Using Semantic Web Technologies for Policy Management on the Web*

Lalana Kagal and **Tim Berners-Lee** and **Dan Connolly** and **Daniel Weitzner**
Massachusetts Institute of Technology
Computer Science and Artificial Intelligence Lab
Cambridge, MA 02139
{lkagal, timbl, connolly, djweitzner}@csail.mit.edu

Abstract

With policy management becoming popular as a means of providing flexible Web security, the number of policy languages being proposed for the Web is constantly increasing. We recognize the importance of policies for securing the Web and believe that the future will only bring more policy languages. We do not, however, believe that users should be forced to conform the description of their policy relationships to a single standard policy language. Instead there should be a way of encompassing different policy languages and supporting heterogeneous policy systems. As a step in this direction, we propose Rein, a policy framework grounded in Semantic Web technologies, which leverages the distributed nature and linkability of the Web to provide Web-based policy management. Rein provides ontologies for describing policy domains in a decentralized manner and provides an engine for reasoning over these descriptions, both of which can be used to develop domain and policy language specific security systems. We describe the Rein policy framework and discuss how a Rein policy management systems can be developed for access control in an online photo sharing application.

Introduction

The Web is one of the most important ways for disseminating information across global boundaries. Though it is a simple and convenient framework for searching and retrieving information, the Web suffers from the lack of easy-to-use and adaptable security required by website administrators, application developers, and people in charge of web content. Several approaches for access control to Web resources have been proposed such as WS-Policy (IBM *et al.* 2006), PeerTrust (Gavriloaie *et al.* 2004), Rei (Kagal, Finin, & Joshi 2003), and XACML (Lockhart, Parducci, & Anderson 2005). Each approach has its own policy language that can be used to develop policies over shared ontologies. This causes not only an interoperability issue between domains that use different languages but also forces users to conform their description of their policy relationships to the system's policy language. Instead of requiring all users to adopt a single policy language for their policy requirements, we instead leverage the power of the Semantic Web to reason across the various languages (such as RDF-S (Brickley & Guha 2004), OWL (Bechhofer *et al.* 2004), and rule languages) that are used to describe policies. Rein is a unifying framework that will help the Web preserve maximum expressiveness for policy communities by allowing users to define policies in their own languages but still use the same mechanisms for deploying policy domains.

Rein is a Web-based policy management framework, which exploits the inherently decentralized and open nature of the Web by allowing policies, meta-policies, and policy languages to be combined, extended, and otherwise handled in the same scalable, modular manner as are any Web resources. Resources, their policies and meta-policies, 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 as long as they are 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) (Berners-Lee, Fielding, & Masinter 2005). 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.

Rein **does not propose a single policy language** for describing policies. It allows every user to potentially have her own policy language or re-use an existing language and if required, a meta policy. Rein provides ontologies for describing Rein policy networks and provides mechanisms for reasoning over them. The ontologies and reasoning mechanisms work with any policy language and domain knowledge defined in RDF-S, OWL, or supported rule languages.

Though authentication is an important part of access control, Rein does not enforce a particular kind of authentication but leaves it up to the individual policy to describe the authentication it requires, if any. This allows domains to either combine authentication and authorization into their access control policies or decouple them and provide the authentication information (such as statements in Security Assertion Markup Language (SAML) (Mishra *et al.* 2006)) as input to the authorization policies. We have developed examples that combine authentication and authorization and rely on simple cryptography techniques and other examples that use

