A Security Model for the Information Mesh

by

Matthew N. Condell

Submitted to the Department of Electrical Engineering and Computer Science

in partial fulfillment of the requirements for the degrees of

Bachelor of Science in Computer Science and Engineering

and

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1996

Author ....
Department of Electrical Engineering and Computer Science
May 9, 1996

Certified by ....
Karen R. Sollins
Research Scientist
Thesis Supervisor

Accepted by ....
F. R. Morgenthaler
Chairman, Department Committee on Graduate Theses

A Security Model for the Information Mesh

by

Matthew N. Condell

Submitted to the
Department of Electrical Engineering and Computer Science

May 9, 1996

In Partial Fulfillment of the Requirements for the degrees of
Bachelor of Science in Computer Science and Engineering
and
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Many distributed systems that are currently being designed are object based. These sytems require a model for authentication and access control which conforms to the object model. They need a model that allows objects to control their own security. In systems where every object in a security domain can not be trusted, there must also be a means for enforcing domain-wide security policies. This must be done without violating the object model.

This paper develops such a security model for the Information Mesh, an infrastructure for facilitating information manipulation. It discusses the goals of the Information Mesh and previous work that has been done towards developing a server-based security model. The paper then proposes a security model that allows objects to control their own security while still providing centralized policy enforcement. Finally, it presents an example of the model in action.

Thesis Supervisor: Karen R. Sollins
Title: Research Scientist

Acknowledgments

I would like to thank my parents, Rand and Freya Condell, for all their love and support they have given me over the years. Without their encouragement I would never have come to MIT. Thanks to my brother, Seth, for his support and proofreading help. Thanks to my girlfriend, Hannah, who has helped me through the highs and lows of my thesis writing, even when she would phase out while I would talk through problems.

I would also like to thank my advisor, Karen Sollins. Her assistance was invaluable throughout this thesis, from helping me decide on a topic to proofreading my drafts. Thanks to all the other members of the Information Mesh project: Lewis Girod, Bien Veléz-Rivera, Tim Chien, Jeff VanDyke, Alan Bawden, and Nancy Cheung, for their help over the years that I have worked on the project.

A special thanks to Garett Wollmann for performing black magic when the disk with my thesis malfunctioned, allowing me to recover all my files.

Finally, I would like to thank all my friends and siblings at Epsilon Theta for making my five years at MIT a great experience.

Contents

List of Figures

Chapter 1

Introduction

Modern authentication systems, such as Kerberos [7], are based on a model of a central policy server that must be queried to obtain permission to access other servers. This model, while effective, does not conform to the object model being used by many emerging distributed infrastructures [1, 5, 6]. This work proposes a security model for one such object-based system, The Information Mesh.

The object model advocates the doctrine that objects manage their own destiny. Following this model, it is necessary that objects control their own security policy, hence they can not be forced to accept tickets from a policy server. Clients must ask each server for permission to use its services. The servers may then choose to query a policy server for advice, but each server makes that choice on its own.

Such a server-based security model means that there is no longer a centrally-defined security policy. A security domain's policy is therefore the union of the policies of all the servers within that domain. If a single server in a domain has a weak security policy, the domain's policy will also be weak. This observation leads to a couple of problems with the server-based security model. First, it is necessary to trust all servers within a security domain to enforce the desired policy. Second, there can be a consistency problem when changing a domain's security policy. This problem exists when the the policy change involves many servers. Since every server needs to be updated individually (unless, of course, they all just defer their decisions to a central server), the potential for different servers to be enforcing inconsistent policies is fairly high. This inconsistency could pose a fairly serious security risk.

Neither of these problems exists in a centralized model, since modifying a security policy only involves updating a single, or small group, of servers. We would like to have a security model that gives us the advantages of centralized control without violating the object model. Objects must still be able to control their own destinies without having to depend upon the decisions of other servers.

This work proposes such a security model for use in the Information Mesh. It starts with a server-based model and expands it to include some centralized control. This is accomplished by allowing a group of servers to behave like a single object. The domain object then may intercept incoming requests and apply its security policy to them. its policy therefore behaves as a centralized policy to the servers encapsulated within the domain. The servers within the domain may still enforce their own security polices.

The model will use a cascaded authentication protocol to allow servers to authenticate requests. The protocol requires servers to provide access certificates for their services. Every server that handles one of these certificates must add its own signature to it. This certificate will allow servers to authenticate all the servers that have handled it.

A demonstration of the model has been built as part of this project. It has been useful both to demonstrate the validity of the model and as a tool to expose the many issues and trade-offs that must be considered when implementing the model. The demonstration implements the security model along with a set of servers that control the reservation of conference rooms around MIT.

The remainder of this document will describe the model, the demonstration of it, and the project for which it has been developed. Chapter two will describe the Information Mesh project and the environment for which the security model is designed. Previous work in the area of server-based security models and cascaded authentication protocols will then be described in chapter three. Chapter four will build upon this work to describe the theoretical security model being proposed. The demonstration of the model is described in chapter five along with many of the trade-offs that must be considered when implementing the model. Finally, chapter six concludes with results of this work and ideas for future work in developing and implementing the model.

Chapter 2

The Information Mesh Project

Before we can develop a security model, we must understand the environment in which it will be used. This is necessary so that the model will provide an appropriate level of security for the environment and that it will conform to the goals and other models of the environment.

The Information Mesh project is developing a long-lived infrastructure to facilitate information access and manipulation. Such a project is necessary for the Internet to adapt to the tremendous growth it has experienced over the last several years, in traffic related to information manipulation. Tools currently available for manipulating information have not been able to keep up with the explosion of information. For example, services that index the World Wide Web have not been able to index the information as fast as it is created, nor have they been able to keep pace with changes in information they have already indexed. The Information Mesh project believes that the solution requires not only better tools to be developed, but it also requires an infrastructure that is designed to better handle the vast amount of information already available and that which will become available.

In order to accomplish this goal, the Information Mesh is developing a substrate that will provide a uniform representation of objects and their relationships and allows for manipulation of this information. At the same time, the Mesh should not force the applications that it supports to all agree on the internal structure of information and how it is manipulated. It should also provide a layer of abstraction between the transport protocols and the applications. The project is working on developing a minimal set of constraints for an infrastructure which can provide the services necessary for information manipulation without limiting the applications that can use the information.

Work towards a long-lived information infrastructure has resulted in a *Mesh kernel*, a *Mesh object system*, and *Mesh links*. The Mesh kernel provides a library of service routines that provide support for object identification and information representation and location. Using the concept of *roles* to describe the nature of *objects*, the Mesh object system provides flexible and evolvable objects for the Information Mesh. *Mesh links* provide a mechanism to express the relationships between Mesh objects.

This chapter describes the goals and requirements necessary for achieving the visions of the Information Mesh. It also describes the Mesh object system and links.

2.1 Goals

It is necessary to understand the goals of the Information Mesh in order to fully understand its design. Its goals can be summarized as: universality, longevity, mobility, evolution, resiliency, homogeneity, and heterogeneity.

- *Universality*

  The Information Mesh should provide a single model of information identification and location for "network-based applications accessing information that is distributed both physically throughout the net and administratively across regions of differing management policies [15]."

- *Longevity*

  The Information Mesh must be able to support long-lived information. Information and references to the information should be able to survive over long periods of time (greater than 100 years). It must be able to support any format the information may take over that time since we can not expect all information to be reformatted to adapt to changing technologies.

- *Mobility*

  Information will move over time to new physical locations as well as new administrative locations. References to the information must remain valid despite these changes.

- *Evolution*

  The Information Mesh must be able to adapt as layers above and below it evolve. This means the Mesh must be able to support changing semantics, syntax, structures, and utilization of information as well as supporting new types of information and relationships. The Mesh must also be able to take advantage of changes in protocols and networks.

- *Resiliency*

  Unreliability is unavoidable in a large network. There are any number of reasons why a piece of information may be unavailable when needed. The Information Mesh should be designed to handle such failures gracefully.

- *Homogeneity*

  In order for distinct applications and information to interact well, it is necessary for the Information Mesh to "provide a single model for information identification, location, and access, as a substrate for distributed systems and applications [15]." This provides a stable abstraction barrier that can allow increased functionality only when it is desired.

- *Heterogeneity*

  The Information Mesh should be flexible enough to provide support in a diverse environment. It should be able to take advantage of current and future network services as well as provide support for a highly varied set of expectations from applications and administrative controls.

These goals may be expressed by two implementation requirements: minimality and flexibility. Since the Information Mesh is designed to support such a diverse and changing set of applications and network support, it must place as few restrictions on its users as possible. In order to accomplish this, it should provide only the minimum necessary to support the goals described above and only require the minimum coordination and agreement necessary to meet the goals. This is important because "we can not depend on any universal agreement on issues like a best way to find information, the internal structure of information or how information is internally manipulated by programs [18]."

The Information Mesh must be highly flexible to adapt to a diverse and ever changing environment. It must be able to support changing information, network infrastructures, expectations from applications, protocols, and whatever else may change in the future. Without such great flexibility, the Information Mesh will not be able to achieve its goals.

2.2 Naming

Naming things is an important way that the Information Mesh achieves its goals. It is also crucial since the security model must be able to identify objects and principals.

Naming generally is used to provide three functions: identification, access, and description. To help support longevity, mobility, and evolution, the Information Mesh has separated these three functions. This is similar to work that's been done in the Uniform Resource Identifiers working group of the IETF [8, 14, 2]. Every object is assigned an *oid* (object identifier) which uniquely identifies the object through both space and time. This is similar to the URI working group's *URN* (Uniform Resource Name). Using the IETF's terminology, Uniform Resource Locators (URL) will provide the location functionality and Uniform Resource Characteristics (URC) will provide the meta-information. The Information Mesh takes advantage of one type of meta-information, called *hints*, to help translate oids to URLs.

