Self-describing Delegation Networks for the Web

Lalana Kagal, Tim Berners-Lee, Dan Connolly, and Daniel Weitzner
Massachusetts Institute of Technology
Computer Science and Artificial Intelligence Laboratory
Cambridge, MA
{lkagal, timbl, connolly, djweitzner}@csail.mit.edu

Abstract

As the necessity of flexible Web security becomes more apparent and as the notion of using policies for access control gains popularity, the number of policy languages being proposed for controlling access to Web resources increases. Instead of defining a single standard policy language, we believe that there should be a way of embracing different policy languages and of allowing interoperability between systems that use different policy languages. We propose Rein - a policy and delegation framework that is grounded in Semantic Web technologies - to help the Web preserve maximum expressiveness for local policy communities by enabling global interoperability of policy reasoning. Rein provides ontologies for describing policy and delegation networks, and provides mechanisms for reasoning over them, both of which can be used to develop domain and policy language specific access control frameworks for Web resources. The focus of this paper is the delegation mechanisms of the Rein policy framework that support both delegation of authorization and trust. In this paper we give a brief overview of the Rein framework, describe its delegation mechanisms, and illustrate their usefulness through some examples.

1 Introduction

As the Web emerges as one of the most important ways of information dissemination across global boundaries, it becomes obvious that though a convenient and simple framework for search and retrieval of information, the Web lacks easy-to-use and flexible security functionality required by users (we use the term "users" to imply website administrators, application developers, or people in charge of web content) for controlling access to pictures, calendar entries, private blogs, draft documents, etc. Several approaches for access control to Web resources have been proposed such as XACML [21], WS-Policy [5], PeerTrust [15], and Rei [18]. Each approach introduces its own policy language and allows policies to be defined over shared ontologies. Instead of requiring everyone on the Web to conform their description of their policy relationships to a single language, we leverage the power of the Semantic Web to reason across the various languages (such as RDF-S [11], OWL [7], and rule languages) that people use to describe policies. Rein will help the Web preserve maximum expressiveness for local policy communities by enabling global interoperability of policy reasoning.

Rein is a framework for policy specification and reasoning, which exploits the inherently decentralized and open nature of the Web by allowing policies to be combined, extended, and otherwise handled in the same scalable, modular manner as are any Web resources. Resources, their policies and meta-policies (policies about how policies are interpreted), the policy languages used, and their relationships together form *Rein policy networks*. Rein allows entities in these policy networks to be located on local or remote Web servers and to be accessible via Hypertext Transfer Protocol (HTTP). It also allows these entities to be defined in terms of one other using their Uniform Resource Identifiers (URI)[1]. Rein policy networks are described using Rein ontologies and these distributed but linked descriptions are collected off the Web by the Rein engine and are reasoned over to provide policy decisions.

Another important aspect of the Rein framework is that it supports delegation of authorization and trust that allow policies to be less exhaustive and provide decentralized security control. Delegation of authorization is very important to the Web because owners of Web resources may not be able to project who should have access to their resources or pre-establish all desirable requirements for access. This kind of delegation allows permissions on a resource to be propagated by a set of trusted entities without explicitly changing the policy or requirements. In order to support the openness of the Web, the Rein framework also includes

[1] http://www.ietf.org/rfc/rfc3986.txt

delegation of trust such that only trusted information on the Web is accepted and reasoned about. Both kinds of delegation can be either key-based or URI-based and can be used with different policy languages defined in RDF-S and OWL.

Rein provides ontologies for describing policy and delegation networks, and provides mechanisms for reasoning over them, both of which can be used to develop domain and policy language specific access control frameworks for Web resources. Except for resources that that need to be secured, all other entities in Rein policy and delegation networks are self describing i.e. all information required to understand the entity is within the entity or linked from the entity. The relation between a secure resource and its policies is not described within the resource itself because this would require the resource to be accessed to retrieve its access control policy in order to decide whether access should be allowed. However, the policies of a resource are known to the Web server or Guard controlling access to the resource. Figure 1 is an example of a Rein policy network. In the figure, *group.jpg* is an image that has an OWL policy defined over an RDF-S policy language. The next image, *seven.jpg*, uses two policies - the OWL policy used by *group.jpg* and an RDF policy over the same RDF-S policy language. The third image, *jamboree.gif*, uses the same the RDF policy as *seven.jpg*, and *first-pic.jpg* has a policy in N3 rules [9, 10] defined over an RDF-S policy language that has a meta-policy in N3 rules.

Some of the main contributions of Rein include :

- Rein is an open and extensible Web-based approach to representing and reasoning over policies.
- Rein supports different mechanisms for delegation that can be grounded out in domain-specific policy languages defined in RDF-S and OWL.
- It allows flexibility in how sophisticated or expressive the policies can be. For example, a policy can be as simple as a list of users and the resources they can/cannot access whereas another policy can be a set of rules that define access rights in terms of specific attributes of users, resources, and the environment and that use information and inferences from other Web resources.
- Rein provides a unified way for reasoning over policy and delegation networks to make access control decisions.
- Except for resources that need to be secured, all other entities in Rein policy and delegation networks are self-describing.