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

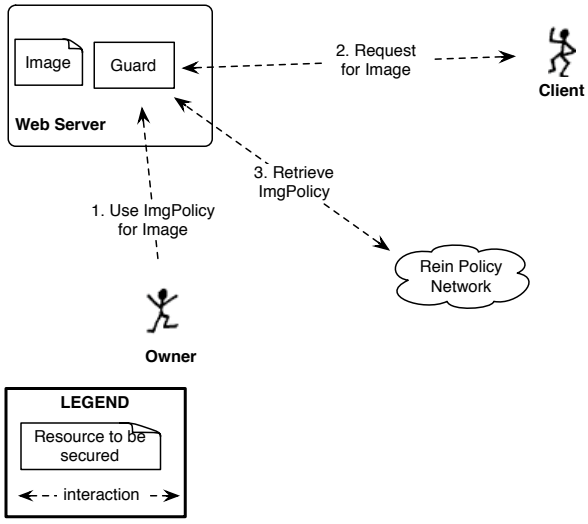

Figure 1: Access Control Model. *The main participants in an access control system developed using the Rein framework include the resource being protected, the Web server that hosts the resource, the owner of the resource, the guard on the Web server that enforces the policy defined by the owner, and the client who wishes to access the resource.*

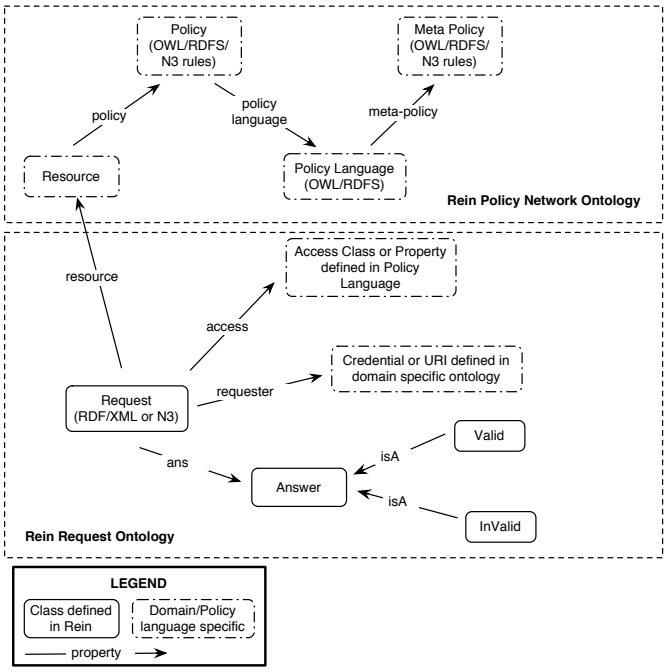

Figure 2: Rein Ontology. *This ontology includes the Rein Policy Network Ontology, which describes the relationships between resources, policies, meta-policies, and policy languages, and the Request class, which is used to perform queries over Rein Policy Networks.*

public keys and signed credentials. The Rein framework can support SAML statements that are translated into RDF. In future work, we plan to investigate integrating Open ID (Fitzpatrick 2005), which is a decentralized authentication mechanism, with the Rein framework.

Some of the main contributions of Rein include, (i) Rein is a Web-based approach to representing and reasoning over policies for Web resources. It promotes extensibility and reusability as it allows every policy to use its own policy language and meta-policy or re-use or extend existing policy languages and meta-policies. (ii) Rein is flexible with respect to how sophisticated or expressive policies can be. (iii) Rein provides a unified mechanism for reasoning over Rein policy networks to make policy decisions. (iv) Rein supports compartmentalized policy development by allowing a division of responsibilities between different parties with different roles and skills. Designing policy languages, writing meta-policies associated with policy languages, developing policies, and enforcing policies are all modular tasks. This allows policy developers to make frequent changes at their high level of understanding without requiring any other changes to the system. (v) All information required to make the policy decision is described within or linked to from the entities in the resource's policy network making our framework self-describing.

Access Control Model

There are several participants in an access control policy system based on Rein - the resource to be protected, the Web server that hosts the resource, the owner of the resource, the guard on the Web server that enforces the policy defined by the owner, and the client or requester who wishes to access the resource. Figure 1 illustrates the interactions between these participants. The owner informs the guard of the access control policies that need to be enforced for the resources she owns. We assume that the owner is authenticated by the Web server and is able to specify policies only for resources that it owns. The guard stores the relationship between a resource and its policy(ies). The resource's policy network, which consists of its policies, policy language(s), and meta policies, can be distributed across the Web. When a resource is requested, the guard collects the description of it's policy network and reasons over it to decide whether to permit access.

