

# A Policy-Based Access Control Mechanism for the Corporate Web\*

Victoria Ungureanu<sup>†</sup>  
MSIS Department  
Rutgers University  
Newark, NJ 07102  
ungurean@cs.rutgers.edu

Farokh Vesuna  
Information Architects  
70 Hudson Street  
Hoboken NJ 0703

Naftaly H. Minsky  
Computer Science Department  
Rutgers University  
New Brunswick, NJ 08903  
minsky@cs.rutgers.edu

## Abstract

*Current Web technologies use access control lists (ACLs) for enforcing regulations and practices governing businesses today. Having the policy hard-coded into ACLs causes management and security problems which have prevented so far Intranets to achieve their full potential.*

*This paper is about a concrete design of a mechanism that supports policies for regulating access to information via corporate Intranet. This mechanism makes a strict separation between the formal statement of a policy, and its enforcement, the latter being carried out by generic policy engines. The proposed mechanism is easy to deploy, requiring no modifications of current web servers. We provide some preliminary performance results that show that the mechanism is quite affordable, even in its present, experimental stage.*

## 1. Introduction

By eliminating physical and geographical barriers, the World Wide Web has become the foremost medium for disseminating information to people all over the world. Increasingly, Web and Internet technologies have been used by companies or government agencies on private networks, or Intranets, as a service to its employees. For example, Intranets represent an attractive alternative to traditional forms of internal communication by providing seemingly limitless access at dramatically reduced costs. Nowadays, many of the internal documents of a company, including procedures, directories, and training materials are converted to electronic format and publicized on corporate webs.

Although the WWW and corporate webs serve the same

purpose and use the same technologies, they differ significantly in regulating access to information. On one hand, an individual user can access any information on the WWW, subject only to the availability of information. On the other hand, an employee, accessing information via the Intranet corporate web, should obey the policies of the company in question. These policies might, for example, stipulate that an employee might have access to the WWW only during predefined time periods, or only to certain sites, or impose a limit on how many documents may be downloaded. More importantly, since not all staff members are allowed access to all information, enterprise policies regulate access to corporate data. It is the formulation and enforcement of such rules that will concern us here.

The prevailing mechanism for enforcing enterprise policies uses access control lists (ACLs) for determining users rights. ACLs associate with each web page (or directory) a list of users or groups allowed to access that particular web page. While such a mechanism proved to be efficient and reasonably suited for the WWW, burying enterprise-rules into ACLs has several serious drawbacks.

First, management of such ACLs is costly and error prone, especially if the enterprise in question has a large number of employees and/or maintains a large collection of documents. (For example, [8] presents the difficulties encountered when establishing and maintaining a corporate web page for Boeing, whose personnel is over one hundred thousand.) In particular, every time an employee leaves the company, or her rights changes, an exhaustive examination of *all* ACLs is needed on *all* servers. Modifying the policy itself is even more problematic.

Second, current mechanisms have been recently criticized as being too weak to support many policies deemed useful for corporate Intranet (cf. [10]). Whenever one needs to enforce a policy which does not fall into the supported patterns, there is no other way, but to build from scratch a specific interface for it (cf. [3]). But, embedding

---

\* Appeared in Proc. of the 16th Annual Computer Security Applications Conference (ACSAC 2000), December 2000, New Orleans

<sup>†</sup> Work supported in part by DIMACS under contract STC-91-19999

a policy into code is time consuming and expensive to carry out.

To deal with these problems we propose in this paper a security mechanism that makes a strict separation between the formal statement of a policy and its enforcement, which is carried out by generic policy engines. Making rules explicit, instead of concealing them into ACLs, makes changes easier to perform. The language proposed here for defining this kind of regulations is expressive enough to support a wide category of enterprise policies, including: policies supported by conventional discretionary models, as well as the more sophisticated role based access models. Time and state constraints are also easily expressible in this formalism.

The rest of the paper is organized as follows. Section 2 describes the system architecture; in Section 3 we describe how policies are formalized under this scheme. We follow, in Section 4 by presenting two policies which are very difficult, if at all possible, to express using conventional mechanisms, and by showing how they can be implemented under our formalism. Section 5 describes the system implementation and presents some preliminary performance results. Section 6 discusses related work, and Section 7 concludes the paper.

## 2. System Architecture

In this section we describe the system architecture: we start by specifying the assumed Intranet infrastructure. We then present the enforcement mechanism and its security.

