Policy Based Protection and Personalized Generation of Web Content

Sergej Zerr*, Daniel Olmedilla*, Juri L. De Coi*, Wolfgang Nejdl* and Piero A. Bonatti†, Luigi Sauro†,

*L3S Research Center
Hannover, Germany
Email: {zerr,olmedilla,decoi,nejdl}@L3S.de
†Università di Napoli Federico II
Napoli, Italy
Email: {bonatti,sauro}@na.infn.it

Abstract—The World Wide Web offers easy sharing of information, but provides only few options for the protection of sensitive information and other sensitive resources. Traditional protection mechanisms rely on the characterization of requesters by identity, which works well in a closed system with a known set of users. Trust negotiation protocols have emerged as a solution for open environments such as the Web, in which parties may make connections and interact without being previously known to each other. In this paper we present an access control framework for the Web that not only provides advanced protection mechanisms for static resources but also personalized generation of content. Our approach separates security from the application logic, integrates a flexible and expressive policy language, enables (possibly automated) interactions with human and software agents, and boosts user awareness and cooperative enforcement of such policies.

Keywords—policies, content protection

I. INTRODUCTION

Open distributed environments like the World Wide Web offer easy sharing of information, but provide few options for the protection of sensitive information and other sensitive resources. Typically, this protection is based on the assumption that a requester is already known by the server (e.g., by means of previous registration and user/password authentication mechanisms). This way, the server is able to map the identity of the requester into a permissions table in order to grant or deny access to a resource.

Nowadays, due to the success of the WWW and therefore to the big amount of potential users a server might have, maintaining a table of authorizations based on identities is no longer desirable. Specifically, the Web provides an environment where parties may make connections and interact without being previously known to each other. In many cases, before any meaningful interaction starts, a certain level of trust must be established from scratch through an exchange of information between the two parties. However, the more a personal information is sensitive the more it cannot be candidly disclosed to an unknown party, thus trust establishment should proceed by means of bilateral steps, i.e. it should be negotiated [15].

In trust negotiation a client and a server iteratively request (possibly verifiable) information (e.g., credentials) in order to satisfy each other’s policies and be able to perform advanced authorization decisions. Examples of such pieces of exchanged information might be whether an entity is an European citizen, a student in a German university or a member of a seal program such as BBBOline or TRUSTe. Policies used in trust negotiation provide a flexible and expressive way of specifying access control requirements. Although trust negotiation provides a promising approach for access control on open environments, no flexible integration on the Web has been provided to date.

Furthermore, many of the protected resources are not static, but rather generated dynamically, and sometimes the content of a dynamically generated web page might depend on the security level of the requester. Currently these scenarios are implemented directly on the script languages used to specify how dynamic web pages are to be built. This typically means that the access control decisions that can be performed are either simple and inflexible, or rather costs for its development and maintenance increase. Today it is commonly accepted that access control and application logic should be kept separate, as witnessed by the design of policy standards such as XACML and the WS-* suite.

This paper presents an advanced approach to access control on the Web. It exploits currently emerging trust negotiation approaches [4], [12], [8], [2] by integrating them in a Web scenario. In such a scenario advance decisions can be made based on expressive conditions, including credentials exchanged among entities in order to establish enough trust to be granted access to a resource, while preserving the privacy of information released. In addition, policies can be used in scripting languages such as JSP in order to personalize dynamically generated content, based on locally stored information or requester information obtained through negotiations. Furthermore, using policies allows us to make use of many of the results in the area, including policy verification techniques and the use of automatically generated natural language explanations describing i.e. the requirements to be satisfied before access to a resource is granted or why a previous attempt has been denied.

