Requirements for Scalable Access Control and Security Management Architectures[*]

Angelos D. Keromytis[†] Jonathan M. Smith[‡]

Abstract

Maximizing local autonomy has led to a scalable Internet. Scalability and the capacity for distributed control have unfortunately not extended well to resource access control policies and mechanisms. Yet management of security is becoming an increasingly challenging problem, in no small part due to scaling up of measures such as number of users, protocols, applications, network elements, topological constraints, and functionality expectations.

In this paper we discuss scalability challenges for traditional access control mechanisms and present a set of fundamental requirements for authorization services in large scale networks. We show why existing mechanisms fail to meet these requirements, and investigate the current design options for a scalable access control architecture.

We argue that the key design options to achieve scalability are the choice of the representation of access control policy, the distribution mechanism for policy and the choice of access rights revocation scheme.

1 Introduction

Technology trends and rapid commercialization have resulted in the rapid deployment of many interconnected, non-research computer networks, particularly those based on Internet technologies [19, 23]. So-called "network effects" apply strongly here, as increasing numbers of online services attract increasing numbers of users (including corporate entities), attracting further online availability of information and services. The resulting communications system has large scale in every dimension, with large numbers of network-attached devices and users, and a variety of protocols and mechanisms[1]. While users desire access to as wide a variety of data and services as possible, some organizations (*e.g.,* financial, military, *etc.*) have networked resources with more restrictive access control policies, and various protection mechanisms in place to enforce these policies. Since the same types of equipment and protocols/applications are used in both "public" and "private" networks (those not directly connected to the Internet), the same, or very similar, security mechanisms are employed.

For example, IP firewalls offer a convenient method for performing access control on packets and connections due to the restrictions they impose on the network topology, as seen in Figure 1. Firewalls do not directly enforce end-to-end security properties; they are systems dedicated to examining network traffic between a protected network and the rest of the world. Thus, a firewall can permit or deny a particular packet (or connection)

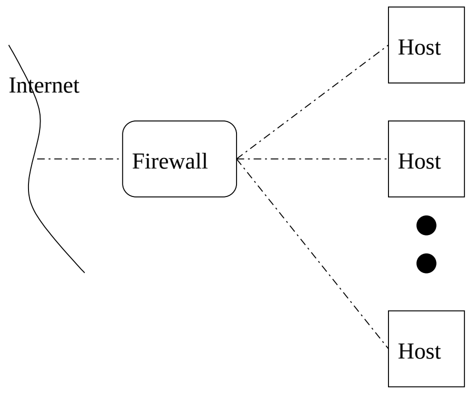


Figure 1: **A firewall's bottleneck topology.**

[*]This work was supported by DARPA under Contract F39502-99-1-0512-MOD P0001.

[†]CS Department, Columbia University, *angelos@cs.columbia.edu*

[‡]CIS Department, University of Pennsylvania, *jms@cis.upenn.edu*

[1]An indication of the number of new services and protocols being deployed can be found in the number of new Request For Comment documents that have been issued the past few years[20]: 1992 - 92 RFCs, 1993 - 173, 1994 - 184, 1995 - 130, 1996 - 171, 1997 - 190, 1998 - 235, 1999 - 260, 2000 - 278. Not all of these documents refer to distinct protocols or services, but they extend or modify existing protocols in some way.

to pass through it based on a policy, but cannot directly protect traffic from eavesdropping or modification once it has passed. Network-layer encryption offers end-to-end secrecy and integrity guarantees, but does not directly address the issue of access control.

Network structure has become sufficiently complex that building blocks such as boundary controllers and encryption are increasingly challenged. Consider, for example, "intranets" and "extranets", where parts of an otherwise protected network are exposed to another entity for the purposes of collaboration, tele-commuting, *etc.* These network structures need access control mechanisms that can operate throughout a network, and enforce a coherent security policy. If we reexamine the use of centralized firewalls, we see several problems:

- The assumption that all insiders are trusted is false.
- It is easy for anyone to establish a new, unauthorized entry point to the network using tunnels or poorly administered access points, such as the increasingly pervasive 802.11 wireless access.
- Some protocols (FTP, RealAudio) require semantic knowledge and demultiplexing which is hard to perform at a firewall, while application-specific gateways are clumsy and introduce new sources of complexity.
- End-to-end encryption can also be a threat to firewall functionality [2], as it inhibits examining packet fields needed for filtering.
- Finally, finer-grained (and even application-specific) access control, which standard firewalls cannot easily accommodate within their processing budget, is increasingly a requirement.

1.1 Middleboxes and endpoints

These problems suggest that access control must become an end-to-end consideration, similar to authentication and confidentiality. This is not surprising, as the IP architecture used the end-to-end argument [22, 8] as the basis for many design decisions. In the present context, we might view the logical end point (for access control) as moving from a centralized firewall to end nodes (*e.g.,* hosts) when a network must support a high degree of decentralized access control.

To manage access control in these networks and deliver the required services, new tools and architectures are needed to cope with the increased scale and complexity of the network entities (devices, users, protocols, security policy enforcement points) and their respective policies for interaction. Since the primary method of addressing scalability issues in networking (and other areas) has been replication, we might attempt a "separation of duty" structure, where different individuals manage different aspects of the network's operations. Unfortunately, current tools either ignore, or do not sufficiently address separation of duty concerns, as we shall see later in Section 2. Even in small networks, administrators have trouble handling the configuration of a small number of firewalls [29]. The results of this can be seen in studies of network intrusions and their causes [13]: an increasing number of vulnerabilities can be directly attributed to misconfiguration, with an even larger percentage of intrusions indirectly caused by administration failures.

