# Promoting Interoperability between Heterogeneous Policy Domains[*]

Lalana Kagal, Tim Berners-Lee, 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 gaining popularity as a means of providing flexible Web security, the number of policy languages being proposed is constantly increasing. We recognize the importance of policies for securing the Web and protecting user information 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 policy management for the Web. 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 systems.

## 1 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 [13], PeerTrust [12], Rei [16], and XACML [17]. 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, we instead leverage the power of the Semantic Web to reason across the various languages (such as RDF-S [8], OWL [2], 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 [15]. 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) [7]. 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. Within the Rein framework, policy languages such as Extensible Access Control Markup

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

Language (XACML) [17], Platform for Privacy Preferences (P3P) [9], KAoS [20], and Rei [14] 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.

Though authentication is an important part of security, 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) [18]) 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 public keys and signed credentials. The Rein framework can support SAML statements that are translated into RDF. In future work, we plan to investigate the use of Open ID [11] as an authentication mechanism for 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.

## 2 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.

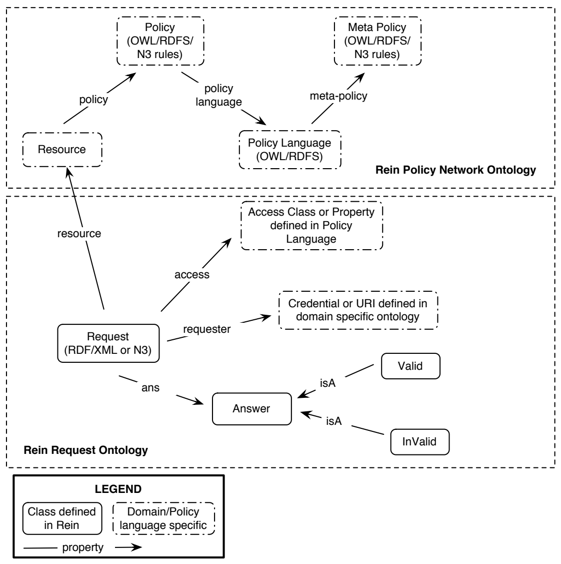
Policy (OWL/RDFS/ N3 rules)
Meta Policy (OWL/RDFS/ N3 rules)
policy
policy language
meta-policy
Resource
Policy Language (OWL/RDFS)
Rein Policy Network Ontology
resource
Access Class or Property defined in Policy Language
access
Credential or URI defined in domain specific ontology
requester
Request (RDF/XML or N3)
Valid
isA
ans
Answer
isA
InValid
Rein Request Ontology
LEGEND
Class defined in Rein
Domain/Policy language specific
property

Figure 1: 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.*

### 2.1 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 1. 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 [3] or N3 [4] or described in a supported rule language. An example of a Rein policy network described using Rein ontologies is shown in Figure 2.

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 [14]. The Rein reasoning engine can be easily modified to handle additional properties for a request including environmental conditions and attributes of the resources.

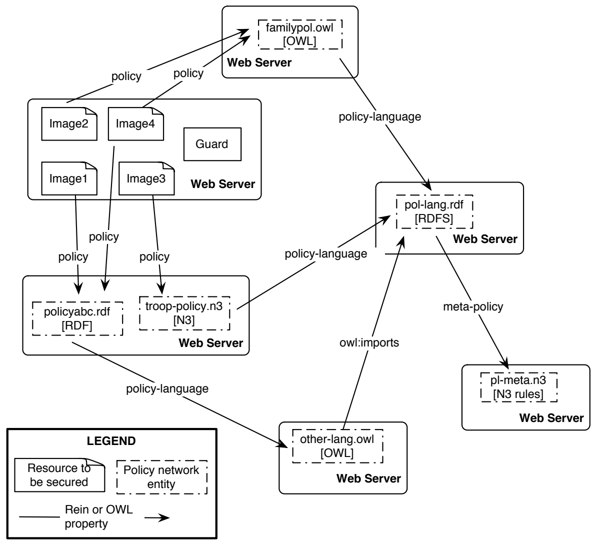

Figure 2: Example of a Rein Policy Network. *Rein policy networks are formed by inter-related resources, policies, policy languages, and meta-policies that can be hosted on different Web servers and that can be extended and reused as required.*

## 2.2 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.

## 3 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 [19], Jena (`http://jena.sourceforge.net/`), cwm [5]), and rule languages (e.g. RuleML [1], N3 rules[6], Flora [21]). We have implemented it in one possible way using python, cwm, and N3 rules.

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 [10] 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 [3] or N3 [4]. On receiving the input, the engine parses it to get the attributes of the requester and the requested resource. The engine uses cwm builtins [5] 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 commandline interface or its Application Program Interface (API) in python.

## 4 Discussion : Rein and Privacy Management

Though Rein was developed for access control on the Web, we propose that it could be used for privacy management as well. In access control, policies describe who should have access to what and under what conditions. To enforce an access control policy, the credentials of the requester are usually checked at the time of the request by the web server or guard that controls access to the requested resource. In privacy management, there are two kinds of policies : user policies and systems policies. User policies describe what information the user is willing to share, with whom, and under what conditions (e.g. APPEL) and system policies describe what information the system needs, what information it will store, with whom it will share this information, and under what conditions (e.g. P3P). Privacy policies are enforced in two ways: (i) the system privacy policies must meet the users policy before the user uses the system, and (ii) it should be possible to check that the system has fulfilled the privacy policy when it uses or shares sensitive user information. Access control and privacy management differ in terms of policy descriptions, time of enforcement, and the enforcer. However, in both cases, policy languages and engines are required for expressing and reasoning over the system and user requirements. As Rein is a policy framework that does not restrict where and when enforcement is done, and provides interoperability between domains that use different policy representations, it can be used as a policy framework for both security and privacy management.

## References

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

[2] 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.

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

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

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

[6] 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.

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

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

[9] 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.

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

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

[12] 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.

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

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

[15] L. Kagal, T. Berners-Lee, D. Connolly, and D. Weitzner. Using Semantic Web Technologies for Policy Management on the Web. In *21st National Conference on Artificial Intelligence (AAAI 2006)*, 2006.

[16] 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.

[17] 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.

[18] 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.

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

[20] 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.

[21] G. Yang, M. Kifer, and C. Zhao. 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)* , November 2003.