reasoning services and the respective access control algorithms does not completely show the advantages of the interactive access control model. This section describes how current logic-based approaches suit our interactive model.

The logical model, as presented so far, abstracts from a specific policy language and presents an execution framework for reasoning about access control. As such, the model fills an important gap between the policy language specification and the policy language enforcement and evaluation.

We skip here the classical access control models (see, e.g., (De Capitani di Vimercati & Samarati, 2001) for a comprehensive survey) and concentrate on the current logic-based access control approaches widely cited in the literature.

The work by Li, Mitchell, and Winsborough (2002) introduces a model for distributed access control, called RT (Role-based Trust management). The core idea of the model is the way

Figure 6. Interactive Access control for stateful systems with controlled disclosure

1. check whether P_A and C_p entail r,
2. if the check succeeds then *grant* access
3. else
   - (a) compute the set of disclosable credentials C_D entailed by P_D and C_p ,
   - (b) compute a set of missing credentials C_M out of the disclosable ones ($C_M \subseteq C_D$) such that:
     - P_A together with C_p and C_M entail r, and
     - P_A together with C_p and C_M preserve consistency.
   - (c) if a set C_M exists then *ask* the client for C_M and iterate
   - (d) else
     - i. compute a set of *excessing credentials* C_E among the client's presented ones ($C_E \subseteq C_p$) such that:
       - *A)* P_A together with $C_p \backslash C_E$ preserve consistency, and
       - B) it exists C_M ($\subseteq C_D$) such that:
         - P_A together with $C_p \backslash C_E$ and C_M entail r, and
         - P_A together with $C_p \backslash C_E$ and C_M preserve consistency.
     - ii. if a set C_E exists then *ask* the client to present C_M and revoke C_E , and iterate
     - iii. else *deny* access.

it classifies principles in a distributed manner. Basically, the model classifies each entity's local attributes (roles) and how other entities relate to those attributes. It classifies how each entity's attributes relates to other entities' attributes (attribute mapping from one domain to another). It also defines attribute-based delegation of attribute authority, i.e. the ability to delegate authority to strangers whose trustworthiness is determined based on their own certified attributes.

A later approach (Li, Li, & Winsborough, 2005) extends the RT framework to cope with different cryptographic schemes (e.g., zero-knowledge proof of attributes, oblivious signature envelope, hidden credential etc.) that are used to improve the privacy protection and effectiveness during a process of bilateral negotiation. The authors proposed a new language, called Attribute-based Trust Negotiation Language (ATNL), that specifies fine-grained protection of resources and their policies.

Another interesting logic-based approach is (Ruan, Varadharajan, & Zhang, 2003). In contrast to what we have seen, this work presents an authorization model that supports both positive and negative authorizations. The model introduces variety of rules that define different authorization and delegation statements, as well as, rules for conflict resolutions. This work targets another type of polices where explicit negation is needed to express the policy requirements.

All of the above described approaches are good candidates for an underlying policy language as the interactive access control model is data-driven by the abduction and deduction reasoning services. So, we will not target a particular policy language throughout the chapter as it is immaterial to the meta-level access control process the actual logical language.

Winsborough and Li (2004) postulate an important property concerning trust negotiation called safety in automated trust negotiation. During a negotiation process a sensitive credential is disclosed when its policy is satisfied by the negotiator. So, the problem comes from the fact that although a sensitive credential itself is not transmitted unless its associated policy is satisfied, the behavior of a negotiator differs based on whether he has the attribute or not. One can reveal additional information about the content of the credential by monitoring the opponent's behavior.

Since the interactive access control model enforces a meta-level negotiation process one can address the safety property requirement by properly defining the structure of the access and disclosure control policies.

THE UNDERLYING LOGICAL MODEL

Syntax

As we have identified, policy-based approaches suit well our needs for having an appropriate policy language depending on particular access control scenario. Still we need to define the syn-

Table 1. Predicates used in the logical model

dominate(Role: *ri*, Role: *rj*)	when role *ri* dominates role *rj* ($\succ$), where *ri* and *rj* are possibly the same.
grant(Resource: *s*, Action: *p*)	when action *p* is granted to be performed on resource *s*.
credential(holder, Attr: *a*, Issuer: *i*)	a certificate attesting that holder has an attribute *a* issued by *i*, where *a* can be a role or other property charcterizing particular access rights.
certificate(subject, Issuer: *i*)	a certificate identifying entity subject issued by *i*.
classify(Issuer: *i*, IssuerType: *t*)	classifies issuer *i* as a particular type *t* certificate authority

tax of the underlying logical model that covers the basic needs of the autonomic communication domain.

In the model we have the following sets of identifiers: *Role* for role identifiers; *Resource* for resource identifiers; *Action* for action identifiers performed on resources; *Attr* for attribute identifiers, where $Role \subseteq Attr$; *Issuer* for certificate issuer identifiers; and *IssuerType* for type/classification of issuers.

Table 1 shows the basic predicates used in the logical model.

An attribute certificate is represented as a combination of two predicates: one defining the holder, attribute and the issuer of the certificate, and the other classifying the issuer authorized (trusted) to issue such attributes.

Analogously, an identity certificate is represented as a predicate identifying the subject and a predicate classifying the trusted issuer.

We do not keep a set of user identifiers because in the autonomic communication's domain anybody is potentially a client and the notion of identity-based access control does not apply. Thus, holder and subject variables, shown in Table 1, do not have a priori fixed values but rather are used to relate with each other in order to express proper identification requirements.

Example 4 *To grant access to a bank report, a client should identify itself by a trusted identity certificate and present a credential for a role bank manager (issued by the bank attribute authority).*

In this situation, we are not particularly interested in the client's identity but in the relation that the subject of the identity certificate correctly maps the holder of the attribute certificate.

The model presented in this section can be adapted to any generic policy framework. The information we need from the underlying policy language is shown in the table above and can be found in (adapted to) most policy languages.

Access policies are written as normal logic programs (Apt, 1990). These are sets of rules of the form:

$$A \leftarrow B_1, \ldots, B_n, \mathit{not}\ C_1, \ldots, \mathit{not}\ C_{m,} \qquad (1)$$

where A, B_i and C_j are (possibly ground) predicates. 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.

$$\leftarrow B_1, \ldots, B_n, \mathit{not}\ C_1, \ldots, \mathit{not}\ C_m \qquad (2)$$

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 situations in which B_i are true and all C_j are false.

Formalization of the Example

Following is the full formalization of the example introduced at the beginning of the chapter. A new predicate used in the example is *authNet(IP, DomainName)*. It is a tuple with first argument the IP address of the authorized network endpoint (the client's machine) and the second argument the domain name where the IP address comes from.

We omit the resource field in the *grant* predicate meaning that for any resource in the system the user is considered to have *disk*, *run* or *configure* access rights. We represent variables with staring capital letter (e.g., Holder, Attr, Issuer) while constants with starting small case letters (e.g., plan-

etLabClass1SOA, institute, juniorResearcher). A variable indicates any value in its field.

Figure 7 shows the formalization of the Planet-Lab policies. Following is the functional explanation of the policies.