2.3 Object Model

The Information Mesh object system [17] provides a typing model for objects that allows for the flexibility and evolution that is necessary to achieve the Mesh's goals. It is also designed to work in an environment where the enforcement of the typing is not guaranteed.

Object behavior is based on the concept of a *role* which all objects must play. A role describes an abstract structure and behavior. If an object behaves in the manner described by a role, then the object *plays* that role. An object may play several different roles and which roles it plays may change over time. To illustrate these concepts, imagine an individual who plays many roles over a lifetime such as a child, teenager, student, parent, friend, etc.

All objects must play the *object-role*. The object-role is the root for an inheritance hierarchy such that an object plays all of a role's super roles in addition to playing a particular role. All objects that play the object-role (and therefore all Mesh objects) are required to be able to answer questions about what roles they play, describe the implementation objects for those roles, and allow the addition of new roles.

The remainder of this section will describe roles in greater detail. It will also describe *implementations* which give objects the ability to play a role by providing a concrete representation of the role.

2.3.1 Roles

Roles are composed of three aspects: *actions*, *parts*, and *makers*. Actions describe the abstract functionality of the role, parts describe the abstract structures of an object that plays the role, and makers define the abstract functions used to create objects that play the role. Each aspect may have some required components and some optional components. All implementations of the role must provide the required aspects, while they may only provide a subset of the optional ones.

As mentioned above, roles inherit actions, parts, and makers from their super roles. This gives roles the ability to provide an extensible typing mechanism by inserting new roles into the hierarchy. This extensibility, along with an object's ability to play multiple and evolving roles, give roles the flexibility and evolvability necessary to meet the goals of the Information Mesh.

Roles are first class objects which play the *role-role*. Objects that play the role-role support the actions, parts, and makers necessary to play a role. Since roles are objects, they must also play the object role and support the required actions of an object.

2.3.2 Implementations

Implementations provide Mesh objects with concrete representations of a role's actions, parts, and makers. This gives the object the ability to play that role. An object may use any implementation of the role it is trying to play.

Implementations have an inheritance mechanism that allows an implementation to use a description of a concrete role capability from a super implementation if it does not have its own description. Implementations are also first class Mesh objects which contain concrete *methods* for actions, parts, and makers.

2.4 Mesh Links

Mesh links [16, 15] give us the means to express relationships between Mesh objects. The discussion here will be brief since they do not relate to the security model.

Mesh links provide two features for expressing relationships. The first is a generic *link-role*. The generic link is an unordered, unnamed set of endpoints which refer to objects. Capabilities such as grouping and distinguishing endpoints, expressing directional links, limiting the number of links, or other functionality can be expressed by adding subroles to the link-role. This makes the linking model highly flexible and extensible. The second feature is the ability to express composition in Mesh objects. By adding an optional *get-required-objects* action to the object-role, an object can indicate other objects that must be included to make the object complete.

Chapter 3

Previous Work

This section describes the two ideas that form the foundation of the Information Mesh security model. The first is a cascaded authentication protocol, developed by Sollins [13], that is used as the basic means of authentication. Cascaded authentication is also used in the server-based model proposed by Bull, Gong, and Sollins [3] which is the second work that forms the base for the Information Mesh's model. This model has been incorporated into the security framework for the ANSA project [1].

3.1 Cascaded Authentication

Cascaded authentication [13] was developed for use in an environment where cooperation must exist despite the absence of complete trust. A request for a service may require the invocation to be cascaded in order to fulfill it. That is, the server may invoke another server which will, in turn, invoke a third server and so on as necessary. The cascaded authentication protocol allows any server in the chain to authenticate all the servers that have previously serviced the request and it can put constraints upon the future progress of the request.

Examples of cascaded requests can be found throughout life. One example is the signing of a legal document by multiple parties. When an agreement is reached between multiple parties, one of the parties' lawyers will draft the agreement. He must sign the document and send it to the lawyers of the other parties involved in the agreement. Once all the lawyers have signed, it may then be sent to the judge, who is presiding over the agreement, for his signature. He will not sign the agreement until all the involved parties have signed. Once he has signed the agreement he must then pass it back to all the parties so they know that it has been signed. There must be a means for all the parties to confirm that the appropriate people have signed the agreement. In this example, the confirmation is accomplished by signatures on a piece of paper. Cascaded authentication gives us a means of providing this sort of confirmation digitally.

3.1.1 Access Certificates

The tool needed to enable the authentication is an `access certificate`[1]. This certificate is signed by every server that handles it. The servers may also add their own constraints to the certificate when they sign it. The certificate allows any server to verify the servers who have handled the certificate and their constraints.

Constraints limit the future progress and the results of a service request. They can limit many different aspects of a request. For example, it could limit the number of servers that may handle the request. They may also limit the amount of money, or other resources, that servicing the request may consume. The principals allowed to handle the request may also be controlled.

When signing a certificate, each server must include several pieces of information in order to verify the certificate's authenticity. First, it must encrypt the ciphertext part of the certificate it received along with the name of the server to which it will be passed next and the constraints that it is adding to the certificate. It then adds its own name and constraints to the cleartext portion of the certificate. The combination of the cleartext and encrypted information allows the authentication server to verify that no one has tampered with the access certificate.

A nonce is included in the initial access certificate to add randomness to the encrypted information. The randomness is necessary to guard against a known cleartext

[1]The access certificate is called a `passport` in Sollins' work [13].

attack. Without the randomness, such an attack is possible since all the encrypted information is also made available in the clear.

Sollins' work assumes a secret key encryption scheme for her protocols, though she notes that it could be easily converted to a public key scheme as demonstrated by Needham and Schroeder [11]. The encryption should be chained[2] so it is possible to add some randomness to the encryption. The trade-offs between secret and public key cryptography will be discussed later in the context of the security model.

Using the following notation, we can illustrate the form of an access certificate:

$\{\}^K$	The material within the brackets is encrypted with key, K.
A, B, D, E	names of principals
I_{A_i}	the i'th nonce unique to principal A
K_A	principle A's secret key, known only to A and the authentication server
C_A	the constraints included by A

In this example, `A` creates a certificate to hand off to `B` who turns around and passes it on to `D`. If `D` were to then hand it off to `E`, `D` would have to sign it in the same way `B` signed it. First, `A` will create an initial access certificate to send to `B` that has the form:

$\{I_{A_i}, B, C_A\}^{K_A}, A, C_A$

If `B` then wants to pass the certificate to D, then it must sign the certificate and add the appropriate cleartext information:

$\{\{I_{A_i}, B, C_A\}^{K_A}, D, C_B\}^{K_B}, A, C_A, B, C_B$

3.1.2 Pairwise Authentication

A pairwise authentication protocol is used to transport the access certificate between two servers. While it is only necessary for a server to authenticate the server from which it receives a certificate, pairwise authentication has several important advantages. Pairwise authentication is needed to detect collusion between two servers.

[2] DES's Cipher Block Chaining (CBC) mode, for example.

Also, the added authentication it provides can reduce the amount of verification that would be needed if the authentication were only one-sided.

Sollins suggests using a pairwise authentication scheme similar to one suggested by Needham and Schroeder [11]. This scheme first requires an exchange with a trusted authentication server[3] to obtain a conversation key which will be used for the pairwise authentication.

The following notation (in addition to the notation above) will be used to illustrate the protocol for getting a conversation key. `A` is requesting a key from the authentication server to enable it to communicate with `B`. The arrows indicate the direction the message moves, while time moves down the page.

T	a time stamp
CK	a conversation key
AS	the authentication server

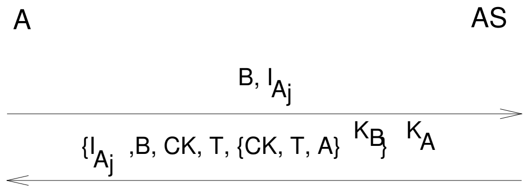

Figure 3-1: Getting a Conversation Key

The first message contains no secret information, so it doesn't need to be encrypted. It contains the name of the server with which `A` would like to communicate and a nonce that is used as a challenge to the authentication server. The authentication server responds by returning this information along with the conversation key, a time stamp that indicates when the key expires, and a ticket encrypted with `B`'s secret key. This information is all encrypted with `A`'s secret key. `A` can be sure

[3]This may actually require more than one authentication server. The servers must at least trust authentications servers that either trust each other or are part of a single chain of authentication servers that trust each other.

that this response is from the authentication server because K_A is only known to the authentication server and A, so only the authentication server could correctly encrypt I_{A_j}. The ticket is used by B to get CK, the timestamp, and the identity of A. It can verify that these came from the authentication server because K_B is only known to the authentication server and B.

The conversation key can be used as many times as necessary until it expires. This eliminates the need to access the authentication server every time a new pairwise authentication is about to start. If the key is compromised, though, it will expire so the integrity of the system will be maintained. Requests to the authentication server may be batched as another way to improve efficiency.

Once A has acquired a ticket and a conversation key, A and B can authenticate each other:

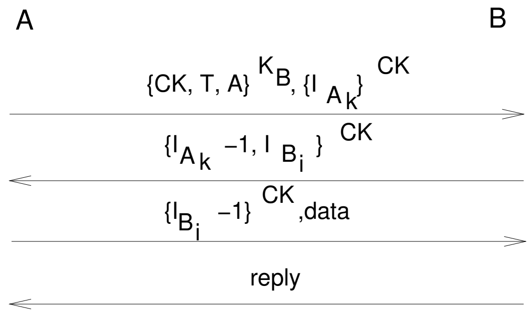

Figure 3-2: Pairwise Authentication

The pairwise authentication starts when A sends the ticket and a nonce encrypted with the conversation key to B. B then can decrypt the ticket, revealing CK and verifying the ticket should have come from A. B also learns when CK expires. B can then use CK to decrypt the nonce. B encrypts and returns a decremented I_{A_k} and its own nonce to A. At this point, A can be sure that B is authentic since only B could have gotten CK from the ticket and used it to decrypt I_{A_k}. A similarly responds to B's challenge by returning a decremented and encrypted version of I_{B_i}. A can also send its data with message three because it is convinced of B's identity. B then replies in the fourth message. The data and reply may also be encrypted with the conversation key to assure that no one has tampered with them.