This model allows the enforcement mechanism to be completely separated from the policy. The owner of a resource can define her own policy and policy language or reuse/extend existing ones. The policy and its language need not be hosted on or controlled by the same Web server as the Web resource. This allows the policy and other entities in the resource's policy network to change as often as required without requiring any modification to rest of the framework.

Rein Framework

Rein is a Web-based framework for representing and reasoning over policies in domains that use different policy languages and domain knowledge described in RDF-S, OWL,

and supported rule languages. It consists of two main parts, (i) a set of ontologies for describing Rein policy networks and access requests, and (ii) a reasoning engine that accepts requests for resources in Rein policy networks and decides whether or not the request is valid.

Ontologies

The Rein framework includes the Rein policy network ontology and the Rein request ontology to model information in the system. These ontologies are illustrated in Figure 2. The Rein ontology is a relatively small base and includes a few powerful terms that define the access control domain, allow the architecture to be decentralized and web-like, and allow policies and policy languages to be re-used and extended.

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. Meta policies are described in one of the supported rule languages. 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 in a supported rule language. 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 (Beckett 2004) or N3 (Berners-Lee 1998) or described in a supported rule language. An example of a Rein policy network described using Rein ontologies is shown in Figure 3.

The *Request* class is a way to query a Rein policy network. *Requests* are created by users or by the guard from the original user requests to verify whether the 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 term (class or property) defined in the policy language for access control (e.g. ispermitted, forbidden, can-write, ReadPermission), and *ans* that is the response set by the engine. Though the *resource* property is usually set to the URI of the resource being requested, the *requester* property is a set of properties or credentials of the requester because the identity of the requester might not always be meaningful in open environments such as the Web (Kagal 2004). The Rein reasoning engine can be easily modified to handle additional properties for a request including environmental conditions and attributes of the resources.

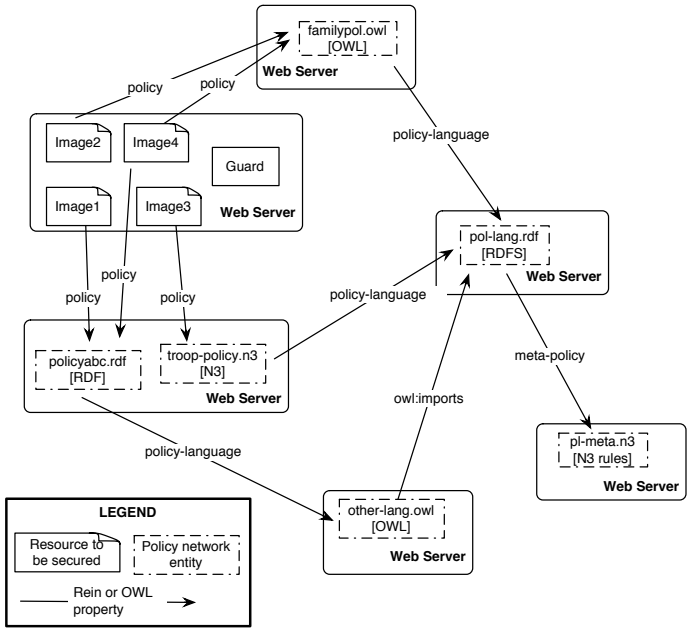

Figure 3: Example of a Rein Policy Network. *Rein policy networks are formed by inter-related resources, policies, policy languages, and meta-policies that can be hosted on different Web servers and that can be extended and re-used as required.*

Reasoning Engine

The Rein reasoning engine accepts requests for resources in Rein policy networks, collects relevant information from these networks, and answers questions about access rights of the requester. It includes a reasoner for RDF-S, OWL, and a reasoning engine for each supported rule language (e.g. N3 rules and RuleML). It is used by guards for controlling access to resources in Rein policy networks. It can also be used by clients to generate proofs of why their request should be allowed. It accepts as input an instance of *Request* and the URI of the policies that describe the access requirements of the resource. It is assumed that the guard is aware of the policies that act on each resource it protects and that these policies are accessible via HTTP.

