The Authorization Service of Tivoli Policy Director

Günter Karjoth
IBM Research
Zurich Research Laboratory

Abstract

This paper presents the Authorization Service provided by Tivoli Policy Director (PD) and its use by PD family members as well as third-party applications. Policies are defined over an object namespace and stored in a database, which is managed via a management console and accessed through an Authorization API. The object namespace abstracts from heterogeneous systems and thus enables the definition of consistent policies and their centralized management. ACL inheritance and delegated management allow these policies to be managed efficiently. The Authorization API allows applications with their own access control requirements to decouple authorization logic from application logic. By intercepting the traffic over well-defined communication protocols (TCP/IP, HTTP, IIOP, and others), PD familiy members establish a single entry point to enforce enterprise policies that regulate access to corporate data.

1 Introduction

As the Internet has become the primary medium for disseminating information to people all over the world, companies and government agencies are increasingly opening their IT infrastructure to give external customers and partners access to resources, such as product support data, and internal users access to various corporate data. These organizations face problems of how to enforce their enterprise policies that regulate access to corporate data, and how to manage these policies efficiently. A viable approach is to provide centralized access control for corporate information. By intercepting the traffic over well-defined communication protocols (TCP/IP, HTTP, IIOP, and others), a single entry point enforces the domain's authorization policy.

There are a number of commercially available products for this type of Web access control [4, 10]. All these products provide a framework for user authentication, authorization management, and access control enforcement for resources within a secured domain. They centralize authorization rules and provide a finer granularity of access control than most native access controls in operating systems, Web servers, and applications do. The notion of authorization server is not new, and authorization servers have been described for instance in [13, 12, 14]. These research prototypes mostly aim at expressivity increase of the policy language, policy neutrality, and for flexible tool provision to configure the security policies; concerns that are orthogonal to the ones addressed by the commercial products in this market, such as scalability and performance.

This paper focuses on Tivoli Policy Director (PD), which provides a facility for centrally managing policy to govern access to resources over geographically dispersed intranets and extranets. Policy Director provides authorization services to applications. Applications that are part of the Policy Director family include WebSEAL, NetSEAL (for TCP-based applications), Application Server (for CORBA applications), and MQSeries (for MQSeries queues). Third-party applications can use Policy Director's authorization service by calling its standard-based Authorization API [11]. Besides its policy management features, Policy Director also supports (additional) external authorization services.

WebSEAL is an HTTP proxy, installed in front of a Web server or group of Web servers, that controls access to Web resources by performing authorization checks on URL names. For that purpose, WebSEAL authenticates users and then acquires user credentials (e.g., group memberships). Subsequently, WebSEAL checks authorization (i.e., makes an access control decision) to protected URL-addressable resources, including "dynamic URLs" generated by applications, based on the user's credentials. Thus, Policy Director performs as a reverse Web proxy; it appears as a Web server to clients and as a Web browser to the back-end servers it is protecting.

Policy Director provides a wide range of built-in authenticators, supports external authenticators, and offers different qualities of protection and accounting. To provide scalability, Policy Director can off-load its authentication and authorization services to separate servers. For example, front-end replicated WebSEAL servers load balance client requests; back-end replicated Web servers mirror resources in a unified name space for high availability.

This paper presents the centralized Policy Director's authorization service and its utilization. Section 2 outlines the Policy Director architecture. Section 3 introduces the elements of the authorization model and explains the logic of the access decision function. Section 4 depicts the administration scheme, where permissions on different regions of the protected object namespace lead to delegated management. WebSEAL is elaborated on in Section 5, followed by an implementation of the Chinese Wall policy in Section 6. Section 7 draws conclusions.

2 Architecture

In a Policy Director environment, access to a resource is managed by an application server, which is the reference monitor for the resource. When a client attempts to perform an operation on the resource, the reference monitor passes the client's identity together with the name of the resource and the set of permissions required to execute the requested operation to the Authorization server.

Policy Director's Authorization Service provides authorization services to applications that are part of the Policy Director family, WebSEAL for example, or to third-party applications. Third-party applications access these services via the Authorization API, a component of the Policy Director Application Toolkit (ADK), which implements The Open Group Authorization API standard [11].

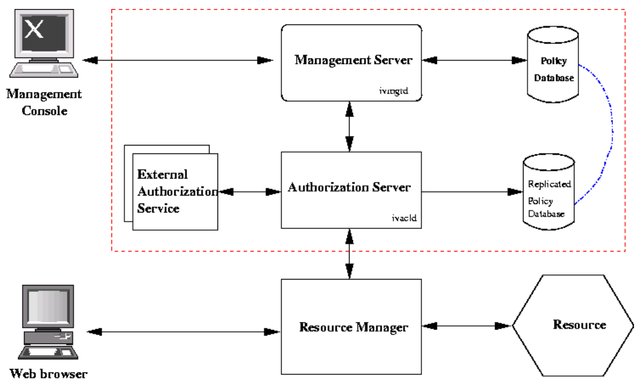


Figure 1. Authorization service.