The access policy says:

- Rules (1), (2), and (3) classify issuers (SOAs) in different logical categories used by the access control logic. Example, Rule (1) categorizes planetLabClass1SOA as a system level SOA.
- Rules (4) and (5) give disk access to the shared network content to everybody from the University of Trento and Fraunhofer institute, regardless the IP and roles at these institutions.
- Rule (6) gives disk access to anybody who has a run access permission.
- Rules (7) and (8) allow run access for those machines that are internal of the two institutions (dedicated only for Planet-Lab access) and distinguished by their fixed IPs.
- Rules (9), (10), and (11) relax the previous two and allow run access from any place of the institutions to those users which present either a Planet-Lab membership certificate or a role-position certificate at one of the two institutions.
- Rule (12) gives run access to anybody who has a configure access permission.
- Rules (13) and (14) give configure access right if a user has a disk access and is at minimum assistant, attested (issued) by a

Figure 7. Planet-lab access and disclosure control policies

Access Policy:

(1) classify(planetLabClass1SOA, system).
(2) classify(fraunhoferClass1SOA, institute).
(3) classify(unitnClass1SOA, university).
(4) grant(disk) ← authNet(*, *.unitn.it).
(5) grant(disk) ← authNet(*, *.fraunhofer.de).
(6) grant(disk) ← grant(run).
(7) grant(run) ← authNet(193.168.205.*, *.unitn.it).
(8) grant(run) ← authNet(198.162.45.*, *.fraunhofer.de).
(9) grant(run) ← grant (disk), credential(*, memberPlanetLab, Issuer), classify(Issuer, system).
(10) grant(run) ← grant(disk), credential(*, Attr, Issuer), classify (Issuer, university), Attr ≻ researcher.
(11) grant(run) ← grant(disk), credential(*, Attr, Issuer), classify (Issuer, institute), Attr ≻ employee.
(12) grant(run) ← grant(configure).
(13) grant(configure) ← grant(disk), credential(*, Attr, Issuer), classify(Issuer, university), Attr ≻ assistant.
(14) grant(configure) ← grant(disk), credential(*, Attr, Issuer), classify(Issuer, institute), Attr ≻ juniorResearcher.
(15) grant(configure) ← authNet(*, *.it), credential(*, Attr, Issuer), classify(Issuer, university), Attr ≻ assProf.
(16) grant(configure) ← authNet(*, *.de), credential(*, Attr, Issuer), classify(Issuer, institute), Attr ≻ seniorResearcher.
(17) grant(configure) ← credential(*, Attr, Issuer), classify(Issuer, university), Attr ≻ fullProf.
(18) grant(configure) ← credential(*, Attr, Issuer), classify(Issuer, institute), Attr ≻ boardOfDirectors.

Disclosure Policy:

(1) credential(Holder, memberPlanetLab, Issuer) ← authNet(*, *.unitn.it), classify(Issuer, system).
(2) credential(Holder, memberPlanetLab, Issuer) ← authNet(*, *.fraunhofer.de), classify(Issuer, system).
(3) credential(Holder, employee, Issuer) ← credential(Holder, memberPlanetLab, IssuerSys), classify(IssuerSys, system), classify(Issuer, institute).
(4) credential(Holder, researcher, Issuer) ← credential(Holder, memberPlanetLab, IssuerSys), classify(IssuerSys, system), classify(Issuer, university).
(5) credential(Holder, AttrX, Issuer) ← credential(Holder, AttrY, Issuer), classify(Issuer, university), AttrX ≻ AttrY.
(6) credential(Holder, AttrX, Issuer) ← credential(Holder, AttrY, Issuer), classify(Issuer, institute), AttrX ≻ AttrY.

trusted university's SOA, or at minimum junior researcher attested by a trusted institutional SOA.

- Rules (15) and (16) relax the previous two and give configure access to associate professors and senior researchers provided that requests come from the respective country domains.
- Rules (17) and (18) give configure access regardless the geographical region only to members of board of directors and to full professors.

The disclosure policy says:

- Rules (1) and (2) disclose the need for a Planet-Lab membership certificate to any request coming from domains of the respective organizations.
- Rules (3) and (4) disclose the need for an employee or a researcher certificate if either a client has already presented its Planet-Lab membership certificate or the certificate is disclosed by other rules of the disclosure policy.
- Rules (5) and (6) disclose (upgrade) the need of higher role-position certificates than those provided either by a client or (disclosed) by other rules of the policy.

Semantics

One of the most prominent semantics for normal logic programs is the stable model semantics proposed by Gelfond and Lifschitz (1988) (see also Apt, 1990, for an introduction). In the following we formally define the reasoning services intuitively introduced in the motivation section.

Definition 1 (Deduction and Consistency) *Let P be a policy and L be a ground literal. L is deducible 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 2 (Security Consequence) *A resource r is a security consequence of a policy P if (i) P is consistent and (ii) r is deducible of P.*

Definition 3 (Abduction) *Let P be a policy, H a set of ground atoms (called hypotheses or abducibles), L a ground literal (called observation) and $\prec$ a partial order (p.o.) over subsets of H. A solution of the abduction problem <L, H, P> is a set of ground atoms E such that:*

1. $E \subseteq H$,
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 partial orders are subset containment or set cardinality.

Definition 4 (Solution Set for a Resource) *Let P be a policy and r be a resource. A set of credentials C_S is a solution set for r according to P if r is a security consequence of P and C_S, i.e. $P \cup C_S \models r$ and $P \cup C_S \not\models \bot$.*

Definition 5 (Monotonic and Non-monotonic Policy) *A policy P is monotonic if whenever a set of statements C is a solution set for r according to P ($P \cup C \models r$) then any superset $C' \supset C$ is also a solution set for r according to P ($P \cup C' \models r$). In contrast, a nonmonotonic policy is a logic program in which if C is a solution for r it may exist $C' \supset C$ that is not a solution for r, i.e. $P \cup C' \not\models r$.*

THE INTERACTIVE ACCESS CONTROL ALGORITHM

Below we summarize all the information we have recalled (policies, credentials, etc.) to this extend.

- P_A security policy governing access to resources.
- P_D security policy controlling the disclosure of foreign (missing) credentials.
- C_p the set of credentials presented by a client in a single interaction.
- C_P the set of active credentials that have been presented by a client during an interactive access control process.
- C_N the set of credentials that a client has declined to present during an interactive access control process.

Now, we have all the necessary material to introduce our interactive access control algorithm for stateless services, shown in Figure 8.

The intuition behind the algorithm is the following. Once the client has initiated a service request r with (optionally) a set of credentials C_p, the interactive algorithm updates the client's profile C_P and C_N (lines 1: and 2:). C_P is updated with the newly presented credentials C_p and C_N is updated with the set difference of what the client was asked in the last interaction (C_M) (set)minus what he presents in the current one (C_p). Next, the algorithm consults for an access decision (line 3:). The first step of the access decision function checks whether the request r is granted by P_A according to the client's set C_P (step 1). If the check fails, the starting point of the interactive framework, then in step 2a the algorithm computes all credentials disclosable from P_D according to C_P and from the resulting set removes all already declined and already presented credentials. The latter is used to avoid dead loops of asking something already declined or presented. Then, the algorithm computes (using the abduction reasoning) all possible subsets of C_D that are consistent with the access policy P_A and, at the same time, grant r. Out of all those sets (if any) the algorithm selects the minimal one.