The engine processes a *Request* by retrieving each policy associated with the requested resource and reasoning over its network including its policies, policy languages, and meta-policies. It also adds information provided by the *Request* such as credentials, certificates, and other attributes of the requester. The Rein engine assumes that all information required to make the policy decision is provided by or linked to from the Request or the policy as it is a self-describing framework. Once the engine collects all the relevant information, it reasons over it using the RDF-S reasoner, the OWL reasoner, and the reasoning engines of the supported rule languages. It then checks whether the inferences include a meaningful relationship between the *resource* property, the *access* property, and an instance that has the same

subset of properties and credentials as those defined by the *requester* property. The reasoning engine has a pre-defined list of possible relationships that it looks for. Finding relationships is possible because we restrict policy languages, which define these relationships, to RDF-S or OWL and the reasoning engine understands the semantics of RDF-S and OWL. For example, if 'foo' is the *access* property some examples of relationships include (i) requester is an instance with foo as a property and the resource as its value, (ii) foo is a class and an instance of foo has requester and resource as values of two of its properties, and (iii) resource is an instance with foo as a property and the requester as its value. If the engine is able to find an appropriate relationship, then the engine infers that the *Request* is valid and sets the *ans* property of the *Request* to *Valid*. The engine can also be run in a certain mode to generate a proof for why a certain *Request* is valid.

Using Rein

Rein is a way of representing and reasoning over policies but does not include any policy enforcement. We propose that the Rein engine be used either by a guard controlling access to the resources or by the client to generate a proof which is checked by the guard. We propose three ways in which policies for Web resources can be enforced:

- Approach where Rein is used by guard : When a client makes a request for a resource, the guard checks if there is a policy associated with the resource. If there is a policy, the guard displays a form that asks the client to provide her (digital) credentials and any other information that is required by the policy. On receiving the submitted form, the guard parses the input and asks the Rein engine to check whether the information provided meet the policy requirements of the requested resource. If the request is valid, the guard allows the request to go through otherwise it returns an error. An analogy for this form of policy enforcement is the process of joining a library or applying for a passport. All the information required by the policy (library or embassy) is provided on a form. The client needs to fill out the form and provide a driver's license or passport in order to meet the server's policy. In this approach, it is possible for the policy to be private, and the guard can iteratively reveal portions of the policy based on the client's credentials. A possible Web implementation includes a client proxy and a modified Apache web server. The Apache web server accepts HTTP requests for the resources it protects and knows which policy applies to each protected resource. It generates the appropriate Rein *Request* from the credentials sent along with the request for the resource and executes the Rein engine. It decides whether or not to permit the request based on the reply from the Rein engine.
- Approach where Rein is used by client : In this approach, when a client requests a resource which has an associated policy, the server returns a link to the portion of the policy that the client needs to satisfy. The client collects the required credentials and uses the Rein engine to generate a proof for why she should have access. Once the engine has generated a proof, the client sends this proof to the guard. The guard checks the proof and if it is valid, the client is allowed to access the resource. Parts of the policy have to be disclosed to the client in this case. The library analogy applies to this case as well. The library provides the policy which states that you have to prove that you have a valid driving license and a local address to join the library. By showing her license, a person can prove that she has the right to join the library. This approach is being used in the Policy Aware Web project (Kolovski *et al*. 2005). The implementation will be similar to the one described above except that the server will first return the policy to be satisfied and the client proxy will reply with the proof. The guard will check the proof and will permit or deny the request depending on whether the proof is valid or not.
- Hybrid Approach : 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 public through the client's web of relationships. The client's company will inform her of possible discounts with car rental companies, AAA will inform her about their discounts, particular newspapers will advertise the car rental agency's coupons. The client will obtain a proof of why she 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 most common case on the Web - some portions of policies will be public and some will be private. A client will have to collect information from trusted sources to develop a proof using Rein for why she 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 unifying mechanisms. Though statements in RDF-S, OWL, and rules in supported rule languages can be loaded and reasoned over in the Rein engine, we have not worked out exactly how this hybrid approach would be supported in the Rein framework.