3.2 Server-based Security

The Information Mesh's security policy is based on the server-based security model that was developed by Bull, Gong, and Sollins [3]. This model proposed that servers control their own security policies. It uses a modified version of the cascaded authentication protocols discussed above. This section first discusses the motivation for a server-based security model, then describes the relevent features of the model.

3.2.1 Motivation

One of the major contributions of Bull, Gong, and Sollins [3] is that they describe the advantages of a server-based security model over an infrastructure-based model. These arguments are important to understanding why a security model for the Information Mesh should be built on a server-based model.

One of the major arguments for a server-based model is that it is a natural model to use in an object-based system. A fundamental characteristic of objects is "*you don't manage objects, objects manage themselves*" [3]. So naturally, objects should control their own security policies otherwise there would be a foreign influence over the management of the objects.

Reducing the granularity of the security policy (from security domains to servers) by giving servers control of their own policies has several positive effects. It allows for immediate revocation of access privileges since access is not granted until the service is requested[4]. Since the servers control their own policies, they are modular so they can migrate from system to system or be included in new systems without updating a central policy server. It also means that server-specific parts of a server's security policy can be built into its design.

A final advantage is the wide range of security policies that it is possible to im-

[4]This can be contrasted to an infrastructure-based model, like Kerberos [7], where access is granted at the time the Kerberos server issues a ticket, not when the service is requested.

plement when the granularity of policy enforcement is the server. Each server can enforce a wide range of policies from being open to everyone to being limited to just a few trusted servers. They also have the ability to enforce their policies on every client or server that has handled the request, not just the client that requested the service. Such flexibility on a server-by-server basis can lead to a very complex set of policies which can both be useful and problematic. The complexity may be difficult to manage on a system-wide basis, since the sum of all the servers' policies characterizes the system's security policy. To solve this, it may be useful to have the servers' default policies set to request help from a central policy server. This still fits the model because it is the server that is choosing to ask the policy server for assistance and not the infrastructure imposing the policy on the server.

A human user would fit into an object-based model quite well. The user would be represented as an object the same as a server. The user would then be in control of her own security policy as she would expect.

3.2.2 Realization

The paper also discusses several design issues for a server-based security model. In order to avoid getting bogged down in unnecessary details, this section only discusses the issues that will be relevent for the design of the Information Mesh's security model.

There are several features of the server-based model that will be used:

- The model assumes that the infrastructure enforces strong encapsulation of objects so that the only access to an object is through its advertised interface. This prevents any access to an object through a "backdoor."

- Each server in this model is an object which has control over its own security policy, even if its policy is to defer the decision to a policy server. The decision about whether or not a service will be granted occurs at the time the service is requested.

- A server will create an access certificate for its service. This certificate must be presented by a client for the service to be granted. These initial certificates may be signed by a key known *only* to the server because it will only need to be decrypted once it returns to the server.

- Access certificates will be passed by some method of cascaded authentication. This will be discussed further below.

- A trusted authentication server is needed to aid in the decryption and verification of access certificates.

- A server that advertises and distributes access certificates will likely be a part of a server-based system. While it will not be part of the demonstration presented in this work, it would be a useful server to implement.

Bull, Gong, and Sollins suggest a different means of cascaded authentication than described above. While it has many similarities, it differs in three fundamental ways. First, it uses one-way hash functions to sign the certificates instead of encryption. Second, it doesn't include any mechanism for including constraints. Finally, it does not include a pairwise authentication mechanism. The lack of the constraints and pairwise authentication mechanisms are the main reasons that this method of cascaded authentication was not used in the Information Mesh's security model.

Chapter 4

The Security Model

While a security model is not part of the core services provided by the Information Mesh, it is necessary for an operational system. It is also needed to help identify and design any core services necessary to support a security system. This model provides integrity, authenticity, and access control, but does not specifically provide privacy.

The *Mesh security model* is based upon the server-based model and the cascaded authentication protocols described in the previous chapter. The Mesh security model generalizes the server-based model to add flexibility and allow some centralized control without giving up the advantages of allowing servers to control their own security. This is accomplished by allowing a collection of servers to be encapsulated as an object in addition to each server being an object. This model allows us to impose a centralized policy as well as having each server control its own destiny without violating the object model that was important to the success of the server-based model. We have essentially generalized the server-based model to an *object-based model.*

The Mesh's object-based security model provides the advantages of the server-based model, but corrects some of its weaknesses by using some of the features of a centralized model. A cascaded authentication scheme similar to the one described in Section 3.1 will be used to take advantage of the power of constraints and the additional security provided by pairwise authentication.

The remainder of this chapter will provide a high-level description of the Mesh's object-based security model. An implementation of the model will be discussed in Chapter 5. This chapter will discuss a threat model for the Information Mesh environment, weaknesses of the server-based model, the Mesh's object-based model, and a generic server that describes all servers conforming to the model.

4.1 Requirements for Security

It is important to understand what problems the Mesh security model is attempting to solve before we propose the model. It is necessary to look at the environment and what threats may be attempted. It is also necessary to look at the weaknesses of the server-based model to understand what improvements we are trying to achieve. We must also understand the model's weaknesses and what problems it does not attempt to solve. We will then not be tempted to apply the model to a problem which it can not solve.

We will now look at the our threat model and the weaknesses of the server-based model.

4.1.1 Threat Model

We must understand what kinds of attacks may be made to our system before we can build up guards against them. In order to do this, we must look at what kinds of attacks may be mounted against servers in the Information Mesh environment and determine which ones we will and will not attempt to guard against. Voydock and Kent [19] provide a good overview of possible threats and will be used as the basis of this discussion.

The Information Mesh depends upon an environment where communication occurs over local and wide area networks. These networks consist of hosts interconnected by links[1]. For this discussion, we will assume that end hosts are secure, but the links are not. It is possible for an intruder to tap into the link and read and send data over it.

[1]Note that these are hardware links such as Ethernet, phone lines, and radio and should not be confused with Mesh links discussed earlier.

The intruder may also compromise a gateway or server within the network.

This environment leaves the Mesh open to both passive and active attacks. Passive attacks can only result in releasing information to unauthorized principals, while active attacks may change what information is sent or may prevent the information from making it to its destination. We will now look at the particular attacks that may be attempted in our environment and if the Information Mesh's security model will try to prevent the attacks.

Passive Attacks:

- *Release of Message Contents.* It is necessary to provide a mechanism to maintain the privacy of data that is transmitted by the Information Mesh. This is accomplished by using an encryption scheme that ensures privacy such as the chain encryption scheme that is found in DES CBC-mode. While it is not necessary to encrypt all data that is transmitted, it is important to leave such an option available.

- *Traffic Analysis.* Traffic analysis is a transport-level threat that is not guarded against in our model. The Information Mesh security model is only designed to secure high-level protocols and, therefore, does not provide any mechanisms to guard against any transport-level threats.

Active Attacks:

- *Message Integrity.* Message integrity refers to detecting modifications to transmitted messages. This can be accomplished by encrypting known or redundant information to allow a high probability of detecting any modifications. This will be discussed in Section 4.3.

- *Message Authenticity.* Attacks on message authenticity include sending messages that are created by an intruder trying to use a false identity and replaying messages that have previously been sent. The cascaded authentication protocols in Section 4.3 are designed to guard against these types of attacks.

- *Message Ordering.* The Information Mesh security model does not provide any mechanisms for detecting attacks that attempt to change the ordering of the messages transmitted.

- *Denial of Service.* We do not provide any mechanisms for detecting or preventing a denial of service attack and leave it up to the user to recognize such an attack.

Voydock and Kent provide full descriptions of these attacks and discuss methods of guarding against them.

4.1.2 Weaknesses of Server-based Model

We must examine the weaknesses of the server-based model in order to illustrate the problems that we are attempting to solve. Since the Mesh security model is built upon the server-based model, we are able to use the security features that it provides and must only correct its weaknesses. We also must be careful not to introduce any new weaknesses in the process of expanding the model.

We have seen that the server-based model (Section 3.2) has two forms of weaknesses – those that can be corrected by using different protocols and those that are fundamental to the model. Weaknesses in the cascaded authentication protocols of the server-based model will be reviewed here, along with the Mesh security model's solution to them. The fundamental problems with the model will be discussed here, though the solutions to them will be discussed in the next section.

As previously discussed, the server-based model did not use a pairwise authentication scheme. This left it open to attack by two or more servers working together to fool a server later in the invocation chain. This collusion will be prevented by using a modified version of the cascaded authentication scheme that will be presented in Section 4.3. These modified protocols are based on the cascaded authentication scheme of Section 3.1. These protocols also have the advantage of using constraints to limit progress of a request beyond specified bounds.

The server-based model has other weaknesses that are more fundamental to the model. They result from the fine granularity of the security policy achieved by the model. Since the *effective* security policy for a collection of servers is the union of the servers' individual policies, an untrustworthy server may cause a discrepancy to form between the *desired* and effective policies for that group. A server may be unintentionally untrustworthy if it has a weak security policy or is poorly implemented or it may be purposefully untrustworthy – actively attempting to damage the security of the security domain. Any untrustworthy server will cause a weakness[2] in the effective policy of the domain, though the weakness will be limited to that untrustworthy server.

The fine granularity of the server-based model may also have problems when a desired policy change affects many servers in the security domain. As the servers are updated, there may be inconsistencies in their policies if some have been updated and others have not. This inconsistency could leave a weakness that leaves the domain open to attack. Additionally, *all* the servers that are affected by the change must have their policies updated. This could be a large and difficult task for changes that affect many servers.