1.2 Access Control Scalability

The situation is equivalently bad in simply scaling the policy enforcement mechanisms; most access control mechanisms become a bottleneck as the level of replication increases in an attempt to meet increased demands in network bandwidth, I/O and processing. To better illustrate this, let us consider a simple example.

Imagine a building with N doors. People wishing to enter the building show up at one of the door; all doors are equivalent for the purpose of accessing the building.

In a simple configuration, each door has a guard that examines the person's identification (authentication) and checks the list of people that are allowed to enter the building (access control). If the person is on the list, he is allowed in the building.

To scale for many visitors, we have to increase the number of doors. In the case of the traditional access control (using guards), we have the problem of distributing the list to all the guards and maintaining that list. Furthermore, if the number of potential visitors is large, the list becomes very large and the guards have to spend time and effort looking up people in that list (let alone lifting the book!). Although we have multiple doors, and we can hire many guards, the work of the guards increases rapidly with the number of users, because that work depends on the size of the list.

Now consider a scenario where the guards are replaced with locks on the doors. Each person has a key and that key grants access to the building. Let us assume momentarily that all visitors have the same key (access control policy); in that case, any visitor can enter through any door. The work in performing an access control decision does not depend on the number of doors. Also since each visitor is supplying the key, the complexity of the locks on each door is independent of the total number of visitors or the number of other doors. As the complexity of the mechanism increases (more sophisticated locks, taking more time to operate) the throughput per door may go down, but this can be fixed by adding more doors.

We shall see in a later section how the problem of giving all the visitors the same key can be addressed.

1.3 Organization

The rest of the paper is organized as follows. Section 2 outlines requirements for modern networks and points out where current systems are inadequate to meet these requirements. Section 3 discusses the various options available to the designer of an access control mechanism, with particular emphasis to a credential-based system. Section 4 concludes the paper with a brief summary of what should be "ideal" access control scheme for a large multi-service network infrastructure.

2 New requirements and existing architectures

Access control management systems appropriate for the scale and complexity of today's networks must meet several requirements:

1. The system must be able to support the security policy requirements of many diverse applications, given the large number in use today. (The term "applications" is used to mean services and protocols that require access control configuration. These applications can be security-oriented, *e.g.,* a network layer security protocol, or they may be consumers of security services, *e.g.,* a web server.)

2. The increasing size and complexity of networks strains the ability of administrators to effectively manage their systems. The traditional way of handling scale at the human level has been decentralization of management and delegation of authority. This approach is evident throughout the complete range of human activities (*i.e.,* most, if not all, effective large "systems" involve the creation and maintenance of an administration service where responsibility for different aspects of the system is handled by different entities). Thus, an access control management system for large networks must be able to adapt to different management structures (web of trust, hierarchical management, *etc.*).

3. The system should be agnostic with respect to the configuration front-end that administrators use. The first reason for this is to allow a decoupling of the management mechanism, which could potentially be used for the whole lifetime of the network, from the method used to configure it, which may change as a result of new developments in Human-Computer Interaction interfaces, or because of a change in administrators. Secondly, such a system, by allowing the use of different management front-ends for configuring different applications' access control policies, encourages the development and use of front-ends (GUIs, languages, *etc.*) that are tailored to the specific application and its particular nuances.

We should note that this requirement is not typical for access control management systems; most such systems promote the use of a single configuration front-end for all the applications in the system. Although more research is needed in this area, one can see the parallels between the all-encompassing languages developed in the 1970s and the more recent trend on "domain-specific" languages (languages specifically designed to address a limited application domain, *e.g.,* active networks).

4. The system must be able to handle large numbers of users, applications, and policy evaluation and enforcement points. As we saw above, corporate (and other) networks are rapidly increasing in size; furthermore, new protocols are being deployed (without necessarily deprecating old ones); finally, these same networks are used in increasingly more complicated ways (intranets, extranets, *etc.*).

5. A corollary of the above is that the system should be able to handle the common operations (such as adding or removing users) efficiently. This is important because, over the lifetime of the system, these overheads will dominate other costs like initial deployment.

6. Last but not least, the system must be efficient. It should not impose significant overheads on existing protocols and mechanisms; it should strive to match the performance curve attained by service replication. Ideally, it should even improve performance by addressing any inefficiencies in existing management systems.

2.1 Systems versus requirements

2.1.1 ACLs and Taos

The work by Lampson [15, 16] established the ground rules for access control policy specification by introduc-

	Multiple Languages	Multiple Applications	Decentralized Management	Scalability	Cheap Updates
OASIS[11]		**x**			**x**
Hinrichs[12]		**x**			
Filtering Postures[9]		**x**			
Firmato[1]					
Molitor[18]		**x**			
Hale *et al.*[10]			**x**		**x**
Bonatti *et al.*[3]	**x**	**x**			
Napoleon[24]		**x**			**x**
SnareWorks[7]					
COPS[4]					**x**
RADIUS[21]					**x**
Bull *et al.*[5]			**x**		**x**
Kerberos[17]		**x**		**x**	**x**

Table 1: **System classification. Bold-face font indicates a system name, otherwise the author name is used.**

ing the access control matrix as a useful generalization for modeling access control. A concept derived from the access control matrix that is used in many security system is the Access Control List (ACL); this is a list of $<$ *Subject, Object, Access Rights* $>$ tuples, that collectively encompass the access control policy of the entire system, in terms of users, and services or data to which access must be controlled. The focus in both that work and in Taos [28] is on authentication. The latter depended on a unified policy specification and enforcement framework, although it identified credentials (in the form of digitally signed statements) as a scalable authorization mechanism.