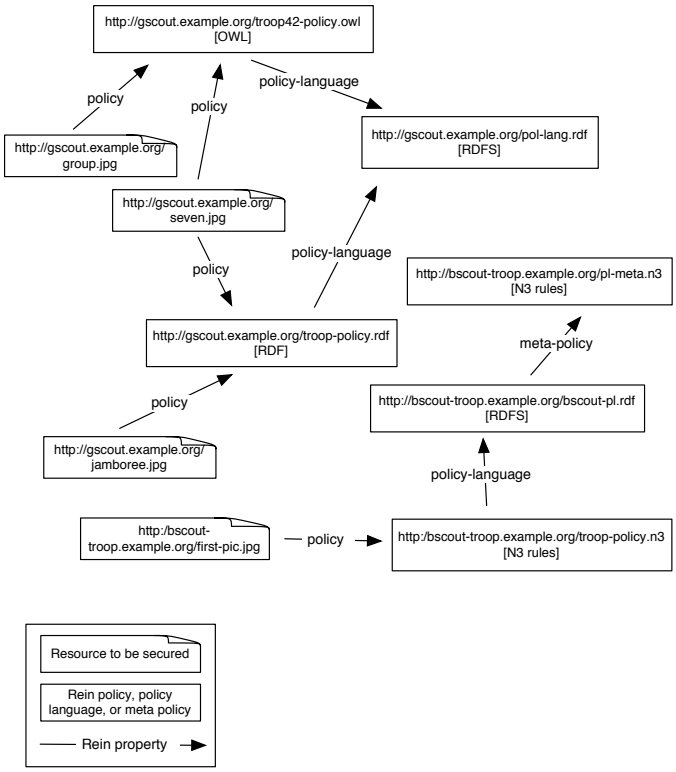

Figure 1. Example of a Rein Policy Network. *Rein policy networks are formed by inter-related resources, policies, policy languages, and meta-policies that can be hosted on different Web servers and that can be extended and re-used as required.*

- Rein supports a compartmentalized approach to policy development as it permits the designing of policy languages, writing of meta-policies associated with policy languages, developing of policies, and enforcing of policies to be modular tasks. This allows policy developers to make frequent changes at their high level of understanding without requiring any other changes to the system.

2 Overview of the Rein Policy Framework

Rein is an open policy framework that can be used in heterogeneous policy domains that use different policy languages and domain knowledge. This domain knowledge can include descriptions of the resources, users, environment, and context in RDF-S or OWL. The domain knowledge can also be defined in XML Schema [14] or XML

DTD [2] such as Common Information Model (CIM XML)[3] as long as the appropriate translation such as XSL Transformations (XSLT) [12] is provided for conversion into RDF-S or OWL.

Rein consists of several ontologies for describing policies, meta-policies, requests, and delegations, and a reasoning engine that accepts access requests for resources in Rein policy networks and decides whether or not the request is valid.

2.1 Rein ontologies

There are three ontologies that are used to model information in the Rein framework namely the policy network ontology, the request ontology, and the delegation ontology as illustrated in Figure 2.

The Rein policy network ontology is made up of three properties that are used to link policy network entities that are located on local or remote hosts via HTTP. The *policy* property is used to associate resources with their access control policies. The *policy-language* property is used by a policy to refer to the policy language(s) it uses, and the *meta-policy* property is used by a policy language to refer to rules that can be used for further policy reasoning. A policy language is represented as an RDF-S or OWL ontology. A meta-policy is a set of rules defined over concepts in a policy language and domain ontologies and is used for additional policy processing such as setting defaults and dynamic conflict resolution. A policy is defined in a policy language and over domain knowledge. It can be a set of RDF-S or OWL instances or a set of rules. A resource could but need not have a description in a machine understandable representation. If it does have a description, it can be used as part of the domain knowledge and policies can be written over this information. Every request for a resource is evaluated against all the policies that control it. If a policy uses a policy language that has a meta-policy, then that meta-policy applies to the policy. Policies, policy languages, and meta-policies can be serialized either in RDF/XML [26] or N3 [8] or described in a supported rule language.

The *Request* class is a way to query a Rein policy and delegation network. *Requests* are created by users or by Web servers from the original user requests to verify whether the access request for the resource is valid. The *Request* class has four properties - *requester* defines the entity making the request, *resource* is the Web resource, service, or action being requested, *access* is a concept (class or property) defined in the policy language for access control (e.g. ReadPermission, can-write, isAble, forbidden, cannot), and *ans* determines whether the request is valid and is set by the engine. Though the *resource* property is a URI

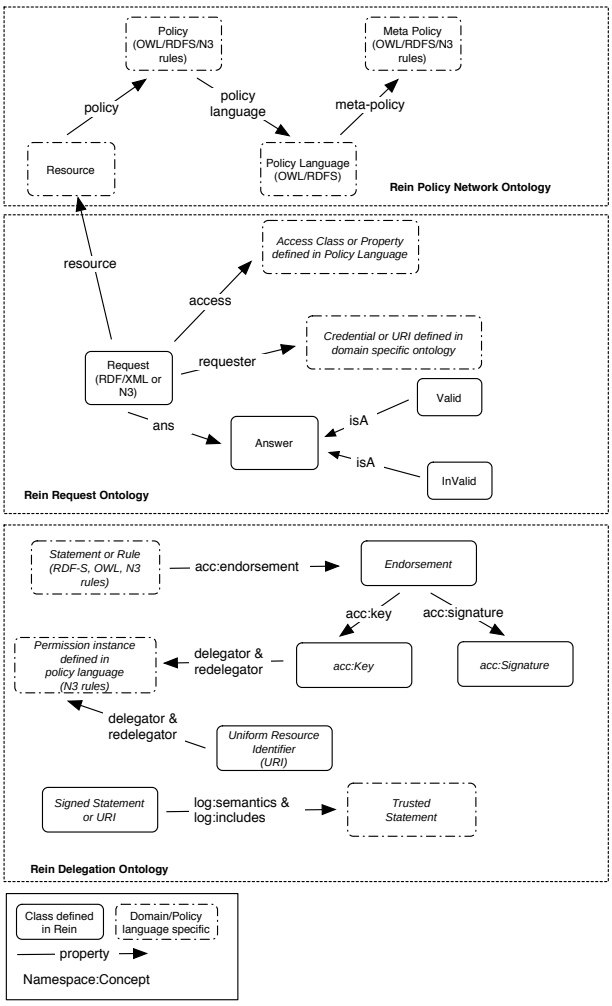

Figure 2. Rein Ontology. *This ontology includes the Rein Policy Network Ontology, which describes the relationships between resources, policies, meta-policies, and policy languages, the Request class, which is used to perform queries over policy networks, and the Rein Delegation Ontology, which is used to describe delegations of authority and trust.*

of the resource being requested, the *requester* property is a set of properties or credentials (such as certificates) of the requester because the identity of the requester might not always be meaningful in open environments such as the Web [17]. The Rein reasoning engine can be modified to handle additional properties for a request including environmental conditions and attributes of the requested resource.