**Intranet Infrastructure** The devised mechanism assumes that Intranet is cordoned off from the public Internet by firewalls, and that employees' access to the Web is mediated by proxy servers. Since terminology varies slightly from vendor to vendor, we will precisely specify the intended meaning of these terms.

Firewalls are designed to prevent unauthorized access to or from a private network. In the following, we will assume that the protection is achieved by filtering packets based on the service required and the IP address of the source and destination.

A proxy server is logically placed between the Web browser (or other client application) and the Web server. A proxy server intercepts HTTP requests and checks if it can fulfill them itself. If not, it forwards the request to the destination server; once the response arrives it is cached by the proxy for a predefined period of time. Caching improves overall performance, thus making proxy servers a well established technology [16].

**Enforcement Mechanism** Enterprise regulations are brought to bear by generic policy engines which can inter-

pret rules written in a specified language. (The language for writing policies will be described in the next section.) Under our scheme, each proxy server has an associated policy engine, to which it passes for evaluation requests and/or responses. Proxy servers are trusted to service only requests sanctioned by the policy engine.

We now present in detail the steps performed to serve a request issued by an employee  $x$ .

- The request is intercepted by the proxy server assigned to  $x$ . If a session was not previously established, the proxy requires  $x$  to authenticate itself<sup>1</sup>. (Note that the authentication step is performed only once. The web browser will automatically add this information to all subsequent requests issued by the user in a particular session.)
- The proxy sends the request to the policy engine. The request is evaluated with respect to the rules in effect, the identity and state of  $x$ .
- In the case the request has been authorized, there are two cases to consider. If it is a request to write (`put`) a document, then the proxy services the request without further ado.
- If it is a request to read a document  $d$ , then the proxy fetches  $d$  either from its own cache or from the Web server. Once  $d$  is retrieved, the policy engine evaluates the policy in order to decide whether the document should be finally delivered to  $x$ . The decision is based, among other things, on the rating and size of the document in question.

The enforcement mechanism presented here describes the general case where a `get` request triggers two policy evaluations: one when the request is issued and another one when the document is retrieved. This is necessary for some policies; in particular, the example policies presented in Section 4 require both evaluations. But, depending on the policy in question, the proxy server can be configured to trigger only the evaluation of the request, or only the evaluation of the reply, thus yielding a better overall performance.

**Security of the proposed mechanism** With regard to the URL of the requested document, requests can be categorized into: (a) requests made by employees for documents on the Intranet—which we call *inbound requests*; and (b) requests made by employees of the enterprise for accessing documents on the Internet—which we call *outbound requests*.

<sup>1</sup>In the current implementation, users are authenticated by means of passwords. We are aware of the drawbacks of this choice, which include sending the password in clear over the Intranet. We plan to replace passwords with certificates; such a change will require, however, modifying the proxy server.

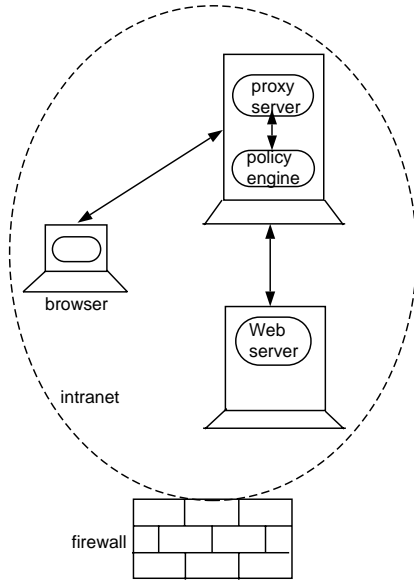


Figure 1. Inbound request.

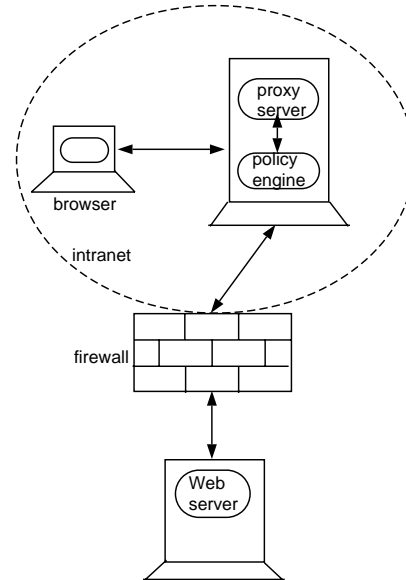


Figure 2. Outbound request.

