Towards a Policy-Aware Web

Vladimir Kolovski[1], Yarden Katz[2], James Hendler[2],
Daniel Weitzner[3], and Tim Berners-Lee[3]
kolovski@cs.umd.edu, yarden@umd.edu, hendler@cs.umd.edu,
djw@w3.org, timbl@w3.org

[1] Maryland Information and Network Dynamics Laboratory Lab, University of Maryland , College Park , MD 20740
[2] Dept. of Computer Science, University of Maryland, College Park, MD
[3] CSAIL, Massachusetts Institute of Technology, Cambridge, MA

Abstract. In this paper, we argue that a new generation of Policy-Aware Web (PAW) technology can hold the key for providing open, distributed and scalable information access on the World Wide Web. Our approach provides for the publication of declarative access policies in a way that allows transparency for information sharing among parties. In addition, greater control over information release can be placed in the hands of the information owner, by employing rule-based discretionary access control. In this paper, we outline the kind of reasoning support which is needed to achieve these goals. Also, we present our initial steps in this direction. Our example application for this purpose is a calendar and photo sharing web site that uses a distributed policy framework (REIN) built on top of a rule-based reasoner (CWM).

1 Introduction/Motivation

While Semantic Web (SW) technologies continue to increase in popularity and scope, certain data owners are still reluctant to make their data public. The reluctance to share information prevents the full effects and benefits of Semantic Web data and tools. The main reason behind the reluctance is the lack of sufficiently sophisticated security and access control. Another issue is privacy; with so much information electronically available and increasingly sophisticated data aggregation techniques, individual privacy might be compromised and liberties put at risk if the information were to be made public with no qualifications. The problem is exacerbated when information must be shared between parties that do not have information sharing policies, or when the components of the data shared are not finely grained (in the cases where access control operates on the level of an entire website or data resource, for example.)

Consider the task of sharing photographs with friends. When sharing a picture with a friend or colleague offline, the context of the interaction and the nature of the information being shared are enough for both parties to understand the social rules governing how the information can be used. When we email pictures, many of the same social conventions are likely to apply. However, when

sharing pictures with informally-defined online communities and unknown audiences, problems soon emerge. The inability to write even very simple rules for specifying who can view this information on the Web leaves us with an inflexible set of choices. We must either share with everyone, share with no one, or engage in the arduous task of managing access via domain-specific security mechanisms (like IP address filtering, or distribution of user names and passwords.)

In this paper, we will argue that a new generation of Policy-Aware Web technology can hold the key for providing open, distributed and scalable information access on the World Wide Web. Our approach provides for the publication of declarative access policies in a way that allows significant transparency for sharing among partners without requiring pre-agreement. In addition, greater control over information release can be placed in the hands of the information owner, allowing discretionary access control to flourish. We will show that it is possible to deploy rules in a distributed and open system, and to produce and exchange proofs based on these rules in a scaleable way. These techniques, properly applied by taking crucial Web architecture issues into account, will extend Semantic Web technology to allow information resources on the World Wide Web to carry access policies that allow a wide dissemination of information without sacrificing individual privacy concerns.

In the next section, we describe in more detail the rule-based infrastructure that we are bulding. Section 3 discusses the kind of reasoning support needed for successful operation of our system. In Section 5, we describe a calendar/photo sharing web application that embodies the core ideas of PAW. Finally, we discuss the related work and future directions of our project.

2 Infrastructure

Most Web access today is performed using identity- and role-based approaches. A disadvantage of these approaches is that the roles (classes of users with specific access rights) have to be defined in advance. Setting up a temporary class for new, unforeseen access rights, in most implementations, difficult. Also, it is difficult to setup these access schemes in a fine-grained way because most web-based access generally works at the file-directory level. Our goal is to be able to describe policies at the level of individual URIs, grounding the system in the smallest externally nameable Web resources.

We are employing a rule-based approach, where a declarative set of rules is used to define fine-grained access to resources. In this framework, requesters are asked to provide a proof showing that they satisfy the policy encoded in the rules. The complexity of the required proof can vary depending on the application and domain.

There are two types of rule-based access mechanisms: Mandatory access control (MAC) and Discretionary access control (DAC). MAC systems usually do not allow the owner of the information to control protection decisions - the system is designed to enforce *a priori* protection decisions, such as enforcing the security policy over the wishes or intentions of the object owner.