Current Implementation

The basic requirements of the Rein framework include (i) reasoners for RDF-S and OWL, (ii) an engine for the supported rule language(s), and (iii) a programming language capable of accessing the Web and of working with the chosen reasoners and engines. 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 (Parsia & Sirin 2004), Jena (`http://jena.sourceforge.net/`), cwm (Berners-

Lee 2000)), and rule languages (e.g. RuleML (Ball *et al.* 2005), N3 rules(Berners-Lee *et al.* 2005), Flora (Yang, Kifer, & Zhao 2003)). 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 networks. N3 rules allow policies to access the Web and objectively check the contents and inferences of documents, without having to 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 on the page is trusted. There are several useful cwm builtins such as cryptography, string functionality, and math operators that are useful for specifying policies. The N3 rule language has the expressivity we required and is convenient to read and write.

We have defined the Rein ontologies in RDF-S and have developed a Rein reasoning engine in N3 rules. We 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 (De Roo 2005) for reasoning over RDF-S and a subset of OWL. The engine accepts as input a *Request* instance and the relationship between the requested resource and its policies. The input can be serialized either in RDF/XML (Beckett 2004) or N3 (Berners-Lee 1998). On receiving the input, the engine parses it to get the attributes of the requester and the requested resource. The engine uses cwm builtins (Berners-Lee 2000) 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 the 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 to decide whether to allow or deny the request. However, 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*. The engine can be accessed by a guard or client through the cwm command-line interface or its Application Program Interface (API) in python.

A requirement in the Rein framework is that the guard 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 framework makes some assumptions in order to achieve policy language independence: (i) The policy language is defined in RDF-S or OWL. (ii) Policies are defined over policy languages and domain knowledge and are described in RDF-S, OWL, or a supported rule language. (iii) It is possible to reason over the policy and infer whether a relationship between *access*, *resource*, and an instance with

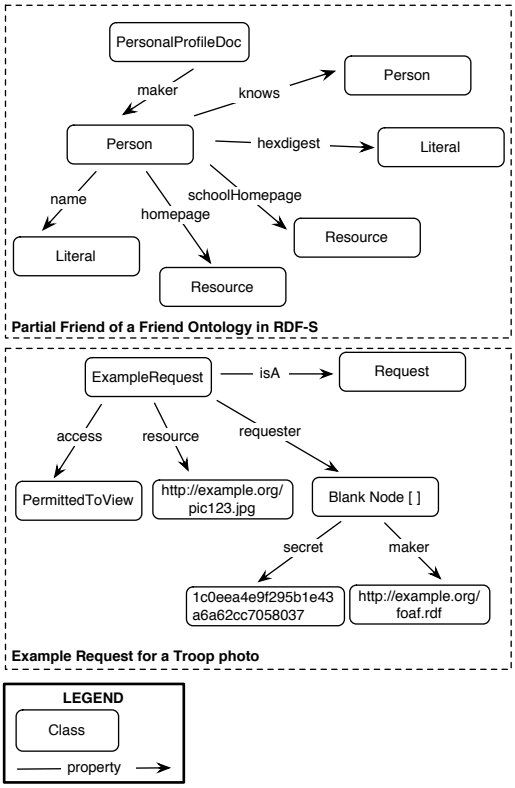

Figure 4: FOAF Ontology and Example Request. *The foaf ontology is used to describe the person who is a member of the troop. The example request illustrates how values are assigned to Request properties in order to query the Rein engine.*

the same credentials and information as the *requester* is entailed. (iv) The programming language used provides accessibility to the Web. (v) The guard is aware of which policies apply to which resource. (vi) The guard is able to generate a valid *Request* instance from the HTTP request sent by the client.

Example : Photo sharing domain

The Web has made sharing photos much easier with online applications such as Flickr (`http://flickr.com`) and Zoomr (`http://zoomr.com`). Though they provide several photo management features such as uploading, tagging, searching for, and leaving comments on photos, their security mechanism is fairly simple. In most of these applications, an image owner can define groups of users and assign photo permissions to them. The owner can decide whether a photo or set of photos is visible to the public in general or to one or more of her groups. This is basically Role Based Access Control (RBAC) (Ferraiolo & Kuhn 1992; Sandhu 1998) where users are assigned to groups (or roles) and groups (or roles) are given permissions.

