

How to Exploit Ontologies in Trust Negotiation

* Travis Leithhead¹, Wolfgang Nejdl², Daniel Olmedilla², Kent E. Seamons¹,
Marianne Winslett³, Ting Yu⁴, and Charles C. Zhang^{3,5}

¹ Department of Computer Science, Brigham Young University, USA
{tleithea, seamons}@cs.byu.edu

² L3S Research Center and University of Hannover, Germany
{nejdl, olmedilla}@l3s.de

³ Dept. of Computer Science, University of Illinois at Urbana-Champaign, USA
{winslett, cczhang}@cs.uiuc.edu

⁴ Dept. of Computer Science, North Carolina State University, USA
yu@csc.ncsu.edu

⁵ Cisco Systems, Inc., USA
cczhang@cisco.com

Abstract. The World Wide Web makes it easy to share information and resources, but offers few ways to limit the manner in which these resources are shared. The specification and automated enforcement of security-related policies offer promise as a way of providing controlled sharing, but few tools are available to assist in policy specification and management, especially in an open system such as the Web, where resource providers and users are often strangers to one another and exact and correct specification of policies will be crucial. In this paper, we propose the use of ontologies to simplify the tasks of policy specification and administration, and to avoid several information leakage problems in run-time trust management in open systems.

1 Introduction

Open distributed environments like the World Wide Web offer easy sharing of information, but offer few options for the protection of sensitive information and other sensitive resources, such as Web Services. Proposed approaches to controlling access to Web resources include XACML [3], SAML [4], WS-Trust [2] and Liberty-Alliance[1]. All of these approaches to trust management rely on the use of vocabularies that are shared among all the parties involved, and declarative policies that describe who is allowed to do what. Some of these approaches also recognize that trust on the Web, and in any other system where resources are shared across organizational boundaries, must be *bilateral*.

Specifically, the Semantic Web provides an environment where parties may make connections and interact without being previously known to each other. In many cases, before any meaningful interaction starts, a certain level of trust must be established from scratch. Generally, trust is established through exchange of information between the two parties. Since neither party is known to the other, this trust establishment process should

* In alphabetical order

be bi-directional: both parties may have sensitive information that they are reluctant to disclose until the other party has proved to be trustworthy at a certain level. As there are more service providers emerging on the Web every day, and people are performing more sensitive transactions (for example, financial and health services) via the Internet, this need for building mutual trust will become more common.

To make controlled sharing of resources easy in such an environment, parties will need software that automates the process of iteratively establishing bilateral trust based on the parties' access control policies, i.e., *trust negotiation* software. Trust negotiation differs from traditional identity-based access control and information release systems mainly in the following aspects:

1. Trust between two strangers is established based on parties' properties, which are proven through disclosure of digital credentials.
2. Every party can define access control and release policies (*policies*, for short) to control outsiders' access to their sensitive resources. These resources can include services accessible over the Internet, documents and other data, roles in role-based access control systems, credentials, policies, and capabilities in capability-based systems. The policies describe what properties a party must demonstrate (e.g., ownership of a driver's license issued by the State of Illinois) in order to gain access to a resource.
3. Two parties establish trust directly without involving trusted third parties, other than credential issuers. Since both parties have policies, trust negotiation is appropriate for deployment in a peer-to-peer architecture such as the Semantic Web, where a client and server are treated equally. Instead of a one-shot authorization and authentication, trust is established incrementally through a sequence of bilateral credential disclosures.

A trust negotiation process is triggered when one party requests to access a resource owned by another party. The goal of a trust negotiation is to find a sequence of credentials (C_1, \dots, C_k, R) , where R is the resource to which access was originally requested, such that when credential C_i is disclosed, its policy has been satisfied by credentials disclosed earlier in the sequence or to determine that no such credential disclosure sequence exists.

The use of declarative policies and the automation of the process of satisfying them in the context of such a *trust negotiation process* seem to be the most promising approach to providing controlled access to resources on the Web. However, this approach opens up new and pressing questions:

1. What confidence can we have that our policies are correct? Because the policies will be enforced automatically, errors in their specification or implementation will allow outsiders to gain inappropriate access to our resources, possibly inflicting huge and costly damages. Unfortunately, real-world policies [9] tend to be as complex as any piece of software when written down in detail; getting a policy right is going to be as hard as getting a piece of software correct, and maintaining a large number of them is difficult as well.
2. How do we avoid information leakage problems in automated trust negotiation? Using very specific policies we may already leak information about what we want

to protect. On the other hand malicious opponents may try to get information which is not relevant to the resource we want to access.