We will show that given our assumptions on the Intranet infrastructure, all serviced requests are subject to the policy of the enterprise in question. We start with inbound requests, illustrated in Figure 1.

Since an inbound request refers to an internal document, it is serviced by a Web server situated inside the Intranet. Such a server can be easily configured to provide service only to the proxy servers belonging to the enterprise in question. Any request not coming from a recognized proxy, and consequently not known to be in compliance with the enterprise policy, is denied.

Similarly, in order to have outbound requests (Figure 2) controlled by the enterprise policy it is sufficient to configure firewalls to filter out all packages requiring http or ftp service whose source is not the IP address and port of a proxy server. Given the fact that, even in large enterprises the number of proxy servers is small, such a configuration is easy to perform and maintain.

### 3. Policy Formulation

Broadly speaking, policies are rules of conduct that characterize the behavior of a group of people involved in a certain enterprise<sup>2</sup>. Accordingly, a policy refers to the *actions* being regulated, the *group* of participants that engage in this actions, and the *guiding principles*.

We are concerned here with policies regulating employees access to web via a corporate Intranet. In this case,

<sup>2</sup>According to the the American Heritage Dictionary, the term policy means “a general principle that guides the actions taken by a person or group.”.

the actions being regulated are requests for documents; the group is comprised of the company employees; and the rules mandate when a request is to be serviced. The following, then, is our definition of a such a policy, which we call a web-policy.

A web-policy  $WP$ , can be defined as the tuple  $\langle \mathcal{R}, \mathcal{A}, \mathcal{CS}, \mathcal{E}, \mathcal{F} \rangle$  where,

1.  $\mathcal{R}$ —is an *explicit and enforced* set of rules regulating access to web.
2.  $\mathcal{A}$  is the set of *agents* belonging to  $WP$ . Typically, it contains the employees of the company in question.
3.  $\mathcal{CS}$  is a set  $\{\mathcal{CS}_x \mid x \in \mathcal{A}\}$  of *control states*, one per member of  $\mathcal{A}$ .  $\mathcal{CS}$  is mutable, subject to rules  $\mathcal{R}$  of the policy.
4.  $\mathcal{E}$  is the set of *events* occurring at members of  $\mathcal{A}$ , whose treatment is subject to rules  $\mathcal{R}$ . These, so called *regulated events*, deal mostly with issuing a request and receiving a reply.
5.  $\mathcal{F}$  is the set of facts representing information about members or URLs.

We will now elaborate on some components of a web-policy.

**Events** The events that are subject to rules are called *regulated events*. Each such event is viewed as occurring at a certain agent  $x$ , called the *home* of the event—strictly speaking, however, events occur at a policy engine. Specifically, the event is triggered at the policy engine assigned to the proxy server by which the request is intercepted.

Three types of events are currently being implemented:

1. `sent(x, request(...))`: occurs at the policy engine when agent `x` issues a request for a document. The parameters of the `request` term are as follows: `protocol(p)`, where `p` is the type of the protocol used (may be `http` or `ftp`); `domain(d)`, where `d` is bound to the name of the domain where document is stored; `port(no)`, where `no` stands for the port number of the web server; `path(pa)`, where `pa` is bound to the path to the document; `file(f)`, where `f` denotes the name of the document; and `method(m)` where `m` is the method to be used (e.g. `get`, `put`). For example, if the original request is to `get` document with URL

```
http://www.anEnterprise.com/  
  aName/public_html/  
  index.html,
```

then parameters of the `request` term are respectively: `protocol(http)`, `domain([com, anEnterprise, www])`, `port(80)`<sup>3</sup>, `path(aName, public_html)`, `file([html, index])`, and `method(get)`.