The basic components of the Authorization Service (dashed box in Figure 1) are the primary (master) Authorization Policy database, the Management server (ivmgrd), the Authorization server (ivacld), and possibly some External Authorization server(s). The Management server maintains the primary Authorization Policy Database, replicates this policy information throughout the secure domain, and updates the database replicas whenever a change is made to the primary database. The Authorization server determines a client's ability to access a protected resource. The Authorization Service also provides a management interface (Management Console or the **pdadmin** utility) that allows the state of the policy database to be changed.

To increase availability and performance, Authorization Service components can be replicated. The database is replicated for each application (local cache mode) or the application uses a shared replica cached by the remote Authorization Server. Maintaining distinct stores of authorization information has also the benefit of making access decisions faster by generating optimized representations [12]. General and static authorization information stored in the master Authorization Policy database is "pushed" to the remote Authorization Server.

The Policy Director Authorization Service relies on external components to provide and maintain the security attributes of principals. User and group membership information, for example, are stored in registries as provided by DCE, LDAP, Domino, etc.

An application's reference monitor intercepts requests of clients to access resources it protects. If necessary, it authenticates the client whose identity is then represented by a set of privilege attributes. To check whether the client possesses the required permission to access the resource, the reference monitor uses the Authorization Service via calls to Authorization API. The Authorization Service compares the client's identity to control attributes associated with the resource. NetSEAL and Application Server, two members of the Policy Director family, intercepts ftp or telnet access or IIOP messages. WebSEAL, another family member, is described in Section 5.

3 Authorization Model

In Policy Director, an authorization database defines the authorization state that determines whether a given request has to be considered authorized. The control attributes are stored in access control lists (ACLs) and protected object policies (POPs). Both entities are named and objects in their own right, called templates. Whereas most ACL systems store the lists at the resource (within the application), Policy Director decouples the authorization information by introducing the notion of *protected objects*, or objects for short, which are the logical representations of resources. Access to a resource is controlled by attaching an ACL and/or POP template to the corresponding object.

In the way the authorization database and thus the authorization state can be changed, Policy Director follows the administration paradigm, where only a restricted group of users (security administrators) can change the authorization state. However, the ownership paradigm is used to control the capabilities of security administrators to change management data, for example ACL templates or user definitions.

The specification of access rights in Policy Director is identity-based – users with similar security properties are collected into groups, and permissions are granted to users and to groups, thus establishing an indirect relationship between users and rights. Only permissions that represent approvals can be given.

3.1 Protected Object Namespace

The *protected object namespace* is a hierarchical portrayal of resources that belong to a secure domain. Its elements are strings whose syntax and structure are similar to absolute URIs [1] but without the scheme, machine, and query components. The slash character ('/') is used to delimit, from left to right, hierarchical substrings of the object's name. Thus, the strings

```
/aaaaaaa
/cgi-bin/test-cgi.exe
/pic/pd.gif
/products.nsf/By+Product+Nbr/$SearchForm
/sales/budget/quarter1/New%20York/travel
/7595ed78b0641e0071ed70/99d37fe852564050
```

are examples of object names.

ACLs and POPs are attached to objects, and objects without attached ACL (POP) inherit the ACL (POP) of the closest ancestor. Authorizations are granted to subjects, which can be single *users* or *groups*. A user may belong to several groups but there is no group hierarchy, i.e. groups cannot be members of other groups. Users are the principals of the secure domain that can be authenticated. Besides users and groups, there are two additional ACL entries: `any-authenticated` matches any authenticated user and `unauthenticated` matches any unauthenticated user.

Access Control Lists. A *permission* is an abstract notion, in fact it is only a name, whose existence is checked for. For ease of administration, a permission might denote a certain type of access (e.g., read, write, execute). However, different objects might employ different access rights for the same permission. Additional permissions can be introduced to denote that an external authorization server should be contacted.

Policy Director's access control is discretionary in the sense that some individual users (administrators) are "owners" of ACL templates and therefore have complete discretion over who should be authorized to perform which action on the object. Ownership is usually acquired as a consequence of creating the ACL template. Granting the control (c) permission gives "ownership" of the ACL template. It allows one to create, delete, and change entries in the ACL, or to delete the ACL template. Subjects with ownership privilege on an ACL template may grant any permission, including ownership, to any other user or group.

Besides the 18 standard permissions, an implementer of an object can define additional 14 permissions for specific access rights. To support large numbers of operations or properties on an object, Policy Director also allows one to define groups of permissions, supporting a total of 32 permission groups (including the primary permission group of the above standard permissions), with up to 32 permissions per group. CORBA Security also uses the notion of permission groups, called rights families, to extend the number of permissions [7]. A different approach is used in Windows 2000, where the object type field of an ACL entry specifies to which portion of an object it refers [9].

Protected Object Policies. Whereas ACL policies are the grounds on which the Authorization service gives a yes/no answer to a given request, *protected object policies* may impose additional conditions on the request that are passed back to the Resource Manager along with the "yes" answer. It is the responsibility of the Resource Manager to enforce the (returned) POP conditions. Conditions imposed by a POP apply to all principals.

A POP is a set of attribute-value pairs. Predefined attributes allow one to express quality of protection and audit levels, to restrict access to a specific time period or to certain IP endpoints, and to set a warning mode. Administrator-defined attributes can be used to store information for use by external authorization services.