Delegations are described using several concepts in Rein ontologies and some defined in the *acc* namespace[4]. A pub-

[2] http://www.w3.org/TR/REC-xml/

[3] http://www.dmtf.org/standards/wbem/

[4] acc namespace is defined at `http://www.w3.org/2000/10/`

lic key or URI can be given the permission to delegate a permission to access a resource using the *delegator* property. This property associates the key or URI with the permission. The permission itself is defined by the policy language and not within Rein. Similarly, *redelegator* is a property between a key or URI and a permission that gives the entity associated with the key or URI the ability to delegate the permission to delegate. The *acc:endorsement*[5] property is used for cryptographic functionality. It is used to associate a signature (*acc:Signature*) and the key (*acc:Key*) used to generate it. After validating the signature, the Rein engine generates a property, *acc:supportedBy*, that associates the statement with the key used to sign it. These signed statements include requests for Rein resources, delegations of trust and authority, delegation rules, and signed credentials such as name and attribute certificates. The properties used for delegating trust include *log:semantics*[6] and *log:includes*. These are cwm builtins [8] that are used to read the contents of a signed statement or URI and extract trusted information from it. The *log:semantics* builtin accesses a web resource, retrieves a representation of it, parses that, and returns the graph. Currently, cwm will retrieve and parse RDF/XML and N3. The *log:includes* builtin checks whether one graph is a subset of the other. Together, *log:semantics* and *log:includes* allow rules to access the web, and to objectively check the contents of documents, without having to load them and believe everything they say.

2.2 Rein engine

The Rein reasoning engine reasons over Rein policy and delegation networks and is able to answer questions about access rights of clients. It includes a reasoner for RDF-S, OWL, and a reasoning engine for the supported rule languages (e.g. N3 rules and RuleML). It is used by the Guard or Web server controlling access to a resource. It can also be used by clients to generate proofs of why their request should be allowed [19]. It accepts as input an instance of *Request*, the policy controlling the resource, and relevant delegations and signed statements. It is assumed that the Guard has access to the *policy* property of the requested resource and that the policies are accessible via HTTP. The engine processes a *Request* by finding each policy associated with the resource and reasoning over its network including its policies, meta-policies, and delegations, as well as attached delegations and signed credentials. Once the engine obtains the result, it checks whether the result includes any relationship between the *resource*, *requester*, and *access*. For example, it checks whether *requester* has a property *access* with the value of *resource*, or whether there exists an instance of *access*, which has as property values the *resource* and the *requester*. As policy languages are in RDF-S or OWL and as the engine understands the semantics of RDF-S and OWL, it is able to look for possible meaningful relationships between the *resource*, *requester*, and *access*. If the engine is able to find an appropriate relationship, then the engine infers that the *Request* is valid.

2.3 Implementation

The conceptual design of Rein allows it to be implemented in several different ways using different programming languages (e.g. python, Java, C++), reasoners (e.g. Pellet [23], Jena [4], cwm [9]), and rule languages (e.g. RuleML [3], N3 rules [9, 10], Flora [1, 28]). We have implemented it in one possible way using python, cwm, and N3 rules. The current implementation of Rein relies on N3 semantics and cwm functionality to integrate and reason over Rein policy and delegation networks. We chose N3 rules for several reasons; (i) N3 rules allow policies to access the web and objectively check the contents and inferences of documents, without having to load them and believe everything they say. This is especially important in open untrusted environments such as the web because a certain web page may be trusted for a certain bit of information but not all the information it contains. (ii) There are several useful cwm builtins for cryptography, string functionality, and math operators that are required for specifying policies. (iii) The rule language has the expressivity we required and is convenient to read and write.

We have defined the Rein ontologies in RDF-S. We have developed a Rein reasoning engine in N3 rules and use cwm as both a reasoning engine for the supported rule language (N3 rules) and as a reasoner for the language of development. The engine includes the Euler rules [24] for reasoning over RDF-S and a subset of OWL. The engine accepts as input a *Request* instance, the relationship between the requested resource and its policies, and the delegations (if any). The input can be serialized either in RDF/XML or N3. On receiving the input, the engine parses it to get the attributes of the requester and the requested resource. The engine uses cwm builtins to read in the associated policies, policy languages, and meta-policies (if any). It then reasons over the files defined in RDF-S or OWL using the Euler rules whereas files defined in N3 and N3 rules are handled by cwm. The results of the policy are passed to the meta-policy and final result is output by stating whether the *Request* is *Valid* or *Invalid*. This output can be serialized in RDF/XML or N3 and is used by the Guard or Web server to decide whether to allow or deny the request. However,

`swap/test/crypto/acc.n3`

[5] acc:concept implies that the concept is defined in the acc namespace at `http://www.w3.org/2000/10/swap/test/crypto/acc.n3`.

[6] log:semantics implies that semantics is a concept defined in the log namespace at `http://www.w3.org/2000/10/swap/log`.

as the Rein engine can be used both by the Guard and the client, the engine has another output. The engine can be run in the *–why* mode, which causes it to output a proof in N3 for why a *Request* is *Valid*.