These are not problems in a centralized system where all policy decisions are made by a single server[3]. Since all service providers must submit to the centralized policy, untrustworthy servers can not enforce incorrect policies. Furthermore inconsistent policies are not a problem since the update is only made in one place.

Centralized authentication systems, such as Kerberos, may experience other inconsistencies when modifying security policies. They often do not have a mechanism to revoke access privileges for a service immediately. Kerberos, for example, issues a principal a ticket for a service. This ticket is good until it expires, even if the security policy changes to exclude that principal in the meantime. The server-based security model, on the other hand, allows for immediate revocation of access privileges.

[2]Here we are considering a weakness to be a deviation from the desired security policy. If the desired policy has weaknesses, then it is a problem with the policy and not the servers.

[3]The centralized system may distribute the policy over several policy servers that act as one. These servers still impose a policy on service providers so, despite having a distributed security policy, it should not be confused with the server-based model.

We would, therefore, like to impose some centralized control on the server-based model. This must be accomplished without sacrificing the advantages of the autonomous model. The next section will describe such a solution that will be used as the Information Mesh's security model.

4.2 Security Domains as Objects

The Information Mesh's security model overlays the server-based model with some centralized control. The centralized control does not violate the object model that is crucial for the success of the server-based model. This is accomplished by allowing groups of servers to be modeled as a single *domain object*.

The domain object controls its own security policy, as all objects do in the server-based model. This allows an administrator to encapsulate a set of servers with a domain object and set the policy of the domain object. The domain object intercepts any messages that are destined for servers which it encapsulates. It then applies its security policy to the request and passes it on to its destination if it passes the security check. The message may then be intercepted by successive domains until it reaches its destination or it fails to pass one of the domain's security policies. Therefore, a domain object's security policy represents a centralized policy imposed on the set of servers it encapsulates.

The domain object may encapsulate a group of one or more servers, domain objects, or a combination of both. This flexibility permits many possible domain formations. At one extreme, the server-based model is the special case where a set of servers is not part of a domain object (Figure 4-1.A). Other configurations include a full hierarchy of domains (Figure 4-1.B), overlapping domains (Figure 4-1.C), domains that contain both other domains and servers (Figure 4-1.D), and any other combination one would like to create.

Domains may be added or removed without the "victim" servers knowing about the change. The servers do not know that any centralized control is being placed on messages sent to them since all they see is an incoming message that follows the

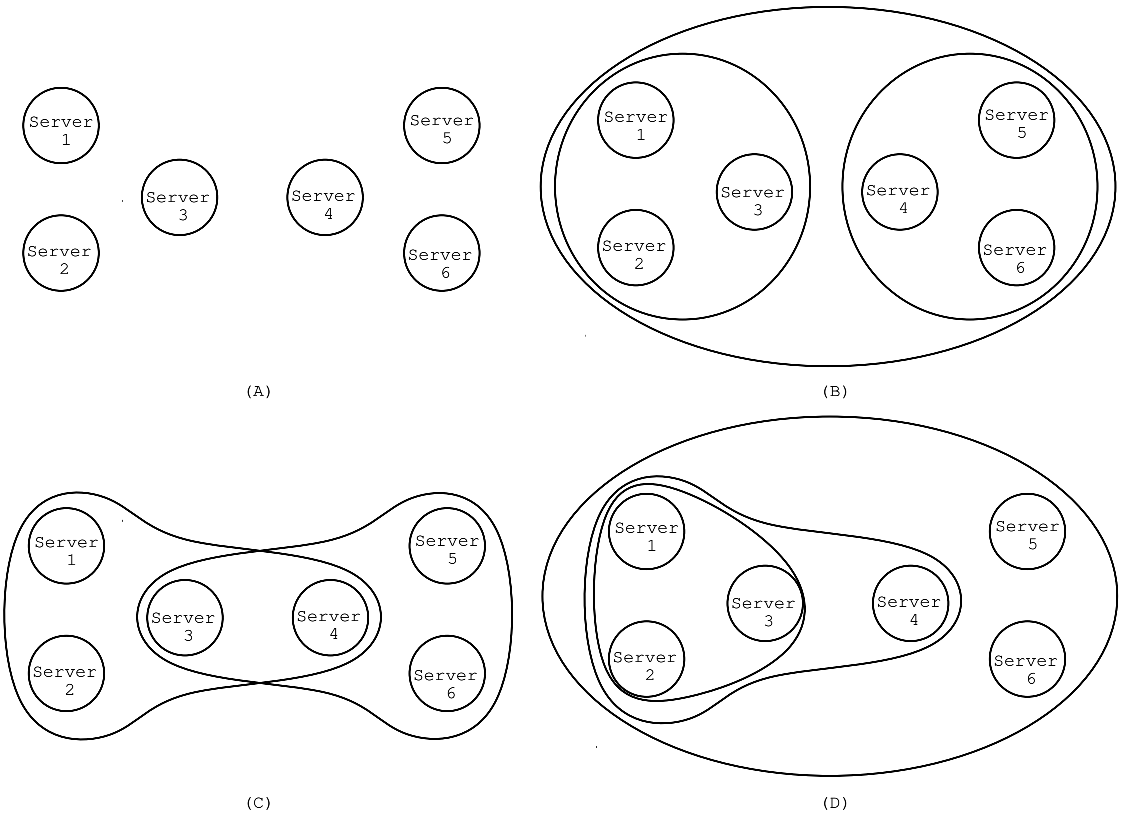

Figure 4-1: A Few Configurations of Six Servers in the Object-based Model. Enclosed areas represent objects. Unlabeled objects are domains. (A) Servers with no domains, equivalent to server-based model; (B) A hierarchical nesting of domains; (C) overlapping domains; and (D) domain that contains both other domain and servers.

same protocols whether coming directly from the requesting server or having been filtered through several domains. The server will only receive messages that have been approved by all the domains that surround it.

The activities, basically the enforcement of the security policy, of a domain is controlled by a *domain server*. The domain server is simply a generic server (as described below) that approves or rejects a request based on its domain's policy. How it shows approval or disapproval may vary between implementations (see Section 5.3). The domain server must be trusted to implement the desired security policy and to perform correctly.

Two questions about domains must be explored in more detail. First, we must determine how to make a group of computers, possibly located over a wide geographical area, behave as a single object. This question can be refined to ask how do we redirect messages to the appropriate domain server so it can be intercepted before it reaches its destination. Second, even with appropriate redirection of messages, there can be difficulties enforcing domain policies on some intradomain and possibly interdomain messages.

4.2.1 Navigating the Domains

The description of domains above assumes that there is a mechanism that enables a domain server to intercept messages. Such a mechanism would allow domains to be added and removed without affecting the operation of servers inside or outside the domain. Ideally, this would be accomplished by providing a secure mechanism for the domain server to specify, at the IP routing level, that network traffic destined for a particular server must be routed through the domain. Unfortunately, no such mechanism currently exists. This section offers some alternative mechanisms and their weaknesses.

In order to simplify our discussion, we will introduce a new term, the *path*. A path is a list of domains through which a request must travel in order to reach the particular server that can process the request. We will impose an ordering on the domains in a path from most general to most specific. A domain is more general than another if it completely encapsulates the other domain. Two domains that overlap, but neither completely encapsulates the other, are considered equal in terms of generality. This ordering can be necessary for enforcing security policies as we will see in Section 5.3.

The solution involves a second trusted server, the *path server*. A path server provides mappings between a server and its path. Adding and removing domains are done by informing the path server of the change. The path server then must update all its paths to reflect the change. The server also has the responsibility of making sure all the domain servers in a path have approved a request before it is passed on to the destination server.

Our problem of redirection is now reduced to having to redirect the requests to the path server so it can then direct the message through the correct path. If all the servers in a domain, or in a set of domains, are in a single geographical area and can all be placed behind a gateway, then the solution is simple – the gateway can redirect the messages to the path server.

Not all domains may have the geographic locality necessary to use gateway to redirect messages. There are several possible ways to solve this problem, though they all have some weaknesses. The first would be to use the Domain Name Service (DNS) [9, 10] to map server (host) names to the path server instead of to the actual host. This has some flaws. First, it depends upon the security of DNS. DNS security is not a safe assumption and has been a problem when implementing security managers for Java [4]. Second, it can not guard against an attacker who knows the IP address and port number of the destination server and can therefore bypass DNS.

Another way to handle this is to designate false ports on each host. These false ports would be advertised as the well known ports for servers. Any messages sent to these ports would be redirected to a path server who knows the real port to send the message. This is obviously flawed if the attacker knows the actual port for the server and sends messages directly to it.

Finally, two options involve putting trust in the servers to do the right thing. The first option would be to trust all the servers within a domain to make sure all requests have travelled the correct path. If they have not, the servers should send the requests to a path server in order to assure that they have traveled the appropriate path. This requires that we trust all the servers to behave correctly, which we were trying to avoid, so it is a useless solution. The second solution requires us to trust all servers outside the domain to send requests to the path server. This is obviously a problem because it forces a domain to trust the servers against which it is trying to protect itself.

This problem of redirecting messages is not solved here and is an issue for future research. The solution will likely involve manipulating routing protocols, as described at the beginning of this section. Since this work only describes a demonstration of the security model where we can control aspects of the environment, we trust all servers to send requests to a path server instead of the servers for which they are destined.

4.2.2 Local Message Problems

A second problem can arise when passing requests between two servers on the same local area network (LAN) or even on the same host. If a request is passed between the two servers it may be impossible to redirect the message before it is read by the destination server. This may not be a problem if all the servers on the same host or LAN are in the same domain. In this case, we can declare that the domains may only apply their security policies to requests originating outside the domain. The servers still control their own policies and will provide security for intradomain requests.