Any object is controlled by an ACL and a POP, which may influence each other. If the POP's warning mode is enabled, the conditions setup by a POP become inactive for every requester, thus providing a way to test ACL and POP policies before they are made active. On the other hand, a granted bypass TOD (B) permission in the controlling ACL overrides the conditions of the time-of-day attribute in the POP.

POPs resemble to some extent the conditions field of an access control rule ("assertion") in KeyNote [3], which consists of several relational expressions and compliance value(s), for example *ApproveAndLog*. POPs also bear some similarity to the concept of provisions [6] that provide *conditional authorizations* – binary decisions can be extended to "allow access provided some actions are taken." For example, POP templates allow one to specify levels of authentication, protection and auditing. If necessary WebSEAL enforces the required condition by creating a secure session to the requester or by initiating a step-up authentication procedure. This implies that access rights are reduced if a resource is requested over an unencrypted or less strongly authenticated channel.

3.2 Access Right Propagation

Whereas traditional operating systems create ACLs by copying entries from the container of an object when it is created, ACLs are explicitly attached to objects in Policy Director. However, by *ACL inheritance*, any object without an attached ACL inherits the nearest ACL attached to an object above it in the hierarchy. This leads to the concept of *regions*, sets of objects that share the same protection properties. In a region, there is one and only one object with an ACL attached to it (the root of the region). The other objects within the region are all the objects below the root that have no explicitly set ACL and are reachable without passing through an object with an explicitly set ACL.

Attached ACL templates form regions in a given protected object namespace. In the protected object namespace below for example, there are four ACL templates A, B, C, and D attached as follows:

A	/
B	/c1/c2/
C	/c1/c2/c3/c4/
D	/c1/c2/c3/c4/c5/f2

They establish four nested regions: object `/c1/` will be in region A; object `/c1/c2/f` is in region B; and object `/c1/c2/c3/c4/f` is in region C.

ACL inheritance allows permissions set on an object to propagate to every object located underneath it until another ACL occurs. This also includes every new object created within the scope of the ACL. Thus regions are open; i.e. objects added later to the tree will be within the scope of the corresponding region. For example, object `/c1/c2/f1` will be within the scope of region B when added.

Objects in a region are accessible if the requester (explicitly or implicitly) holds the traverse (T) permission on each object on the path of the region. Note that POPs are inherited in the same way as ACLs, except that inheritance cannot be blocked by a missing Traverse permission on a higher POP.

3.3 Fine-grained Access Control

Policy Director provides mechanisms to provide fine-grained access control for legacy data, whose HTML representation is generated dynamically by a gateway program. Queryable resources are dynamic document bases that, when addressed by a URL request supplemented with parameters (query string), will return a dynamic content displayable by a Web browser. Dynamic URL mappings allow access control on URL requests with parameters.

Dynamic URLs. Whereas ordinary URLs are links to static documents on the Web, a dynamic URL is a URL with name-value pairs in the query component. Depending on the values provided, a request command might respond with different content encoded in HTML. For example, if the resource is the Java servlet `Savings`, which needs a value for user and a value for property, then the URL `http://server/Savings?user=alice&property=job+title` addresses this resource.

```
/db/redshirt   /db.cgi*product=shirt*color=red*
/app/snoop     /rt[25]/servlet/snoop
/app/snoopA    /rt?/servlet/snoop
/app/cnt/ejb   /examples/HitCount\?src=EJB
/app/cnt       /examples/HitCount*
```

Figure 2. Example of URL mapping.

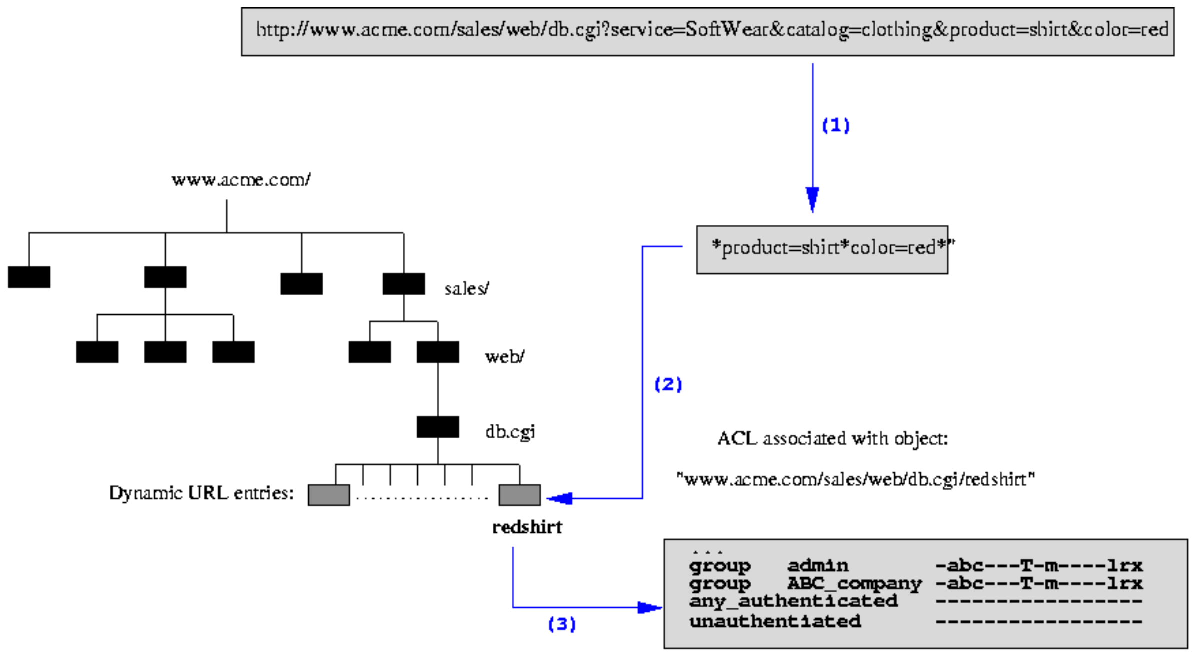
To apply access control to specific values of a dynamic URL, Policy Director offers the possibility to map sets of object names to single object names. When a dynamic URL has been resolved to a namespace object, PD uses the object's ACL for the subsequent authorization check. Otherwise, PD uses the URL itself.