Though a resource can have several policies acting on it, we currently support only disjunction of policies - if the request is valid in any one of the policies, it is considered valid. We will look into supporting conjunction of policy decisions as well as defining meta-policies over multiple policies in the future.

A requirement in the Rein framework is that the Guard or Web server needs to know which policies apply to which resources. In our implementation this can be done in using RDF-S, OWL, or N3 rules. If RDF-S or OWL is used, policies can be associated with resources using individual statements for each resource or using restrictions. The use of N3 rules allows policies to be assigned to resources grouped by common attributes.

The engine can be accessed directly by a Guard, a server, or client through the cwm commandline interface or its Application Program Interface (API) in python[2].

3 Delegation in Rein

In keeping with the motivation of Rein, delegation is also not coupled to any policy language. Rein includes a high level ontology for talking about delegations of authorizations and trust and allows domains to use their own policy language. The delegation mechanisms themselves are in N3 rules. A policy decision about whether a request should be granted is dependent on the (i) policy(ies), (ii) meta-policy(ies), (iii) credentials of the requester, (iv) delegations of authority, and (v) delegations of trust. Rein supports signed credentials in OWL, RDF-S, and N3 rules, and delegations in N3 rules described over policy languages in OWL, RDF-S, and N3 rules. Delegations can be self-describing. They can perform authentication and check online white and black lists including certificate and delegation revocation lists making delegation management more modular and easier.

3.1 Delegation of Authority

Delegation of authority in Rein is basically a way to distribute the policy of a Web resource and to allow more than one entity to be responsible for propagating access to the resource. Rein provides two properties for describing delegations - *delegator* and *redelegator* - but allows the permission being delegated to be policy language specific. Rein distinguishes between the permission to delegate a permission and the permission to delegate the permission to delegate. For example, Judy, a professor, delegates to Alice, her student, the permission to write a review for a paper and also delegates the permission to further delegate the permission to write the review. This implies that Alice can either review the paper herself or delegate the permission to someone else. An important property of these delegations is that even though the delegations maybe syntactically correct, they may not be valid. A delegation is only valid if the delegator has the permission to make the delegation; this holds for both delegation of a permission and delegation of the permission to delegate.

Delegations can be made to a key or a document identitied by a URI or a group of keys or several documents identified by URIs. For *key-based delegations*, Rein supports an approach similar to Simple Public Key Infrastructure (SPKI) [7] where keys are associated directly with permissions and the requester needs to own one of these keys in order to get access. The requester proves ownership of the key by signing the *Request*. There is a possibility of replay attack where someone can intercept the signed *Request* and use it to gain access. However, we believe using techniques such as time stamps and a secure channel will mitigate this problem. In order to model key-based delegations Rein provides two properties for defining delegation properties *delegator* and *redelegator* and a property for describing signatures namely *endorsement* that associates a signature and the key used to generate the signature. When a delegation is made to a key, requests and further delegations from that key are expected to be signed. The Rein engine checks the signature of the request, delegations, and attached signed credentials. A delegation is considered valid if the signature is valid and the key that made the delegation has the permission to delegate. A signed request is valid if its signature is valid and there is a valid delegation chain from the resource to the key used to sign the request.

In *URI-based delegations*, a document identified by the URI is delegated the permission to delegate or redelegate. It is assumed that this document can be modified by a set of trusted people. This is an indirect delegation to those people. Rein uses the *delegator* and *redelegator* properties in a similar fashion as key-based delegations to associate a URI with a permission. Delegations made to a URI cause the Rein engine to read in the contents of the URI and reason over related delegations and redelegations it includes. A delegation on a URI is valid if there is valid delegation chain to that URI. An unsigned request is valid is there is a valid delegation to it from a URI or if the requester is permitted by the policy.

Delegation Example 1 : Key-based Delegation A boy scout troop has a set of photos online that are managed by a policy administrator. The policy administrator defines a policy that permits certain keys to access certain pictures. The policy administrator is going on leave and permits