Discretionary Access Control, on the other hand, allows the information owner to locally determine the access policy. The problem with DAC is that it generally requires the information requesters to prove that they have access, and for the objects in the system to have a finer-grained control of the information than in MAC systems. The system we're trying to build is of the discretionary, rule-based access type - thus, our work addresses the DAC challenges.

For an illustration of discretionary access control, consider the following example of a Girl Scout troop photo sharing. In this particular Girl Scout organization, pictures of the events can only be seen by Girl Scouts or their parents. Mary is a Girl Scout and her mother Jane wants to look at the pictures. In order to access these photos, Jane has to prove two things: she is a parent of Mary and that Mary is a girlscout. Our goal is to provide a general framework for providing those proofs. This framework will consist of a proof-exchange mechanism and also a way by which the system receiving the proof can check is correctness. On the Web, we cannot assume that every user will employ the same proof-checking software, so a set of standards is required be sure that all participants evaluate proofs on the semantic web in a consistent manner.

The central piece of the PAW project is the rules language that we are using for representing access policies and the inference engine that handles the proof checking. Since we are building on already existing Web standards, we need a rules language that can be serialized in a form where rules can be published, searched, browsed and shared using HTTP. Our language of choise is Notation 3 (N3), RDF-based rule language that was designed to be consistent with a number of Web Architecture principles. N3 is also designed to work closely with Closed World Machine (CWM), a forward-chaining reasoner that can be used for querying, checking, transforming and filtering information on the Web. Its core language is RDF, extended to include N3 rules. More detail and examples are available in the next section.

It is important to note one distinct feature of our approach. The current use of CWM requires the bulk of the reasoning to be done on the server's side. We are working on a system that is more scalable because CWM (or a CWM-like reasoner) will be used to generate proofs on the *client* side, so the only job of the server will be to check the proof to see if its grounded and consistent. Also, the authentication used by the client is not the key feature of our work (for example we might use transparent distributed authentication like openID [4]). The main contribution of our project will be combining the authentication, the proof generation on the client side and server-side proof checking to achieve a truly transparent, policy-aware web.

By shifting the burden of finding the access justification to the requesting party (and leaving only the task of checking the justification to the authorizing party), the resulting system (a) has a much smaller trusted computing base (only the part that verifies justifications) and (b) is much more transparent: any third party can audit that the justification is valid.

3 Reasoning Support for PAW

Reasoning is a critical issue when dealing with policies. Our recent work [11] shows the usefulness of having policies written in a formally defined language, OWL-DL in specific, for which there are reasoners available. The generic services of an OWL-DL reasoner were capable of capturing key tasks related to policies, such as policy containment ("If I meet policy X, do I also meet policy Y?") and detection of contradictory policies ("Is it impossible to meet policy X?"), among others.

Rule-based languages have also played a dominant role in reasoning on the Semantic Web, the primary example being N3. In the space of policies, the REIN language, based off of N3, was proposed in [10] as a generic framework for representing rule-based policies. In this section, we focus on the kind of inference licensed by languages like N3 and REIN. We sketch a brief outline of requirements from policy-aware inference engines that work with these languages, based on our preliminary work. Lastly we offer some directions in which future work may follow.

3.1 Rules, policies and proofs

Rules are desirable on the Semantic Web, and policies are no exception. The numerous proposals for rule languages offered recently suggest that rules are desirable due to their expressivity, and in some cases, because of their attractive computational properties. The latter is particularly important in PAW, where inference will be performed over a large number of policies, and will be addressed in a later section.

Once a policy is expressed as a set of rules, several challenges occur. A key challenge, and one which PAW reasoners must address, is *proof checking*. Abstractly, consider the following: agent B would like agent A to reveal certain information.[4] A has a publicly available policy (a collection of rules) P which mandates who is allowed to view the information B wishes to receive–B must meet this policy.