We can not completely pass the problem off that easily. What if a domain needs to apply some restrictions on intradomain messages? Or what if multiple domains exist on a single LAN or host? We need some way of redirecting these messages to a path server. Many of the flawed options above will also work to solve this problem. The option of trusting all servers to send their requests to a path server will “solve” this problem as it did above. Since we can control the environment of the demonstration, the trust is a reasonable assumption.

4.3 Cascaded Authentication

Cascaded authentication is needed to provide authentication for cascaded invocations. The Mesh security model uses a version of cascaded authentication very similar to the one presented in Section 3.1. This section describes modifications to the protocols that are necessary for the security model. This section also describes a third trusted server that is required by the model – the *authentication server*. Efficiency considerations with the protocol are also discussed. The section concludes with a discussion of the trade-offs between using public and private key encryption schemes for the protocol.

We also must take a very brief look at the representation of constraints. Constraints are information that may be added to an access certificate to limit future invocations on behalf of the associated request. They can be implemented in a similar manner to the security policy described in Section 5.2.4. We will use this abstract concept of a constraint until that discussion.

4.3.1 Modifications

The Information Mesh's security model makes several minor modifications to the cascaded authentication protocols described in Section 3.1. Beyond the modifications described here, all other aspects of the protocol remain the same. One modification results from communication with authentication servers, while the others are modifications to the access certificate.

The first modification occurs when a server performs a pairwise authentication with an authentication server. It is not necessary for the server to obtain a conversation key because it already shares a secret key, K_A, with the authentication server. The pairwise authentication protocol can be modified to use this key:

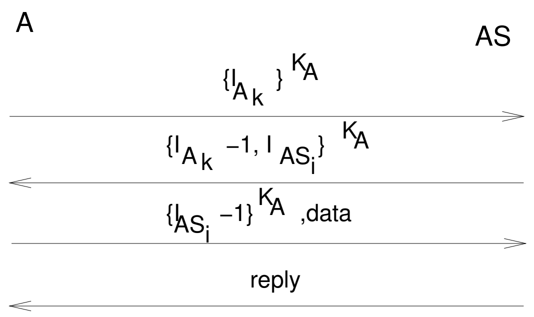

Figure 4-2: Pairwise Authentication with an Authentication Server

These changes do not affect the protocol very much. The first message of the protocol no longer needs the ticket since it was used to pass the conversation key to B and assure B that the ticket came from the authentication server. For the rest of the protocol, K_A replaces CK as the conversation key. K_A is known only to A and the authentication server so it behaves the same as CK. A problem arises when the key is compromised. A can no longer depend upon the conversation key to expire and will have to arrange a new shared key with the authentication server.

This problem with a broken conversation key may be compounded by the concern that a key is more likely to be broken if it is used more often. This suggests that we may desire different modifications to the protocol. This modification would involve leaving the ticket out of both the original pairwise authentication protocol and the protocol for getting a new conversation key. The ticket's purpose was to prove to a third party (B in Figures 3-1 and 3-2) that the conversation key came from the authentication server. There is no third party when communicating with an authentication server so the ticket is not needed. Therefore, a server that would like to communicate with the authentication server would use these modified protocols to get a conversation key from the authentication server and use the key to perform a pairwise authentication with it.

The `data` in the protocols contains two pieces of information. First it has the access certificate that is required to access the service. Second, it contains service-specific information. We will describe examples of service-specific information in the discussion of particular servers in Chapter 5.

We should note that we often want to be assured the `data` and `reply` are also authentic. If privacy is not an issue, we can accomplish this by encrypting the MD5 [12] of the information with the conversation key. This then can be used as proof of the information's authenticity. If privacy is an issue, though, we can just encrypt the information with the conversation key. This guarantees that an attacker has not tampered with the `data` or `reply`. Without one of these protections, an attacker could modify the unprotected `data` or `reply` and gain unauthorized access.

The remaining changes to the protocol involve minor modifications to the access certificate. The first was suggested as part of the protocol for the server-based model [3]. It was noted that access certificates often start and end at the same server. This allows the initial certificate to be signed using a secret key that is known only to that server for additional security. The initial access certificates in the Mesh security model will be signed using secret keys that are known only to the creators of the certificates.

The second modification to the access certificate results from a need to identify which service the access certificate is requesting. Every server will support several services, so it will need to supply access certificates for each service. When it receives a request, it would like to be able to identify which for service the access certificate was created. This could be accomplished by having a different secret key for each service. The client would tell the server which service it was requesting and the server could confirm it by using the appropriate key to decrypt the certificate. We have opted for another solution that involves adding the name of the service inside the encrypted part of the initial access certificate and placing it at the beginning of the cleartext part of the certificate. The server can immediately see which service is being requested by looking at the cleartext portion of the certificate and this can be confirmed by checking with the service encrypted in the initial certificate.

We need to add the following to our established notation:

SK_A a secret key known only to `A`

S the name of a service provided by A

We can now see the access certificate used by the Mesh:

$\{I_{A_i}, B, C_A, S\}^{SK_A}, S, A, C_A$

Successive signing of the certificate remains the same as before.

4.3.2 Authentication Server

The authentication server is the third trusted server required by the Mesh security model. An authentication server will service a defined group of servers. How this division is made is very flexible. There may be an authentication server per domain for medium-sized domains, one authentication server for many small domains, or many authentication servers within a large domain. The job of an authentication server may also be distributed over several servers.

Each authentication server stores keys for every server it assists. It must be able to generate and distribute conversation keys when requested. Finally, it will decrypt access certificates upon request and verify their validity. In order to support decrypting certificates that have passed through areas serviced by other authentication servers, each authentication server must share a key with all authentication servers that it trusts. Such a key would have to be pre-negotiated between the servers' administrators as must be done in a Kerberos system.

4.3.3 Efficiency

There are several efficiency issues concerning authentication. One issue, using a private or public key cryptosystem will be discussed in the next subsection. We would like to keep the overhead incurred by the authentication to a minimum and be able to incur little overhead when authentication is not required.

There are several features of the protocol that are included for efficiency reasons. The timestamp associated with conversation keys allow them to be reused to save communication with the authentication server. By adjusting the expiration time on the conversation key, the number of messages necessary for pairwise authentication can be brought arbitrarily close to four. The modifications to the pairwise authentication protocol discussed above were also made to improve efficiency.

Batching requests to the authentication server, or any other server, will reduce the number of network accesses. It also saves on the number of encryptions and decryptions necessary to process the requests which saves time and computational resources. In the example presented in Chapter 5, we will see this idea used when requesting multiple access certificates at the same time.

It is also necessary to minimize the overhead for requests that do not require any authentication. These requests still need to traverse the entire path dictated by the path server, but they can avoid the pairwise authentication and the access certificates which account for much of the overhead. These requests may be done through a different interface than requests requiring authentication. When a server receives such a request, it must check its security policy to see if the request needs authentication. If it does, the request must be denied, even if it is from a principal that has access to the service, since there is no means to authenticate the request. If it does not require authentication, then the request may be granted. This is not a required addition to the security model, but is desired for efficiency.

4.3.4 Private vs. Public Keys

All the protocols discussed so far have assumed the use of a private key encryption scheme. The protocols are easily converted to use a public key encryption scheme as demonstrated by Needham and Schroeder [11]. While the Information Mesh security model does not, and should not, specify which encryption scheme should be used[4], it is important to look at the trade-offs between the two schemes. This section only discusses the trade-offs in terms of the effect the schemes have on the speed of the authentication protocols. There are other factors to consider when choosing a method

[4]Specifying a single encryption scheme would violate the goals of the Information Mesh. Besides, over a hundred years, new encryption schemes may be realized and the trade-offs discussed here may change.

of encryption.

The cascaded authentication scheme requires many encryptions and decryptions, both for the pairwise authentication and the access certificate. This indicates that the speed of the encryption algorithm will have a significant effect on the speed of the authentication, especially when processing a request involves a cascaded invocation. In general, private key encryption schemes have faster implementations than public key schemes, so the speed of encryption is an advantage for private key schemes.

A close look at decrypting access certificates, though, exposes a potential for parallel access to keys when using a public key encryption scheme. When decrypting an access certificate to check its authenticity, each signature must be stripped off one at a time. If all the keys needed to decrypt the access certificate are stored on the same authentication server, then the time needed to decrypt it is limited by the encryption scheme. If the keys are located on different authentication servers, though, many network accesses will be required. This will affect the speed of the authentication. Using a private key scheme, the access certificate must be passed to each authentication server in turn to decrypt the layers for which it has keys. Using a public key scheme, though, enables the authentication server to request all the keys in parallel and can then decrypt the access certificate as the keys are returned.

Therefore, the overall speed of a public or private key encryption scheme depends upon the locality of the servers signing the access certificate. If the common case will be that most access certificates will only be signed by servers with keys on the local authentication server, then a private key scheme will be faster, otherwise a public key scheme may be faster.

4.4 The Generic Server

As a result of this security model, all servers will share a common interface. This allows us to create a *generic server*. In terms of the Mesh object system, we can create a *generic-server-role*. All servers must play the generic-server-role or one of its subroles. This section will briefly describe what actions a generic server must be able to support, while implementation considerations will be discussed in Chapter 5.

The generic server must be able to support a cascaded authentication interface. This means it must support pairwise authentication with other servers and an authentication server, be able to get a conversation key from the authentication server, and be able to request that the authentication server decrypt an access certificate. It may also provide a fast path to avoid the cascaded interface when possible, as described above. For our example, the generic server must also know to send requests to a path server.

In order to provide access control, the generic server must have a security policy. The security policy should be extensible so it can evolve over time and adapt to different applications. Possible implementations of security policies are discussed in Section 5.2.4.

A generic server must also provide at least two services. First, it must be able to create new access certificates when requested. This is required to support cascaded authentication. Second, it must provide a means for changing the security policy so it can provide access control that can change over time.