Mappings are defined by entries in a configuration file and become activated by command execution. In Figure 2, a sample dynamic URL mapping is given. The objects on the left represent some Web applications, whose corresponding dynamic URLs are denoted by wildcard patterns, a subset of shell-style pattern matching. Note that the mapping is dependent on the order of the entries in the configuration file.

Performing access control on a dynamic URL is shown in Figure 3, where a dynamic URL mapping maps objects from the set `/sales/web/db.cgi*product=shirt*color=red*` on object `/sales/web/db.cgi/redshirt` (2). Thus, the request `http://www.acme.com/sales/web/db.cgi?service=SoftWear&catalog=clothing&product=shirt&color=red` is mapped to object `/sales/web/db.cgi/redshirt` (1), and the ACL associated with this object (3) will be used for the authorization check.

Although wildcard patterns can be used everywhere in an object name, their use should be restricted to the query component of the object name to avoid clashes with the concept of ACL inheritance. If wildcard patterns occur before the query component, they may possibly introduce deviations from the longest matching prefix rule, whose consequences are discussed at the end of Section 3.4.

External Authorization Service. The expressivity of Policy Director can also be extended by the use of External



http://www.acme.com/sales/web/db.cgi?service=SoftWear&catalog=clothing&product=shirt&color=red
(1)
www.acme.com/
*product=shirt*color=red*"
sales/
web/
(2)
db.cgi
ACL associated with object:
"www.acme.com/sales/web/db.cgi/redshirt"
Dynamic URL entries:
redshirt
(3)

```
. . .
group     admin         -abc---T-m----lrx
group     ABC_company   -abc---T-m----lrx
any_authenticated       -----------------
unauthentiated          -----------------
```

Figure 3. Authorization on a dynamic URL.

Authorization services, which are freely programmable, and included into the access decision evaluation process. Each External Authorization service (EAS), a separate (external) authorization server program, is represented by a new permission. When encountering such a permission during an authorization check, the corresponding external authorization service is referred to for additional authorization decisions. For example, a customized EAS might map unregistered users, which provide an appropriate attribute certificate, to a pseudo user or group owning the corresponding permissions. The EAS architecture allows the full integration of an organization's existing security service by integrating legacy servers into the authorization decision-making process.

Policy Director's external authorization services resemble the *PolicyEvaluator* objects in CORBA's resource access decision service [2]. By encapsulating the representation and evaluation of arbitrary authorization policies, policy evaluation objects can be dynamically added to and removed from the authorization service.

3.4 Access Decision Function

Whereas ACLs are the units of granting or revoking, access decisions are made based on individual permissions. Policy Director employs a two-step procedure, checking whether the requester holds the necessary permissions on the region of the object *and* whether the region is accessible for the requester. Both authorization checks use an evaluation scheme on ACLs, as found in Posix or DCE [8].

For a given client, determined by its user identifier and a possibly empty set of group identifiers, the algorithm determines the set of permissions granted by a specific ACL, performing a sequence of *attempted matches* against ACL entry types. First, it checks whether the user identifier matches one of the ACL's user entries. If so, it returns the associated set of permissions. Otherwise, the algorithm computes the union of all permissions the user holds by matching group entries. If the computed union of permissions does not match the required set of permissions, the algorithm next checks the `any-authenticated` entry, and finally the `unauthenticated` entry. For the last entry, the set of effective permissions is determined by a bitwise "and" operation against the `any-authenticated` entry.

When PD has determined the ACL and POP templates that define the access policy to the requested object, it first checks the IP endpoint authentication method attribute, then the ACL permissions, the time-of-day attribute of the POP, and finally determines the audit level. To check ACL permissions, the decision function first checks whether the required permissions are granted to the requester. Next, it sends an authorization request to each external authorization service whose permission appears in the ACL. If a denial occurs, Policy Director will deny the authorization request. If an external authorization service is not available, access will also be denied.