Example 5 *A senior researcher at Fraunhofer institute FOKUS wants to reconfigure an online service for paper submissions of a workshop. The service is part of a big management system hosted*

Figure 8. Interactive access control algorithm

Input: r, C_p
Output: grant/deny/ask(C_M)
***i*AccessControl**(r, C_p){
1: $C_P = C_P \cup C_p$;
2: $C_N = C_N \cup (C_M \backslash C_p)$, where C_M is from the last interaction;
3: *result* = *i*AccessDecision(r, P_A, P_D, C_P, C_N);
4: return *result*;
}
***i*AccessDecision**(r, P_A, P_D, C_P, C_N){
1. check whether r is a security consequence of P_A and C_P , namely
   - $P_A \cup C_P \models r$, and
   - $P_A \cup C_P \not\models \perp$.
2. if the check succeeds then return *grant* else
   (a) compute the set of disclosable credentials C_D as
   $C_D = \{c \mid c \text{ credential that } P_D \cup C_P \models c\} \setminus (C_N \cup C_P)$,
   (b) use abduction to find a set of missing credentials C_M ($\subseteq C_D$) such that:
   - $P_A \cup C_P \cup C_M \models$ r, and
   - $P_A \cup C_P \cup C_M \not\models \perp$.
   (c) if no such set exists then return *deny*
   (d) else return *ask*(C_M).
}

at the University of Trento's network that is part of the Planet-Lab network, formalized in the previous section. For doing that, at the time of access, she presents her employee certificate, issued by a Fraunhofer certificate authority, presuming that it is enough as a potential user. Formally speaking, the request comes from a domain fokus.fraunhofer.de with an attribute credential for an employee. The set of credentials is:

{authNet(198.162.193.46, fokus.fraunhofer.de),
credential(AliceMilburk, employee, fraunhoferClass1SOA)}

So, according to the access policy the credentials are not enough to get configure access and the request would be denied (see rule 14 in Figure 7). Then, following the algorithm (step 2a in Figure 8) it is computed the set of disclosable credentials from the disclosure policy and the user's set of active credentials C_P. In our case, C_P is the set of credentials mentioned above. Next, the algorithm computes C_D as the need of all roles higher in position than memberPlanetLab (see Figure 7, Disclosure Policy part) and the abduction step (Figure 8 step 2b), with criterion minimal set cardinality, computes the following missing sets that satisfy the request:

{credential(AliceMilburk, juniorResearcher, fraunhoferClass1SOA)},
{credential(AliceMilburk, seniorResearcher, fraunhoferClass1SOA)},
{credential(AliceMilburk, boardOfDirectors, fraunhoferClass1SOA)}

Then, using role minimality *criterion, the algorithm returns back the need for {credential(AliceMilburk, juniorResearcher, fraunhoferClass1SOA)}.*

In the next interaction, since Alice is a senior researcher, she declines to present the requested credential by returning the same query but with no entry for presented credentials ($C_p = \emptyset$). So, the algorithm updates the user's profile marking the requested credential credential(AliceMilburk, juniorResearcher, fraunhoferClass1SOA) declined.

The difference comes when the algorithm recomputes the disclosable credentials as all disclosable credentials from the last interaction minus the newly declined one. Next, abduction computes the following sets of missing credentials that satisfy the request:

{credential(AliceMilburk, seniorResearcher, fraunhoferClass1SOA)},
{credential(AliceMilburk, boardOfDirectors, fraunhoferClass1SOA)}

According to role minimality criterion, the algorithm returns the need for a credential {credential(AliceMilburk, seniorResearcher, fraunhoferClass1SOA)}. On the next interaction, Alice presents a certificate attesting her as a senior researcher and the algorithm grants the requested service.

Remark 1 *Using declined credentials is essential to avoid loops in the process and to guarantee successful interactions in presence of disjunctive information.*

For example suppose we have alternatives in the partner's policy (e.g., "present either a VISA or a Mastercard or an American Express card"). An arbitrary alternative can be selected by the abduction algorithm 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. The process continues until all credentials have been declined (and access is denied) or a solution is found (and access is granted).

Technical Guarantees

In the following we show the summary of the technical results that the access control algorithm provides. We refer the reader to (Koshutanski, 2005) for full details on the theoretical framework.

Following are the basic guarantees that the interactive framework provides:

- **Termination**: The interactive access control algorithm always terminates, that is, in a finite number of interactions either grant or deny is returned by the algorithms (resistant against DoS attacks).
- **Correctness**: If a client gets grant for a service then he has a solution for the service, that is, the algorithm does not grant access to unauthorized clients.
- **Completeness**: If a client has a solution for a service request then the algorithm will grant him access.

The most important thing, also the most difficult, is to model and prove that a client who has the right set of credentials and who is willing to send them to the server will not be left stranded in our autonomic network and will get grant.

First we need to define two different types of clients.

Definition 6 (Powerful client) *A powerful client is a client that whenever receives a request for missing credentials returns all of them.*

Definition 7 (Cooperative client) *A cooperative client is a client that whenever receives a request for missing credentials returns those of them that he has in possession.*

Defining the notion of good clients with respect to the interactive algorithm is still not enough to state the practical relevance of the access control model. We need to introduce the notion of fairness regarding the access and disclosure control policies. We define the following two properties:

Definition 8 (Fair Access) *A fair access property guarantees that whenever there is a request for a service it exists a solution in the access control policy which unlocks (grants) the service.*

In other words, for each resource protected by the access policy there should exist a set of credentials (a solution) that grants the resource according to the policy. Fair access property avoids cases where the policy specifies a solution for a service but the solution itself makes the policy state inconsistent, so that even a client with the right set of credentials for the service cannot get it.

Definition 9 (Fair Interaction) *A fair interaction property guarantees that if a solution for a service request exists (according to the access policy) then this solution should be disclosable by the disclosure control policy.*

In other words, any solution for a service should be potentially disclosable to a client requesting the service. In an autonomic scenario, where a service is potentially accessible by any client, fair interaction property would disclose a solution for a service to potentially any client requesting it. So, on one side, we want to be fair and disclose solutions to clients but, on the other side, we want to protect and restrict the disclosure of information only to selected clients (not to anybody). To approach this problem we introduce the notion of hidden credentials.

Informally speaking, a credential is hidden if an access control system needs it for taking an access decision, but does not disclose the need to anybody. Thus, an autonomic server can dynamically protect the privacy of its policies by specifying which credentials are hidden and which are not. This allows a server to restrict access to certain services only to selected clients.

Now we can define a client with hidden credentials.

Definition 10 (Client with Hidden Credentials for a Service) *A client with hidden credentials for a service is any client that has in possession the hidden credentials for that service and knows that these are to be pushed to the server.*

Now, we have to redefine the fair interaction property with respect to hidden credentials.

Definition 11 (Fair Interaction with Hidden Credentials) *If a solution for a service exists and there are hidden credentials for that solution then all credentials from the solution set which are not hidden must be disclosable by the disclosure policy and the set of hidden credentials.*

So far, we have introduced all we need to formulate the main guarantees showing the practical relevance of the access control framework.

- **Completeness for a powerful client**: If access and disclosure control policies guarantee fair access and interaction, respectively, then a powerful client requesting access to a service will get grant with the interactive access control algorithm.
- **Completeness for a cooperative client**: If access and disclosure control policies guarantee fair access and interaction, respectively, then if a cooperative client has a solution for a service request then he will get grant with the interactive access control algorithm.

We have the same claims for powerful and cooperative clients with hidden credentials.