2. `arrived(x, reply(...), forRequest(...))`: occurs at a policy engine when a response, for a previously request issued by `x`, is retrieved by the proxy. The `reply` term has the following parameters: `time(t)` where `t` is the time when the document was last modified; `size(s)` where `s` is the size of the file (in KB); and `rating(r)` where `r`, if available, is bound to the rating of the document. The term `forRequest` contains information identifying the corresponding request (protocol type, domain name, port number, etc).
3. `obligationDue(...)`: the occurrence of this event means that it is time to enforce an *obligation* previously imposed on the home of this event. Obligations are widely considered essential for the specification of enterprise policies, along with permissions and prohibitions [11, 13]. Informally speaking, an obligation is a kind of *motive force*. Once an obligation is imposed on an agent `x`—which can be done as part of the ruling for some event at `x`—it ensures that a certain action (called *sanction*) is carried out at this agent, at a specified time in the future, when the obligation is said to *come due*—provided that certain conditions on the control state of the agent are satisfied at that time. The

circumstances under which an agent may incur an obligation, the treatment of pending obligations, and the nature of the sanctions, are all governed by the policy. (For a detailed presentation of obligations the reader is referred to [15].)

**Control States** In our formalism, each member `x` has an associated control state, denoted by  $CS_x$ , consisting of a *mutable* set of Prolog-like terms, called the *attributes* of `x`. The function of control states is to enable the policy to make distinctions between different agents. The meaning of such attributes, is determined by the rules of the policy. Typically, the control-state of an agent represents such things as the role of this agent, or record a history of its activity—and it can change dynamically, subject to the rules of the policy. Consider, for example, a company where access to documents is determined by the role held by an employee. Under this policy, each employee has in her control state a term `role(r)`, where `r` denotes the role of this agent (for example `r` may be `manager`, `secretary`, `programmer`).

Control-state  $CS_x$  is not directly accessible to agent `x` (or to any other agent). It is maintained by the proxy server assigned to `x`, and can be changed only by operations included in the ruling for events at `x`.

**Facts** Are used to represent general information about URLs, domains and roles; facts are currently implemented as Prolog unit clauses. As an example of facts usage consider an enterprise whose policy is to allow access only to local documents. Such a policy might be implemented by using a set of facts enumerating the domains belonging to the Intranet of this enterprise.

**Rules** Are defined over a certain type of events occurring at members of  $\mathcal{A}$ , and are evaluated by the policy engine whenever one of the above events is triggered. The evaluation produces a *ruling* which is a possibly empty sequence of primitive operations, to be carried out in response to the event in question. (An empty ruling simply implies that the event in question has no consequences—such an event is effectively ignored.) A ruling may require modification of the control state of the issuer, may authorize or reject requests, and may demand addition of extra tags to the request.

Enterprise regulations can be quite naturally expressed by means of any language based on *event-condition-action* (ECA) kind of rules. We are here using an extension of a language devised for support of security policies [12] built on top of Prolog [6]. We note, that we might easily replace the language with a simpler and more restrictive language, if this will become necessary. Such a change will be easy to accomplish, because it requires no change in the model, and only a minor change in the toolkit that implements it.

<sup>3</sup>since no port was given in the request, the default port of the http protocol is used)

Operations on the control-state	
$t@CS$	returns true if term $t$ is present in the control state, and fails otherwise
$+t$	adds term $t$ to the control state;
$-t$	removes term $t$ from the control state;
$t1 \leftarrow t2$	replaces term $t1$ with term $t2$ ;
$incr(t(v), d)$	increments the value of the parameter $v$ of term $t$ with quantity $d$
$dcr(t(v), d)$	decrements the value of the parameter $v$ of term $t$ with quantity $d$
Communication operations	
authorize	communicates to the proxy that the request is authorized
reject	communicates to the proxy that the request is rejected
append( $t, v$ )	instruct the proxy to append tag $t$ with value $v$ to the request
Miscellaneous	
$t@L$	returns true if term $t$ is present in list $L$ , and fails otherwise
imposeObligation( $type, dt$ )	causes the triggering of an obligationDue( $type$ ) event $dt$ seconds later.

**Figure 3. Some primitive operations.**

For now, the rules are defined by means of a Prolog-like program which, when presented with a goal, representing a regulated-event triggered by a request of an agent  $x$ , is evaluated in the context of the control-state of this agent. In addition to the standard types of Prolog goals, the body of a rule may contain a distinguished type of goal, called a *do-goal*. A do-goal has the form  $do(p)$ , where  $p$  is a *primitive-operation*. When such a statement is evaluated the primitive operation  $p$  is appended to the ruling. Primitive op-

erations currently supported include operations for testing the control-state of an agent and for its update, communication with the proxy-server, and operations on obligations. A sample of primitive operations is presented in Figure 3.