2.1.2 Policy algebras

In [3] the authors propose an algebra of security policies that allows combination of authorization policies specified in different languages and issued by different authorities. The algebraic primitives presented allow for considerable flexibility in policy combination. As the authors discuss, their algebra can be directly translated to boolean predicates that combine the authorization results of the different policy engines. The main disadvantage of this approach is that it assumes that all policies and (more importantly) all necessary supporting information is available at a single decision point, which is a difficult proposition even within the bounds of an operating system. Our observation here is that in fact the decision made by a policy engine can be cached and reused higher in the stack. Although the authors briefly discuss partial evaluation of composition policies, they do so only in the context of their generation and not on enforcement.

2.1.3 Domain specific languages

The approach taken in Firmato[1] is that of use of a "network grouping" language that is customized for each managed firewall at that firewall. The language used is independent of the firewalls and routers used, but is limited to packet filtering. Furthermore, it does not handle delegation, nor was it designed to cover different, interacting application domains (IPsec, web access, *etc.*). Policy updates are equivalent to policy initializations in that they require a reloading of all the rules on the affected enforcement points. Finally, the entire relevant policy rule-set has to be available at an enforcement point, causing scale problems with respect to the number of users, peer nodes, and policy entries. Other similar work includes [12, 9, 18].

2.1.4 Names and role dependencies

In the OASIS architecture [11], the designers identify the dependencies between different services and the need to coordinate these. They present a role-based system where each principal may be issued with a name by one service on condition that it has already been issued with some specified name of another service. Their system uses event notification to revoke names when the issuing conditions are no longer satisfied, thus revoking access to services that depended on that name. Each service is responsible for performing its own authentication and policy enforcement. However, credentials in that system are limited to verifying membership to a group or role, thus making it necessary to keep policy closely tied to the objects it applies to. Furthermore, OASIS uses delegation in a very limited scope, thus limiting administrative decentralization.

2.1.5 Policy mediation, proxying and delegation

The work described in [10] proposes a ticket-based architecture using mediators to coordinate policy between different information enclaves. Policy relevant to an object is retrieved by a central repository by the controlling mediator. Mediators also map foreign principals to local entities, assign local proxies to act as trusted delegates of foreign principals, and perform other authorization-related duties. Coordination policy must be explicitly defined by the security administrator of a system, and is separate from (although is taken in consideration along with) access policy.

2.1.6 Group-based access control

The Napoleon system [24, 25] defines a layered group-based access control scheme that is in some ways similar to the distributed firewall concept presented in [14], although it is mostly targeted to RMI environments like CORBA. Policies are compiled to access control lists appropriate for each application (in our case, that would be each end host) and pushed out to them at policy creation or update time.

2.1.7 Specializing security with wrappers

SnareWork [7] is a DCE-based system that can provide transparent security services (including access control) to end-applications, through use of wrapper modules that understand the application-specific protocols. Policies are compiled to ACLs and distributed to the various hosts in the secured network. Connections to protected ports are reported to a local security manager which decides whether to drop, allow, or forward them (using DCE RPC) to a remote host, based on the ACLs.

2.1.8 Decentralized enforcement and delegation

[5] describes an open, scalable mechanism for enforcing security. It argues for a shift to a more decentralized policy specification and enforcement paradigm, without discussing the specifics of policy expression. It emphasizes the need for delegation as a mechanism to achieve scale and decentralization, but focuses on design of protocols for accomplishing this rather than the more high-level requirements on policy expression.

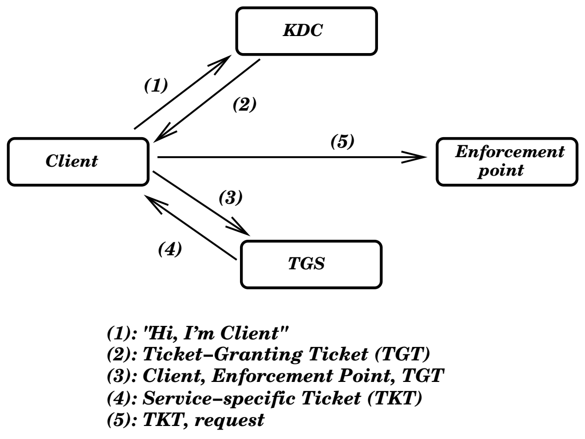

KDC
(1)
(2)
Client
(5)
Enforcement point
(3)
(4)
TGS
(1): "Hi, I'm Client"
(2): Ticket–Granting Ticket (TGT)
(3): Client, Enforcement Point, TGT
(4): Service–specific Ticket (TKT)
(5): TKT, request

Figure 2: **The Kerberos authentication protocol.**

2.1.9 RAP, COPS, RADIUS and DIAMETER

In the IETF, the RAP (RSVP Admission Policy) working group has defined the COPS [4] protocol, as a standard mechanism for moving policy to the devices. This protocol was developed for use in the context of QoS, but is general enough to be used in other application domains.

RADIUS [21] and its proposed successor, DIAMETER [6], are similar in some ways to COPS. They require communication with a policy server, which is supplied with all necessary information and is depended upon to make a policy-based decision. Both protocols are oriented toward providing Accounting, Authentication, and Authorization services for dial-up and roaming users.

2.1.10 Kerberos

Kerberos [17] is an authentication system that uses a central server and a set of secret key protocols, as shown in Figure 2, to authenticate clients and give both a client and an application server a secret key for use in protecting further communications. Initially, the client authentication to the Key Distribution Center (KDC), which gives it a Ticket Granting Ticket; this step occurs infrequently (typically, once every 8 hours). For each service the client needs to contact, it must then contact the Ticket Granting Service (TGS), which responds with a Ticket (TKT) that is service-specific. The client then contacts the service, providing TKT. Often, the KDC and the TGS are co-located.