IMPLEMENTING THE ACCESS CONTROL FRAMEWORK

This section emphasizes on the practical relevance of the access control framework and, particularly, on how the access control model can be of practical use.

There are two main points relevant to the implementation of the framework. This first one is how to cope with the implementation of the interactive access control algorithm and the second one is how to integrate the logical model with the current security standards widely adopted by IT companies.

For the first point, we will use a logical-based reasoning system, called DLV (see *http://www.dlvsystem.com*) and, particularly, how to employ DLV in order to perform the basic computations of abduction and deduction. As for the second one, we will show how to integrate the logical model with X.509 certificate framework and OASIS SAML standard.

Integration with the Automated Reasoning Tool DLV

For the implementation of the interactive access control algorithm (presented in Figure 5) we use the DLV system (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. Figure 9 shows the implementation using the DLV system. The input of the function *i*AccessDecision is the requested service *r*, the policy for access control P_A, the policy for disclosure control P_D, the set of active credentials C_P and the set of declined credentials C_N . Step 1 uses the DLV's deductive front end. It specifies

as input the service request *r* marked as a query over the models (*r*?) computed over $P_A \cup C_P$. The output of this step are those models of in which *r* is true.

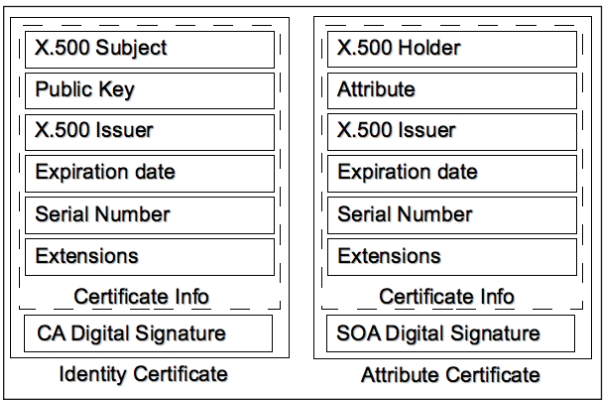
If it exists a model in Step 1 that satisfies *r* then it is returned grant (step 1). If no model for *r* exists then we use the DLV's deductive front end with input $P_D \cup C_P$ (step 3). In this case, DLV computes all credentials disclosable from $P_D \cup C_P$. Then from the computed set we remove all credentials that belong to C_N and C_P .

Once the disclosable credentials are computed then, in Step 4, we use the abductive diagnosis front end with the input: the requested service *r*, stored in a temporary file with extension .obs (observations), the just computed set of disclosable credentials C_D stored in a temporary file with extension .hyp (hypotheses or also called abducibles) and the third argument is the access policy together with the active credentials $P_A \cup C_P$. The two input files (.hyp and .obs) have particular meaning for DLV system in the abductive mode.

Figure 9. Implementation of the basic functionalities of deduction and abduction

***i*AccessDecision**(r, P_A, P_D, C_P, C_N){
1: if doDeduction(r, $P_A \cup C_P$) then return *grant*
2: else
3: $C_D = \{c \mid P_D \cup C_P \models c\} \setminus (C_N \cup C_P)$;
4: *result* = doAbduction(r, C_D, $P_A \cup C_P$);
5: if *result* = = $\perp$ then return *deny*
6: else return ask(*result*);
}
doDeduction(R: Query, P: LogProgram){ check for P |= 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) $P \cup C_i \not\models \perp$;
3: if no C_i exists then return $\perp$
4: else select a minimal C_{min} and return C_{min};
}

Figure 10. X.509 identity and attribute certificates structure



The output of that 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 according to some minimality criteria and select the minimal set out of them. If no missing set is found then we return deny else we return the missing set back to the client.

The automated reasoning tool depends on the one's own choice. It can be used any other tool that supports the basic reasoning services.

Integration with X.509 and SAML Standards

The framework described so far processes credentials on a high (abstract) level: defines what can be inferred and what missing is from partner's access policy and user's set of credentials. There is a need of a suitable certificate infrastructure for describing participant's identities and access rights. A good choice is the widely adopted certificate standard X.509 (X.509, 2001).

There are two certificate types considered by the standard: identity and attribute certificates. Figure 10 shows the structures of the two certificates.

X.509 identity certificate is used to identify entities in a network. The main fields of the certificate's structure are the subject information, the public key identifying the subject (corresponding to the subject's private key), the issuer information and the digital signature on the document, signed by the issuer (with its private key).

X.509 attribute certificate has the same structure like the identity one with the difference that instead of a public key field there is a field for listing attributes and the Subject field is called Holder (of the attributes).

Referring to the message level, one can adopt to use the OASIS SAML standard (SAML) for having standard semantics for authorization statements among participants in an autonomic network. SAML offers a standard way for exchanging authentication and authorization information between on-line partners.

The basic SAML data objects are assertions. Assertions contain information that determines whether users can be authenticated or authorized to use resources. The SAML framework also defines a protocol for requesting assertions and responding to them, which makes it suitable when modeling interactive communications between entities in a distributed environment.

We list below the SAML Request/Response protocol and how we employ it in the interactive access control framework.

- **SAML Request:** Use the Authorization Decision Query statement for expressing access decision requests. Specify the resource and action in the respective standard fields of the access statement.
  Once an access decision is taken use the SAML response part.
- **SAML Response:** Use the Authorization Decision Statement
  - **Permit / Deny:** When explicit grant/deny is returned by the *i*AccessControl protocol (see Figure 8).
  - **Indeterminate:** When ask(C_M) is returned. In this case, list the missing credentials in the standard SAML attribute fields, for example,
    <attribute name=``MISSING_CREDENTIAL''>Employee ID</attribute>
    <attribute name=``MISSING_CREDENTIAL''>Full Professor</attribute>

  To make the access decision engine Web Services compatible we also adopted the W3C SOAP (see *http://www.w3.org/TR/soap*) as a main transport layer protocol. SOAP is a lightweight protocol for exchanging structured information in a decentralized, distributed environment. It has an optional Header element and a required Body ele-

ment. Informally, in the body we specify what information is directly associated with the service request and in the header additional information that should be considered by the end-point server.

So, to request for an access decision on a message level we have to:

- First, attach X.509 Certificates in the SOAP Header using WS-Security (WS-Security) specification for that,
- Then, place the SAML Request in the SOAP Body thus making it an input to the decision engine being invoked.

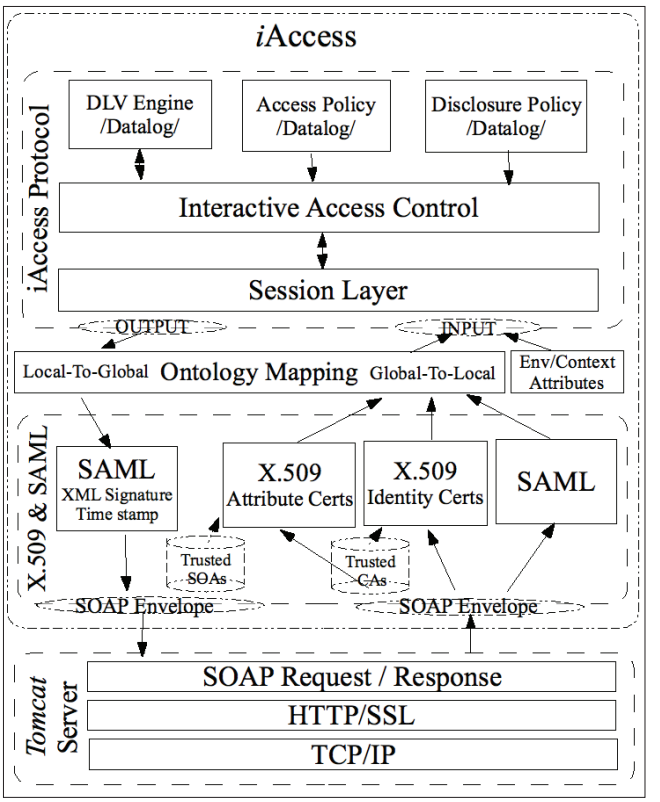
Having the needed technologies in hands, the next section describes how the just introduced standards and protocols can be integrated into one architecture.