Default rules ensure the consistency of policies defined in Policy Director. For each possible request, a *unique* access decision exists because there are no negative authorizations, and the decision will either grant or deny access, because there is always an ACL template attached to the root, and there is at least one matching entry as there is an implicit ACL entry for subject `unauthenticated`.

By removing the Traverse permission, the security administrator can deny access to a subtree of objects. However, as mentioned before, a dynamic URL mapping may introduce a second (access) path to an object in the subtree that bypasses the root of the subtree. To guarantee consistency on the accessibility of objects we postulate the following rule:

> The authorization state of a policy database is consistent if for any two objects o_1 and o_2, with o_1 being a prefix of o_2, the object $\mathcal{D}(o_2)$ determined by the URL mapping is only accessible if o_1 is accessible.

This is the least assumption we can make. To introduce an ordering on the strength of permissions is not feasible because, depending on the purpose, the administrator might either only give access to some parameters of an executable or might simply exclude some parameters.

4 Management

Advanced services can employ complex access models, which must be both economically implemented and securely managed. The large number of users and objects, the varying working relationships among users in such environments, and the frequent changes of access control information pose challenges to the design of the authorization system. In this section, we elaborate on Policy Director's concept of regions and their use for delegating administration tasks to subordinated security administrators, providing leverage for dealing with the problem of scale in security policy management.

4.1 Regions

Manipulation of protected objects are controlled by permissions to create or modify (m), to delete (d), and to list or view (v) an object. To browse the namespace below an object, browse (b) permission is needed.

Policy Director's protected object namespace has different categories of objects, and each category has its own permissions. Policy Director uses the following standard namespace categories:

I Web objects (`/WebSEAL`)

II Network objects (`/NetSEAL`)

III Management objects (`/Management`)

IV User-defined objects (third-party namespace)

In the following, we describe in more detail the four categories of objects and the permissions associated with them.

Web objects. The objects in region `/WebSEAL` represent WebSEAL servers, directories, files, and executables (CGI programs, Java Servlets, JSP). The node below `/WebSEAL` identifies the machine on which the WebSEAL server is running. This node is the root for the local file system of the WebSEAL server. The namespaces of other parts of the local file system of the WebSEAL machine as well as of other Web servers can be appended via a junction[1] to any node in this subtree.

To access resources directly provided by the WebSEAL Web server, the list (l) permission is needed for a directory, the read (r) permission for a host of file, and the execute (x) permission for an executable. Access to objects, which include directories and CGI programs, across junctions is only controlled using the read (r) permission.

Management objects. The objects in region `/Management` represent ACL templates and POPs, WebSEAL servers, customized actions (new permissions), and policy databases.

Permissions on object `/Management/ACL` control ACL templates. The control (c) permission gives "ownership" of the ACL, i.e. it allows one to create, delete, and change entries in the ACL, or to delete the ACL template. Subjects with ownership privilege on an ACL template may grant any permission, including ownership, to any other user or group. The attach (a) permission allows one to attach/remove ACL templates to/from objects. On region `/Management/POP`, permission Bypass TOD (B) overrides the time-of-day POP attribute.

Permissions on object `/Management/Server` control the creation/deletion of a server definition, the execution of server administration tasks (such as start, stop, suspend, resume), and the listing of servers or to view a server's properties. Permission view (v) granted on object `/Management/Replica` allows one to read the primary authorization database and permission modify (m) authorizes modifications of the replica database(s).

The capability to create new actions (permissions) and action groups or to delete an existing action/action group is controlled by corresponding permissions on object `/Management/Action`.

[1] A junction is a physical TCP/IP connection between a front-end WebSEAL server and a back-end application server. The back-end server can be another WebSEAL server or a third-party application server. See Section 5 for more details.

The management of user accounts and of groups and group membership are controlled by objects `/Management/Users` and `/Management/Groups`, respectively. Whereas the create (N) permission is needed to create a new user account and optionally to assign that user to a group, or to create a new group and to import group data from the user registry, the modify (m) permission is restricted to the update of user account details and group descriptions. There are also two special permissions: The password (W) permission on `/Management/Users` allows password resets, and the add (A) permissions on `/Management/Groups` allows one to add an existing user to a group. In Section 4.2, we show how Policy Director uses the above regions to support the delegation of certain management activities and can restrict an administrator's ability to set security policy to a subset the object space.

User-defined objects. To extend Policy Director's authorization service to objects belonging to a third-party application, the user-defined object namespace must be described. User-defined object namespace regions can be created by **pdadmin objectspace** commands or alternatively through a special mapping file. The name of the root object for a third-party object namespace and the location of the mapping file are listed in the `[object-spaces]` stanza of the Management server configuration file (`ivmgrd.conf`). This file lists the objects belonging to the third-party object namespace and indicates their hierarchical relationship. The namespace is appended at the root (`/`) of the protected object namespace. Each third-party application is free to select its permissions and to define their meaning.

All Policy Director servers maintain a local copy of the authorization database. Policy Director servers include all Security Managers (`secmgrd`) and Authorization servers (`ivacld`). Initially, all servers have view permission. In particular, group `ivacld-servers` needs the view permission to be able to apply changes to the authorization database.