[7] http://world.std.com/ cme/html/spki.html

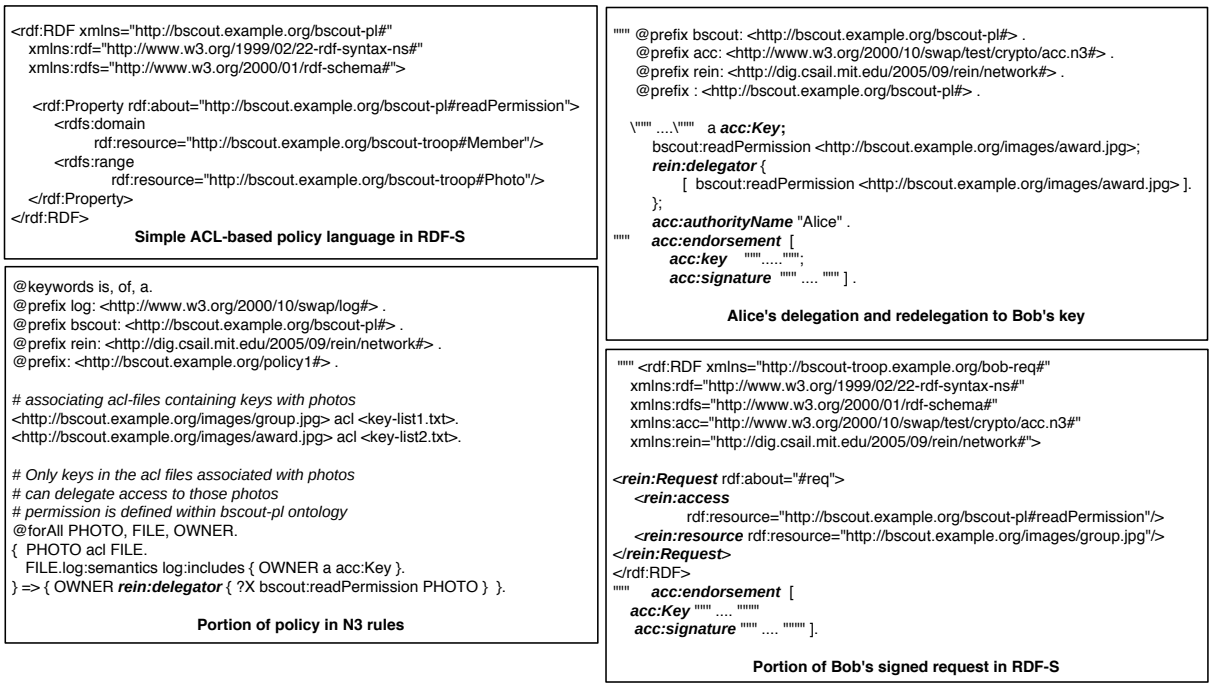

Figure 3. Key-based Delegation. *This figure includes an RDF-S policy language for describing permissions about photos, a portion of the policy that describes delegation to a set of keys, the signed delegation from Alice to Bob's key, and a portion of Bob's signed request. The policy uses the delegator property to assign delegation permissions to a set of keys (including Alice's). Alice's delegation gives Bob's key the permission to access the picture as well as the permission to delegate this permission. However, as Alice does not have the permission to redelegate, any delegation from Bob will be invalid.*

a set of keys to be responsible for defining access to different pictures. The owner of one of these delegated keys, Alice, makes a delegation to a key belonging to Bob. She permits Bob's key to access a picture `http://bscout.example.org/images/award.jpg` and to further delegate this permission. However, as Alice does not have the permission to redelegate, any delegation by Bob will be invalid. Bob wants to access the picture. He creates a *Request* with *resource* set to `http://bscout.example.org/images/award.jpg` and *access* set to `http://bscout.example.org/bscout-pl#ReadPermission`. He signs this request with his key and sends the signed request and the delegation from Alice to the Guard of the photos. The Guards sends this information to the Rein engine. The Rein engine reasons over the policy of the photo, the delegation made to Alice in the policy, the delegation made from Alice and Bob, and Bob's signed request. The Rein engine decides that the Request is valid. Figure 3 includes a portion of this example. The figure shows the policy language in RDF-S that is used for describing permissions about photos, a portion of the policy that describes delegation to a set of keys, part of Bob's request, and the signed delegation from Alice to Bob's key.

4 Delegation of Trust

Along with delegating authority, delegation of trust is important on the Web in order to distribute responsibility for authenticating credentials and to decide what information on the Web is trustworthy while making a policy decision. Keys or URIs are only trusted with respect to a certain credential or attribute of the requester or with respect to particular information. Trust is not transitive i.e. if a URI is trusted with information about members of a troop and the URI trusts a key, signed statements from the key are not automatically trusted.

In order to delegate trust to a key, Rein uses cwm builtins *log:semantics* and *log:includes*, and the Rein *supportedBy* property. In case a key is delegated the responsibility of vouching for credentials, the requester is responsible for obtaining the correct signed credential and attaching it to the