System Architecture

Figure 11 shows the architecture of a prototype that has been developed, called *i*Access. The bottom most layer in the figure comprises the integration of the prototype with the Tomcat (see *http://tomcat.apache.org)* application server. We perform all requests over SSL connection. Thus, assuring message confidentiality and integrity on the transport layer.

Once an access request is received by the Tomcat server, it invokes the *i*Access engine for an access decision. As shown in the figure, first, the engine parses the SOAP envelope, containing the body and the header elements. Then, it extracts X.509 (see X.509 technology provider: *http://www.bouncycastle.org*) identity and attribute certificates, and the SAML (see SAML technol-

Figure 11. iAccess architecture



ogy provider: *http://www.opensaml.org*) request protocol. Next, the engine performs validation and verification of the certificates: first for expiration dates and second for trustworthiness. The latter is performed according to local databases listing the trusted identity issuers and, respectively, the trusted attribute issuers (their public keys). The two databases are service provider specific.

Remark 2 *We point out that the check for trusted CAs and SOAs is to filter out those certificates that are issued by unknown (distrusted) certificate authorities. The fine-grained granularity on trusted attributes and identities is performed on the logical level and according to the access policy.*

Once the certificates are validated and verified, *i*Access invokes an ontology alignment module for mapping the global certificate information to a local, provider specific, representation. The same mapping is also performed for the SAML request protocol information.

The ontology mapping transforms global-to-local and local-to-global the following information:

- Certificate attributes
- Certificate issuers
- Resource names (service requests)
- Service actions

These transformations leverage access control management on the logical level because on this level there is local (domain specific) syntax for the representation of the above items.

After the transformation is performed *i*Access invokes the *i*AccessControl module for an access decision. The *i*AccessControl module transforms certificates' information and SAML request to predicates suitable for the logical model, as described below.

- Identity certificates are transformed to certificate(subject, Issuer: *i*) predicates,
- Attribute certificates are transformed to credential(holder, Attr: *a*, Issuer: *i*) predicates,
- SAML access request to grant(Resource: *r*, Action: *p*).

Once an access decision is taken (returned by the *i*AccessControl protocol), *i*Access maps the information grant, deny or additional credentials to their global representation and then generates the respective SAML Response protocol. After that, *i*Access places a time-stamp for validity period on the access decision statement and then digitally signs it to ensure integrity of the information. Next, Tomcat server returns the SAML decision to the entity requested it.

TRUST NEGOTIATION

In an autonomic network scenario servers must have a way to find out what credentials are required for clients to get access to resources. Clients, once asked for the missing credentials, may be unwilling to disclose them unless the server discloses some of its credentials first, that is, negotiate the need of sensitive credentials.

If we merge the two frameworks we have the following open 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 stepwise fashion the missing credentials.

To combine automated trust negotiation and interactive access control we assume that both clients and servers have the three logical security policies:

1. P_{AR} a policy for access to *own* resources on the basis of *foreign* credentials,
2. P_{AC} a policy for access to *own* credentials on the basis of *foreign* credentials,
3. P_{D} a policy for disclosure the need of (missing) *foreign* credentials.

Technically speaking we could merge 1 and 2 into a flat policy for protecting sensitive resources as in (Yu & Winslett, 2003; Yu, Winslett, & Seamons, 2003). However, the 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.

For example the negotiation of a sensitive credential may require activation of credentials that are not considered from the business logic for the actual access control process and even they may be inconsistent with the business logic rules. Thus, forcing separation between policies 1 and 2 we free the access policy P_{AR} to be arbitrarily complex with almost everything that is on the (Datalog) access control market (say with negation as failure, constraints on separation of duties, or other credentials such as those by Li and Mitchell (2003).

Rather, the policy for access to own credentials P_{AC} we restrict to be monotonic because of its particular nature: once the need for a credential is disclosed (granted), it is disclosed! In contrast, a credential needed for access to resources may come and go due to separation of duty or other constraints.

The Negotiation Protocol

This sections shows how one can bootstrap from the simple security policies a comprehensive negotiation protocol that establishes proper trust relationships via bilateral exchange of credentials.

We introduce a new set notation O indicating a set of own credentials with respect to a negotiation opponent.

Now, let us recall the interactive access control protocol with the following modification. Instead of returning the set of missing credentials C_M we will transform in into a sequence of single requests each asking for a foreign credential from the missing set. Figure 12 shows the new version of the protocol.

We extended the protocol to work on client and server sides so that they automatically request each other for missing credentials. Step 1 of the protocol updates the set of active (foreign) credentials with those presented at the time of request. Those presented credentials are typically pushed by the opponent when initially requests for a service. After the initial update we go in a loop where *i*AccessDecision algorithm is run for an access decision.

The purpose of the loop is to keep asking the opponent new solutions (missing credentials) until a final decision of grant or deny is taken. The technicality of the protocol is in step 6 where we represent the request for a missing credential as a remote invocation of the *i*AccessNegotiation protocol on the opponent side. In this way, the new protocol has the same functionality as

Figure 12. The core of the negotiation protocol

```
Session vars: C_P and C_N. Initially C_P=C_N = ∅;.
iAccessNegotiation(r, C_p){
 1: C_P = C_P ∪ C_p;
 2: repeat
 3:   result = iAccessDecision(r, P_A, P_D, C_P, C_N);
 4:   if result = = ask(C_M) then
 5:     for each c ∈ C_M do
 6:       response = invoke iAccessNegotiation(c, ∅)@Opponent;
 7:       if response = = grant then
 8:         C_P = C_P ∪ {c};
 9:       else
 10:        C_N = C_N ∪ {c};
 11:    done
 12:  fi
 13: until result = = grant or result = = deny.
 14: return result;
}
```

*i*AccessControl protocol if the client does not negotiate but just replies whether he has a credential or not.

Step 6 invokes *i*AccessNegotiation protocol with an empty set of presented own credentials. One would ask why we do not push own credentials when requesting for foreign credentials. We simply want to enforce a negotiation process only on the basis of opponents' policies for protecting credentials rather than burdening the process with an additional reasoning for deciding what are one's own credentials that must be pushed for each request for a foreign credential. Of course one can slightly modify the protocol by introducing a function PushedCredentials(*c*) that decides what own credentials an opponent has to present (O_{push}) when requesting for a foreign credential *c*. Since it is not directly relevant for the protocol itself we will omit this function in the rest of the section.