The rest of the paper is organized as follows: §II describes two motivating scenarios and analyzes the features required for their support. Policies and explanations as well as the PROTUNE policy framework, our choice in this paper, are presented in §III. §IV introduces the general content protection and personalization mechanisms needed to address our running scenarios and its implementation is described in §V. §VI compares our work with related approaches and §VII concludes the paper.
II. WEB CONTENT GENERATION AND PROTECTION

A. Buying a protected resource on the Web

Alice searches on-line for a book she would like to read and finds a web site where it is available. At such web site, it is said that the book is not available for free but it is on sale instead. She would like to know how she can buy it and therefore selects the “How to buy it” link. A new window pops up with a natural language description of the process required to buy the book. Navigating the explanation Alice finds out that several payment methods are at her disposal including bank account transfer (to a German bank) and credit cards. Furthermore, it states that discounts are provided for students and also for members of the ACM. Alice is indeed a student and decides to buy the book. Therefore, she selects the buy option and at that time a (semi-)automatic trust negotiation between Alice and the on-line shop is started. Alice policies state that she would reveal her student card to anyone but her Master Card only to companies certified by the seal program Better Business Bureau. The on-line shop has such a membership credential and is willing to disclose it to any requester. Alice can visualize how the negotiation performs but she turned on the “automatic” switch of her software, so she is not requested for confirmation every time one of her credentials is disclosed. The negotiation terminates and unfortunately, the on-line shop returns an “access denied” web page, indicating that the negotiation was unsuccessful. Luckily, within such a page Alice has a link to the explanation of the negotiation failure. Alice can therefore navigate a natural language explanation of her previous interaction and finds out that although she disclosed her credit card, it was not accepted, the reason being that the shop only accepts VISA cards. Now that Alice knows what went wrong, she tries again with her VISA card and this time the negotiation is successful and she is able to download the book she has just bought.

B. Dynamically generation of personalized content

Bob is a web designer and developer of an organization and is also in charge of the development of the web pages that list all clients of the company. Such a list should contain different information depending on the requester:

- for members of the direction team all data including projects together with such a client and the budget of such projects should be included.
- for any employee of Bob’s organization or any of its partners the list should only include the client name and the contact person.
- for any other person visiting the website, only the list of client names should be displayed.

Bob creates a server page which will be used to generate dynamic web pages. In order to decide which information is included in the list, he simply states the condition that must be fulfilled for each attribute of the client in order to be included in the page. This way, Bob specifies “name” to be always included. Then if the condition “member of direction team” is satisfied, the page will include name, project, budget and contact person. Else if the condition “consortium employee” is satisfied, then only contact person is added. The definition of who is member of the direction team or who works for a consortium partner is not required by Bob to develop the server page, and he is not necessarily allowed to access such information. Instead, a policy manager will specify that information, including that any external requester should certify with a credential that she is an employee of a partner company.

C. General Requirements

From the previous paragraphs we can extract some of the main requirements needed to be addressed when building such scenarios. This section summarizes and briefly describes the most important ones:

From the point of view of the user:

- Flexible privacy preserving mechanisms. Before a user discloses a sensitive credential such as a credit card, she would like to have some verifiable information about the server she is releasing it to. Therefore, interactions are not anymore a take-it-or-leave-it approach but a bilateral negotiation.
- User awareness. The user must be informed in an easy and understandable way about what steps and information must be released in order to successfully perform the transaction.
- No extra software required. Ideally, the user should be able to transparently (negotiate) access resources as she would normally do, without requiring the installation and management of extra software. She would only be required to create or personalize her policies\(^1\).

From the point of view of the web server provider:

- Expressive access control mechanisms. Allowing for more articulated policies, better fitting the business model of the organization or owner and providing better and more advanced services to the users (possibly clients).
- Ease of specification and separation of duties. Access control policies, as well as business rules, should be easily and declaratively specified by policy managers and content owners, not requiring being a developer or computer expert.

\(^1\)In a general scenario, default policies should be provided by credential issuers in order to support users that will not specify or personalize their own policies (e.g., to support people not used to computer work).
In many cases, policies should even be kept confidential to administrators and developers.

- **Fine-grained protection and content generation.** Such access control policies and business rules shall be used to protect both static resources (e.g., static HTML pages or PDF documents) but also to generate personalized content (e.g., dynamic HTML pages).