In this paper, we will address these questions by exploring, in section 2, the use of ontologies for providing abstraction and structure for policy specification, and, in section 3, for providing additional knowledge which can be used during the trust negotiation process to avoid giving out too much information during the trust negotiation process.

2 Using Ontologies to Ease Policy Specification and Management

Using structure and abstraction helps for maintaining complex software and it also helps for maintaining complex sets of policies. In the context of the Semantic Web, ontologies provide formal specification of concepts and their interrelationships, and play an essential role in complex web service environments [6], semantics-based search engines [11] and digital libraries [19].

One important purpose of these formal specifications is sharing of knowledge between independent entities. In the context of trust negotiation, we want to share information about credentials and their attributes, needed for establishing trust between negotiating parties. Figure 1 shows a simple example ontology for credential IDs.

Each credential class can contain its own attributes; e.g., a Cisco Employee ID credential has three attributes: name, rank and department. Trust Negotiation is attribute-based and builds on the assumption that each of these attributes can be protected and disclosed separately. While in some approaches (e.g. with X.509 certificates) credentials and their attributes are signed together as a whole by the credential issuer, in this paper we will rely on cryptographic techniques such as [17] which allow us to disclose credentials with different granularities, hiding attributes not relevant to a given policy.

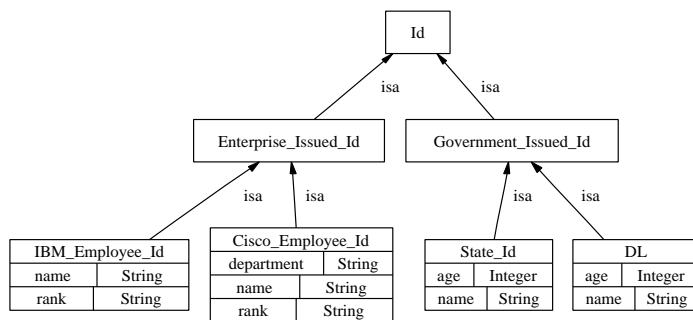


Fig. 1. Simple ID Credential Ontology

In trust negotiation, a party's security policies consist of constraints that the other party has to satisfy; e.g. it has to produce a proof that it owns a certain credential, and

that one of the credential attributes has to be within a certain range. Assuming a casino requires any customer's age to be over 21 and requires a state Id to testify that, the policy for its `admits` service can be represented as⁶:

Casino:

```
allowedInCasino(Requester) ←
  type(CredentialIdentifier, "State_Id") @ Issuer @ Requester,
  issuedFor(CredentialIdentifier, Requester) @ Issuer @ Requester,
  age(CredentialIdentifier, Age) @ Issuer @ Requester,
  Age > 21.
```

In this example, the first two statements in the body of the rule require the requester to prove that he owns a credential of type `State_Id` issued by `Issuer`. If the requester proves that he has it (notice that information about attributes has not been disclosed so far, except for the `issuedFor` attribute), the casino asks for the value of the attribute `age` in the presented credential. Then it verifies whether the requester's age is over 21 and, if successful, admits the `Requester` into the casino.

2.1 Sharing Policies for Common Attributes

Often, credentials share common attributes, and these attributes might share the same policies. Figure 1 shows an example of a simple credential hierarchy, where the concrete credential classes used are depicted in the leaves of the hierarchy. The upper part of the hierarchy represent the different abstract classes: the root represents any ID, which is partitioned into different subclasses according to the issuer of the credential, distinguished between `Government_Issued` and `Enterprise_Issued` IDs. The leaf nodes represent concrete classes which contain the attributes, i.e. `name`, `age`, `rank` and `department`.

This somewhat degenerated hierarchy however does not yet allow for policy re-use. For this we have to exploit attribute inheritance. In our example, all leaf nodes share the `Name` attribute, which therefore can be moved up to the root class `Id`. We are now able to specify common policies for the `Name` attribute at the `Id` level. Similarly, we will move `Rank` up so that it becomes an attribute of `Enterprise_Issued_Id`, and `Age` an attribute of `Government_Issued_Id`. A subclass automatically inherits its superclass's attributes, which might be local or inherited from the superclass's superclass. This leads to the refined ontology as described in figure 2, where each leaf node has the same set of attributes as in figure 1, but inherited from higher levels. This makes it possible to specify shared policies for these shared attributes, similar to method inheritance in object oriented programming languages.