To approach bilateral negotiation first we have to take into account the following two issues:

- Each request for a credential spurs a new negotiation thread that negotiates access to this credential.
- During a negotiation process parties may start to request each other credentials that are already in a negotiation. So, the notion of suspended credential requests must be taken into account.

Figure 13 shows the updated version of the *i*AccessNegotiation protocol. With its new version, whenever a request arrives it is run in a new thread that shares the same session variables C_P, C_N and O_{susp} with other threads running under the same negotiation process. The set O_{susp} keeps track of the opponent's own credentials that have been requested and which are still in a negotiation. We called it a set of one's own suspended credentials meaning that each request for an own credential must be suspended if it appears in O_{susp}.

Now, if a request for a credential, which is already in a negotiation, is received the protocol suspends the new thread until the respective negotiation thread finishes (step 3). Then, when the original thread returns an access decision the protocol resumes all threads awaiting on the requested credential and informs them for the final decision (step 20).

Figure 13. The negotiation protocol with suspended credentials

```
Session vars: C_P and C_N and O_susp. Initially C_P=C_N = O_susp= ∅;.
iAccessNegotiation(r, C_p){ runs in a new thread
 1: C_P = C_P ∪C_p;
 2: if r ∈ O_susp then
 3:   suspend and await for the result on r's negotiation;
 4:   return result when resumed;
 5: else
 6:   O_susp = O_susp ∪ {r};
 7:   repeat
 8:     result = iAccessDecision(r, P_A, P_D, C_P, C_N);
 9:     if result = = ask(C_M) then
 10:       for each c ∈ C_M do
 11:         response = invoke iAccessNegotiation(c, ∅)@Opponent;
 12:         if response = = grant then
 13:           C_P = C_P ∪ {c};
 14:         else
 15:           C_N = C_N ∪ {c};
 16:       done
 17:    fi
 18:  until result = = grant or result = = deny.
 19:  O_susp = O_susp \ {r};
 20:  resume all processes awaiting on r with the result of the negotiation;
 21:  return result;
 22:elseif
}
```

Figure 14 shows the full-fledged negotiation protocol. The *i*AccessDispatcher module manages the negotiation session information. Its role is to dispatch (assign) to each request/response the right negotiation process information. It works in the following way. Whenever a request for a service is received the dispatcher runs *i*AccessNegotiation in a new session process and initializes C_P, C_N and O_{susp} to an empty set (step 2). Then each counter-request for a credential is run in a new thread under the same negotiation process (step 4).

On the other hand, whenever an entity requests a service *r* at the opponent side, presenting initially some own credentials O_p, the *i*AccessDispatcher module invokes *i*AccessNegotiation (at the opponent side) and creates a new session process so that any counter-request from the opponent is run in a new thread under the new negotiation process.

The intuition behind the negotiation protocol is the following:

1. A client, Alice, sends a service request *r* and (optionally) a set of own credentials O_p to a server, Bob.
2. Bob's *i*AccessDispatcher receives the requests and runs *i*AccessNegotiation(*r*, C_p) in a new process. Here $C_p = O_p$ with respect to Bob.
3. Once the protocol is initiated, it updates the over all set of presented foreign credentials with the newly presented ones and checks whether the request should be suspended or not (steps 1 and 2).
4. If no suspended, then Bob looks at *r* and if it is a request for a service he calls *i*AccessDecision with his policy for access to resources P_{AR}, his policy for disclosure of foreign credentials P_D, the set of foreign presented credentials C_P and the set of foreign declined credentials C_N (step 9).
5. If *r* is a request for a credential then Bob calls *i*AccessDecision with his policy for

access to own credentials P_{AC}, his policy for disclosure of foreign credentials P_D, the set of presented foreign credentials C_P and the set of declined foreign credentials C_N (step 11).

6. In the case of computed missing foreign credentials C_M, Bob transforms it into requests for credentials and awaits until receives all responses. At this point Bob acts as a client, requesting Alice the set of credentials C_M. Alice runs the same protocol with swapped roles.
7. When Bob receives all responses he restarts the loop and consults the *i*AccessDecision algorithm for a new decision.

Figure 14. The negotiation protocol

Session vars: C_P, C_N and O_{susp}. Initially $C_P = C_N = O_{susp} = \varnothing$;.
*i*AccessDispatcher{
OnReceiveRequest: *i*AccessNegotiation(r, C_p)
1: if isService(r) then
2: reply response = *i*AccessNegotiation(r, C_p); in a new negotiation session process.
3: else
4: reply response = *i*AccessNegotiation(r, C_p); in a new thread under the original negotiation session.
OnSendRequest: <r, O_p>
1: if isService(r) then
2: result = invoke *i*AccessNegotiation(r, O_p)@Opponent; in a new negotiation session process.
}
*i*AccessNegotiation(r, C_p){
1: $C_P = C_P \cup C_p$;
2: if $r \in O_{susp}$ then
3: suspend and await for the result on r's negotiation;
4: return result when resumed;
5: else
6: $O_{susp} = O_{susp} \cup \{r\}$;
7: repeat
8: if isService(r) then
9: *result* = *i*AccessDecision(r, P_{AR}, P_D, C_P, C_N);
10: else
11: *result* = *i*AccessDecision(r, P_{AC}, P_D, C_P, C_N);
12: if *result* = = ask(C_M) then
13: AskCredentials(C_M);
14: until *result* = = grant or *result* = = deny.
15: $O_{susp} = O_{susp} \setminus \{r\}$;
16: resume all processes awaiting on r with the *result* of the negotiation;
17: return *result*;
18:elseif
}
AskCredentials(C_M){
1: parfor each $c \in C_M$ do
2: *response* = invoke *iAccessNegotiation(c, ∅)@Opponent*;
3: if *response* = = grant then
4: $C_P = C_P \cup \{c\}$;
5: else
6: $C_N = C_N \cup \{c\}$;
7: done
8: await untill all responses are received (await until $C_M \subseteq C_P \cup C_N$);
}

8. When a final decision of grant or deny is taken, the respective response is returned back to Alice.

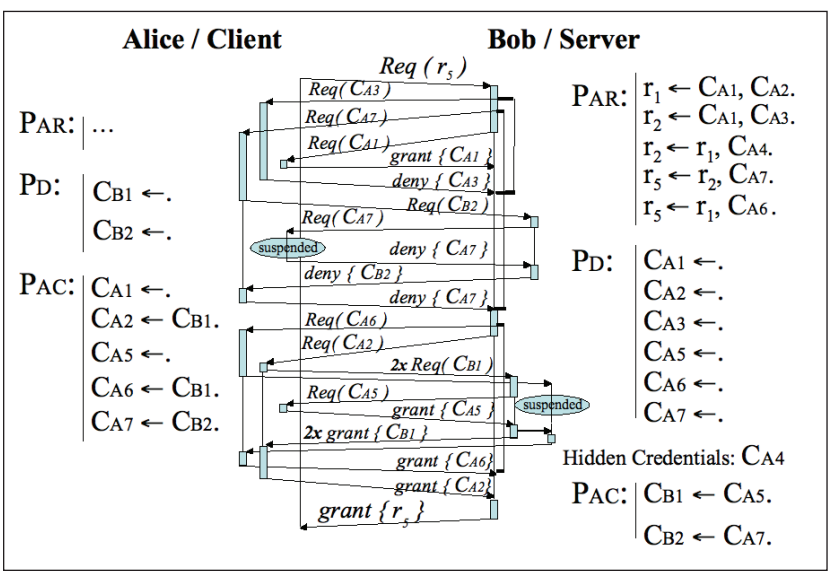
Technicality in the protocol is in the way the server requests missing credentials back to the client. As indicated in the figure, we use the keyword parfor for representing that the body of the loop is run each time in a parallel thread. Thus, each missing credential is requested independently from the requests for the others. At that point of the protocol, it is important that each of the finished threads updates presented and declined sets of foreign credentials properly without interfering with other threads. We note that after a certain session time expires each credential request that is still awaiting on an answer is marked as declined.