- **User awareness.** A server benefits from transparency and therefore it is interested in transmitting the policy behind an access control decision or business process as clear and understandable to the user as possible. Natural language explanations provide a perfect solution for that. Note that decoupling policies and application logic is a necessary prerequisite to keep policies declarative, which in turn is essential for building explanations automatically.

- **Keep costs low.** Ideally, addressing the previous requirements should not increase the investment required for development and maintenance of the system nor the overall load of the web servers. On the contrary, a similar approach should lead to better systems in shorter time and at lower costs, due to factors such as improved readability/maintainability, declarative policy re-use, ontology re-use (say, roles and user categories of common interest), automated documentation (produced by the explanation facility) and automated interactions with human and software agents (through the negotiator). Roughly speaking, the goal is that programming activities related to access control are being converted into declarative configuration tasks.

### III. Policies: Representation and Explanations

Policies are pervasive in all web-related contexts and allow (among others) for security and privacy descriptions in a machine understandable way. More specifically, service or information providers may use security policies to protect any system open to the internet and control access to resources by describing the conditions a requester must fulfill. At the same time, service or information consumers may regulate the information they are willing to disclose by protecting it with privacy policies. In addition to this, business rules specify which conditions apply to each customer of a web service. Other policies specify constraints related to Quality of Service (QoS). In E-government applications, visas and other documents are released according to specific eligibility policies. This list is not exhaustive and is limited only by the class of applications that can be deployed in the world wide web.

Most of these policies make their decisions based on similar pieces of information [1] – essentially, properties of the peers involved in the transaction. For example, age, nationality, customer profile, identity, and reputation may all be considered both in access control decisions, and in determining which discounts are applicable (as well as other eligibility criteria). It is appealing to integrate these kinds of policies into a coherent framework, so that (i) a common infrastructure can be used to support interoperability and decision making, and (ii) the policies themselves can be harmonized and synchronized.

In the general view depicted above, policies may also establish that some events must be logged (audit policies), that user profiles must be updated, and that when a transaction fails, the user should be told how to obtain missing permissions. In other words, policies may specify actions whose execution may be interleaved with the decision process. Such policies are called **provisional policies.** In this context, **policies act both as decision support systems and as declarative behaviour specifications.** An effective user-friendly approach to policy specification could give common users (with no training in computer science or logic) better control on the behaviour of their own system.

Of course, the extent to which this goal can be achieved depends on the policy’s ability to **interoperate** with legacy software and data – or more generally, with the rest of the system. Then a policy specification language should support suitable primitives for interacting with external packages and data in a flexible way.

However, it is not only important how flexible and expressive the policy language is, but also whether administrators and general users will be able to understand their policies and the decisions made when reasoning over them. Web applications need to help new users in obtaining the services that the application provides, so potential customers should not be discouraged. Whenever prerequisites for accessing a service are not met, web applications should explain what is missing and help the user in obtaining the required permissions. As part of this cooperative enforcement, advanced **explanation mechanisms** are necessary to help users in understanding policy decisions and obtaining the permission to access a desired service.

#### A. Protune Framework

The PRovisional TrUst NEgotiation framework **PROTUNE [2]** addresses all these requirements. It aims at combining distributed trust management policies with provisional-style business rules and access-control related actions. Protune’s rule language extends two previous languages: PAPL [4], which until 2002 was one of the most complete policy languages for trust negotiation, and PeerTrust [8], which supports distributed credentials and a more flexible policy protection mechanism. In addition, the framework features a powerful declarative meta-language for driving some critical negotiation decisions, and integrity constraints for monitoring negotiations and credential disclosure.

**PROTUNE** provides a framework with:

- A trust management language supporting general provisional-style\(^2\) actions (possibly user-defined).
- An extensible declarative meta-language for driving decisions about request formulation, information disclosure, and distributed credential collection.
- A parameterized negotiation procedure, that gives a semantics to the meta-language and provably satisfies some desirable properties for all possible meta-policies.

\(^2\)Authorizations involving actions and side effects are sometimes called provisional.
• Integrity constraints for negotiation monitoring and disclosure control.