4.2 Delegating Administration Tasks

Policy Director supports delegated management of objects in subregions of the object namespace. If the object namespace of a large organization is organized into regions representing departments or divisions, then a manager familiar with the issues and needs of that branch can receive sub-management responsibilities.[2]

The chief security administrator can create management accounts and can assign to these accounts appropriate controls for specific regions of the object namespace. For example, an "ACL administrator" can attach ACL templates to objects within its subregion if (s)he holds the permissions attach and browse, and an "ACL policy administrator" can be the only user allowed to create, delete, and modify ACL templates (requires permissions browse, delete, manage, and view on `/Management` or `/Management/ACL`). Server management and authorization action management are other delegated administration tasks.

A "group administrator" can create groups and fill these groups with existing users (add (A) permission) as well as with new users (create (N) permission). The group administrator can perform an operation on a user of his group(s) if that group administrator has the appropriate permission assigned to that group. Note that if an 'outside' user is placed into a group, the administrator of that group now has also gained control over that user, shared with the administrator of `/Management/Users` and possibly with other group administrators.

A helpdesk operator, who has the password (W) permission on a group or a group container, can force a user in that group or groups to change the password at the next login.

5 WebSEAL

WebSEAL controls access to back-end servers, where the use of firewalls and filtering router technology establishes a buffer network (DMZ) between the private network and the Internet. A WebSEAL server connects to back-end servers via junctions. Each junction links a back-end server to a particular branch in the namespace. A client can request resources from a particular back-end server by prefixing the path of the URL with the configured junction name, thus providing a uniform URL namespace. Figure 4 shows a typical WebSEAL configuration, where two WebSEAL servers load-balance across a number of back-end servers. Depending on the security requirements, a WebSEAL junction can be configured with different protection: A junction over a TCP connection does not provide secure communication whereas SSL junctions additionally encrypt all communication. An SSL junction can be unauthenticated, (back-end) server authenticated, or mutually authenticated.

As Policy Director performs access decisions based on the names of protected objects, it is important that WebSEAL and the back-end servers treat URLs the same way. By default, WebSEAL expects servers to be case-sensitive. To avoid back doors that bypass WebSEAL, it can be configured to treat URLs as case-insensitive, to remove trailing dots from file names, and to disallow the 8.3 file name format.

When a client accesses a junctioned back-end server, the returned data can be plain HTML or a client application, for

[2]Permissions could also be transfered to all members of a group to establish self-managing teams if wanted.

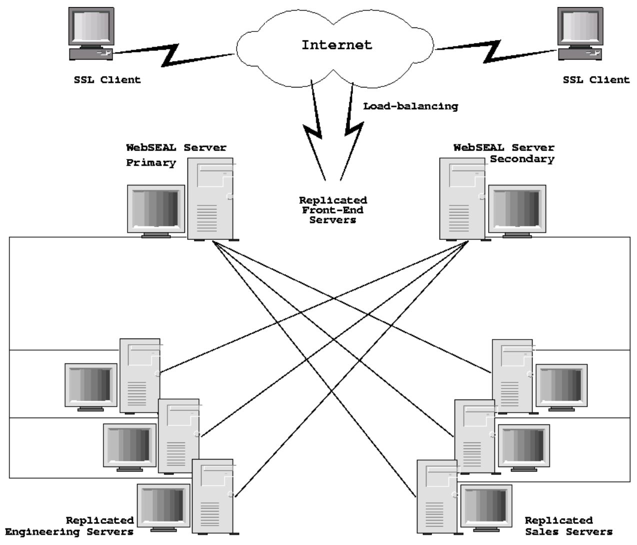


Figure 4. WebSEAL configuration.

example a Java applet or a JavaScript. Any page generated by this data is likely to contain links to other resources on that back-end server or elsewhere. WebSEAL filters data of type "text/html" that it receives from junctioned servers; server-relative and absolute URLs are modified to reflect the junction point of the junctioned server. To avoid the complexity of filtering scripts, WebSEAL can also be configured to provide junction information when a failed request contains an absolute or server-relative URL.

WebSEAL recognizes four authentication methods for use in the step-up authentication mechanism: unauthenticated, password, token-card, certificate. Any order of the methods can be defined, as long as unauthenticated is the last element.

The three strikes login policy enables the specification of a maximum number of failed login attempts *n* and a penalty lockout time *x*, such that after *n* failed login attempts a user is locked out for *x* seconds (or the account is disabled). Thus, for example, some users might be governed by a stronger (weaker) minimum password length policy.

5.1 Access Enforcement

When WebSEAL intercepts a request to access a resource, it (1) maps the user operation (request) to the set of required permissions to grant access; (2) maps the name of the resource to a protected object name; and (3) obtains information about the initiator (a credential). Provided with above information, the Authorization service checks whether access could be granted according to the information stored in the authorization database. If access is denied, WebSEAL returns a "Forbidden" page, no matter whether the object maps to an existing resource or not. If access is granted, then the requester, however, may observe two different behaviors: There will be a "Not Found" page if the object does not map to a resource that can be retrieved by the Web server. Otherwise, the content of the resource will be returned.