## 4. Examples of Enterprise Policies

To illustrate the expressive power of the proposed mechanism, we present here the implementation of two policies: the synchronized update policy and the traffic control policy.

### 4.1. Synchronized Update Policy

The maintenance of corporate webs poses several problems [10] which are hard to overcome with today's Web technologies. First, not every employee should be allowed to modify internal pages—a classical access control concern. Second, if, for example, two persons, A and B are allowed to modify a page, then there is the risk that A's modifications would be overwritten by B. While the first problem can be solved by current mechanisms—with high managerial costs, however—the second problem addressed above may be solved only by pushing the responsibility for correct updates to the page writers. To show, how both problems may be addressed under our mechanism, we are introducing here a policy, called synchronized update (*SU*) policy. The requirements of this policy are that updates can be done only by designated persons, and an update is carried on only if no modifications are lost.

The *SU*-policy is established by the three rules displayed in Figure 4. Each rule is followed with a comment (in italic), which together with the following discussion, should provide the reader with some understanding of the nature of rules. Under this policy only employees that have a term *writer* in their control state may modify internal documents<sup>4</sup>. Now, by Rule  $\mathcal{R}1$ , any *get* request is authorized. Rule  $\mathcal{R}2$  mandates that all responses are authorized to be delivered to employees. Additionally, if the issuer of the request is a *writer*, a term *modified* recording the address, name, and the time of last update of the document in question is added to her control state. Finally, Rule  $\mathcal{R}3$  deals with *put* requests, i.e. requests to modify/write documents. Such a request is authorized only if the issuer has the term *writer* in her control state. Additionally, it mandates that the tag '*If-Unmodified-Since*' is to be added to the HTTP request. The value of the tag is the time of the last update of the document in question as recorded in the *modified* term. The addition of the tag ensures<sup>5</sup> that the

<sup>4</sup>This is only a finger exercise, meant to illustrate the mechanism; a full implementation should grant update rights only on certain documents. This can be achieved using, for example, roles and facts as in the next example.

<sup>5</sup>Provided that web servers use version 1.1 of HTTP protocol (or newer).

file will be updated only if not modified in between.

*Initially:* An employee  $x$  might have in her control state a term `writer` denoting that she is allowed to update documents.

```
R1. sent(X,request(Protocol,Domain,Port,
    Path,File,method(get))) :-
    do(authorize).
```

*All get requests are authorized.*

```
R2. arrived(X,response(LastUpdate,-,-),
    forRequest(Protocol,Domain,Port,
    Path,File,Method)) :-
    writer@CS →
    (do(+ modified(Domain, Path,
    File,LastUpdate))
    ;
    true),
    do(authorize).
```

*If the employee  $X$  issuing the request is allowed to update documents; the term `modified(...)` recording the time when the document was last modified is added to  $X$ 's control state.*

```
R3. sent(X,request(Protocol,Domain,Port,
    Path,File,method(put))) :-
    writer@CS →
    ( modified(Domain,Path,File,
    LastUpdate)@CS,
    do(-modified(Domain,Path,File,
    LastUpdate)),
    do(append('If-Unmodified-Since',
    LastUpdate))),
    do(authorize))
    ;
    do(reject).
```

*A put request is authorized only if issued by a writer. Also, a tag recording the last time the document was modified is appended to the request.*

**Figure 4. The rules of the *SU* Policy**

## 4.2. Traffic Control Policy

A general concern in enterprises is assuring quality of service for all agents operating on the Intranet. One way to prevent congestion, and thus provide better service, is to control employees access to Internet. The currently prevailing methods to achieve this purpose are denying access to Internet altogether, or allowing it only during specified time periods (e.g. lunch hours). While these coarse policies attain their primary goal, they have limited utility in an enterprise where employees may need to access the Web to carry out their work. We will present here, a policy for traffic con-

trol that allows a more flexible usage of the Internet. Under this policy, there is a quota specifying for each role in the enterprise, the maximum volume of external data which can be accessed in a given amount of time. An employee is thus free to access any site as long she does not exceeds her quota; however, if the quota is exceeded her requests are denied. A system administrator can thus precisely evaluate and regulate the maximum amount of traffic due to Internet access.