The simplistic proof checking scenario might then go as follows. B presents a proof Π (a sequence of assertions) to A, which is intended to establish B's eligibility to receive A's information. That is, the set of assertions $\pi_1, ..., \pi_{n-1} \in \Pi$ are supposed to justify the last line in Π ($\pi_n \in \Pi$), i.e., the assertion that B is authorized to view the information. A naive proof checker might check that $\{\pi_1, ..., \pi_{n-1}\} \cup P \vdash \pi_n$. The use of "proof" in this context of a forward-chaining is just a set of assertions, and nothing more. A proof will be a successful one (allow B access to information of A) if it (the set of facts) plus the shared policy rules between B and A entail B's granted authorization to access A's documents. Note that under the assumption that both A and B can read and process policies

[4] Note that we are assuming that these agents are acting on behalf of users. In other words, we assume that it is possible for an agent to provide information about the person for which it is performing a task if the person wishes to share this information.

written in the same language (say, REIN), there should be no disagreement about what statements are provable. Thus, determining a valid assertion in the deductive sense is trivial; simply run the rules by your own trusted reasoner and see whether the client's reported consequences follow. However, there can certainly be disagreement about the truth of the premises used in a proof, as we will now see. Suppose that A's specific policy is the following: an agent can access our information if (i) it is affiliated with the MIND lab institution, and (ii) the information is requested on a workday. Furthermore, one meets (i) just in case one's provided email address is a MIND lab email address, which take on a specific form. The "truth of the premise" disagreement is solved by having the client and server agree (via public key signing) on the grounded assertions, at which point it is sufficient to rely on deductive validity.

Policy P in N3

R_1: Conditions for being an authorized agent

```
{?agent policyP:hasMINDAcct ?email.
 ?agent policyP:requestsAccess ?request.
 ?request policyP:requestTime ?time.
 ?time a policyP:validRequestTime.}
  => {?agent a policyP:authorizedAgent.}
```

R_2 : Conditions for having the proper email credentials

```
{?agent foaf:mbox ?email.
 ?email string:matches ".*?@mindlab.umd.edu$".}
  => {?agent policyP:hasMINDAcct ?email}.
```

R_3 : Conditions for the times when a request can be processed

```
{?agent policyP:requestsAccess ?request.
 ?agent policy:requestTime ?time.
 "" time:localTime ?localTime.
 ?localTime time:dayOfWeek ?day.
 ?day math:greaterThan "0".
 ?day math:lessThan "6".
  => {?time a policyP:validRequestTime.}
```

Table 1. Policy P

In the case of policies, the naive approach is not sufficient. Certainly, there can be "proofs" presented to B which fail the naive proof checking test immediately. For example, if the triple `:AgentB foaf:mbox "bob@umbc.edu".` is part of Π, then rule R_2 will not fire, preventing us from concluding that B has the proper email address. In this case, the policy will not be satisfied. The same holds true for the case where the email address given is syntactically proper by our rule, i.e. ends with "mindlab.umd.edu", but simply does not exist.

One might object that in light of the requirement that policies be open, an agent can simply *falsify* the necessary triples satisfy the antecedents of every rule in the policy, thus tricking the naive proof checker. Let us consider such a case. Imagine B were a malicious agent, and used the valid address "bernardo@mindlab.umd.edu" without permission. Logically, this is sufficient to derive that B satisfied the email requirement. But this is clearly not enough; in a case like this, A must rely on an additional *callback confirmation* to B to guarantee the truth of the premise `:AgentB foaf:mbox "bernardo@mindlab.umd.edu"`. If B is not truly representing the individual to which this email address belongs, the callback will fail and the policy will be unsatisfied.

Notice that in both malicious scenarios, the work done to ensure the authenticity of the proof was split between the client and the server. Traditional authentication mechanisms would generally overload the server-side. For example, in the case of a falsified email, the traditional approach might require the server to contact an official server (the MIND lab server in our case) to verify the existence of the email address. The callback approach instead requires the client to do work in the process of a request (answering the callback), thus preventing potential denial-of-service attacks from the client, and relaxing the commitment to a trusted third-party server such as the university.

3.2 Engineering for scalability

For the goal of scalability, several insights from research into rule-based expert systems in the AI community are relevant. A primary example is the Rete algorithm, developed by Forgy [5], which allows for efficient processing of very large rule bases. While other approaches may be equally relevant, the use of Rete can prove useful in the context of rule-based policies for the following reasons: (i) policies can be quite complex (composed of many rules) while facts sent interactively by client may only trigger a small number of rules[5] and (ii) there will naturally be policies that are commonly used by a certain source (i.e. a company's policy to serve sensitive documents to its employees working from a remote location) and implementations of Rete lend themselves to optimization of such a use. For example, the Rete network, once constructed for a given large policy, can be either kept persistent memory, or stored locally and reloaded. The Rete network corresponding to a policy need not be rebuilt two separate client interactions.