Also an important point here is to clarify the way we treat declined and not yet released credentials. In a negotiation process, declining a credential is when an entity is asked for it and the same entity replies to the same request with answer deny. In the second case, when the entity is asked for a credential and, instead 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 the negotiation process the client either supplies the originally asked credential or declines it.

Example 6 *Figure 15 shows an example of Alice's and Bob's interactions using the negotiation protocol on both sides. The policies for access to resources and access to sensitive credentials are in notations like in Yu, Winslett and Seamons (2003) where the Alice's local credentials are marked with subscript "A" and Bob's with "B", respectively. Bob's access policy P_{AR} says that access to resource r_1 is granted if $\{C_{A1}, C_{A2}\}$ are presented by Alice. To get access to r_2 Alice should either present $\{C_{A1}, C_{A3}\}$ or satisfy the requirements for access to r_1 and present C_{A4}. To get access to r_5 Alice should either satisfy the requirements for r_2 and present C_{A7} or satisfy the requirements for r_1 and present C_{A6}.*

We read Bob's disclosure policy as to disclose the need for a credential C_{A2} there should be al-

Figure 15. Example of interoperability of the negotiation protocol



ready disclosed a credential C_{A1}, which by default is always disclosable. But in contrast, the need for a credential C_{A4} is never disclosed by P_{AR} but expected when r_2 is requested. It is an example of a hidden credential that must be pushed.

Analogously, Bob's P_{AC} says: to grant access to Bob's C_{B1} Alice must present C_{A5} and to grant access to Bob's C_{B2} Alice must present C_{A7}.

Following is the negotiation scenario. Alice requests r_5 to Bob presenting empty set of initial credentials. Alice's TN Dispatcher detects the request and creates a new session process awaiting on Bob's reply. Next, Bob runs the interactive algorithm on his P_{AR}. The outcome of the algorithm is the set of missing credentials {C_{A1}, C_{A3}, C_{A7}} (computed as the minimal one). Then, Bob transforms the missing credentials in single requests and asks Alice for them.

Alice's TN Dispatcher receives the requests and runs them in three new threads for each of them, respectively. Next, Alice runs the interactive access control algorithm on her P_{AC} for each of the requests and returns grant C_{A1}, deny C_{A3} and counter request for Bob's C_{B2}. Bob replies to the request for C_{B2} with a counter request for Alice's C_{A7}. Since C_{A7} has been already requested by Bob, now Alice suspends the new request and awaits on the original one to finish its negotiation.

If we look again in the sequence of requests we recognize than the original thread depends on the outcome of the suspended one and we come to a recursive loop (interlock). Since Alice's suspended thread has a session timeout, so after it expires Alice returns to Bob a decision deny. At this point Alice can choose (automatically) to extend her session time to allow the negotiation to continue and eventually to successfully finish.

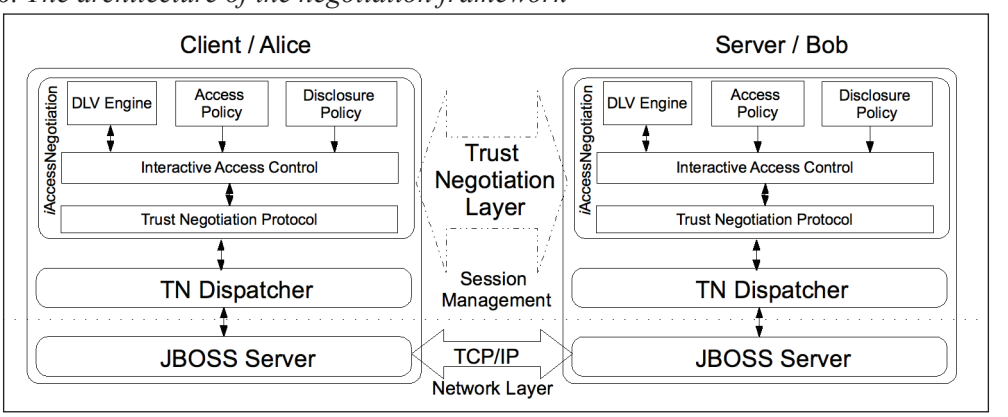
Next, Bob recalls its interactive access control for a new decision for r_5. The next set of missing credentials is {C_{A2}, C_{A6}} which Bob transforms to single requests. The rest of the scenario follows analogously.

After Alice and Bob successfully negotiate on Bob's requests for missing credentials, Bob grants access to the service request r_5.

However, we have not solved the problem of stepwise disclosure of missing foreign credentials yet. The intuition here is that Bob may not want to disclose the missing foreign credentials all at once to Alice but, instead, he may want to ask Alice first some of the less sensitive credentials assuring him that Alice is enough trustworthy to disclose her other more sensitive credentials and so on until all the missing ones are disclosed. Here we point out that the stepwise approach may require a client to provide credentials that are not directly related to a specific resource but needed for a fine-grained disclosure control.

To address this issue we extend the negotiation protocol with an algorithm for stepwise disclo-

Figure 16. The architecture of the negotiation framework



sure 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 a one-step deduction over the disclosure policy P_D to determine the next set of potentially disclosable credentials. We refer the reader to (Koshutanski & Massacci, 2004) for details on the stepwise algorithm.

Implementing the Trust Negotiation Framework

Figure 16 shows the architecture of the trust negotiation framework. JBOSS application server (see *http://www.jboss.org/products/jbossas)* uses TCP/IP sockets to send/receive information. The functionality of the server has been extended with the possibility to transform high-level credential/service requests, understandable by the TN Dispatcher, to low-level raw data requests suitable for transmission over TCP/IP connections.

Whenever the TN Dispatcher is initially run it internally runs the JBOSS application server. So, when the TN Dispatcher is run it resides in the memory awaiting for new requests. Once the JBOSS server receives a request it transforms it from raw data to a high-level representation and automatically redirects it to the dispatcher.

On each received request the TN Dispatcher analyzes the session data from the request and its local database, and acts as following. If no session data is specified in the request then the dispatcher generates new session information (new session data sets, see Figure 14) and runs the negotiation protocol with the new session info. If it exists a session data in the request and the session data correctly maps to the corresponding one in the dispatcher's local database then the dispatcher runs the negotiation protocol under the existing session. We remind that the negotiation protocol is anyway run in a new parallel thread and it internally updates the session information. The Trust Negotiation Protocol uses the JBOSS server methods to send/receive requests.

CONCLUSION

In this chapter we presented a framework on policy-based access control for autonomic communications. The framework is grounded in a formal model with the stable model semantics. The key idea is that in an autonomic network a client may have the right credentials but may not know it and thus an autonomic server needs a way to interact and negotiate with the client the missing credentials that grant access.