The implementation of this policy, denoted by *TC*-policy, assumes that every employee  $x$  has in her control state the following attributes: (a) `role(r)` where  $r$  denotes her position within the enterprise, and (b) an attribute `servedRequests(sr)` where  $sr$  represents the total volume of data fetched for  $x$  during the current time frame. The database used by *TC* policy contains two sets of facts. A predicate `internal(d)` specifies the domains contained by the Intranet, and a predicate `quota(r,q,dt)`, defining for every role  $r$ , the maximum volume of data  $q$ , that a person holding role  $r$ , is allowed to download in  $dt$  seconds.

The rules of *TC*-policy along with few examples of facts are presented in Figure 5. To see how these rules achieve the informal requirements presented above we will show now the treatment of a document request issued by an employer,  $x$ . There are two cases to consider. If the request is for an internal document, then by Rule *R1* it is approved. If, however the request refers to an external document, then by Rule *R2*, the request is approved only if  $x$  did not exceed her quota. Now, when the response for  $x$ 's request is fetched, the term `servedRequests(sr)` from her control state is increased by  $sr$ , the size of the requested document (Rule *R3*). Finally, by Rule *R4*, every  $dt$  seconds, the value of term `servedRequests` is set to zero, thus allowing an employee to make new requests.

**Discussion** We note that this policy is a nice example of a scenario when evaluations of both request and reply are needed. Since the size of a document is not known at the time the request is made, the term `servedRequests` can be updated only after the document is retrieved, making the evaluation of the arrived event mandatory. And the evaluation of the sent event is necessary in order to prevent fetching documents when the requester quota has been exceeded.

## 5. Implementation and Performance Results

**Implementation** In the current implementation, we are using the Jigsaw proxy server [7], developed by W3 Consortium, which is modified to communicate with our policy engine. The Jigsaw proxy is also used to store the web policy components: the rules of the policy; the facts; and the

*Initially:* Any employee  $x$  has in its control state a term `servedRequests(sr)` where  $sr$  denotes the volume of documents fetched on behalf of  $x$  in the current time frame (initially  $sr$  is 0). An initial obligation to reset  $x$ 's quota is set to fire after  $dt$  seconds.

```
R1. sent(X,request(Protocol,Domain,Port,
    Path,File,Method)) :-
    internal(Domain) →
    do(authorize)
    ;
    do(reject).
```

*All requests for internal documents are authorized.*

```
R2. sent(X,request(Protocol,Domain,Port,
    Path,File,Method)) :-
    servedRequests(SR)@CS,
    role(R)@CS, quota(R,Q,DT),
    SR<=Q →
    do(authorize)
    ;
    do(reject).
```

*A request for an external document is authorized only if the agent in question did not exceed the quota allotted for its role.*

```
R3. arrived(X,response(_,size(S),_),
    forRequest(Protocol,Domain,Port,
    Path,File,Method)) :-
    servedRequests(SR)@CS,
    do(incr(servedRequests(SR),S),
    role(R)@CS,
    quota(R,Q,DT),
    (SR < Q →
    do(authorize)
    ;
    do(reject))).
```

*A response is delivered to the requester  $X$  only if  $X$  did not exceed her quota. Also, the term `servedRequests` is increased by  $S$ , the size of the fetched document.*

```
R4. obligationDue(reset) :-
    do(servedRequests(SR)
    ←servedRequests(0),
    role(R)@CS, quota(R,Q,DT),
    do(imposeObligation(reset,DT)).
```

*When the obligation comes due, the number of served requests is reset and a new obligation is set to fire in  $DT$  seconds.*

```
F1. internal(domain([com,nj,aCompany,www])).
```

```
F2. internal(domain([com,ny,aCompany,www])).
```

```
F3. quota(manager,100,10).
```

```
F4. quota(secretary,1,10).
```

*Examples of facts.*

names, passwords and control states of the employees assigned to it.

The policy engine, written mostly in Java, is generic, in the sense that it can interpret any well formed set of rules. The policy engine evaluates the ruling for events pertaining to an employee  $x$  *sequentially*, in the chronological order of their occurrence, and carries out this ruling atomically, so that the sequence of operations that constitutes the ruling for one event does not interleave with those of any other event whose home is  $x$ .

**Performance** As a gauge for the efficiency of the proposed mechanism we will use the *relative overhead* incurred when serving a request using our mechanism. We define this quantity to be the overhead incurred when servicing a request—i.e., the difference between the service time of a request using our mechanism and the service time of the same request by a unmodified proxy server—divided by the latter.