Pychinko [13] is a forward chaining rule engine, written in Python, that implements Rete for such purposes. Its rules are expressed in the N3 language and its facts as RDF triples. The use of Rete allows it to scale far better than a similar and more popular rule engine, CWM [15], the engine supporting the REIN policy

[5] This is the case where the efficiency of the Rete algorithm shines: many rules and a newly added fact. A naive algorithm would be forced to check this newly added fact against all the rules for a possible match, while the Rete will channel the fact only to the relevant rules (i.e. rules that might be fired due to the addition of this new fact)

language, which uses a naive rule processing algorithm. It is important to note that while forward-chaining rule engines such as these add facts to the server's knowledge base (in order to determine whether's a clients authorization follows from its submitted proof), they do not in any way alter the state of the protocol interactions. The application of the reasoner is merely a server-side computation ordinarily performed in other stateless protocls.

4 Initial Results: Calendar/Photo Sharing Application

As an initial prototype, we have developed a calendar sharing application to illustrate how the rule-based policy infrastructure can be used. The application consists of a calendar view that displays events and photos associated with these events. Access to every event and photo is controlled by rules specified in N3. All of the calendar data is stored in an ontology and the policies that control access to it are in separate files. We are employing finest level of granularity - policies can be specified down to a single calendar event or a picture. REIN is used to connect the policies, the calendar data and the reasoner.

Whenever a user (who may or may not be authenticated) attempts to browse the calendar, the application first retrieves all of the events in the calendar view, filters them through the REIN policies associated with them and returns the accessible ones back to the user [6].

As an example of how policies are defined in the calendar application, consider the calendar sharing rules of a student, member of Mindswap research group at UMD). The student would like to specify that his public calendar can be seen by any member of Mindswap, but his work calendar can only be seen by his advisor.

In the first release, authentication is performed by an account-based mechanism where individuals in the ontology are mapped to user names. Our primary goal for the prototype was to use a distributed policy framework (REIN) as a resource manager to demonstrate discretionary access control on the web. A distributed authentication mechanism will be added in a future release.

5 Related Work

We are certainly not the first group to tackle access control on the web [7–9, 12, 14, 16]. In this section we will discuss two projects that have provided the most direct influence to our work.

Proof-carrying Authorization (PCA) and related distributed proof systems [2, 1] is an authorization framework that is based on a higher-order logic. We have built our work on top of theirs (ideas of a client generating and server checking proofs, integrating the proof exchange protocol into HTTP, etc.) The higher-order logic (AF logic) used to check the proofs is undecidable, though this problem is avoided by forcing clients to generate proofs on their own, using only

[6] A demo is available at http://www.policyawareweb.org/2005/calendar/

Any member of Mindswap can access events belonging to the public calendar

```
@forAll :x, :y.
{  ?x a ont:User.
   ont:Mindswap ont:hasMember ?x.
   ?y a ical:Vevent.
   ont:PrivateCalendar ont:hasEvent ?y.
} => { ?x reine:canAccess ?y}.
```

Any professor, member of Mindswap can access events belonging to the work calendar (telecons, etc.)

```
@forAll :x, :y.
{  ?x a ont:Professor.
ont:Mindswap ont:hasMember ?x.
?y a ical:Vevent.
ont:WorkCalendar ont:hasEvent ?y.
\} => {?x reine:canAccess ?y}.
```

Table 2. Calendar sharing policies in REIN P

a decidable subset of AF logic. Consequently, the authorizing server's task of proof-checking is reduced to a tractable type-checking problem - but this leads to large rate of increase of sizes of the client proof. We are currently in the process of defining our own logic for the task of representing policies, by formalizing key concepts in REIN and N3.