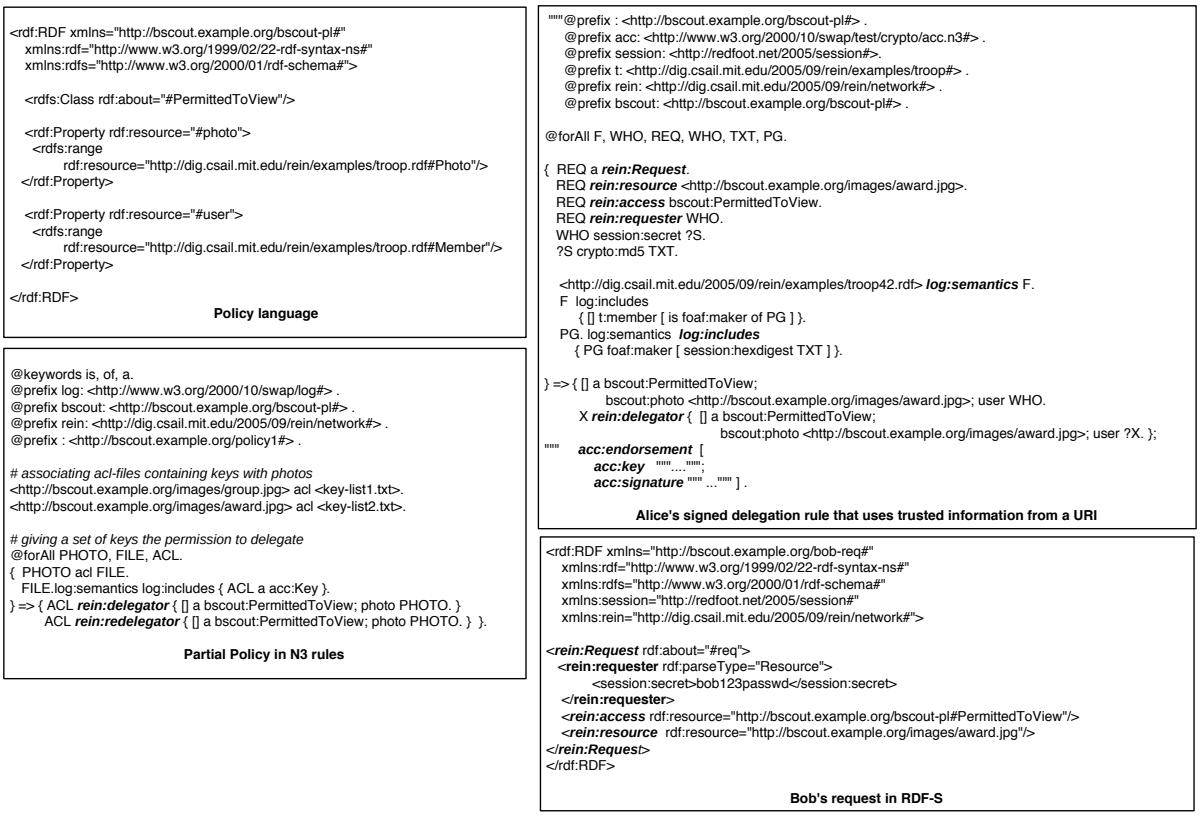

Figure 4. Key-based Delegation Rule. *This figure includes an RDF-S for describing permissions about photos, a portion of the policy that describes delegation to a set of keys (including Alice's), the signed rule-based delegation from Alice, and Bob's request. Alice's delegation gives members of the troop the permission to access the picture. Alice's delegation also uses information from a URI in order to authenticate Bob's credentials.*

Request. The Rein framework reads the attached signed credentials, verifies the signatures, extracts only the credential portions, and reasons over them to decide whether the request is valid. If a URI is trusted with certain information, the Rein framework uses *log:semantics* to read in its contents and *log:includes* to extract the trusted part of the page.

Delegation Example 2 : Key-based delegation rule with delegation of trust to a URI We extend Example 1 to include delegation rules. The policy administrator gives Alice the permission to delegate the permission to view the picture as well as the permission to redelegate the permission to delegate. Inside of specifically giving Bob's key the permission to access the picture, Alice defines a rule giving this permission to a set of entities based on certain attributes. In this case the attributes need to be validated and Alice's delegation of trust allows a URI to be trusted with this information. Alice decides to validate the requester's password using a URI that defines the members of the scout troop. The URI is the troop page and it lists current members and their foaf pages. Alice assumes that the requester is a member of the troop if the secret password provided with the *Request* can be verified against the foaf:hexdigest of the foaf page of one of the members of the troop. Bob's request does not need to be signed but it must include his password. Figure 4 illustrates this example. We use a different policy language to demonstrate the range of policy languages that Rein can support.

Delegation Example 3 : URI based delegation with delegation of trust to a key We restate our earlier example using a delegation of authority to a URI `http://bscout.example.org/alice-policy.n3` and the delegation of trust to a key. We use the same policy language as example 2 (Please refer to figure 4). The policy administrator gives a certain URI the permission to delegate access to a set of photos taken at the jamboree. This URI has a rule that gives any member of the troop the permission to access

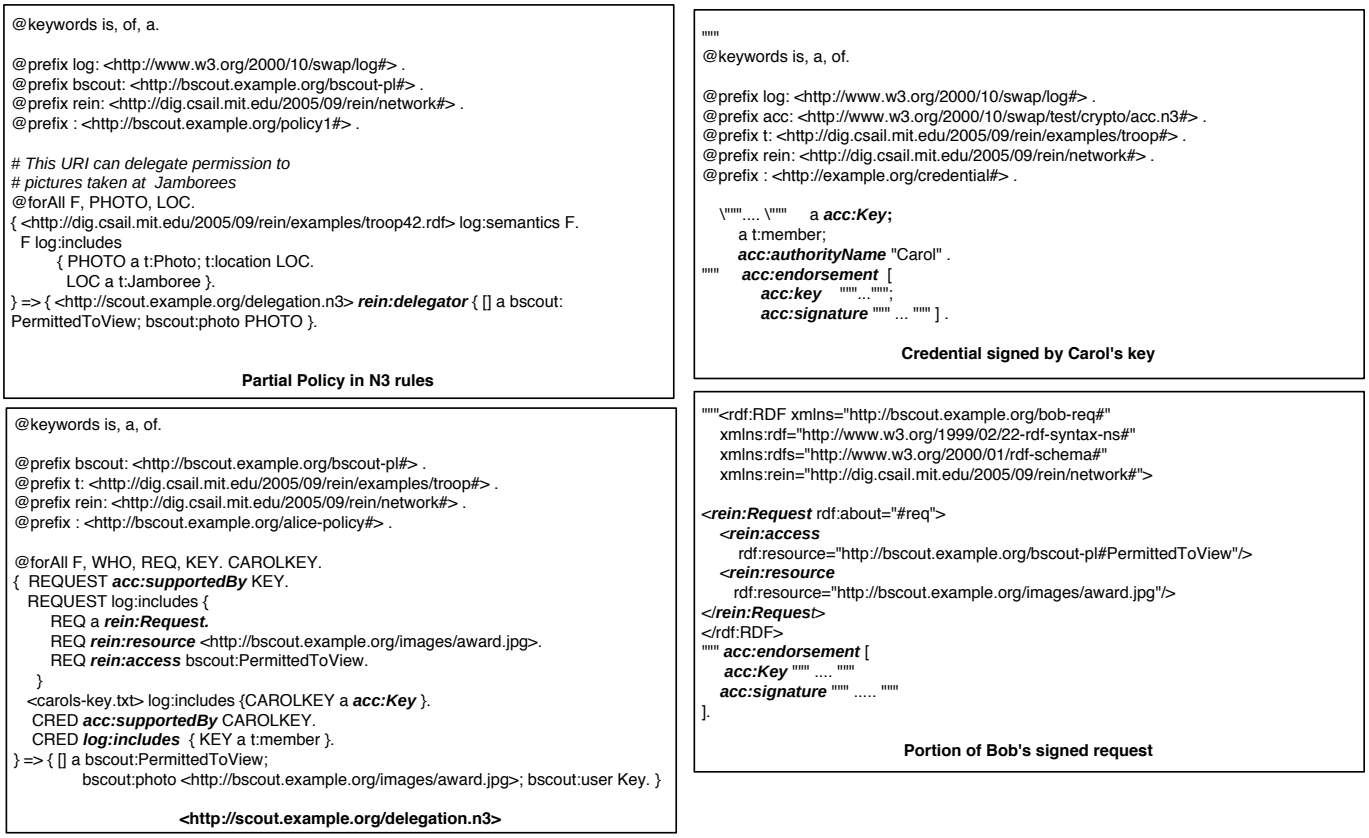

Figure 5. URI-based Delegation. *This is an example of a delegation to a URI and a delegation of trust to a key. If a URI is made a* **delegator** *by a policy, the Rein engine reads in the contents of URI and uses any delegations in it. The URI delegates trust to a key, namely Carol's key. Carol creates a signed statement that Bob's key is a member of the troop. Bob sends his signed request along with Carol's signed statement to the Rein engine and his request is allowed.*