2.2 Composing and Overriding Policies

Now, given such credential ontologies, we can specify security policies at different levels. Being able to inherit and compose these security policies simplifies policy maintenance, though of course we have to distinguish between the case where we compose

⁶ Our examples in this paper are specified using a simplified version of the PeerTrust [16, 14] language

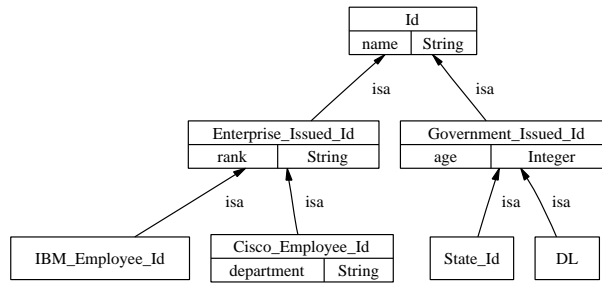


Fig. 2. Refined Ontology

inherited and local policies and the case where the policy specified for an attribute of a specific class overrides the policy inherited from the superclass. In this paper we will describe *mandatory policies* and *default policies*.

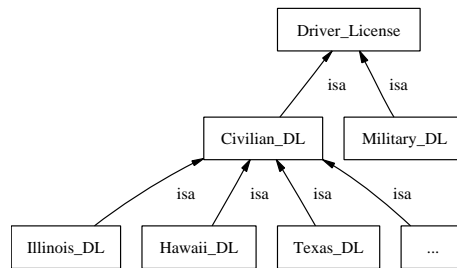


Fig. 3. Driver License Ontology

Mandatory Policies Mandatory policies are used when we want to mandate that policies of a higher level are always enforced at lower levels. Assume the ontology depicted in 3 and that we want to hire an experienced driver to accomplish a certain highly classified and challenging task. Before we show the details of the task to an interested candidate, we want the candidate to present a driver's license, which can be proven to satisfy the following mandatory policies as specified at the different levels:

At the *Driver_License* level, we enforce generic requirements for driver licenses; e.g., a driver license has to be signed by a federally authorized certificate authority and must not have expired.

At the *Civilian_DL* level, we require that the driver license is non-commercial, assuming commercial drivers may have a conflict of interests in the intended task.

At the *Illinois_DL* level, we require that the category of the driver license is not *F*, assuming *F* licenses are for farm vehicles only. At the *Military_DL* level, we

can specify policies such as “the driver license must be for land passenger vehicles” as opposed to fighter planes or submarines.

So for an Illinois driver, the overall policy is: must hold a valid driver license, as qualified by the policy at the `Driver_License` level; must hold a non-commercial driver license, as required by the `Civilian_DL` policy; and the driver license must not be for farm vehicles only. The advantage of using mandatory policies here is twofold: firstly, shared policies such as the generic driver license requirements are only specified once at a higher level, which means a more compact set of policies; secondly, it gives a cleaner and more intuitive logical structure to policies, which makes the policies easier to specify and manage.

Default Policies In this example, we assume that all driver licenses show driving experience (expressed in years of driving). Now suppose that a specific task requires the following policy: in most cases, 4 years’ driving experience is required; however, if the driver comes from Texas, he/she needs only 3 years’ experience (assuming it is harder to get a driver’s license in Texas).

To simplify the specification of this policy, we can use the default policy construct. A parent’s default policy is inherited and enforced by a child if and only if the child does not have a corresponding (overriding) policy. In our example, we can specify at the `Driver_License` level that the driving age has to be at least 4 years; then at the `Texas_DL` level, specify an overriding policy that the driving age has to be at least 3 years.

It’s of interest to note that the same result can be achieved here without using default policies: we can move the shared 4-year mandatory policy down to *every* concrete driver license class except `Texas_DL`, where we require 3 years. However, the power of policy sharing is lost.

3 Using Ontologies to Protect Sensitive Information

In the previous section we have used ontologies to structure credentials and policies, making policy maintenance easier. This section concentrates on how to use ontologies to offer additional protection for sensitive information.