All servers must provide these minimum services. Chapter 5 will describe an implementation of a generic server and show how other servers, including the domain, path, and authentication servers may be built from it.

4.5 Summary

This chapter has described the security model that will be used for the Information Mesh. After exploring the threat model for the Information Mesh and the weaknesses of the server-based model, we developed the model for the Information Mesh.

The model is founded in the server-based model, but is extended to allow a group of servers, a domain, to behave like an object. This domain object has the ability to intercept messages destined for servers in the group it controls and can apply its own security policy to those messages. This effectively applies a centralized security policy to messages entering a domain. The model uses a modified version of the cascaded authentication protocols seen in the previous chapter.

The model presents three types of servers that are required and must be trusted. A domain server controls the security policy for the domain. An authentication server is needed to decrypt access certificates and provide conversation keys. As a result of the difficulty involved in redirecting messages to the appropriate domain servers, a path server is required to direct a message through the correct path.

Finally, we were able to develop a generic server that provided the interface and services that all servers must provide. This includes the ability to communicate using the cascaded authentication protocols and services to provide new access certificates and modify its security policy.

Chapter 5

A Room Reservation System

In order to fully understand the Information Mesh security model, it is necessary to look at an extended example of the model. The example will also expose implementation issues that were not handled as part of the model. This chapter will detail an implementation of the model that allows principals to reserve conference rooms around MIT.

This chapter begins with a discussion of the environment and the assumptions on which the example is built. It then details the implementation of a generic server which is used as a foundation for all the servers that are discussed: the path server, domain server, authentication server, room reservation servers, and signature servers. As each server is described, so will the issues associated with that server. Bootstrapping is a common problem for the servers and will be discussed separately. Finally, the example world will be built and run.

5.1 Environment

In order to simplify the example, it is necessary to limit the environment and the threats associated with it. These simplifications will help make the example easier to understand and implement. The simplifying assumptions will affect the design, but will still indicate how the security model would behave in a full Information Mesh environment. This section describes the assumptions about the environment and describes the room reservation service that has been developed within that environment.

The example is implemented on one host. This includes all the servers, domains, and clients. All servers are trusted to send service requests to the path server. These assumptions allow us to avoid the issue of how to redirect messages to the path server. This would not be appropriate for a full implementation of the security model. We chose not to address the routing problem in order to focus more clearly on the security problems.

The example also only uses one authentication server and one path server. Since we are implementing a fairly small world, more servers are not necessary. It also speeds examples up by not requiring cascaded authentication requests, beyond the request that is being processed. These simplifications avoid some confusion while running examples.

Finally, we will take a moment to discuss how we would like the room reservation service to work since it will influence the design of the servers that will implement it. Every room that may be reserved has a server that controls the authoritative list of reservations for the room. A *room reservation server* may be the authority for one or more rooms. It may also *know of* rooms for which it is not an authority, though it will know which servers are the authorities for those rooms. Before these authorities may schedule a reservation, the request must be signed by someone who is considered "in charge" of the room. This is accomplished by a *signature server*. It is possible for one signature server to be in charge of multiple rooms.

5.2 The Generic Server

The *generic server* embodies all the functionality of a server, except for the specific services it might provide. It can perform cascaded authentication, work with authentication servers to verify access certificates, and check permissions using the server's security policy. It implements the required services to provide new access certificates and allow for modification of the server's security policy.

This section describes the implementation of the generic server. It also discusses issues related to implementing a security policy that expresses the generality and flexibility necessary to achieve the goals of the Information Mesh.

The generic server has two encryption keys: a key shared with the authentication server and a secret key that only it knows (for encrypting the initial access certificates). The server keeps a *policy file* that stores its security policy. It also keeps a list of access certificates it holds in an `ac-list`.

The generic server has the ability to manage both ends of the unmodified pairwise authentication protocol, so it can both send and receive requests with other servers. When sending a request, it first requests a conversation key from an authentication server. Additionally, it can send requests to authentication servers using the pairwise authentication protocol that we modified for communication with authentication servers.

5.2.1 Processing a Request

When a server receives a request for a service, it must determine if the request passes its security policy. This section will outline the steps necessary to authenticate a request. Servers may skip some of these steps as their policy dictates. Here is an outline for a full authentication of a request:

1. Check security policy to see if the request must be authenticated.
2. If the request must be authenticated, send the access certificate to an authentication server and wait for the reply. This involves:
   (a) Get an access certificate for the authentication server.
   (b) Sign the access certificate.
   (c) Send the original access certificate to the authentication server, along with the access certificate for the authentication server, and wait for the reply.
3. If the response is positive, apply the security policy, if necessary.
4. If the request passes the security policy, invoke the service.

The server invokes the requested service using arguments provided in message three of the pairwise authentication (Figure 3-2). If, at any point, the request fails to pass any of these tests, an error message is returned to the client requesting the service.

5.2.2 Sending a Request

The process of sending a request can be a bit more complicated than the process of receiving one. This results from having to keep track of the number of access certificates the server is holding for a service, so it can get more when it runs out. Here are the steps required to send a request:

1. Check to see if there is an access certificate for the service in `ac-list`.
2. If there are no access certificates for that service, we must get some before trying to send the request. When getting certificates, it is best to get many of them to reduce the number of `get-ac` requests.
   (a) Save the request in a `request-list`.
   (b) Get some access certificates for the service.
      i. Send a request using the service `get-ac` to get more certificates for the desired service. This is accomplished by starting at Step 1 with the service `get-ac`. This procedure will not recurse because there must be access certificates for `get-ac` or else it has encountered a bootstrapping problem.
      ii. If this only leaves one remaining access certificate for `get-ac`, then use it to get more access certificates for `get-ac`.
   (c) Remove the request from the `request-list` and goto step 1. This will also not recurse because now there is a guarantee that there will be access certificates available for the service.
3. If the access certificate is in `ac-list` get it and remove it from the list.

4. If this leaves no more access certificates for the service, get some more (see step 4-b).

5. Sign the access certificate.

6. Send the request using pairwise authentication.

We should note that this process requires the server to hold some `get-ac` access certificates initially. This bootstrapping problem will be discuss in Section 5.7.

5.2.3 Generic Services

A generic server provides two services that are required both for cascaded authentication and to allow the servers to evolve. `get-ac` allows clients to request new access certificates and `update-policy` allows authorized principals to change the server's security policy.

`get-ac` requires the following information in order to process the request: `(number destination service)`. `get-ac` replies with `number` access certificates for `service`. `destination` refers to the destination server that is to be encrypted in the access certificate. `destination` must correspond to the server that is requesting the access certificates.

The information necessary for an `update-policy` request will vary depending upon the implementation of the security policy. For our security policy (described below) the necessary information is: `(service attribute add? value)`. `service` indicates which service's security policy is being modified. `add?` indicates whether the bit associated with `attribute` should be set high or low. `value` specifies any values that may be associated with the attribute in the security policy. Finally, the attribute `all` allows policies for services to be added and removed. Specifics about the security policy and attributes are discussed below.

5.2.4 Security Policies

The representation of its security policy can severely restrict the flexibility and evolvability of the Mesh security model. We would like a representation that allows the security policy to change over time. Not only should the values of the attributes be able to change, but which attributes are checked should be allowed to change. For example, one server may only care about which hosts are trying to access it, while another may care how many servers have handled the request, how much the client is willing to spend on the service, and the speed of the client's internet connection. In the future, attributes we do not consider important now may be checked for access control purposes.

For the example, we have developed a simple security policy representation. It has some flexibility about which attributes it checks, but does not evolve well. We will discuss an idea for a more sophisticated representation later.

The simple security policy has the following form:

```
(service (authenticate? constraint? security-policy) init-constraints)
```

Where `security-policy` and `init-constraints` (and other constraints) have the form:

```
(bit-list value value ..)
```

Each server has such a policy for every service it provides. These are distinguished by `service`. The bits `authenticate?` and `constraint?` indicate whether or not it is necessary to authenticate a request for this service or necessary to check all the constraints in the access certificate.

The remainder of the security policy and the constraints have a similar form. They have a set of bits where each bit represents an attribute that can be checked. If the bit is set high, then the attribute must be checked and any information necessary to define the policy for that attribute is included as a `value`. The values are listed in the order of the bits that are set high. For example, a policy may check four attributes: The number of servers that may handle the request, the principals that may handle the request, the price the client is willing to pay for the service, and the time that has passed since the access certificate was created[1]. A server may then have the policy:

```
(1010 (4) ($1))
```

This policy says that the request may be handled by no more than four servers and the client must be willing to pay at least one dollar for the service.

This policy representation does not evolve well because it relies on some universal agreement on the attribute which each bit represents. A solution to this problem may be suggested by the Hydra [20] capability system. Hydra designed a flexible security policy based on twenty-four bits. The first sixteen bits represented access control privileges to a fixed set of general operations. The last eight bits represented type specific privileges. Those bits represented access control for different operations in policies of different types.

It may be possible to transfer the idea of different types of policies to the Information Mesh. This could be accomplished by making a security policy a first-class object. This would allow us to define a *security-policy-role* and different types of policies would be represented by subroles of the security-policy-role. Each attribute could be defined as a different action of the role, allowing each subrole to support a different set of attributes. Every security policy would have to play one of the subroles of the security-policy-role. This and other representations for security policies are subjects for future research.

5.3 Path Server

A *path server* is a trusted server that directs a request through the appropriate domains to get to its destination. The path server must keep track of the servers and domains that exist and must be able to direct the request to each domain for approval. A path server does not check its security policy for any requests that are not destined for itself; its job is not to approve requests, just to distribute them to the

[1]This attribute is easy to verify if a timestamp is used for the nonce in the initial certificate.

servers that do approve requests. For this reason, it is not necessary to get an access certificate when trying to access a different server through the path server.