Bonatti et al. [3] discuss a uniform formal framework to formulate and reason about both service access and information disclosure constraints on the web. It introduces the ideas of servers publishing their policies as sets of rules and allowing usage of uncertified declarations in the client proof. Related to [3], PeerTrust [6] also deals with discretionary access control on the web using semantic web technologies. PeerTrust is a language for policies and trust negotiation, and this paper describes their implementation of implicit registration and authentication that runs over a Prolog engine. The work differs from ours in a couple of aspects. First, the authors make the assumptions that information owners are not willing to freely share their access control rules on the web, where one of the main postulates of the PAW project is transparency. Also, the trust negotiation protocol in [6] is keeping state, whereas our proof exchange protocol is stateless.

6 Conclusion and Future Work

In this paper, we have shown the need for a policy-aware infrastructure for the web, and shown our work towards this infrastructue. We have described our current work in developing a rule-based policy management system that can be deployed in an open and distributed manner on the World Wide Web. We have also shown in a demo application how it is possible to combine a Semantic Web rules language (N3), a theorem prover designed for the Web (CWM), and

a distributed policy management system (REIN) to provide discretionary access control for users. However this is still preliminary work and there are a lot of open issues, just to name a few:

- An important issue is the proof generation/checking on the client side. Future plans on this front include extending the Pychinko engine with proof checking capabilities that are sensitive to CWM's builtins.
- The party that requests access to data has to provide authentication as part of its proof. We have been experimenting with email callback mechanisms as means of proving identities, however this approach does not seem to generalize well. Other interesting mechanisms we are exploring are distributed authentication systems such as OpenID [4]
- In an open system such as the Web inconsistencies are inevitable - most logic-based systems built to date, however, are intolerant of inconsistencies. Developing a model where inconsistency can be tolerated, and kept from causing harm, is one of the key areas of research in our work.

References

1. L. Bauer, S. Garriss, and M. K. Reiter. Distributed proving in access-control systems. In *SP '05: Proceedings of the 2005 IEEE Symposium on Security and Privacy*, pages 81–95, Washington, DC, USA, 2005. IEEE Computer Society.
2. L. Bauer, M. A. Schneider, and E. W. Felten. A proof-carrying authorization system. Technical Report TR-638-01, Princeton University, 2001.
3. P. A. Bonatti and P. Samarati. A uniform framework for regulating service access and information release on the web. *J. Comput. Secur.*, 10(3):241–271, 2002.
4. B. Fitzpatrick. Openid:an actually distributed identity system, July 2005.
5. C. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 12:17–37, 1982.
6. 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 *European Semantic Web Symposium*, May 2004.
7. S. Godik and T. Moses. Oasis extensible access control markup language (xacml) version 1.1. oasis committee specification, July 2003. http://www.oasis-open.org/committees/download.php/4103/cs-xacml-specification-1.1.doc.
8. JAAS. Java authentication and authorization service (jaas). http://java.sun.com/products/jaas/.
9. S. Jajodia, P. Samarati, V. S. Subrahmanian, and E. Bertino. A unified framework for enforcing multiple access control policies. In *SIGMOD '97: Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, pages 474–485, New York, NY, USA, 1997. ACM Press.
10. L. Kagal and T. Berners-Lee. Rein : Where policies meet rules in the semantic web. Technical report, MIT, 2005.
11. V. Kolovski, B. Parsia, Y. Katz, and J. Hendler. Representing web service policies in owl-dl. In *International Semantic Web Conference (ISWC)*, 2005.
12. N. H. Minsky and V. Ungureanu. A mechanism for establishing policies for electronic commerce. In *ICDCS '98: Proceedings of the The 18th International Conference on Distributed Computing Systems*, page 322, Washington, DC, USA, 1998. IEEE Computer Society.

13. B. Parsia, Y. Katz, and K. Clark. Pychinko: Rete-based CWM clone, October 2004. http://www.mindswap.org/~katz/pychinko/.
14. M. Roscheisen and T. Winograd. A communication agreement framework for access/action control. In *SP '96: Proceedings of the 1996 IEEE Symposium on Security and Privacy*, page 154, Washington, DC, USA, 1996. IEEE Computer Society.
15. W3C. cwm - a general purpose data processor for the semantic web, August 2004. http://www.w3.org/2000/10/swap/doc/cwm.html.
16. T. Yu, X. Ma, and M. Winslett. Prunes: an efficient and complete strategy for automated trust negotiation over the internet. In *CCS '00: Proceedings of the 7th ACM conference on Computer and communications security*, pages 210–219, New York, NY, USA, 2000. ACM Press.