In order to give an owner of photos more control over who can access her photos and to provide more expressivity

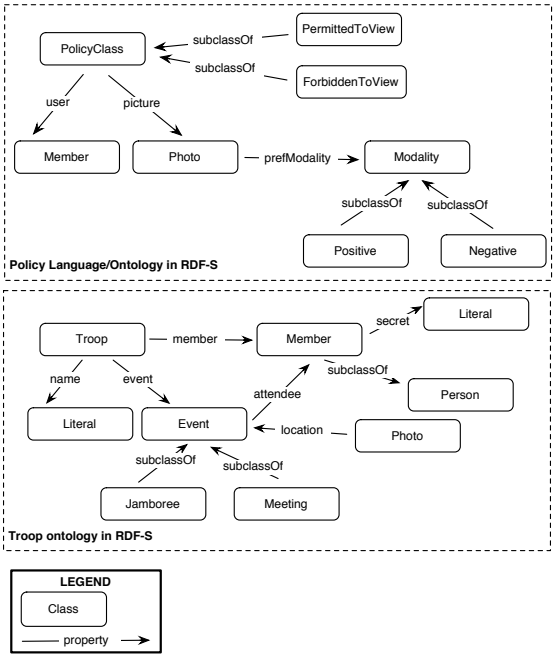

Figure 5: Policy Language and Troop Ontology. *The policy language consists of two classes PermittedToView and ForbiddenToView, both of which have two properties, photo and user. It also includes a property for resolving policy conflicts, prefModality. The troop ontology is used to describe information about the troop, its members, and its activities.*

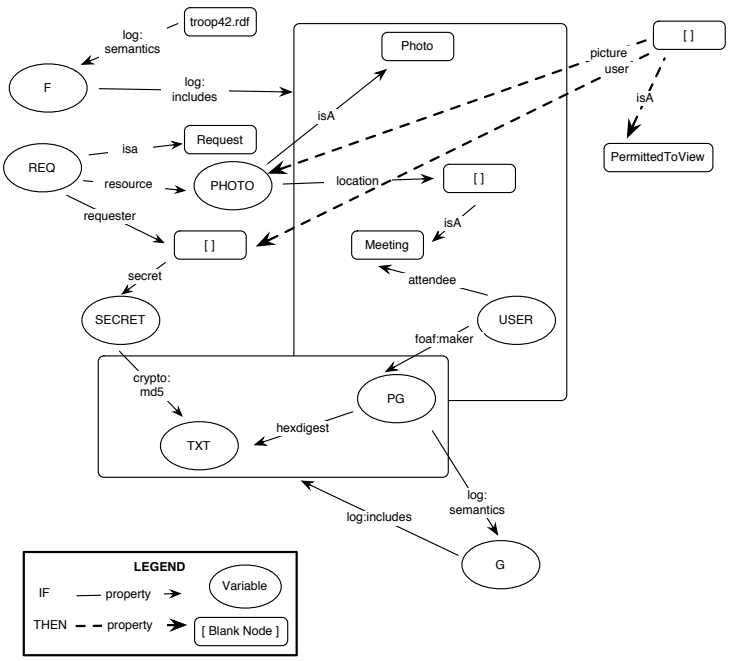

Figure 6: Girl Scout Policy. *The policy implements a simple authentication scheme in which every member has a secret key on her foaf page. In order to prove that she is a member of the girl scout troop, the requester must provide a secret password whose MD5 hash is the value of the hexdigest property on a member's foaf page. The policy states that only members who attended a certain meeting can view pictures of the meeting. Cwm builtin functions, log:semantics and log:includes, are used to access Web resources and check whether one graph is a subset of another.*

in defining these requirements, we suggest that Rein be used for policy management in these photo sharing applications. Instead of associating photos with groups by enumerating group members, the owner will associate declarative policies by informing the guard of the URIs of these policies. The owner can use RDF-S, OWL, or N3 rules to associate policies with photos. Some examples of this association include (i) all my photos are controlled by PolicyA, (ii) all photos with certain tags are controlled by PolicyB, and (iii) all photos without a certain tag are controlled by PolicyC.