a certain photo as long as Carol believes that they are members. This is a delegation of trust to Carol's key. It implies that the request must be signed with a key and a credential signed with Carol's key stating that the owner of key used to sign the request is a member of the troop must be attached to the request. Figure 5 shows the delegation to the URI by the policy, the rule defined on the URI, the signed request from Bob, and the credential signed by Carol's key. In this example, Carol explicitly states that Bob's key is a member. However, it is also possible to have rules in signed credentials whose inferences are assumed to be signed by the authority's key.

5 Delegation Use Case

In order to explain the extremely general policy and delegation framework of Rein, we describe a use case that we believe will be a common scenario on the Web. We believe that delegation and authentication rules will be accessible to the users but that it will be the responsibility of the users to put them together in order to prove that they fulfill the policy associated with a resource thereby giving them access to the resource.

An example of this kind is the process of obtaining a discount at a car rental agency. Car rental agencies usually have discounts for various things such as AAA membership, corporate negotiations, coupons, and promotions. When a client asks for a discount, the car rental agency cannot (and usually does not) ask for all possible credentials such as AAA memberships, proof of affiliation with companies that they have negotiated a discount with, coupons in newspapers, etc. In this case, certain parts of the policy are known through the client's web of relationships. The client's company will inform her/him of possible discounts with car rental companies, AAA will inform her/him about their discounts, particular newspapers will advertise the car

rental agency's coupons. The client's company will have its own way of authenticating the client and his credentials (e.g. through a digital ID card) and AAA will have its own way (e.g. by possessing a AAA card). The client will obtain a proof of why she/he should get a discount by putting together different rules and policies from different domains (company, AAA, the car rental agency) and provide it to the car rental agency. We believe that this will be the case on the Web - some portions of policies will be known and some will unknown or private. A client will have to collect information from trusted sources to develop a proof using Rein for why she/he should have access to a certain resource or get a certain discount. In order to support this scenario, a general policy framework such as Rein is required that is able to work with different kinds of policy languages and delegation mechanisms.

6 Related Work

Proof Carrying Authorization (PCA) proposes that the underlying framework of a distributed authorization system be a higher-order logic and that different domains in this system use different application-specific logics that are subsets of the higher-order logic [6]. They also propose that clients develop proofs of access using these application specific logics and send them to servers to validate. Rein draws inspiration from PCA but modifies it to leverage the distributed nature and linkability of the Web and the power of Semantic Web technologies.

Extensible Access Control Markup Language (XACML) [21] is a policy language in XML for expressing policies. It can be used to describe policies and rules in terms of boolean combinations of attribute-value pairs based on the subject, resource, and environment. It also provides conflict resolution through its combining algorithms. The Platform for Privacy Preferences (P3P) is a standard developed by the World Wide Web Consortium (W3C) that enables websites to describe their privacy policies and allows browsers to reason over these policies to decide whether they match the user's preferences [13]. KAoS is a policy language developed in OWL [25]. It is similar to Rei [17] and can be used to develop positive and negative authorization and obligation policies over actions. KAoS policies are OWL descriptions of actions that are permitted (or not) or obligated (or not). This limits the expressive power, so that there are policies that can described in N3 rules that KAoS cannot. Using OWL, however, allows the classification of policy statements, enabling conflicts to be discovered from the rules themselves.

We consider XACML, P3P, KAoS, and Rei to be specific policy languages that can be used within Rein policy networks to describe policies. If their semantics can be represented in RDF-S, OWL, or N3 rules, it will be possible to integrate them seamlessly into the current Rein implementation.

RuleML [3] is a proposed standard for a rule language, based on declarative logic programs. SWRL is a closely related rule language standard for describing Horn like rules in first order logic, intended to be combined with OWL-DL [16]. SWRL's syntax for Horn rules is essentially a subset of RuleML's. Unlike SWRL, which is based on classical logic, full RuleML can represent non-monotonicity and conflict handling because it is based on logic programs that have negation as failure and, in the courteous extension, priorities. Rule Interchange Format (RIF) is a W3C activity aimed at developed a standard rule language for the Web [27]. As part of our future work, we will look into representing policies in RuleML, SWRL, and RIF and using these rule languages to provide functionality similar to Rein.

PeerTrust provides a mechanism for gaining access to secure information/services on the web by using semantic annotations, policies and automated trust negotiation [15]. It defines digital credentials as online equivalents of paper credentials that make trust establishment possible. These credentials are signed assertions by the issuer about the properties of one or more entities. In PeerTrust, trust is established incrementally through an iterative process that involves gradually disclosing credentials and requests for credentials. PeerTrust's policy language for expressing access control policies is based on definite Horn clauses. An entity has its own access control rules for the resources it controls and may use rules from other entities in its proofs. Rules may also be signed for authenticity but the verification of signature is not handled by the PeerTrust framework. PeerTrust is a very interesting approach that expects both parties to exchange credentials in order to trust each other and assumes that policies are private. But PeerTrust introduces its own policy language whereas Rein is a policy framework that tries to support different policy languages. Also PeerTrust trusts all information within verified signed statements but in Rein selected subgraphs within signed statements can be trusted. Rein is a basically representation framework for policies and does not include protocol for policy exchange or enforcement like PeerTrust. We would, however, like to support an iterative disclosure approach such as PeerTrust within the Rein framework in the future.