• General, ontology-based techniques for importing and exporting meta-policies and for smoothly integrating language extensions.

• Advanced policy explanations in order to answer why, why-not, how-to, and what-if queries [3].

The advanced policy explanation facility has been designed to clarify in natural language control ³, the explanation is broken into manageable pieces and the user is given a navigable structure that can be explored, starting with a general overview of the policy and allowing to focus on details of parts of it. Explanations can for instance describe how to access a resource or why a negotiation has failed.

For instance, once Alice’s negotiation from §II-A is finished and for the case where the credit card was disclosed but not accepted, a why-not explanation as follows would be generated:

I can’t prove that it is allowed to download book.pdf
because:
A. Smith is authenticated [details]
but it is not the case that
A. Smith has paid 99 euros for book.pdf [details]
and
A. Smith is authenticated [details]
A. Smith is a student [details]
but it is not the case that
A. Smith has paid 90 euros for book.pdf [details]
and
A. Smith is authenticated [details]
but the following conditions cannot be simultaneously satisfied
A. Smith is an ACM member [details]
A. Smith has paid 90 euros for book.pdf [details]

Descriptions of the policy are given in natural language and in order to get more details on different concepts used, a user can click on the details link and refine the explanation by focusing on that concept in particular. For example, focusing on “User is a student” it would be possible to find out that only student credentials from European universities are accepted. Equally, expanding the line “User has paid 99 for book.pdf” would show that payment requires two steps, namely disclosure of a credit card and confirmation from the user that payment should be performed. With this type of explanation, the user can then explore why the payment was not performed even though she provided a valid credit card and can find out that her credit card is not among those types accepted by the on-line seller. This kind of explanations use domain independent heuristics in order to filter out irrelevant information and highlight important one.

IV. WEB CONTENT PROTECTION AND PERSONALIZATION

Policies, negotiations and explanations are key elements that contribute to enhance current Web authorization decisions and interactions among users and web servers. We illustrate both, a descriptions and formalizations of these elements on following simple example policies, defined on the client and server side.

Client policy: The user is willing to disclose his University credential including student ID and University name only to a institution possessing a credential issued by “BBB”.

Formally:
%The student credential is disclosed
%only to BBB proven institutions:
allow(sentCredential('Student at Hannover University') :-
  sentCredential(Credential),
  Credential.issuer:‘BBB’.

Server policy: The server grants access to books only to students of either Hanover University, or the Open University. We assume that the server holds a credential issued by “BBB” and is willing to disclose it to everybody.

Formally:
%Access policy:
allow(access(Book)) :-
  protected(Book,Role),
  sentCredential(Credential),
  Credential.issuer:Issuer,
  university(Issuer),
  Credential.role:Role.
%The list of allowed Universities:
university(‘University Hannover’).
university(‘Open University’).
%NL explanations:
access(Book) → explanation:
  “The requester can access “ & Book.
university(Issuer) → explanation:
  “The credential is issued by ” & Issuer.

The list of protected documents on the server can be created manually or automatically (e.g., listing all documents located in a specific folder). A set of rules is generated from this list and added to the policy:

%The list of protected documents:
protected(‘Formal Theories of Information.pdf’, student).
protected(‘LNCS Titles Published in 2003.pdf’, student).

This section presents several mechanisms in which
• static web resources are protected by means of explainable policies and support for negotiations.
• content is generated dynamically and personalized based on policies (possibly involving negotiations).

In both cases, we make use of the technologies available nowadays (HTTP, Java applets and standard Web browsers) so clients are not required to install any additional software in order to access policy-enhanced web sites.

A. Static Content Protection

Nowadays, a large amount of the resources provided by web servers is static, that is, such resources already “exists” (they are not created “on the fly”) and are stored locally. An authorization
process in this context involves

1) Identifying the resource requested.
2) Checking whether the requester is allowed to retrieve it (authorization decision).
3) If granted, send the requested resource.