Access control policies need not be hosted on the photo server and can be described in RDF-S, OWL, or N3 rules. The owner can define her own policy and language or use/extend someone else's policy or policy language. If another user has a policy that states that MIT graduate students are allowed to access his pictures, then by using this policy, the owner gives MIT graduate students access to her pictures without worrying about how to authenticate the students. The owner can also use only the policy language and define her own policy that states that UMBC students are allowed to access her pictures taken at the award ceremony.

A user who wishes to access a certain photo or set of photos that meet a certain search criteria will send an HTTP request. If the photo or photos is protected by a policy, the guard will return an HTTP 401 response, which means that the request requires user authentication. This response will be intercepted by a proxy on the client side. The proxy will get together the credentials and information required based on the user's preferences and resubmit the request. The guard will use the information sent to form a Rein *Request* and query the Rein engine with it. If the Rein engine infers that the *Request* is valid, the request is allowed to go through and the photo is returned to the client.

As an example, consider members of a girl scout troop who use an online photo sharing application to host their photos. The troop has a website that lists its members, their Friend of A Friend (foaf) (Brickley & Miller 2006) pages, how long they have been members, etc. The troop members upload photos of their meetings, jamboree, and award ceremonies. Please refer to Figure 4 for the foaf ontology and an example of a *Request*. The troop has a policy language and troop ontology in RDF-S. The troop ontology is used to describe information about the troop, its members, and its activities and this information also available on the troop website. Troop members upload photos and define policies for them using the policy language and troop ontology. The troop's policy language consists of two classes *PermittedToView* and *ForbiddenToView* class both of which have two properties, *picture* and *user*. The policy language also includes a property for defining meta-policies called

prefModality, which is a property associated with photos. This property is used to resolve conflicts in the policy i.e. if a requester is both permitted and prohibited from accessing a picture, the *prefModality* property is used by the meta-policy to decide which one policy overrides the other. Figure 5 shows the policy language being used and the troop ontology.

The girl scout troop has a rule-based policy in N3 for defining the following rules: (i) only members who attended a certain meeting can view pictures of the meeting, (ii) members who missed the last Jamboree cannot see any pictures. This example implements a simple authentication scheme in which every member has a secret key on her foaf page. In order to prove that she is a member of the girl scout troop, the requester must provide a password whose MD5 hash is the value of the *hexdigest* property on a member's foaf page. The meta policy is also defined in N3 rules and resolves any conflicts using the preferred modality of the requested photo. If there is a conflict, the meta-policy checks the *prefModality* associated with the photo. If the *prefModality* is *Positive*, the permission is the final result, otherwise the prohibition overrides. Please refer to Figure 6 for a pictorial representation of a portion of this policy.

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 (Appel & Felten 1999). 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.

PeerTrust provides a mechanism for gaining access to secure information/services on the web by using semantic annotations, policies and automated trust negotiation (Gavriloaie *et al.* 2004). In PeerTrust, trust is established incrementally through an iterative process which involves gradually disclosing credentials and requests for credentials. PeerTrust's policy language for expressing access control policies is based on definite Horn clauses. PeerTrust is an interesting approach that expects both parties to exchange credentials in order to trust each other and assumes that policies are private. This is appropriate for highly secure resources such as ecommerce sites, however, our work is aimed at controlling access to resources such as pictures, blogs, and calendar entries. However, Rein is a representation framework for policies and does not include a protocol for policy exchange or enforcement. We would like to support an iterative disclosure approach like PeerTrust within the Rein framework in the future.

Within the Rein framework, policy languages such as Extensible Access Control Markup Language (XACML) (Lockhart, Parducci, & Anderson 2005), Platform for Privacy Preferences (P3P) (Cranor *et al.* 2002), KAoS (Uszok *et al.* 2004), and Rei (Kagal 2004) can be considered domain-specific policy languages. In fact, if their semantics can be represented in RDF-S, OWL, or N3 rules, it will be possible to integrate them into the current Rein implementation.