3.1 Avoiding Information Leaking Requests

We assume the ontology presented in figure 2 and an equipment provider which has the following external policy for its big customer Cisco:

```
Cisco Purchase Records:
permits(Requester) $ Requester ←
  type(CredentialIdentifier, “Cisco_Employee_Id”) @ Issuer @ Requester,
  issuedFor(CredentialIdentifier, Requester) @ Issuer @ Requester,
  authorizedEmployee(CredentialIdentifier) @ Issuer @ Requester.
authorizedEmployee(CredentialIdentifier) ←
  rank(CredentialIdentifier, Rank),
```

```
Rank > "Manager".
authorizedEmployee(CredentialIdentifier) ←
  department(CredentialIdentifier, "Sales").
```

This policy gives access to Cisco employees which are either working at the sales department or are at least a manager. If the request for a valid Cisco employee ID is already considered leakage of confidential business information, we can obscure the policy by abstracting it to a higher level in the type hierarchy:

```
Cisco Purchase Records:
permits(Requester) $ Requester ←
  type(CredentialIdentifier, "Enterprise_Issued_Id") @ Issuer @ Requester,
  issuedFor(CredentialIdentifier, Requester) @ Issuer @ Requester,
  type(CredentialIdentifier, "Cisco_Employee_Id"),
  authorizedEmployee(CredentialIdentifier).
```

In general, we can summarize this abstraction process as follows: elevate the type of a required sensitive credential to one of its ancestors, which is more generic and discloses less information when requested from the other party; the policy body stays unchanged except that an additional type check is necessary to offset the abstraction.

3.2 Avoiding Answering Unnecessary Requests

We have focused on protecting sensitive information on behalf of the requester of a resource. Similarly, the party providing resources also wants to disclose only information that is relevant to the task at hand. Ontologies describing standard types of negotiations help accomplish this goal. These ontologies contain properties that will describe typical attributes required in the specified negotiation, without specifying any additional constraints. A simple ontology of this kind is depicted in figure 4. This kind of ontology leads to two additional uses of ontologies: *need-to-know disclosure* and *predisposed negotiation*.

Need-to-Know Disclosure. A negotiator may use properties collected from such ontologies to screen received policies and detect unusual credential solicitations, a technique we call need-to-know disclosure. Need-to-know disclosure occurs when the service provider possesses the credentials and properties necessary to solicit a requester's sensitive property, but the property in question is not relevant to the current negotiation. For example, a trust negotiation agent is entering into a home loan transaction with a legitimate bank. The bank releases a policy requiring a credit card. The requester's trust agent ascertains that the credit card request is not relevant to the home loan negotiation because that property is not found within the ontology for the current negotiation. Though the bank server - acting as a legitimate financial institution - could request the credit card, the home loan transaction as defined by the given ontology, doesn't typically require a credit card. Thus, information that the bank server did not need-to-know is not disclosed. Need-to-know disclosure based on knowledge assimilated with the aid of ontologies facilitates a safeguard to the resource requester against malicious property phishing attacks or poorly formed policies without revealing the presence or absence of the requested property.

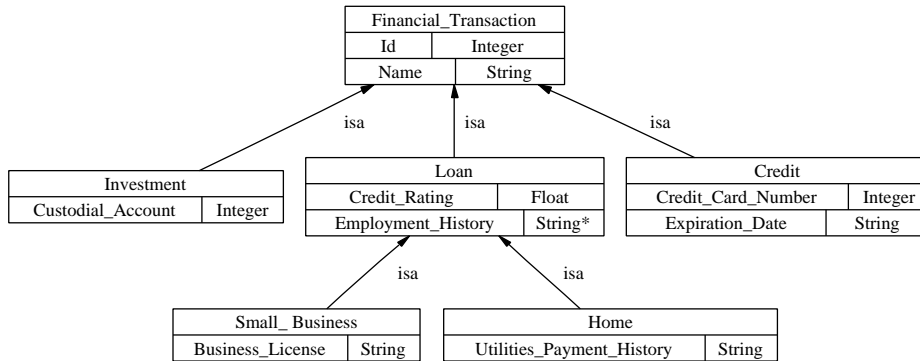


Fig. 4. Negotiation-Type Ontology with Associated Properties for a Typical Financial Transaction