The service for directing a message through the domains is called `enter-domain`. The service request requires three arguments, `(info server service)`, the information necessary to process the request at the destination server and the `server` and `service` for which the request is destined. The access certificate for the destination service is passed to the path server as normal since no access certificate is required for the path server.

Once the path server receives such a request, it then contacts all the domain servers in the path. There domain servers may be contacted sequentially or all at once. These methods lead to different solutions that can affect the security policies that are possible. If the request is approved by all the domain servers, then it is passed on to the destination server.

When contacting the domain servers sequentially, the path server first signs the access certificate, then passes it (along with any other relevant information) to each domain server from the most general to the most specific, as discussed above. The domain server will check the certificate and information against its security policy. If the request passes, the domain server signs the certificate and returns it to the path server, otherwise it returns a negative response. This method results in all the domain servers signing the access certificate in order.

There are several advantages to this method. It allows servers (both domain and local) to base at least part of their security policies on the approval of a more general server. It also leaves a distinct audit trail since the signatures on the access certificate indicate *all* the servers that have handled it. The method has a very distinct disadvantage, though. It will be slow, especially if there are several domains in the path. The many encryptions necessary and the ordering of each approval will add a long delay for the request to get to its destination.

In the interest of developing a more efficient method to get approval from the domains, the path server may contact all the domains in the path at the same time. The path server sends the access certificate and other relevant information to the domain servers. All it expects in return is an approval or disapproval of the request. If all the domain servers return a positive response, then the path server signs the request and passes it to the destination server. If any domain server returns a negative response, then the request is rejected.

This method is much more efficient than the previous one. It also leaves an audit trail, only not as explicit as in the previous method. The audit trail can be found by contacting the path server that signed the access certificate and querying it for the path that it contacted for approval. This method is limited in that security policies can not depend upon the response of a more general server.

We have chosen the latter method in the interest of efficiency, but do not limit other implementations to do the same.

Finally, we must discuss other services that a path server must support. The server should allow domains and servers to be added and removed. Additionally, domains need to be allowed to change over time. These services are reflected by `modify-domain` and `modify-server`. It is incumbent upon domains and servers to tell the path server when they come into existence or when the go away in order to keep the server current.

`modify-domain` allows an authorized principal to add, remove, and change domains. The changes are indicated by three arguments: `(domain modify victims)`. `domain` indicates the name of the domain that will be affected, while `modify` indicates what will be done to `domain`. `modify` is a number that tells whether `domain` is to be added, removed, or modified. `victims` is a list that describes which objects will be under the influence of the domain. It is a list of pairs `(name . type)` where `name` is the name of the victim and type is one of the following: `server`, `domain`, or `group`. `server` means that the named victim is a local server, `domain` that the victim is another domain, and `group` means that the victim is a group of servers that can be named as one, such as: `*.mit.edu`, `*.lcs.mit.edu`, `18.26.0.*`, etc.

When `modify-domain` is invoked, it is necessary to compare every server and domain indexed by the path server against the victim list to see if it is affected. If it is, then appropriate changes must be made to indicate the change in the domain.

`modify-server` allows an authorized principal to add or remove a server from the path server. This requires the request to include `(server add?)` to name the server and whether it should be added or removed. Adding a server means that the path server must check all the domains that it knows about and determine the path required to get to the new server.

5.4 Domain Server

The *domain server* is the second trusted server. As we saw above, its purpose is to apply a domain-wide security policy and inform a path server of the results. A domain server needs `(destination ac info)` in order to `approve` the access certificate. This provides the name of the destination server, access certificate, and any additional information needed for judging the request. The domain server uses the tools provided by a generic server to check the access certificate against its security policy and returns its approval or disapproval of the request as discussed above.

5.5 Authentication Server

The final trusted server is the *authentication server*. While it is based on the generic server, it has some subtle variations that will be discussed here. The authentication server must perform two services – authenticating access certificates and distributing conversation keys.

The first variation is a result of distributing conversation keys. There is no need for pairwise authentication when requesting a conversation key because only authentic servers will be able to decrypt the response containing the key. This service, therefore, by-passes the pairwise authentication interface.

The second variation from the generic server is the requirement that the authentication server must be able to communicate using both sides of the modified pairwise authentication protocol that was described in Figure 4-2. All its communications will use this modified protocol. Local and domain servers that trust the authentication server will share a key with the server. Furthermore, authentication servers that trust each other must similarly share a pre-negotiated key. Since these groups represent all the servers that will communicate with the authentication server, only the modified protocol is needed.

Authentication servers will contact each other for authentication services when they do not control all the keys necessary to verify an access certificate. They may only contact other authentication servers that they trust and with whom they share a key. This communication may require further invocation before the authentication is complete. These invocations must be accomplished using the modified cascaded authentication protocol.

The only data needed to `authenticate` an access certificate is the access certificate itself. `authenticate` decrypts one layer of encryption at a time and makes sure that the certificate was formed properly. If it encounters a signature for which it does not have a key, it will request help from a different, but trusted, authentication server. It can not decrypt the initial certificate since it is encrypted with a key that only the creator knows. It returns this initial certificate to the requesting server to finish verifying the certificate.

Now that we have described the servers that are required for the security model, we can now look at the local servers that are specific to the example.

5.6 Local Servers

Reserving conference rooms requires two types of servers: a signature server to grant official approval for the principal to use a room and a room reservation server that keeps track of the reservations for a room. It is also necessary to have a client to request the reservations.

The servers are modeled after the process that is necessary to reserve conference rooms. To reserve a room, you must go to the person in charge of the room, find out if the room is free when you need it, and get a signature to approve your use of the room at that time. The servers are a slight variation on the process. First a client must receive permission to use a room, then it checks with the server that has authority over the room to see if the room is available when it is needed. This section describes these servers and a client.

5.6.1 Signature Server

The *signature server* has the job of signing a request for a room. If the request passes the security policy of the signature server, the server then signs the request. The signature server (as well as the room reservation server) may want to know any number of details to `sign-request`. This information should be listed as an extensible list of attribute-value pairs such as the following:

((cert ac)	access certificate for reservation service
(for value)	group for which the room is being reservered
(contact value)	who to contact about the request
(reply value)	where to send replies
(room value)	which room is being reserved
(time (day, starttime, length))	when the room is needed
(period value)	periodic use of the room: value is `onetime`, `daily`, `weekly`, `monthly`
(number value)	number of people the room needs to accommodate
(cost value))	amount the group is willing to pay to reserve the room

The signature server signs `ac` and replaces it in the list. It then passes the request on to the appropriate reservation server. If the request does not pass the security policy of the server, then an error is returned to the client.

5.6.2 Room Reservation Server

The *room reservation server* is responsible for scheduling the use of one or more rooms. It must keep track of the use of each room for which it is the authority and make sure there are no scheduling conflicts. It may know about rooms for which it is not the authority. It will pass such requests to the appropriate signature server for further processing.

In order to `reserve` a room, the room reservation server needs the same list of information that was used to access the signature server. It is likely that its security policy will require that the access certificate be signed by a particular signature server, though this may not be the case for all rooms. In all cases, the room must be unused for the entire time requested.

Additional functionality may be added to the room reservation server. If the server is given some knowledge of characteristics of each room, it would be possible for the reservation server to suggest rooms when a room is not specified in the information list. For example, if the server knows about the seating capacity of each room, it could suggest a room based on the number of people that will be using it. It could include other characteristics such as the presence of audio/visual equipment, conference tables, whiteboard space, etc.

5.6.3 Client

A `client` has much of the same functionality as the generic server. While it must be able to send cascaded requests, it does not need to be able to receive the requests. It must also have the added functionality of being able to initiate a request for a particular service or set of services. This involves collecting the necessary information to initiate the request (along with an access certificate) and sending it via the pairwise authentication protocols.

5.7 Bootstrapping

This model requires that a server must have an access certificate to get more access certificates. This implies that a server must be primed with some access certificates before it may function in its environment. This section details what information is required to solve this bootstrapping problem and a way that it may be achieved.

Before a server or client can use a service, it must have an access certificate for that service. To get the access certificate, the server needs an access certificate for `get-ac`. This implies that a server needs access certificates for `get-ac` for every server with which it ever wants to communicate.

While this may be possible to create by hand in the little world of our example, it does not scale to a real system. In the real world, there will likely be services that trade access certificates. They could provide `get-ac` and other access certificates for large numbers of servers and services. A new server would then only need to be primed with a few access certificates, including at least one trading service. It would then have access to certificates for nearly any service that it wanted.

5.8 The Example

We have used the servers described above to create an example of the Mesh security model in action. This section describes our example world and some tests and their results that have been performed on this world. We also use the example to illustrate where bottlenecks may occur in the system and possible solutions to them.

5.8.1 A Miniature World

Our example world uses the servers described above to implement a small room reservation service around MIT. There is one path server and one authentication server for the entire world to simplify the problem. Figure 5-1 shows all the servers and domains in the example world.

The example world presents five hierarchical domains. The Laboratory for Computer Science (LCS) and Research Laboratory for Electronics (RLE) domains fall under the control of the Department of Electrical Engineering and Computer Science (EECS). EECS and the Department of Mechanical Engineering (MechE) both fall under the jurisdiction of the MIT domain. These hierarchical domains are for demonstration purposes only and do not reflect the actual control structure within MIT.