The flexibility of the mechanism used in step 2 determines the expressiveness of the complete access control decision. Traditional systems use a model based on triples of the form \((S, R, M)\) where \(S\) is the set of subjects (known users), \(R\) is the set of resources to be protected and \(M\) is a matrix that contains in each cell the list of permissions a subject has over a resource. While these systems allow for mapping the identity of an already registered user with the appropriate permissions, in this section we present a more flexible and powerful approach, in which users do not need to register at all but will be requested to provide (possibly verifiable) information required to perform the access control decision (e.g., proof of enrolment in an European university). Furthermore, users can at any time request an explanation of how to satisfy the server’s policy, why access to the resource has been denied (or granted) as well as what would be the result of the access control decision if some conditions were satisfied (e.g., “would I access the resource if I was a member of Open University?”). Using our example, defined above, a user policy would match the server policy and access would be granted.

1) Negotiations: Since user policies and credentials are sensitive and must be kept private, the negotiation software (agent hereafter) has to run on the local machine, so no other party can access such information. Integration of negotiations should be performed in a transparent way to the user, besides requiring her set of policies describing how the negotiation agent should be driven. Therefore, the agent should be integrated with standard web browsers, since this is the main way users access web content nowadays. Two options are available here:

- Browser plug-in. Developing the trust agent as a plug-in allows for more controlled operations and integration with the browser. However, it reduces the compatibility of the agent since a different plug-in has to be developed for each different web browser.
- Java applet. Developing the trust agent as an applet has the benefit that one single implementation can be delivered to any browser supporting it (most current browsers do). On the other hand, integration with the browser is very limited.

In both cases, from a conceptual point of view, the communication between the agent and the browser works in the same way, and therefore the procedure described in the rest of this section can be applied to any of both solutions (§V describes later the details of our implementation). The steps to be performed are briefly described in the following:

A local trust agent on the client’s browser initializes his policy engine with the policies and credentials that are locally available (Step 0). The negotiation agent takes over the control by catching each new request for a Web document from the browser and forwards the HTTP request to the web server (Step 1). Assuming that the document is protected by policies that require a negotiation, the server returns a standard response message with a customized error code meaning that a negotiation is required. This message will be shown in usual browsers which do not have a local trust agent integrated. (Step 2). The trust agent receives the answer and after interpreting the error code received and detecting the need for trust negotiation, it forwards the response to the local policy engine (Step 3). The local policy engine sends a negotiation request to the server’s engine (Step 4). After the exchange of a (possibly empty) set of extra negotiation messages, the server returns a final response message indicating whether the negotiation is successful or not. If the negotiation failed, the trust agent shows an access denied page together with an explanation describing why it failed. In case the negotiation was successful, the answer is supplied with a session token that allows for the web document retrieval. Using the token the document is requested through the normal HTTP POST request. The web server receives the new HTTP request with a session token attached and inquires the policy engine about the previously performed negotiation. After ensuring that (a) the negotiation was indeed successful and (b) the requester and the requested resource are the same as the ones included in the new HTTP request, the web server sends an HTTP response with the requested resource to the client. In our example, defined above, the user agent would start a negotiation asking for a “BBB” credential before it discloses the user’s University credential.

2) Explanations: In case the negotiation is not successful, the user should be provided with an explanation describing what went wrong, or which pieces of required information were not provided. Since our explanation facility makes use of hyperlinks for navigation, it is straightforward to integrate it into web pages that can be accessed by the user.

It is important to note that the generation of the explanations is performed entirely at client’s side. During negotiations, apart of required credentials, also the policies of both parties are exchanged, in order to let the other party know what information is required to be disclosed. These exchanged policies allow to analyze not only the next step of the negotiation but possibly several steps ahead\(^4\), and in addition, it provides enough information to automatically generate explanations without requesting any extra information from the server.

B. Dynamic Content Generation

In addition to access control over static resources, it is even more interesting to be able to generate personalized content based on the policies specified in the server. The process followed in this case involves

1) Identifying the resource requested.
2) Checking whether the requester is allowed to retrieve it (authorization decision).