The two most important deficiencies of Kerberos are that it does not implement any kind of authorization (applications are expected to make their own access control decisions, based on information they acquire through other means, *e.g.,* Directory Services, local ACLs, database queries), and it is expensive, in terms of administrative effort, to do cross-realm authentication, as this requires all clients to have complete knowledge of the trust relationships between realms (a Kerberos realm is the collection of systems and users managed by a single administrative entity). Although there has been some recent work towards addressing these issues [27, 26], there remain significant problems with using Kerberos in a truly large scale environment.

A more important deficiency, however, lies in the nature of the secret key authentication employed by Kerberos: Referrals (used for cross-realm authentication) can solve the problem of securely determining the identity of the principals and KDCs involved in a request, but they cannot be used to convey hierarchical policy information to the enforcement point, beyond any policy included in the ticket issued by the enforcement point's KDC. While access policy could be encoded in the referrals themselves, these would not be verifiable to the enforcement point (since it does not share a secret key with any of the intermediate KDCs). The intermediate KDCs cannot make an access control decision at the time the referral must be issued, since they do not have any information about the application request itself; even if they did, however, this would be an extremely inefficient approach to access control, since all such KDCs would have to be contacted each time a request is made — with no possibility of ticket and referral caching, as is currently possible.

Similar inefficiencies arise when the enforcement point contacts each KDC for every request made by the client. Either of these approaches effectively converts a fairly decentralized authentication mechanism into an extremely centralized access control mechanism. Finally, a referral-based architecture that supports policy dissemination, requires duplication of client information at both the client and the enforcement point's KDC. This is necessary because only the enforcement point's KDC can provide policy information to the enforcement point (encoded inside a ticket), and therefore has to have knowledge of the client's privileges.

2.2 Summary: Access Control System Classification

Table 1 classifies the various systems based on the requirements we enumerated. For the real system requirements we enumerated at the beginning of this section, no single system gets right all of the policy and mechanism interaction challenges. The next section outlines design choices needed for such a system.

3 Designing a Scalable Access Control Architecture

The concerns outlined in Section 2 must guide the design of an access control architecture. Such a system must effectively scale in two different, but related, areas: system and management complexity (and size).

Addressing system complexity requires policy specification, distribution, and enforcement *mechanisms* that can handle large numbers of users, enforcement points, and applications. Furthermore, the system must be able to handle the increased complexity of mechanism interactions. We can critique three obvious models rather easily.

Fully-centralized (Figure 3) approaches demonstrate poor scaling properties. Here, the enforcement points contact the server with the user request details, and expect an answer. Policy evaluation is done at the central repository, for each request. Responses may be cached at the enforcement points, as long as the details of the request do not change, but systems implementing this approach must therefore also address policy consistency issues. Interactions between services and protocols are easy to define, since all the information is centrally available.

Semi-centralized (Figure 4) approaches are those where policy is centrally specified but distributed (synchronously, or "simultaneously") to all enforcement points. Interactions between protocols and services are easy to define, since all the information is centrally available. Changes to the running system require communication with the affected enforcement points. Such approaches require the enforcement points to maintain large amounts of potentially unneeded state, and require communication for common (and thus frequent) security operations such as adding/removing users or modifying their privileges.

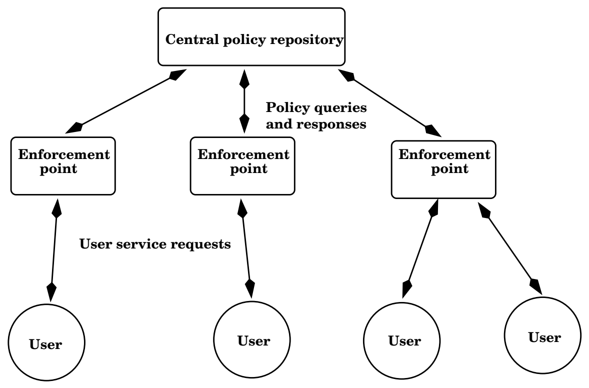


Figure 3: **Centralized policy specification and enforcement.**

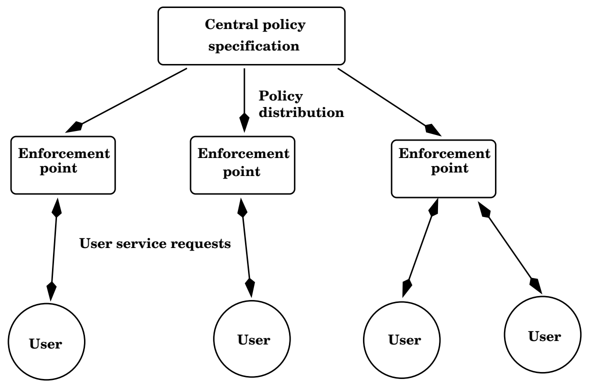


Figure 4: **Central policy specification, decentralized enforcement.**

Fully-decentralized (Figure 5) approaches do not easily allow for interaction between different applications. Policy is specified by different administrators for the different applications, users, and enforcement points. Policy may be distributed directly to the enforcement points, or may be made available to the users in the form of certificates or tickets. Interactions between protocols and services are difficult to express, unless an additional "coordination" layer is added, which re-introduces a measure of centralization to the system; the coordination layer may be explicit (in the form of a meta-policy server), or implicit (in the form of a meta-policy language).

No real system follows any of these three approaches (especially the centralized ones) in their pure form (*e.g.,* caches are employed at enforcement points), but they outline the separation of policy from mechanism in access control architectures.