7 Summary and Future Work

Rein is an open web-based policy and delegation framework, which supports heterogeneous domains that use different policy languages and domain knowledge. Rein provides ontologies for describing policy networks, delegations, keys, and signatures and provides mechanisms for reasoning over them, both of which can be used to develop domain and policy language specific frameworks for pro-

viding access control to Web resources.

Though only valid delegators and redelegators can make valid delegations, there is no way of controlling to whom a delegator can delegate to. This is something we would like to address.

Another problem is that Rein does not take possible attacks such as denial of service into consideration and imports all required rules.

Though cwm provides a way to trust certain pieces of information for certain purposes, it does not provide sandboxing for reasoning over untrusted rules. We believe some kind of mechanism will be required so that the computing resources available to untrusted rules are restricted.

The current querying mechanism using *Requests* is very and only checks whether a requester has a certain access type on a resource. We would like to extend that to allow different kinds of queries such as who is permitted to perform a printing kind of service, what kind of resources can John access etc.

Acknowledgements

This work is sponsored by the National Science Foundation Awards 0427275 and 0524481.

References

[1] Flora-2: An Object-Oriented Knowledge Base Language. http://flora.sourceforge.net.

[2] The Official Python Programming Language Website. http://www.python.org/.

[3] The Rule Markup Initiative. http://www.ruleml.org/.

[4] Jena : A semantic web framework for java. http://jena.sourceforge.net/, 2005.

[5] Web Services Policy Framework (WS-Policy). http://www-106.ibm.com/developerworks/library/specification/ws-polfram/, March 2006.

[6] A. W. Appel and E. W. Felten. Proof-Carrying Authentication. In *6th ACM Conference on Computer and Communications Security*, 1999.

[7] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL Web Ontology Language Reference, W3C Recommendation. http://www.w3.org/TR/owl-ref/, February 2004.

[8] T. Berners-Lee. Notation 3 (N3) A readable RDF Syntax. http://www.w3.org/DesignIssues/Notation3.html, 1998.

[9] T. Berners-Lee. Cwm : General-purpose Data Processor for the Semantic Web. http://www.w3.org/2000/10/swap/doc/cwm, 2000.

[10] T. Berners-Lee, D. Connolly, E. Prud'homeaux, and Y. Scharf. Experience with N3 rules. In *W3C Workshop on Rule Languages for Interoperability*, April 2005.

[11] D. Brickley and R.V.Guha. RDF Vocabulary Description Language 1.0: RDF Schema, W3C Recommendation. http://www.w3.org/TR/rdf-schema/, February 2002.

[12] J. Clark. XSL Transformations (XSLT) Version 1.0. W3C Recommendation. http://www.w3.org/TR/xslt, November 1999.

[13] L. Cranor, M. Langheinrich, M. Marchiori, M. Presler-Marshall, and J. Reagle. The Platform for Privacy Preferences 1.0 (P3P1.0) Specification. W3C Recommendation. http://www.w3.org/TR/P3P/, April 2002.

[14] D. Fallside and P. Walmsley. XML Schema Part 0: Primer Second Edition. W3C Recommendation. http://www.w3.org/TR/xmlschema-0/, October 2004.

[15] 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 European Semantic Web Symposium, May. 2004, Heraklion, Greece*, 2004.

[16] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: Semantic Web Rule Language Combining OWL and RuleML. http://www.daml.org/rules/proposal/, 2004.

[17] L. Kagal. A Policy-Based Approach to Governing Autonomous Behavior in Distributed Environments. Dissertation, September 2004.

[18] L. Kagal, T. Finin, and A. Joshi. A Policy Based Approach to Security for the Semantic Web. In *Second Int. Semantic Web Conference (ISWC2003), Sanibel Island FL*, October 2003.

[19] V. Kolovski, Y. Katz, J. Hendler, D. Weitzner, and T. Berners-Lee. Towards a Policy-Aware Web. In *Semantic Web and Policy Workshop at the 4th International Semantic Web Conference*, 2005.

[20] Liberty Alliance: Digital Identify Defined. http://www.projectliberty.org/, 2006.

[21] H. Lockhart, B. Parducci, and A. Anderson. OASIS eXtensible Access Control Markup Language (XACML). http://www.oasis-open.org/committees/tc-home.php, February 2005.

[22] P. Mishra, H. Lockhart, S. Anderson, J. Hodges, and E. Maler. OASIS Security Services (Security Assertions Markup Language) . http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security, 2006.

[23] B. Parsia and E. Sirin. Pellet : An OWL DL Reasoner. In *International Semantic Web Conference, Poster Session*, 2004.

[24] J. D. Roo. Euler proof mechanism. http://www.agfa.com/w3c/euler/, 2005.

[25] A. Uszok, J. M. Bradshaw, R. Jeffers, M. Johnson, A. Tate, J. Dalton, and S. Aitken. Policy and Contract Management for Semantic Web Services. In *AAAI Spring Symposium, First International Semantic Web Services Symposium*, 2004.

[26] W3C. RDF/XML Syntax Specification (Revised). http://www.w3.org/TR/rdf-syntax-grammar/, 2004.

[27] W3C RIF Working Group. Rule interchange format. http://www.w3.org/2005/rules/Overview.html, 2006.

[28] G. Yang and M. Kifer. Implementing an Efficient DOOD System Using a Tabling Logic Engine. In *Intl. Conference on Computational Logic*, July 2000.