A framework for semantic web policies*

P.A. Bonatti
Università di Napoli Federico II
bonatti@na.infn.it

J.L. De Coi
University of Hannover
decoi@L3S.de

D. Olmedilla
University of Hannover
olmedilla@L3S.de

L. Sauro
Università di Napoli Federico II
sauro@na.infn.it

ABSTRACT

Trust and policies are going to play a crucial role in enabling the potential of many web applications. In this paper we illustrate Protune, a system for specifying and cooperatively enforcing security and privacy policies.

1. INTRODUCTION

Trust is the top layer of the famous semantic web picture. It plays a crucial role in enabling the potential of the web. While security and privacy do not cover all the facets of trust, still they play a central role in raising the level of trust in web resources. Web services obviously need some form of access control: The application of suitable policies for protecting services and sensitive data may determine success or failure of a new service. In a near future, we might see web services compete with each other by improving and properly advertising their policies.

A major issue in moving towards such a policy-aware web is *usability*, that in turn has several facets. It is well known that as protection increases, usability is affected by the extra steps required for authentication and other operations related to access control. Moreover, it is frequently not clear to a common user which policy is actually applied by a system, and which are its consequences. Similarly, common users may find it difficult to formulate their own privacy requirements and compare them with whatever privacy policy is advertised by a web service.

In this paper we describe the policy framework *Protune*, designed and implemented to support the creation and enforcement of advanced policies, supporting not only traditional access control but also trust negotiation (to automate security checks and privacy-aware information release) and second generation explanation facilities (to improve user awareness about—and control on—policies).

2. POLICIES AS SEMANTIC MARKUP IN PROTUNE

Policies are semantic markups because they specify declaratively part of the semantics (in terms of behavior constraints and admissible usage) of the static or dynamic resources policies are attached to. Accordingly, semantic techniques have several roles in Protune

- Policies are formulated as sets of axioms and meta-axioms with a formal, processing-independent semantics; this is the basis for consistent treatment of policies for different tasks: enforcement, negotiation, explanations, validation, etc.

- The aforementioned tasks involve different automated reasoning mechanisms, such as deduction for enforcement, abduction and partial evaluation for negotiation, pruning and natural language generation for explanations, etc.

- The auxiliary concepts needed to formulate policies (such as what is a public resource or an accepted credit card, ...) and the link between such concepts and the evidences needed to prove them (e.g. which X.509 credentials are needed or what forms need to be filled in) are defined by means of lightweight ontologies that may be included in the policy itself or referred to by means of suitable URIs; therefore, unlike XACML *contexts*, Protune's auxiliary concepts are machine understandable and allow agent interoperability

3. NEGOTIATIONS

In response to a resource request, a server may return its policy for accessing the resource. The policy may contain (a reference to) an auxiliary ontology, as explained in the previous section. In the simplest case, user agents may use such machine understandable information to check automatically whether the policy can be fulfilled and how, thereby (partially) automating the operations needed for traditional access control and facilitating navigation in the presence of articulated policies. In advanced scenarios, a user agent may reply with a counter-request in order to enforce the user's privacy policy.

The availability of a framework capable to enforce access control and negotiations automatically given the two policies has remarkable consequences on privacy as well as usability. On the one hand a direct intervention of the user in the decision process would be required less frequently, since the user's decision would be already embedded to some extent into the policies (s)he defines and sensitive resources would (not) be disclosed without necessarily asking the user every time. On the other hand, such usability improvement may encourage users to refine their policies by specifying articulated policies, thereby improving privacy guarantees.

4. PROTUNE'S POLICY LANGUAGE

Protune's policy language is a logic programming language enhanced with an object oriented syntax. For example, the rule that allows to buy a book by giving a credit card could be encoded with a set of rules including:

*This work has been partially supported by the network of excellence REWERSE, IST-2004-506779.

$allow(buy(Resource)) \leftarrow$
$credential(C),\ valid\_credit\_card(C),\ accepted\_credit\_card(C).$

$valid\_credit\_card(C) \leftarrow$
$C.expiration : Exp,\ date(Today),\ Exp > Today.$

where $C.expiration : Exp$ is an O.O. expression meaning that Exp is the value of C's attribute *expiration*.