We will evaluate the relative overhead under various conditions. The general picture that emerges from this section is as follows: the mechanism is quite affordable, even in its present, experimental stage. To be more specific the relative overhead ranges between 0.02 and 0.19.

For our experiment, we used a more complex policy, containing 24 rules and 40 facts. Under this policy, designed for an academic environment, the rights a person has depend on her role in the University (faculty, graduate students, undergraduate students and staff), the department to which the person belongs, and the courses taught/attended. For example, this policy allows a student to access the web pages of a course only if she is registered for the course in question.

The experiment consisted of fetching documents with sizes ranging from 10KB to 1M. Every regulated request triggered on the average the evaluation of 3 rules and 5 facts. Each document was fetched 100 times (in batches of 10) to average over the network load. The proxy server and the policy engine were running on a SUNW, Ultra-2 machine operating at 296 MHz, using Solaris 2.6 operating system and Java 1.2. We took measurements with and without proxy server caching enabled. The measurements, in the case the proxy server and the web server are on the same LAN are presented in Table 1. And in Table 2 we present the results obtained when the proxy and the web server are on different LANs (the proxy was running in New Jersey and the web server in New York). In both tables, average service times are given in milliseconds.

Several conclusions can be drawn from these measurements. As expected, the overhead incurred diminishes with the increase in size of fetched documents. This is because the communication time grows with the size of the document, while the evaluation time is practically constant for any given policy. For large documents, communication time

**Figure 5. The rules of the  $TC$  Policy**

Document size (KB)	No Caching		
	regulated requests	unregulated requests	relative overhead
10	99	84	0.17
100	257	245	0.04
1000	2185	2112	0.03

Document size (KB)	Caching		
	regulated requests	unregulated requests	relative overhead
10	89	75	0.18
100	161	151	0.06
1000	1237	1197	0.03

**Table 1. Average time to service a request when the proxy and the web server are on the same LAN (ms).**

Document size (KB)	No Caching		
	regulated requests	unregulated requests	relative overhead
10	122	106	0.15
100	274	254	0.07
1000	2205	2110	0.04

Document size (KB)	Caching		
	regulated requests.	unregulated requests	relative overhead
10	91	76	0.19
100	164	152	0.07
1000	1278	1209	0.05

**Table 2. Average time to service a request when the proxy and the web server are on different LANs (ms).**

dwarfs the time needed for policy evaluation. The experiment shows that the relative overhead is negligible for files bigger than 100K (between 0.03 and 0.07), and it is acceptable for smaller files (between 0.15 and 0.19).

We expected, however, a wider gap in performance, between the measurements for LAN and WAN. Although, the relative overhead incurred for WAN is smaller, the results for LAN are far from being prohibitive. We also believed the relative overhead incurred for a file not found in the cache—measured by disabling the cache of the proxy—to

be much smaller than the relative overhead for a file previously cached. However, the difference between a “hit” and a “miss” is less than 10%.

## 6. Related Work

The need for a mechanism for specifying access control policies regulating web access, as an alternative to hard coding, was emphasized by several researchers. The PolicyMaker [5, 4] trust management system presents a generic approach for supporting such policies. Like our mechanism, PolicyMaker uses a formal language for expressing policies; unlike us, a policy is evaluated by a web browser whenever a document is fetched. Consequently, PolicyMaker can be used adequately only for scenarios where the web user has a vested interest in obeying the policies. For example, PolicyMaker cannot guarantee that the traffic control policy (presented in Section 4) is followed by every employee of an enterprise, since a malicious agent may easily bypass the browser.

Baker and Grosse [1] developed Signet, a software system to be used by teachers and parents to control students’ access to resources on the Web. Like in our approach, the filtering is done by the proxy server according to a specified policy; their policies, however, have less expressive power, in the sense that neither the identity of the requester, nor its state can be taken into consideration. Consequently, neither traffic control policy nor synchronized access can be implemented under Signet.

Ferraiolo and all [2, 9] suggested a role based access mechanism where the rights an employee has are determined by its position in the enterprise. The decision to grant or deny access is delegated to the web server, which maintains the mapping between agents and roles, and a database listing the privileges of each role. While a very effective technique for supporting policies regulating access to *internal* documents, it cannot be used for regulating access to *external* documents, since the policy enforcement is done at the web servers, local to a given enterprise.