\(^4\)From a client point of view, it is not desirable to be requested first an id, and after it has been disclosed, be requested the next credential. In case the client does not have or does not want to disclose the “next” credential (in which case the negotiation would fail), there would have already been an unnecessary disclosure of credentials.
3) If granted, generate the dynamic web page (possibly according to the identity of the requester).
4) Send the requested resource if access was granted.

In this case, while step 2 is the same as the one in the previous section, step 3 allows to personalize content based on i.e. information received from the requester. In this case, policies are a flexible and expressive choice for the declarative specification of personalization rules to be used when generating the content. Moreover, benefits of using policies in this context include (i) they are declarative, therefore making easy the personalization specification, (ii) they are inherently dynamic, allowing for easy changes without needing to rewrite software or recompile sources, (iii) they separate the web communication and visualization of content from the business rules of the content owner and (iv) they do not only support integration of legacy systems (e.g., databases) but also add reasoning capabilities. In our particular case of dynamic content generation, our policies used for personalization do not include only access control policies (in order to find out information from the requester), but also any other kind of condition that can be used to decide which content has to be shown under such conditions. For instance, it is be possible to:

- show different information based on different access control levels (see §II-B).
- display location-aware content based on the contextual information provided by the client trust agent.
- infer and visualize the portfolio of a person (including her direct clients, the clients of the person she supervises, and so on).
- extract the VIP clients of a company while retaining control of who knows about the procedure used to define such clients.
- personalize web page content based on released information from the client such as age or citizenship.

Applying access restrictions to a document’s elements should be easy and just as declarative as policies are. This can be achieved by introducing suitable access control tags (semantic markup) in server pages that will be evaluated when the dynamic web page is being generated. Those tags consider the result of the policy evaluation process which is then used to choose and generate the appropriate content to be delivered. The following briefly describe the different steps involved in the process:

A local trust agent on the client’s browser initializes his policy engine with the policies and credentials that are locally available (Step 0). The negotiation agent takes over the control by catching each new request for a Web document from the browser and forwards the HTTP request to the web server (Step 1). The servlet filter identifies the request as a dynamic web page and invokes the JSP handler (Step 2). The JSP handler evaluates the server page, generating the content to be sent to the requester. For each policy tag found, a request is sent to the policy engine in order to be evaluated (Step 3). If required, the policy engine will perform a negotiation with the requester (Step 4). For each evaluated request the negotiation result is stored in a session table. After evaluating all requests and a session token is returned to the local trust agent on the user side. Using this token the local trust agent starts retrieving the JSP page through the normal HTTP POST request. The JSP handler generates the policy-related content according to the results of the evaluated requests. In case the request fails, the negative tag body is evaluated. In case the request was successful, the positive tag body is evaluated. All positives results are listed in case a "foreach" clause is present within the positive tag body.

This simple process ensures that the content generated in the delivered web page conforms to the specified personalization policies. In our example, defined above, a web master would include policy tags in his JSP pages in order to personalize a view based on user credential. In this case all documents accessible to current user are listed. The code would look like:

```html
<html>
<body>
<h1>A list of accessible documents</h1>
<foreach container="docs" as="doc">
<poljsp:policycondition
   policyname="access(doc)"
   evaluator="self">
   <a href="get?id=<%=doc%>">
   <%=doc%>
   </a>
</poljsp:iftrue>
</foreach>
</body>
</html>
```

V. Implementation

The architecture presented in previous sections is compatible with most trust negotiation languages and application web servers supporting servlet technology. In our implementation, we used the PROTUNE framework in order to perform negotiations and policy reasoning. It is entirely developed in Java and provides an API that permits its easy integration in applications such as applets, Swing/RCP applications or web servers. It also includes integration of legacy systems such as relational databases, RDF stores and LDAP servers, and access to local files and extraction of content based on regular expressions (see Figure 2 for an overview of its architecture).