Protune supports two pre-defined predicates: *credential* and *declaration*. An atom $credential(x)$ is true when an object x representing an X.509 credential is stored in the current negotiation state. A peer may make $credential(x)$ true on the other peer by sending the corresponding credential. Predicate *declaration* is analogous but its argument x is an unsigned semi-structured object similar to a web form that, for example, can be used to encode a traditional password-based authentication procedure as in:

$authenticated \leftarrow$
$declaration(D),\ valid\_login\_data(D.username, D.password).$

When a set of rules like the above ones is disclosed by a server in response to a client's request, the client—roughly speaking—works back from $allow(Request)$ looking for the credentials and declarations in its portfolio that match the conditions listed in the rules' bodies. In logical terms, the selected credentials and declarations (represented as logical atoms) plus the policy rules should entail $allow(Request)$: this is called an *abduction problem* by the automated reasoning community. After receiving credential and declarations from a client, a server checks whether its policy is fulfilled by trying to prove $allow(Request)$ using its own rules and the new atoms received from the client, as in a standard *deduction problem*.

When a client enforces a privacy policy and issues a counter-request as in Alice's scenario, the roles of the two peers are inverted: the client plays the role of the server and viceversa. For example, the client may publish rules governing credit card release such as:

$allow(release(C)) \leftarrow credit\_card(C),\ bbb\_member(Server), \ldots$
$bbb\_member(Server) \leftarrow$
$credential(BBB),\ BBB.issuer = \text{“}BBB\_CA\text{”}, \ldots$

5. EXPLANATIONS: PROTUNE-X

Protune-X, the explanation facility of Protune, plays an essential role in improving user awareness about—and possibly control over—the policy enforced by a system. Protune-X is also a major element of Protune's *cooperative enforcement* strategy: Since a crude denial may discourage new users from using a system, the explanation system is meant to enrich the denials with information about how to obtain the permissions (if possible) for the requested service or resource.

For this purpose four kinds of queries are supported: *How-to* queries provide a description of a policy and may help a user in identifying the prerequisites needed for fulfilling it or may be used to verify a complex policy. *What-if* queries are meant to help users *foresee* the results of a hypothetical situation, which may be useful for validating a policy before its deployment. Finally, *why* and *why-not* queries explain the outcome of a concrete negotiation (i.e. provide a *context-specific* help) and can be used both by end users who want to understand an unexpected response, and by policy administrators who want to diagnose a policy.

Some of the major desiderata that guided Protune-X's design are

- *explanations should not increase significantly the computational load of the servers.* For this reason explanations are produced by a distinct module, ProtuneX, which operates client-side
- *almost no further effort should be added to the policy instantiation phase.* This is achieved by exploiting *generic heuristics* as much as possible. In most cases, the only extra effort needed for enabling explanations consists in writing verbalization metarules in order to specify how domain-specific atoms have to be rendered, e.g.

  $passwd(X, Y) \rightarrow verbalization :$
  $Y\&\text{“ is the password of ”}\&X.$

- *explanations should support so-called second generation features.* Such features include methods to present the explanation in manageable pieces, highlight relevant information while pruning irrelevant parts, present explanations in a user friendly fashion rather than following the engine's artificial reasoning method

6. DEMO: POLICY-DRIVEN WEB CONTENT PROTECTION & PERSONALIZATION

We have integrated Protune in a Web scenario capable of advanced decisions based on expressive conditions, including credential negotiation to establish enough trust to complete a transaction while obtaining some privacy guarantees on the information released [1, 2]. We have developed a component that is easily deployable in web servers supporting servlet technology, which adds support for negotiations and policy reasoning. It allows web developers to protect static resources by assigning policies to them. In addition to protection of static content, it also allows web developers to generate parts of dynamic documents based on the satisfaction of policies (possibly involving negotiations). We provide an extension to the web design tool Macromedia Dreamweaver in order to help web designers to easily and visually assign policies to their dynamic web pages[1]. A live demo is publicly available[2] as well as a screencast[3].

7. DISCUSSION AND CONCLUSIONS

We have illustrated the policy framework Protune and its implementation, reporting some positive, preliminary performance evaluation experiments. More information about Protune and the vision behind it can be found on the web site of REWERSE's working group on Policies: `http://cs.na.infn.it/rewerse/`.

8. REFERENCES

[1] Piero A. Bonatti and Daniel Olmedilla. Driving and monitoring provisional trust negotiation with metapolicies. In *6th IEEE Policies for Distributed Systems and Networks (POLICY 2005)*, pages 14–23, Stockholm, Sweden, June 2005. IEEE Computer Society.

[2] Piero A. Bonatti, Daniel Olmedilla, and Joachim Peer. Advanced policy explanations on the web. In *17th European Conference on Artificial Intelligence (ECAI 2006)*, pages 200–204, Riva del Garda, Italy, Aug-Sep 2006. IOS Press.

[1] As described in `http://skydev.l3s.uni-hannover.de/gf/project/protune/wiki/admin/?pagename=Integration+with+Dreamweaver`

[2] `http://policy.l3s.uni-hannover.de/`

[3] `http://www.viddler.com/olmedilla/videos/1/`