We have proposed a solution to this problem by extending classical access control models with an advanced reasoning service: abduction. Building on top of this service, we have presented the key interactive access control algorithm that, in case service request fails, computes on-the-fly missing credentials that entail the request. We have also introduced the notion of disclosable and hidden credentials. The distinction allows servers to dynamically protect the privacy of their policies by specifying which credentials are hidden and which are not and notifying selected clients for that.

We have identified the interactive access control model as a way for protecting security interests with respect to disclosure of information and access control of both server and client sides. We have proposed a protocol for leveraging trust negotiation between two entities involved in an autonomic communication. The protocol communicates and negotiates the missing credentials until enough trust is established and the service is granted or the negotiation fails and the process is terminated. The protocol is run on both client and server sides so that they understand each other and automatically interoperate until a desired solution is reached or denied.

One of the advantages of the approach is that we do not pose any restrictions on partner's policies because the basic computations of deduction and abduction, performed on the policies, do not require any specific policy structure. We have also presented an implementation of the framework using X.509 and SAML standards.

Open Problems and Future Work

Future work is in the direction of characterizing the complexity of the framework. Proving which guarantees the protocol can offer in terms of interoperability, completeness and correctness when applied to a practical policy language is still an open process and will be a subject of future research.

In the direction of mutual negotiation, future work is to explore the interoperability of the negotiation framework with the TrustBuilder prototype (Yu, Winslett, & Seamons, 2003). We believe that this is an important step toward building a secure open computing environment.

ACKNOWLEDGMENT

This work was partly supported by the projects: 2003-S116-00018 PAT-MOSTRO, 016004 IST-FP6-FET-IP-SENSORIA, 27587 IST-FP6-IP-SERENITY, 038978 EU-MarieCurie-EIF-iAccess, 034744 EU-INFSO-IST ONE, 034824 EU-INFSO-IST OPAALS.

REFERENCES

Apt, K. (1990). Logic programming. In J. van Leeuwen (Ed.), *Handbook of theoretical computer science*. Elsevier.

Atluri, V., Chun, S. A., & Mazzoleni, P. A (2001). Chinese wall security model for decentralized workflow systems. In *Proceedings of the Eighth ACM conference on Computer and Communications Security,* 48-57.

Bertino, E., Catania, B., Ferrari, E., & Perlasca, P. (2001). A logical framework for reasoning about access control models. In *Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies (SACMAT),* 41-52.

Bertino, E., Ferrari, E., & Atluri, V. (1999) The specification and enforcement of authorization constraints in workflow management systems. *ACM Transactions on Information and System Security (TISSEC), 2*(1), 65-104.

Bonatti, P., & Samarati, P. (2002). A unified framework for regulating access and information release on the Web. *Journal of Computer Security, 10*(3), 241-272.

Damianou, N., Dulay, N., Lupu, E., & Sloman, M. (2001). The Ponder policy specification language. In *Proceedings of the International Workshop on Policies for Distributed Systems and Networks (POLICY),* 18-38.

De Capitani di Vimercati, S., & Samarati, P. (2001). Access control: Policies, models, and mechanism. In R. Focardi & F. Gorrieri (Eds.), *Foundations of security analysis and design - tutorial lectures* (vol. 2171 of LNCS). Springer-Verlag.

Ellison, C., Frantz, B., Lampson, B., Rivest, R., Thomas, B. M., & Ylonen, T. (1999, September). SPKI certificate theory. *IETF RFC,* 2693.

Gelfond, M., & Lifschitz, V. (1988). The stable model semantics for logic programming. In Kowalski R., & Bowen, K. (Eds.), *Proceedings of the Fifth International Conference on Logic Programming (ICLP'88),* 1070-1080.

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

Kang, M. H., Park, J. S., & Froscher, J. N. (2001). Access control mechanisms for interorganizational workflow. In *Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies,* 66-74.

Koshutanski, H. (2005). *Interactive access control for autonomic systems.* Unpublished doctoral dissertation, University of Trento, Italy.

Koshutanski, H., & Massacci, F. (2004, August). An interactive trust management and negotiation scheme. In *Proceedings of the Second International Workshop on Formal Aspects in Security and Trust (FAST),* 139-152.

Li, J., Li, N., & Winsborough, W. H. (2005). Automated trust negotiation using cryptographic credentials. In *Proceedings of the 12th ACM conference on Computer and communications security,* 46-57.

Li, N., Grosof, B. N., & Feigenbaum, J. (2003). Delegation logic: A logic-based approach to distributed authorization. *ACM Transactions on Information and System Security (TISSEC), 6*(1), 128-171.

Li, N., & Mitchell, J. C. (2003). RT: A role-based trust-management framework. In *Proceedings of the Third DARPA Information Survivability Conference and Exposition (DISCEX III),* 201-212.

Li, N., Mitchell, J. C., & Winsborough, W. H. (2002). Design of a role-based trust management framework. In *Proceedings of IEEE Symposium on Security and Privacy (S&P),* 114-130.

Lymberopoulos, L., Lupu, E., & Sloman, M. (2003). An adaptive policy based framework for network services management. *Plenum Press Journal of Network and Systems Management, 11*(3), 277-303.

Ruan, C., Varadharajan, V., & Zhang, Y. (2003). A logic model for temporal authorization delegation with negation. In C. Boyd & W. Mao (Eds.), *Proceedings of the Sixth International Conference on Information Security (ISC), 2851,* 310-324.

SAML. (2004). *Security assertion markup language (SAML).* Retrieved from http://www.oasis-open.org/committees/security

Sandhu, R., Coyne, E., Feinstein, H., & Youman, C. (1996). Role-based access control models. *IEEE Computer 39*(2), 38-47.

Seamons, K., Winslett, M., & Yu, T. (2001). Limiting the disclosure of access control policies during automated trust negotiation. In *Network and Distributed System Security Symposium*. San Diego, CA.

Shanahan, M. (1989). Prediction is deduction but explanation is abduction. In *Proceedings of IJCAI'89* (pp. 1055-1060). Morgan Kaufmann.

Sloman, M., & Lupu, E. (1999). Policy specification for programmable networks. In *Proceedings of the First International Working Conference on Active Networks,* 73-84.

Smirnov, M. (2003). Rule-based systems security model. In *Proceedings of the Second International Workshop on Mathematical Methods, Models, and Architectures for Computer Network Security (MMM-ACNS),* 135-146.

SPKI. (1999). SPKI certificate theory. *IETF RFC,* 2693.

Weeks, S. (2001). Understanding trust management systems. *IEEE Symposium on Security and Privacy.*

Winsborough, W., & Li, N. (2004). Safety in automated trust negotiation. In *Proceedings of the IEEE Symposium on Security and Privacy,* 147-160.

WS-Security. (2002, April). Web services security (WS-security). Retrieved from *http://www-106.ibm.com/developerworks/webservices/library/ws-secure*

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

XACML. (2004). eXtensible Access Control Markup Language (XACML), from http://www.oasis-open.org/committees/xacml.

Yu, T., & Winslett, M. (2003). A unified scheme for resource protection in automated trust negotiation. In *Proceedings of the IEEE Symposium on Security and Privacy,* 110-122.

Yu, T., Winslett, M., & Seamons, K. E. (2003). Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM Transactions on Information and System Security (TISSEC), 6*(1), 1-42.