Future Work

The querying mechanism using *Requests* is very simple and only checks whether a there is a relationship between the requester, resource, and access properties. 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, can all faculty members access the faculty printer etc.

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. Rein does, however, include both disjunction and conjunction of rules within policies. We will look into supporting conjunction of policy decisions as well as defining meta-policies over multiple policies in the future.

As part of our future work, we will also look into representing policies in proposed rule languages for the Web including RuleML (Ball *et al.* 2005), SWRL (Horrocks *et al.* 2004), and Rule Interchange Format (RIF) (W3C RIF Working Group 2006) and use these rule languages to provide functionality similar to Rein.

Rein does not take possible attacks such as denial of service and replay attacks into consideration. This is something we would like to address. Another problem is that 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.

Summary

Rein is a policy framework for the Web grounded in Semantic Web technologies. It provides ontologies for describing diverse policy scenarios and mechanisms for reasoning over them, both of which can be configured to work with different policy languages and domain knowledge in RDF-S and OWL. Rein also includes *delegation mechanisms*, independent of the policy language and domain information, that support both delegation of authorization and trust (Kagal *et al.* 2006). Rein ontologies have been defined in RDF-S and the framework has been implemented using python, cwm, and N3 rules.

References

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

Ball, M.; Boley, H.; Hirtle, D.; Mei, J.; and Spencer, B. 2005. Implementing RuleML Using Schemas, Translators, and Bidirectional Interpreters. In *W3C Workshop on Rule Languages for Interoperability*.

Bechhofer, S.; van Harmelen, F.; Hendler, J.; Horrocks, I.; McGuinness, D. L.; Patel-Schneider, P. F.; and Stein,

L. A. 2004. OWL Web Ontology Language Reference, W3C Recommendation. http://www.w3.org/TR/owl-ref/.

Beckett, D. 2004. RDF/XML Syntax Specification (Revised). W3C Recommendation. http://www.w3.org/TR/rdf-syntax-grammar/.

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

Berners-Lee, T.; Fielding, R.; and Masinter, L. 2005. Uniform Resource Identifier (URI). http://www.ietf.org/rfc/rfc3986.txt.

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

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

Brickley, D., and Guha, R. V. 2004. RDF Vocabulary Description Language 1.0: RDF Schema, W3C Recommendation. http://www.w3.org/TR/rdf-schema/.

Brickley, D., and Miller, L. 2006. Friend of a Friend (FOAF) project. http://www.foaf-project.org/.

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

De Roo, J. 2005. Euler proof mechanism. http://www.agfa.com/w3c/euler/.

Ferraiolo, D., and Kuhn, R. 1992. Role-based access controls. In *15th NIST-NCSC National Computer Security Conference*, 554–563.

Fitzpatrick, B. 2005. OpenID: an actually distributed identity system. http://openid.net/.

Gavriloaie, R.; Nejdl, W.; Olmedilla, D.; Seamons, K.; and Winslett, M. 2004. 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*.

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

IBM; Systems, B.; Microsoft; AG, S.; Software, S.; and VeriSign. 2006. Web Services Policy Framework (WS-Policy). http://www-106.ibm.com/developerworks/library/specification/ws-polfram.

Kagal, L.; Berners-Lee, T.; Connolly, D.; and Weitzner, D. 2006. Self-describing Delegation Networks for the Web. In *IEEE Workshop on Policy for Distributed Systems and Networks (POLICY 2006)*.

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

Kagal, L. 2004. A Policy-Based Approach to Governing Autonomous Behavior in Distributed Environments. Dissertation. University of Maryland, Baltimore County.

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

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

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

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

Sandhu, R. S. 1998. Role-based access control. In Zerkowitz, M., ed., *Advances in Computers*, volume 48. Academic Press.

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

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

Yang, G.; Kifer, M.; and Zhao, C. 2003. FLORA-2: A Rule-Based Knowledge Representation and Inference Infrastructure for the Semantic Web. In *Second International Conference on Ontologies, Databases and Applications of Semantics (ODBASE)* .