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 [2]. We have developed a component that is easily deployable in web servers supporting servlet technology (we currently support Apache Tomcat), which adds support for
There is a considerable amount of work described in the literature on policy based information access control. The P3P standard [14] focuses on deciding the disclosure of a user’s sensitive private information based on the privacy practices of the server. P3P is based on data submitted by the server that are claims it makes about itself. [11] enhances the P3P framework with a more expressive language [10] for defining privacy policies and a new trust model for websites based on both decentralized (social previous experiences) and centralized sources (e.g., Google’s pagerank or BizRate). Policy driven negotiation generalizes this by basing resource disclosure on verifiable properties of interest that can be represented in i.e. credentials as well as those coming from integrated trust models.

An approach for protection of the shared web resources within a social network platform using ACE [7] - controlled natural language policies was introduced recently in [6]. Thereby the owner can define access policies to her resources based on their properties. The policies are defined using a natural language policy editor and stored on server. They are evaluated each time the resource is accessed. This mechanism provides a suitable easy-to-use solutions for highly dynamic social platforms and in this way replaces a selection of a predefined security options like used in the most of the modern applications.

Trust negotiation mechanisms have been presented in different research papers [4], [12], [8], [2] though not all of them have been implemented and only some of them have been integrated into a Web related scenario. In particular, [8] integrates trust negotiation on the Web in order to regulate the disclosure of private information between the client and the server, though it does not allow support advanced protection mechanisms as the ones presented here. [9] uses document classification methods to find out whether information to be disclosed by a client is sensitive and if it is associates an access control policy to it, therefore protecting its disclosure. However, though this method might have high-accuracy for clients within a given domain, it does not apply to the more general scenarios described in this paper. Recently, [13] presents an approach for authentication and single sign-on on the Web by relying on e-mail or instant messenger providers but does not address at all authorization mechanisms.

In addition to all previous approaches, this paper introduces the possibility for personalized and access control oriented dynamic generation of content based on declarative policies. This reduces the granularity of the data that can be protected and provides a needed separation between application logic, utilization of content and security[10]. Moreover, it integrates natural language explanations mechanisms as part of the policy driven negotiation process.

VI. RELATED WORK

The World Wide Web offers easy sharing of information, but provide few options for the protection of sensitive information and other sensitive resources. Traditional protection mechanisms rely on the characterization of requesters by identity, which works well in a closed system with a known set of users. Trust negotiation protocols have emerged as a solution for open environments such as the Web. In this paper, we present an access control framework for the Web that not only provides advanced protection mechanisms for static resources but also personalized generation of content.

In particular, this paper presents the benefits derived from the integration of policies and negotiations into Web scenarios in order to

- provide powerful access control mechanisms in open distributed environments based on flexible and expressive declarative policies
- enables (possibly automated) interactions with human and software agents
- personalize the dynamic generation of web content based on information of the requester or other kind of sources and conditions.

[5] is related to this in the sense of fine-grained access control to data. In particular, it presents how authorizations regulating access to XML documents at all levels of granularity are supported.

---

6As described in http://skysdev.l3s.uni-hannover.de/gf/project/protune/wiki/admin/?pagename=Integration+with+Dreamweaver
8http://policy.l3s.uni-hannover.de/
9http://www.viddler.com/olmedilla/videos/1/. We recommend viewing it in full screen.
• separate access control and application logic, therefore clearly distinguishing between web designers and developers from policy managers and content owners.
• help users understand access controls decisions and results of interactions, as part of the cooperative enforcement of server’s policies.

As a side effect, our approach contributes to cost reductions. Factors such as improved readability/maintainability, declarative policy re-use, ontology re-use (say, roles and user categories of common interest), automated documentation (produced by the explanation facility) and automated interactions with human and software agents (through the negotiation agent) are expected to lead to better systems in shorter time with lower costs.

We are currently developing management tools that help policy managers to specify, organize and visualize their policies and organizations to merge policies defined at different levels from different people in the organization. These tools are also designed to inform web developers about the policies available for their developments. Another interesting issue we are currently exploring involves the personalization and access control based on contextual information of the requester, such as time and location.

ACKNOWLEDGMENT

This research has been partially funded by the European Commission and by the TENCompetence Integrated Project (contract 027087) and the Interdisciplinary Research on Security and Safety; Leibniz University Hannover.

REFERENCES