3.1 Use of flexible credentials

As a first design choice then, a system should exhibit the scaling properties of a decentralized policy specification, distribution, and enforcement system, while retaining the ability to let different applications and protocols interact as needed. Therefore, policy should be expressed in a way that is easy to distribute to enforcement points "on the fly", and which is easy for the enforcement points to verify and process efficiently. One way of expressing low-level policy is in the form of public-key credentials (roughly, public-key certificates with authorization information embedded inside them); an administrator can issue signed statements that contain the privileges of users; enforcement points can verify the validity of these credentials and enforce the policies encoded therein. An additional benefit is that, since credentials are integrity-protected via a digital signature, they need not be protected when transmitted over the network (thus avoiding a potential security bootstrap problem). Thus, it is possible to distribute policies in any of the following three ways:

1. Have the policies "pushed" directly to the enforcement points. While this is the simplest approach, it requires all policy information to be stored locally at an enforcement point, which may present problems for embedded systems or routers. For example, assume a system that any of 100,000 users may access; identifying each user would require knowledge of their public key, for authentication purposes. Assuming a typical RSA key of 128 bytes (1024 bits), simply storing this information requires about 13 MB, excluding any access control information. Typical certificate encodings multiply this by 3 or 4, and access control information will further add to this.

Furthermore, under this scheme, changes in the policy (*e.g.,* adding a new user) require all affected systems to be contacted and their local copy of the policy updated. If such changes are frequent, or the number of affected systems is large, the cost can prove prohibitive.

Finally, the enforcement point will also have to incur a processing cost for examining potentially "useless" policy entries when trying to determine whether a specific user request should be granted. The exact cost depends on the particular scheme used to store and process this information.

2. Have the policies "pulled" by the enforcement points from a policy repository as needed, and then stored locally. This exhibits much better behavior in terms of pro-

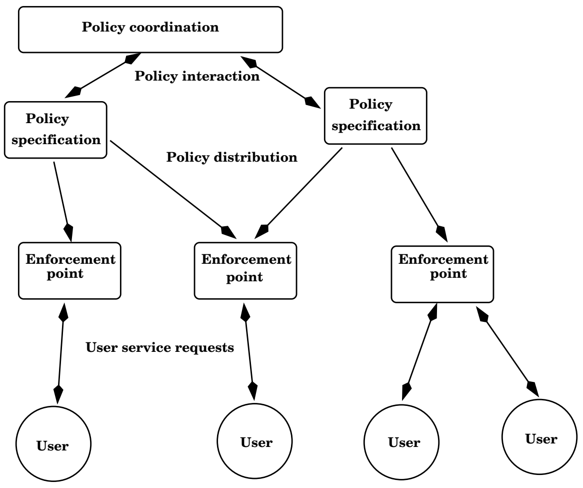


Figure 5: **Decentralized policy specification and enforcement.**

cessing and storage requirements, but requires that the enforcement point perform some additional processing (and incur some communication overhead) when evaluating a security request. System availability can be addressed via replicated repositories; an attacker that compromises one or more of these can deny service to legitimate users, but cannot otherwise affect a policy decision. This approach offers two additional advantages: first, it is relatively easy to deploy since it requires modification of only the enforcement points (as opposed to modifying all the clients and other network elements). Secondly, it effectively addresses privilege revocation (which we discuss later in this section).

3. Have the policies distributed to the client (user) systems, and make these responsible for delivering them to the enforcement points. While this approach requires modification of the client, most security protocols already provide certificate exchange as part of the authentication mechanism; it is often relatively straightforward to modify such protocols to deliver the kind of credentials used in our system instead. Furthermore, since the end systems hold all the credentials that are relevant to them, it is possible to determine in advance under what conditions a request will be granted by an enforcement point (*e.g.,* how strong the encryption should be to be able to see confidential information on the corporate web server).

The three approaches to policy distribution are shown in Figure 6. These approaches are: (1) policy is pushed

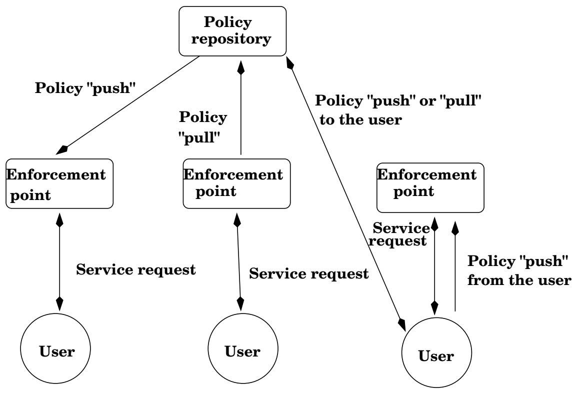


Figure 6: **Policy distribution models.**

to the enforcement points; (2) policy is "pulled" by the enforcement points from a repository; and (3) policy is supplied to the end users which must deliver it to the enforcement points as needed. A combination of (2) and (3) may be used in the system: if the client system provides credentials during the authentication phase, these are used to determine the user's privileges; otherwise, the system may contact a repository to retrieve the relevant information or, if it is overloaded, deny the request and ask that the user provide the missing information in a subsequent request. One advantage of this approach is that policy can be treated as "soft state," and periodically be purged to handle new users and requests (using LRU, or some other replacement mechanism). If the policy is needed again, it will be re-instantiated. This mechanism is conceptually similar to virtual memory page replacement algorithms used by modern operating systems, and thus many such algorithms can be reused here for purposes of policy state. We call this mechanism "lazy policy instantiation" in our context.

One benefit of choosing to use credentials as a means for distributing policy is the fact that one of the frequently-done operations (adding a user, or giving additional privileges to an existing user) is cheap: we simply have to issue the necessary credentials for the user in question, and make them available in the repository. Under any of the distribution schemes already described, the new policy will take effect as soon as the next request that requires is appears.

On the other hand, one other frequent operation (removing a user, or revoking some existing user's privileges) is more complicated in an environment where policy is not centrally stored and maintained. We defer discussion of this issue until Section 3.4.