The visible part of the object namespace, as seen at the PD Management Console for example, is determined by

- the filesystem of the WebSEAL server,
- the local filesystem on the WebSEAL server machine (if mounted into the PD namespace),
- the "filesystem" of the back-end Web servers (as returned by `query-contents`, a customizable CGI script),
- third-party objects introduced by `[object-spaces]` stanzas (in file `ivmgrd.conf`),
- dynamic URLs (as defined in the configuration file `www/lib/dynurl.conf`).

It is critical to understand that the URL of the request is mapped on two (related) namespaces, the set of resources and the above set of object names. An access policy is only consistent if the identified object is indeed the proxy for the requested resource.

Let $\Sigma : L \rightarrow R$ denote the mapping from object names to resources. This mapping is defined by the way Web servers interpret a given URL to identify an abstract or physical resource. For example, the IBM HTTP Server also executes the CGI program `test-cgi.exe` even if the HTTP request only provides its name `test-cgi` without extension `.exe`. This means that both object names map to the same resource, `C:/Program Files/IBM HTTP Server/cgi-bin/test-cgi.exe` for example, and that $\Sigma($`/cgi-bin/test-cgi`$) =$ $\Sigma($`/cgi-bin/test-cgi.exe`$)$.

Assume that an ACL template, which denies access for requester Alice, is attached to object `test-cgi.exe` but Alice may access any other object otherwise. WebSEAL shows different behavior depending how the program is addressed and where the program is located. If fully named (`test-cgi.exe`), WebSEAL denies access no matter which HTTP server hosts the program. However, if the requester only partly names the executable, WebSEAL grants access. Depending where the resource is located, the result would be either the execution of the program or the page "Not Found".

There are also situations in which POP-defined policies might not be correctly implemented. For example, a resource controlled by a POP time constraint might also be accessible in the time-out period when it is stored in the cache of the Web browser (or the WebSEAL server). The IP address used by WebSEAL to enforce the network-based authentication policy should be the IP address of the originator of the TCP connection. However, WebSEAL might not be able to definitely identify the true client IP address in the case of HTTP proxies or IP address attacks.

5.2 Personalization

To support security-aware applications, WebSEAL can be instructed to insert PD-specific client identity and group membership information into the headers of the HTTP requests. This enables applications on junctioned servers to perform user-specific actions based on the client's PD identity. This information is accessible as global variables for CGI scripts or as header names for Java servlets. Additionally, encoded as a PD credential, it can be used by the application to call the Authorization server via the Authorization API.

A personalization junction instructs WebSEAL to provide to a personalization service, for example implemented by a servlet, the list of applications that the requesting user has access to. This list of applications, in fact a list of protected objects, is placed in the HTTP header and passed across to the junction to be processed by the personalization service. After processing the list, the personalization service can return information such as a customized menu for this user. When WebSEAL generates this list of protected objects it actually checks for specific ACL permissions. The list of ACL permissions and the region of the protected object namespace to be searched is configurable. A portal stanza in file `iv.conf` such as

/portal/cgi-bin/script = /objectspace/obj:r

maps a server-relative back-end Web object to a protected object and a permission.

A personalization junction uses the Authorization API to instruct the Authorization Service to obtain an entitlement that consists of a list of objects within a given region for which the given user credential has the specified access privileges. In general, an entitlement is a data structure that contains externalized policy information, formatted in such a way that it is understandable to a specific application. For example, an entitlement service may use a specific attribute-value pair stored in POPs to provide "expense limit" information of customers.

Policy Director's capability to pass user information to security-aware Web applications, enables them to tailor their responses to the user's access rights, needs, and preferences. This allows application integration (for example by employing user directory information) to serve up personalized content with less development time.

6 Chinese Walls

In this section, we illustrate the power of the Policy Director authorization service by showing how a commercial non-disclosure policy can be implemented by employing an External Authorization Service. In the Chinese Wall policy, resources are grouped into company datasets, and a (symmetric) conflict-of-interest relation denotes whether two companies are in competition. For example, people are not permitted to advice an organization when they have insider knowledge of another competing organization.

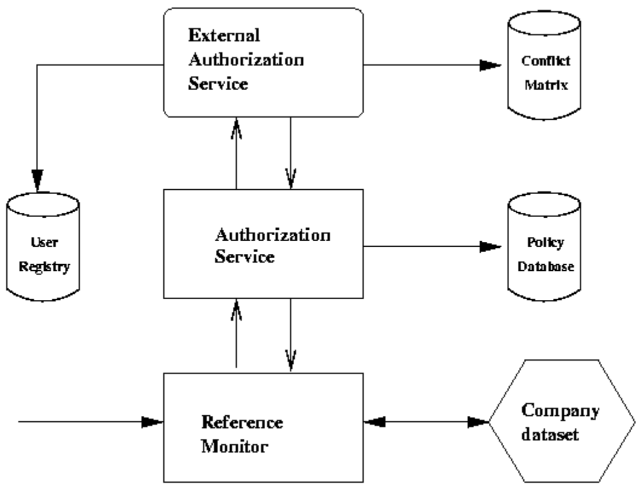


Figure 5. Implementing the Chinese Wall policy.