Finally, we are mentioning here two toolkits, Bob and Akenti, designed to deal with a set of specific issues raised in controlling Intranets. Bob [17] has been developed to protect sensitive information, even when the web servers on which data is stored are not trustworthy. Akenti toolkit [14] has been designed to support policies regulating access to resources that have multiple stakeholders. (In this case access to a resource is granted only if allowed by all stakeholders.) Both approaches deal with a fairly narrow range of scenarios, and incur a large overhead (varying from 200ms to 2260ms in Ahenti’s case) which limits their applicability.

## 7. Conclusion

We have argued that in order for corporate webs to reach their full potential, access control mechanisms that can express regulations and practices governing businesses are needed, and showed that current Web technologies provide only limited support for this purpose. The mechanism presented in this paper remedies some of the current problems. First, by providing a clear separation between policy and mechanism, it creates grounds for easy maintenance. Second, the language proposed here for expressing regulations is expressive enough to support a wide category of enterprise policies. Finally, policies expressed in this formalism are sensitive to the state of agents. This property is critical to the traffic control policy, and to many others.

Moreover the mechanism is easy to deploy, requiring only minor modifications to proxy-servers and no modification to web servers. And as experimental results have shown its usage incurs only a minimal overhead.

## References

- [1] B. Baker and E. Grosse. Local control over filtered www access. In *Fourth International World Wide Web Conference*, December 1995.
- [2] J. Barkley, A. Cincotta, D. Ferraiolo, S. Gavrilu, and R. Kuhn. Role based access for the world wide web. In *National Information Systems Security Conference*, October 1997.
- [3] X. Blanc, M.P. Geravis, and R. Le-Delliou. Using the UML language to express the ODP enterprise concepts. In *Proceedings of the Third International Enterprise Distributed Object Computing (EDOC99) Conference*, pages 50–59. IEEE, September 1999.
- [4] M. Blaze, J. Feigenbaum, P. Resnick, and M. Strauss. Managing trust in an information-labeling system. *European Transactions on Telecommunications*, (8):491–501, 1997. Special issue of selected papers from the 1996 Amalfi Conference on Secure Communication in Networks.
- [5] Y.-h. Chu, J. Feigenbaum, B. LaMacchia, P. Resnick, and Strauss. Referee: Trust management for web applications. In *Proceedings of the 6th International World Wide Web Conference*, 1997.
- [6] W.F. Clocksin and C.S. Mellish. *Programming in Prolog*. Springer-Verlag, 1981.
- [7] World Wide Web Consortium. Jigsaw - the w3c's web server. website:<http://www.w3.org/Jigsaw/>.
- [8] M. Crandall and M. Swenson. Integrating electronic information through a corporate web. In *Fifth International World Wide Web Conference*, 1996.
- [9] D. Ferraiolo, J. Barkley, and R. Kuhn. A role based access control model and reference implementation within a corporate internet. *ACM Transactions on Information and System Security*, 2(1), February 1999.
- [10] R. Khare and A. Rifkin. Trust management issues on the world wide web. *Computer Networks and ISDN Systems*, 30:651–643, 1998.
- [11] P.F. Linington. Options for expressing ODP enterprise communities and their policies by using UML. In *Proceedings of the Third International Enterprise Distributed Object Computing (EDOC99) Conference*. IEEE, September 1999.
- [12] N.H. Minsky and V. Ungureanu. Unified support for heterogeneous security policies in distributed systems. In *7th USENIX Security Symposium*, January 1998.
- [13] M. Sloman. Policy driven management for distributed systems. *Journal of Network and Systems Management*, 1994.
- [14] M. Thomson, W. Johnston, S. Mudumbai, G. Hoo, K. Jackson, and A. Essiari. Certificate-based access control for widely distributed resources. In *Proceedings of 8th USENIX Security Symposium*, August 1999.
- [15] V. Ungureanu. *A Mechanism for Supporting Communication Policies in Distributed Systems*. PhD thesis, Rutgers University, 2000. Obtainable from [ungurean@cs.rutgers.edu](mailto:ungurean@cs.rutgers.edu).
- [16] J. Wang. A survey of web caching schemes for the internet. *ACM Computer Communication Review*, (29):36–46, October 1999.
- [17] T. Wilkinson, D. Hearn, and S. Wisemani. rustworthy access control with untrustworthy web servers. In *Proceedings of 15th Annual Computer Security Applications Conference*, December 1999.