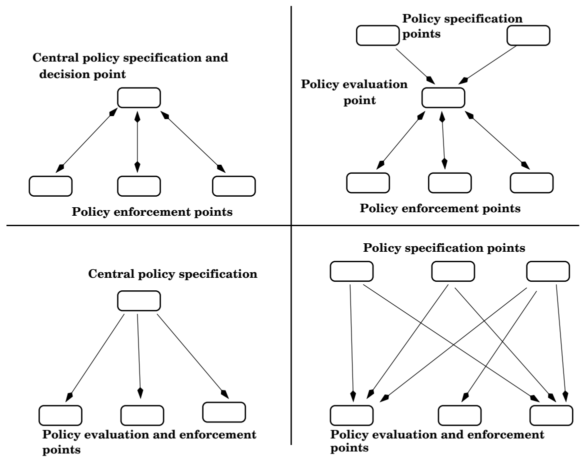

Central policy specification and decision point
Policy enforcement points
Policy specification points
Policy evaluation point
Policy enforcement points
Central policy specification
Policy evaluation and enforcement points
Policy specification points
Policy evaluation and enforcement points

Figure 7: **Different combinations of policy specification and decision making with respect to (de)centralization.**

3.2 Ease of administration

The second scale-related problem area our system must address is administrative complexity; the increased system scale stretches the ability of human administrators to handle its complexity. One well-known and widely used solution is that of "separation of duty": different administrators are made responsible for managing different aspects of the larger system. In computer networks, this separation can be implemented across network boundaries (*e.g.,* LAN or WAN administrators) or across application boundaries (*e.g.,* different administrators for the firewalls, the web servers, the print servers, *etc.*). Multiple layers of management may be used, to handle increasing scale. Thus, our system must support this management approach. One commonly-used mechanism that implements hierarchical management in decentralized systems is delegation of authority.

Note that the degree of (de)centralization of policy specification and enforcement are independent of each other: decentralized policy specification may be built on top of a centralized enforcement system, by providing a suitable interface to the different administrators; similarly, a centralized policy specification system can easily be built on top of decentralized enforcement architecture, as shown in Figure 7. Although the actual enforcement is done at the different network elements (marked as "enforcement points"), enforcement typically refers to the decision making (policy evaluation).

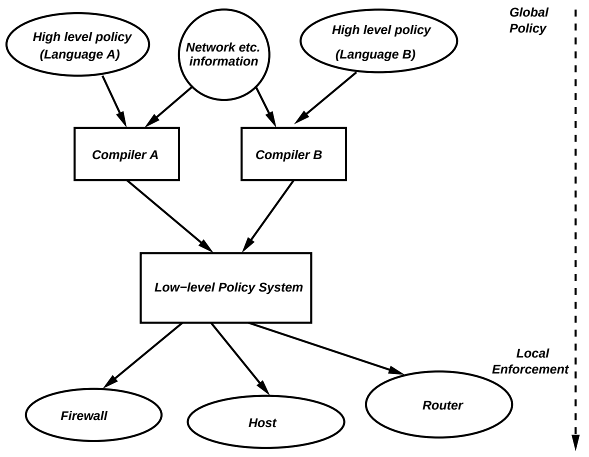


Figure 8: **A multi-layer access control architecture.**

3.3 Layering considerations

These considerations argue for a multi-layer design, such as shown in Figure 8. Administrators can use any number of different interfaces in specifying access control policy. Thus, administrators can pick an interface they are already familiar with or one that is not very different from what they have been using. Furthermore, it is possible to construct application-specific interfaces, that capture the particular nuances of the application they control. This architecture has an intentional resemblance to the IP "hourglass", and resolves heterogeneity in similar ways, *e.g.,* the mapping of the interoperability layer onto a particular enforcement device, or the servicing of multiple applications with a policy *lingua franca*.

Is is important to realize that the design in Figure 8 refers to the logical flow of policy; the system itself follows the decentralized policy specification and enforcement approach. High-level policy is specified separately by each administrator. This interface takes as input the stated policy and information from a network/user database, and produces policy statements in the common language of the low-level policy system. Thus, the low-level policy system (the policy interoperability layer, as it were) must be powerful and flexible enough to handle different applications. These low-level policy statements are then distributed *on-demand* to the enforcement points, where policy evaluation and enforcement is performed locally.

To accommodate management delegation, one of two approaches may be taken: delegation may be implemented as part of the low-level policy mechanism, or as a function of the high-level policy specification system,