Like Foley who represents Chinese Walls as user groups [5], we allocate for every organization $a \in Org$ a unique group-id, denoted $gorg(a)$. Each resource containing data from organization a is controlled by an ACL template that gives access rights only to members of group $gorg(a)$. For example, the ACL protecting a resource that contains data from company IBM contains the following entries:

`any-authenticated`	`k`
group `ibm`	`lrx`
user `cell_admin`	`c`

Only group entry `ibm` grants `lrx` permissions. At any moment $mbrs(gorg(a))$ represent the users who may access resources of organization a. Initially, group $gorg(a)$ is empty for every organization a.

However, when a user u who is not member of group $gorg(a)$ requests access to a resource belonging to organization a, Policy Director calls the External Authorization Sercice associated with the `k` permission set in entry `any-authenticated`. The EAS checks whether user

u can be granted access to organization a's resources. If the conflict-of-interest relation indicates no conflict with the current user memberships, the user is added to group $gorg(a)$ and EAS returns a "granted" result. Access decisions for subsequent calls of the same user do not need to invoke the EAS. However, if there is a conflict, EAS returns a "rejected" result. Figure 5 shows the involved components and its interfaces to the conflict-of-interest database and the user registry.

7 Conclusion

In this paper we presented the Authorization Service provided by Tivoli Policy Director and its use by PD family members as well as distributed applications. Policy Director centrally stores its authorization information in the form of ACL and POP templates attached to protected objects that represent resources. POP templates allow one to express time and state constraints and thus to increase the expressivity of a traditional ACL model. The dynamic URL mapping facility extends access control to method parameters.

Object names form a hierarchical namespace and Policy Director uses this hierarchy to implement a sparse ACL model using inheritance. This abstraction facilitates maintenance, and makes answering questions about a user's access rights very simple. However, beside the question of whether it is possible to abstract all resources into object names, one also has to be aware that Policy Director protects the namespace but not the physical resources. Therefore, setting up a correct policy requires a good understanding of the behavior of the controlled Web servers.

The protected object namespace, which separates access control information from their storage at the resources, is a flexible concept and makes Policy Director the foundation for other access control systems. For example, Policy Director can be used as an authorization engine from a pure Java 3 environment, and is used by the Tivoli Privacy Manager to support authorization based on dynamic roles for privacy and other applications.

Acknowledgments

The author would like to thank his colleagues Bob Blakley, Vaughan Harper, Heather Hinton, Tony Nadalin, Avery Salmon, Brian Turner, and Michael Tuton for their help in the preparation of this paper. Thanks also to the anonymous reviewers for their informative feedback.

References

[1] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform resource identifiers (URI): Generic syntax. RFC 2396, Aug. 1998.

[2] K. Beznosov, Y. Deng, B. Blakley, C. Burt, and J. Barkley. A resource access decision service for CORBA-based distributed systems. In *15th Annual Computer Security Applications Conference (ACSAC'99)*, pg. 310–319, 1999.

[3] M. Blaze, J. Feigenbaum, and A.D. Keromytis. KeyNote: Trust management for public-key infrastructures. In *Security Protocols—6th Int'l Workshop*, Lecture Notes in Computer Science 1550, pg. 59–66. Springer, 1999.

[4] S. Blount. Securing Your Insecurities on the Web. In *15th Annual Computer Security Applications Conference (ACSAC'99)*, 1999. `www.acsac.org/1999/papers/thu-c-1300-netegrity.pdf`

[5] S. Foley. Implementing Chinese Walls in Unix. *Computers and Security Journal*, 16(6):551–563, 1997.

[6] S. Jajodia, M. Kudo, and V.S. Subrahmanian. Provisional authorization. In *Recent Advances in Secure and Private E-Commerce*, Kluwer Academic Publishers, 2001.

[7] G. Karjoth. Authorization in CORBA security. *Journal of Computer Security*, 8(2/3):89–108, 2000.

[8] J. Pato. *DCE Access Control Lists (ACL's)*. OSF DCE Specifications, 1990.

[9] M. Swift, C. Van Dyke, P. Brundrett, P. Garg, A. Hopkins, M. Goertzel, S. Chan, and G. Jensensworth. Improving the granularity of access control in Windows NT. In *6th ACM Symposium on Access Control Models and Technologies (SACMAT 2001)*, pg. 87–96, 2001.

[10] J. Snyder. The New Holy Grail? *Information Security Magazine*, Oct. 2000.

[11] The Open Group. Authorization (AZN) API. Open Group Technical Standard C908, Jan. 2000.

[12] V. Varadharajan, C. Crall, and J. Pato. Authorization in Enterprise-wide Distributed System – A Practical Design and Implementation. In *14th Annual Computer Security Applications Conference (ACSAC'98)*, 1998.

[13] T.Y.C. Woo and S.S. Lam. A Framework for Distributed Authorization. In *1st ACM Conference on Computer and Communications Security*, pg. 112–118. ACM Press, 1993.

[14] M.E. Zurko, R. Simon and T. Sanfilippo. A user-centered modular authorization service built on RBAC foundation. *1999 IEEE Symposium on Security and Privacy*, pg. 57–71. IEEE Computer Society, 1999.