Predisposed Disclosure Strategies. Sometimes a party does not want to wait for specific detailed requests for certificates, but provides information about a set of predisposed credential or policies in one step. This is a modification of the eager strategy described in [21]. Methods similar to those described in need-to-know disclosure provide the basic framework - a set of properties related to a negotiation type which are collected from an ontology (we leave the details of the discovery and collection of these ontologies for future work). Predisposed disclosure involves selecting the credentials or policies that best fit the anticipated needs of the service provider and pushing them along with the resource request. Predisposed disclosure expedites rounds of negotiation in an attempt to satisfy the service provider’s disclosure policy without receiving it. This disclosure technique compliments the eager strategy by narrowing the scope of the credentials disclosed to a more relevant set (preserving the sensitivity of credentials that do not pertain to the negotiation) and ensuring the privacy of sensitive policies, since the eager strategy requires no policy disclosures.

4 Related Work

Recent work in the context of the Semantic Web has focused on how to describe security requirements. KAoS and Rei policy languages [15, 20] investigate the use of ontologies for modeling speech acts, objects, and access types necessary for specifying security policies on the Semantic Web. Hierarchies of annotations to describe capabilities and requirements of providers and requesting agents in the context of Web Services are introduced in [10]. Those annotations are used during the matchmaking process to decide if requester and provider share similar security characteristics and if they are compatible. Ontologies have also been discussed in the context of digital libraries for concepts and credentials [5]. An approach called “most specific authorization” is used for conflict resolution. It states that policies specified on specific elements prevail over policies specified on more general ones. In this paper we explore complementary uses of ontologies for trust negotiation, through which we target iterative trust establishment between strangers and the dynamic exchange of credentials during an iterative trust negotiation

process that can be declaratively expressed and implemented. Work done in [8] defines *abstractions* of credentials and services. Those abstractions allow a service provider to request for example a credit card without specifically asking for each kind of credit card that it accepts. We add to this work in the context of policy specification the concept of *mandatory* and *default* policies.

Ontology-based policy composition and conflict resolving have also been discussed in previous work. Policy inheritance is done by *implication* in [12], but it does not provide any fine-grained overriding mechanism based on class levels. *Default properties* are discussed in [13], short of generalizing the idea to policies. The approaches closest to our default and mandatory policy constructs are the *weak* and *strong* authorizations in [7], where a strong rule always overrides a weak rule, and SPL in [18], which forces the security administrator to combine policies into a structure that precludes conflicts. Compared to these approaches, we find ours particularly simple and intuitive, while its expressiveness well serves general trust negotiation needs.

The concepts forming the basis for need-to-know credential disclosures within automated trust negotiation are suggested in [21], where two dichotomous negotiation strategies are meticulously analyzed: eager and parsimonious. The strength of the eager strategy is its simplicity, yet it discloses all non-sensitive credentials, which raises a privacy concern. A need for a smarter, need-to-know, credential disclosure strategy is recommended in which credentials relevant only to the present negotiation are disclosed. The parsimonious strategy focuses and guides a negotiation, but does so only according to the received credential request policies. Our work relies upon ontologies to provide additional semantics that supplement these negotiation strategies and enable genuine need-to-know disclosure.

5 Conclusions and Future Research Directions

Ontologies can provide important supplemental information to trust negotiation agents both at compile time (to simplify policy management and composition) and at run-time (to avoid certain forms of information leakage for all peers participating in a negotiation). This paper has explored some important benefits of using ontologies.

For compile time usage, ontologies with their possibility of sharing policies for common attributes provide an important way for structuring available policies. In this context we discussed two useful strategies to compose and override these policies, building upon the notions of mandatory and default policies. Further investigations into these mechanisms should draw upon previous work on inheritance reasoning and method composition in object oriented programming languages, and will improve the maintenance of large sets of policies for real-world applications.

For run-time usage, ontologies provide valuable additional information, which can be used to avoid information leaking requests and enable negotiators to detect credential requests irrelevant for the service currently negotiated. Need-to-know disclosure, and predisposed negotiation are two ways to harness ontologies in a useful manner at runtime to provide such privacy benefits.

Need-to-know disclosures make assumptions that require further research: the existence of standard negotiation typing ontologies; implicit trust of the content of such

ontologies and ontologies in general; determination of credential relevancy and efficiency models for trust negotiation which include the overhead of using ontologies. For predisposed negotiation, further work is required to determine how local credential sets are condensed to reduce the scope of a negotiation and to select appropriate credentials to push. Analysis of the effects of predisposed negotiation should focus on the effects of creating a localized set of negotiation parameters (policies and credentials) specific to the scope of the current negotiation, as well as the overall effects of ontology information inference.