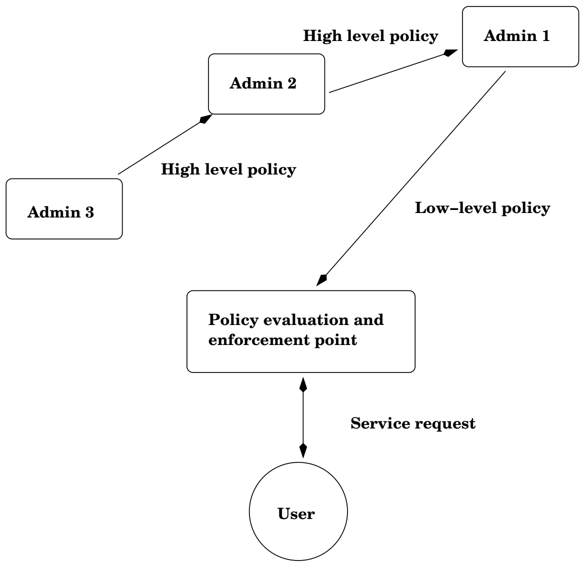


Figure 9: **Delegation as a function of high-level policy specification.**

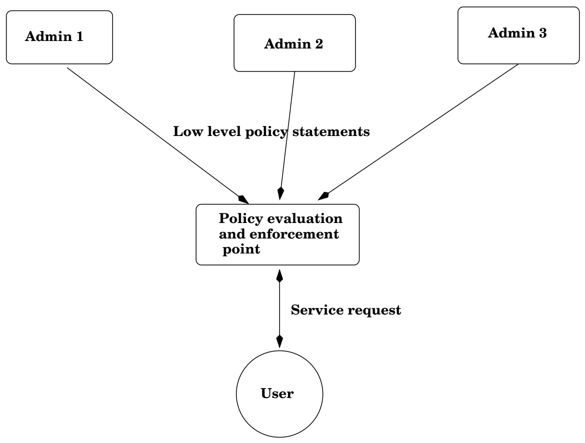
as shown in Figures 9 and 10. We differentiate between high and low levels in the following way. High-level policy statements by different administrators at level N of the management hierarchy are imported and combined at level *N-1*, recursively. The top-level administrator produces the final low-level policy statement, as a result of the composition of all the policies. In contrast, low-level policy statements from all (relevant) administrators are combined at the policy evaluation point.

The high-level approach offers considerable flexibility in expressing delegation and related restrictions, but causes the higher echelons of the administrative hierarchy to become bottlenecks, since they have to be involved in all policy specification. One advantage of following the "low-level" approach is that administration hierarchies can be built "on the fly", simply by delegating to a new administrator.

To summarize, our choice for a low-level policy mechanism is dictated by:

1. Flexibility in the types of applications it can support.
2. Efficiency in evaluating policy.
3. Ability to naturally and efficiently express and handle delegation of authority.
4. Simplicity, as a desirable property of any system[2].

[2]To paraphrase Albert Einstein, "every system should be as simple as possible, but no more."



Admin 1
Admin 2
Admin 3
Low level policy statements
Policy evaluation and enforcement point
Service request
User

Figure 10: **Delegation as part of the low-level policy system.**

3.4 Policy Updates and Revocation

In a credential-based access control system, adding a new user or granting more privileges to an existing user in a credential-based system is simply a matter of issuing a new credential (note that both operations are equivalent in terms of sequence of operations in our system).

The inverse operation, removing a user or revoking issued privilege, means notifying entities that might try to use the relevant credential that it is no longer valid, even though the credential itself has not expired. Potential reasons for the revocation include theft or loss of the administrator key used to sign the credential (in which case, all certificates signed by that key need to be revoked), theft or loss of the user or administrator key authority has been delegated to, or discovery that the information contained in the certificate has become inaccurate.

There are four main mechanisms for certificate revocation:

1. The validity period of the credential itself; if it is set to a sufficiently small value, then the window of revocation is effectively limited to that. On the other hand, a short lifetime means that the a user's credential has to be re-issued much more often, which implies increased work for the administrator (in terms of credential generation and distribution). In the extreme case, where credentials are made valid for a few minutes only, the CA is effectively involved in (almost) every authentication protocol exchange. This approach works well when credentials are used in a transient manner (*e.g.,* to authorize temporary access to a resource). On the other hand, if credential revocation is rare in a given deployed system, the amount of unnecessary work done by the system (reissuing short lived policy statements) can be quite high.

2. Certificate revocation lists (CRLs), and their variants. The idea is that the administrator compiles a list of credentials that must be revoked, and distributes this to the enforcement points (or, as is more typical, the enforcement points periodically retrieve the list from a repository). The CRL is signed by the administrator, and contains a timestamp. An enforcement point can verify that it has received a valid and reasonably recent copy of the CRL by verifying the signature and examining the timestamp. Revoked credentials can be removed from the CRL as soon as their validity period expires. This approach works well when, on the average, only a small number of credentials are revoked. Various approaches, such as Delta-CRLs or Windowed Revocation, attempt to address scalability issues with this approach.

3. Refresher credentials. In this scheme, the owner of a long-lived credential has to periodically retrieve a short-lived credential that must be used in conjunction with the long-lived one. They can do this by simply contacting the issuer of the credential (or some other entity that handles refresher credentials). The advantage of this approach over direct short-lived credentials is that a refresher credential is only issued if the user actually needs one. On the other hand, it requires some communication on the part of the credential owner (as do all revocation schemes, except lifetime-based revocation).

4. Online certificate-status protocols, such as OCSP, have the credential verifier query the credential issuer (or other trusted entity) about the validity of a credential. One drawback is that it is the verifier that must do this status check; if the verifier is a web server or other (potentially overloaded) service, this approach places additional burden on it. On the other hand, this approach does not require even roughly synchronized clocks, as solutions (1), (2), and (3) do. However, since the exchange needs to be secured, the protocol can be fairly expensive.

In cases (2), (3) and (4), the credential issuer (or other trusted entity) must issue statements as to the validity of an issued credential. Since such statements must be verifiable, these approaches require that this issuer's private key is available online (especially for cases (3) and (4)). However, separate keys can be used for issuing and revoking credentials; both keys can be present in the credential. In the event that the machine where the revoking key is stored is compromised, an attacker can extend the lifetime of any issued credential that uses the compromised key for revocation to its maximum validity period; but, the attacker cannot issue new credentials, nor can they affect the revocation of credentials issued after the intrusion has been detected (at which point, a new revocation key is used).

The decision as to which revocation mechanism to use depends on the specifics of the system; in particular, how often are credentials revoked (and for what reason), how stringent the revocation requirements are, what the communication and processing costs and capabilities are, *etc.* For environments where quick revocation is not necessary, time-based expiration may be sufficient; at the other end of the spectrum, a certificate status check protocol may be used to provide near real-time revocation services. (Note however that even Kerberos uses an 8-hour window of revocation, by issuing tickets that are valid for that long, as a tradeoff between efficiency and security.) Luckily, the exact revocation requirements for any particular credential can be encoded in the credential itself; so an administrator's credentials may require an online status check for every use, whereas a user's revocation requirements may be considerably more lax. Furthermore, these requirements can change over time (with each new version of the credential that is issued).

4 Conclusions

The use of credentials has many attractive properties in terms of flexibility. Yet, as in other systems, the schemes for distributing them are important to the overall scalability and correctness of the system. The use of caches creates some challenges for credential revocation, yet these appear to be addressable with a menu of techniques, the choice of which is dependent on particular system requirements for credential expiry.

We have outlined requirements for scalability, and provide a survey of viable approaches to meeting these requirements. Our belief is that from this analysis, one should definitely favor the flexibility of credential-based policy management, while using the lazy evaluation technique. Refresher credentials have the most appeal to us in terms of scalability and consistency with respect to the rest of the system, but may not be "safe" enough for all security applications.

References

[1] Y. Bartal, A. Mayer, K. Nissim, and A. Wool. Firmato: a novel firewall management toolkit. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, pages 17–31, May 1999.

[2] S. M. Bellovin. Distributed Firewalls. *;login: magazine, special issue on security*, November 1999.

[3] P. Bonatti, S. D. C. di Vimercati, and P. Samarati. A Modular Approach to Composing Access Policies. In *Proceedings of Computer and Communications Security (CCS) 2000*, pages 164–173, November 2000.

[4] J. Boyle, R. Cohen, D. Durham, S. Herzog, R. Rajan, and A. Sastry. The COPS (Common Open Policy Service) Protocol. Request for comments (proposed standard), Internet Engineering Task Force, January 2000.

[5] J. Bull, L. Gong, and K. Sollins. Towards Security in an Open Systems Federation. In *Lecture Note in Computer Science 648, ESORICS '92*, pages 3–20. Springer-Verlag, 1992.

[6] P. Calhoun, A. Rubens, H. Akhtar, and E. Guttman. DIAMETER Base Protocol. Internet Draft, Internet Engineering Task Force, Dec. 1999. Work in progress.

[7] J. Chinitz and S. Sonnenberg. A Transparent Security Framework For TCP/IP and Legacy Applications. Technical report, Intellisoft Corp., August 1996.

[8] D. D. Clark. The Design Philosophy of the DARPA Internet Protocols. In *Proc. SIGCOMM 1988*, pages 106–114, 1988.

[9] J. D. Guttman. Filtering Postures: Local Enforcement for Global Policies. In *IEEE Security and Privacy Conference*, pages 120–129, May 1997.

[10] J. Hale, P. Galiasso, M. Papa, and S. Shenoi. Security Policy Coordination for Heterogeneous Information Systems. In *Proc. of the 15th Annual Computer Security Applications Conference (ACSAC)*, December 1999.

[11] R. Hayton, J. Bacon, and K. Moody. Access Control in an Open Distributed Environment. In *IEEE Symposium on Security and Privacy*, May 1998.

[12] S. Hinrichs. Policy-Based Management: Bridging the Gap. In *Proc. of the 15th Annual Computer Security Applications Conference (ACSAC)*, December 1999.

[13] J. D. Howard. *An Analysis Of Security On The Internet 1989 - 1995*. PhD thesis, Carnegie Mellon University, April 1997.

[14] S. Ioannidis, A. Keromytis, S. Bellovin, and J. Smith. Implementing a Distributed Firewall. In *Proceedings of Computer and Communications Security (CCS) 2000*, pages 190–199, November 2000.

[15] B. Lampson. Protection. In *Proc. of the 5th Princeton Symposium on Information Sciences and Systems*, pages 473–443, March 1971.

[16] B. Lampson. Protection. *Operating Systems Review*, 8(1):18–24, January 1974.

[17] S. P. Miller, B. C. Neuman, J. I. Schiller, and J. H. Saltzer. Kerberos Authentication and Authorization System. Technical report, MIT, December 1987.

[18] A. Molitor. An Architecture for Advanced Packet Filtering. In *Proceedings of the 5th USENIX UNIX Security Symposium*, June 1995.

[19] Network Wizards. Internet Domain Survey. http://www.isc.org/ds.

[20] RFC Editor. RFCs issued by year. http://www.rfceditor.org/num_rfc_year.html.

[21] C. Rigney, A. Rubens, W. Simpson, and S. Willens. Remote Authentication Dial In User Service (RADIUS). Request for Comments (Proposed Standard) 2138, Internet Engineering Task Force, Apr. 1997.

[22] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in System Design. *ACM Transactions on Computer Systems*, 2(4):277–288, November 1984.

[23] Telcordia Technologies. Evaluating the size of the Internet. http://www.netsizer.com/.

[24] D. Thomsen, D. O'Brien, and J. Bogle. Role Based Access Control Framework for Network Enterprises. In *Proceedings of the 14th Annual Computer Security Applications Conference*, December 1998.

[25] D. Thomsen, R. O'Brien, and C. Payne. Napoleon Network Application Policy Environment. In *Proceedings of the 4th ACM Workshop on Role-Based Acess Control (RBAC)*, pages 145–152, October 1999.

[26] J. Trostle, I. Kosinovsky, and M. M. Swift. Implementation of Crossrealm Referral Handling in the MIT Kerberos Client. In *Proceedings of the 2001 Network and Distributed System Security Symposium (SNDSS)*, pages 109–124, February 2001.

[27] A. Westerlund and J. Danielsson. Heimdal and Windows 2000 Kerberos — how to get them to play together. In *Proceedings of the 2001 USENIX Annual Technical Conference, Freenix Track*, pages 267–272, June 2001.

[28] E. Wobber, M. Abadi, M. Burrows, and B. Lampson. Authentication in the Taos Operating System. *TOCS*, 12(1):3–32, February 1994.

[29] A. Wool. Architecting the Lumeta Firewall Analyzer. In *Proceedings of the 10th USENIX Security Symposium*, pages 85–97, August 2001.