Acknowledgments

The research of Nejdil and Olmedilla was partially supported by the projects ELENA (<http://www.elena-project.org>, IST-2001-37264) and REVERSE (<http://reverse.net>, IST-506779). The research of Leithead, Seamons and Winslett was supported by DARPA (N66001-01-1-8908), the National Science Foundation (CCR-0325951, IIS-0331707) and The Regents of the University of California. The research of Yu was partially supported by Faculty Research and Professional Development Fund, NCSU.

References

1. *Liberty Alliance Project*. <http://www.projectliberty.org/about/whitepapers.php>.
2. *Web Services Trust Language (WS-Trust) Specification*. <http://www-106.ibm.com/developerworks/library/specification/ws-trust/>.
3. Xacml 1.0 specification <http://xml.coverpages.org/ni2003-02-11-a.html>.
4. Assertions and protocol for the oasis security assertion markup language (saml); committee specification 01, 2002.
5. N. R. Adam, V. Atluri, E. Bertino, and E. Ferrari. A content-based authorization model for digital libraries. *IEEE Transactions on Knowledge and Data Engineering*, 14(2):296–315, 2002.
6. A. Ankolekar. Daml-s: Semantic markup for web services.
7. E. Bertino, S. Jojodia, and P. Samarati. Supporting multiple access control policies in database systems. In *IEEE Symposium on Security and Privacy*, pages 94–109, Oakland, CA, 1996. IEEE Computer Society Press.
8. P. Bonatti and P. Samarati. Regulating Service Access and Information Release on the Web. In *Conference on Computer and Communications Security*, Athens, Nov. 2000.
9. Cassandra policy for national ehr in england. <http://www.cl.cam.ac.uk/users/mywyb2/publications/ehrpolicy.pdf>.
10. G. Denker, L. Kagal, T. Finin, M. Paolucci, and K. Sycara. Security for daml web services: Annotation and matchmaking. In *Proceedings of the 2nd International Semantic Web Conference*, Sanibel Island, Florida, USA, Oct. 2003.
11. M. Erdmann and R. Studer. How to structure and access xml documents with ontologies. *Data and Knowledge Engineering*, 36(3), 2001.
12. W. Emayr, F. Kastner, G. Pernul, S. Preishuber, and A. Tjoa. Authorization and access control in iro-db.
13. R. Fikes, D. McGuinness, J. Rice, G. Frank, Y. Sun, and Z. Qing. Distributed repositories of highly expressive reusable knowledge, 1999.

14. R. Gavriloaie, W. Nejdl, D. Olmedilla, K. Seamons, and M. Winslett. No registration needed: How to use declarative policies and negotiation to access sensitive resources on the semantic web. In *1st First European Semantic Web Symposium*, Heraklion, Greece, May 2004.
15. L. Kagal, T. Finin, and A. Joshi. A policy based approach to security for the semantic web. In *2nd International Semantic Web Conference*, Sanibel Island, Florida, USA, Oct. 2003.
16. W. Nejdl, D. Olmedilla, and M. Winslett. PeerTrust: automated trust negotiation for peers on the semantic web. In *Workshop on Secure Data Management in a Connected World (SDM'04)*, Toronto, Aug. 2004.
17. P. Persiano and I. Visconti. User privacy issues regarding certificates and the tls protocol. In *Conference on Computer and Communications Security*, Athens, Nov. 2000.
18. C. Ribeiro and P. Guedes. Spl: An access control language for security policies with complex constraints, 1999.
19. S. B. Shum, E. Motta, and J. Domingue. Scholonto: an ontology-based digital library server for research documents and discourse. *Int. J. on Digital Libraries*, 3(3):237–248, 2000.
20. G. Tonti, J. M. Bradshaw, R. Jeffers, R. Montanari, N. Suri, and A. Uszok. Semantic web languages for policy representation and reasoning: A comparison of KAoS, Rei and Ponder. In *2nd International Semantic Web Conference*, Sanibel Island, Florida, USA, Oct. 2003.
21. W. H. Winsborough, K. E. Seamons, and V. E. Jones. Automated trust negotiation. DARPA Information Survivability Conference and Exposition, IEEE Press, Jan 2000.