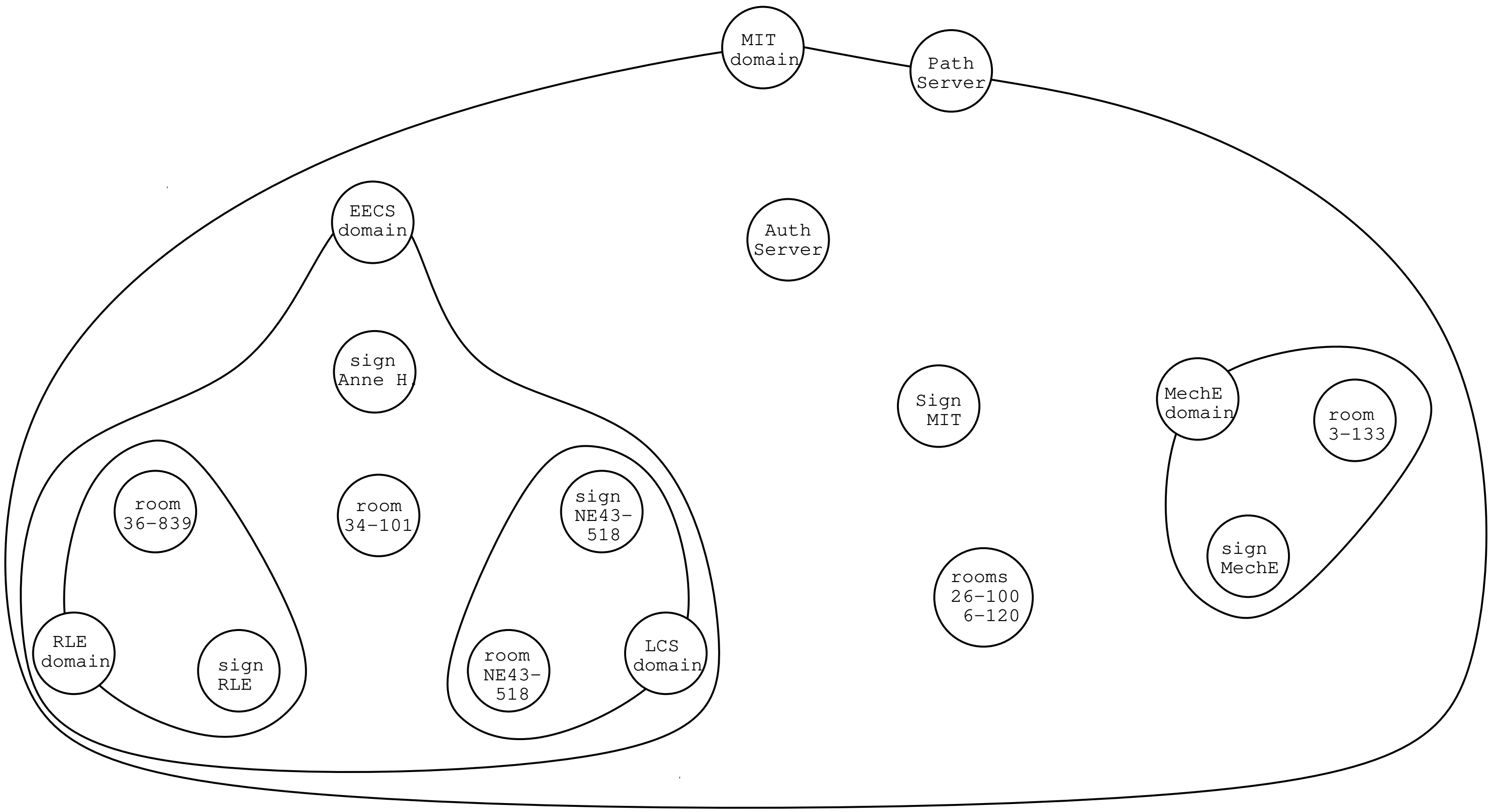

Figure 5-1: The Example World.

Within each domain are the room reservation servers and their associated signature servers (See Figure 5-1). The room reservation server for rooms 26-100 and 6-120 know about the rooms 3-133, 34-101, and NE43-518 and vice versa. Additionally, the room reservation server for 34-101 knows about room 36-839 and NE43-518 (and vice versa).

The room reservation servers have a policy of not accepting any `reserve` requests unless they have been signed by their associated signature servers. The MIT domain server has a policy of not accepting any domain requests that originate outside of `*.mit.edu`. While the ability exists to change these policies, they should not be modified. All other policies are subject to change for demonstration and testing purposes.

5.8.2 Tests and Results

Although this example was mainly built to expose issues surrounding the Information Mesh's security model, we can still use it to run some informative tests. First, the example is a proof of concept. By using it, we can prove the validity of the model. Second, although the implementation was not designed for speed or efficiency, we can run some tests that will expose where the model imposes most of its overhead. This information will aid us in determining where bottlenecks may occur.

There is not much that can be said about how these tests demonstrate the validity of the model. All the following tests prove that the model works within the assumptions we made at the beginning of this chapter. All the security policies and path information were set up by running the demonstration. Many other tests have been made to show that the model enforces its security policies, the path server directs requests along the appropriate paths, and that cascaded requests cascade correctly.

There are several timing tests that we would like to run. We would like to know the overhead caused by: getting a conversation key, authenticating an access certificate (and how much overhead each signature adds to this process), getting approval from a domain, getting approval from multiple domains, and getting new access certificates when making a request. Since the actual times will vary greatly with different implementations, we will look at relative data. All results represent the average of at least five runs of each test started when the load on the host was at or near zero. The methods used to arrive at the numbers will be described with each test. While these numbers may also change with different implementations, they are much more informative than actual times.

Getting a conversation key

Unless the time it takes to get a conversation key is very large, this time is not very significant per-request, since it will be amortized over many requests. Where it may be significant is determining if it will contribute to the authentication server being a bottleneck to the system. For this test, we will measure the time that it takes to get the conversation key and compare it to the time it takes the client to send a request to the path server.

Getting the conversation key was only responsible for approximately 20% of the time needed to make the request. The majority of the time (approximately 80%) was spent on signing the request and the pairwise authentication. The results of this test were very consistent with each running of the test. These numbers should shift even more heavily towards the signing and authentication as the access certificate gets larger and the encryption takes longer. So we can see that getting the conversation key does not contribute much overhead.

Authenticating an access certificate

Another action that may cause the authentication server to be a bottleneck and will slow a request is the verification of an access certificate. The time this process takes will vary with the number of signatures on the certificate. We would like to know what fraction of a simple request is a result of verifying the access certificate and how much time additional signatures add to the verification.

We will test these by timing how long it takes to verify certificates that have four and five signatures on them. They can be represented by the following requests: a request for the MIT domain server to update its policy and a request to reserve room

26-100 when the client asks the MIT signature server.

Verifying the access certificate with four signatures required only an additional 10% in time over the benchmark described below. Verifying a certificate with a fifth signature on it took about four and a half times longer than verifying the certificate with four signatures. This result is a bit suspect. Part of the discrepancy is due to the additional load placed on the host running the servers by adding the server necessary for the fifth encryption. There may also be an efficiency problem with our representation of the encrypted string, which causes its length, and hence time needed for decryption, to grow faster than necessary. While the additional signature should add some time in the verification of the access certificate, it should not add as much as our tests show.

Getting approval from domains

Another task which we expect to add time to a request is getting the approval of the domains in a path. We would like to compare the time it takes to approve a request through a domain with the time it takes without a domain. First, we will ask the MIT domain server to update its policy. This request will not need the approval of any domains and can be used as a benchmark. This test will be compared to a similar request to the MIT signature server which only needs the approval of the MIT domain server. Results show that the latter test takes approximately 30% longer than the benchmark.

We can postulate that asking for the approval of multiple domains in parallel will speed up the approval time. Unfortunately, our environment does not allow us to test this appropriately. Since all the servers are running on one host, the parallel requests must be processed serially. Furthermore, the numerous requests increase the load on the machine, slowing down the results. For example, a policy update request for the LCS signature server must get the approval of three domains: the MIT, EECS, and LCS domains. This test takes approximately 110% longer than the benchmark.

Getting new access certificates

Our final test is to see how much time getting more access certificates adds to the processing of a request. As with the time for getting a conversation key, this expense will be amortized over many requests since the request asks for multiple certificates at once. For this test, the client will be making requests to update the MIT domain's security policy so no domain approval is necessary. We will compare the running times of the request to update the policy and of that request coupled with a request for ten additional access certificates for the service.

The request that was coupled with getting more access certificates took approximately twice the time of the request that did not require additional access certificates. This is to be expected, since getting more access certificates during a service request adds a second full request to the one necessary for the service request.

5.8.3 Bottlenecks

We can imagine that three types of servers might become bottlenecks in a real implementation of the Mesh security model: the authentication server, the path server, and the domain servers. This section will briefly examine where the bottlenecks may occur and possible means for alleviating them.

The authentication server may become a bottleneck if it serves too many servers or if a high percentage of the requests must be authenticated. In either case, the authentication server may have more requests for verifying access certificates than it can reasonable handle. As we discovered above, getting conversation keys will not contribute much to a bottleneck.

One solution would be to reduce the granularity of the authentication server. There would be more authentication servers serving fewer servers, each with a reduced load. This would work well if there is not much communication between the groups of servers served by each of the new authentication servers. If there is much communication between the groups then the authentication servers would often need help from each other to verify access certificates. This would probably negate the value of reducing the granularity of the authentication server. A second solution would be to replicate the authentication server.

Domain servers for large domains may also become a bottleneck since they are the only entrance point to their domains. We may also solve this bottleneck using either replication or distribution. Replication would allow many access points to all servers. It has a couple of drawbacks, though. First, it is necessary to have a means of keeping their security policies consistent. Second, we must have a means for servers, or protocols, to choose the domain server where they will send their request. Distribution, on the other hand, provides different access points for different servers. Each domain server for a single domain would allow access to different servers within that domain. Now we only have the problem of determining to which domain server a particular request must be directed.

The final possible bottleneck is a relic of our assumptions for the example, the path server. Since the path server services all requests for a set of domains, it can be more of a bottleneck than the domain servers. This bottleneck could also be relieved using either replication or distribution. Since the path server should not be part of a real implementation, the possibilities do not need to be discussed here.

These represent all the possible bottlenecks that may be imposed by the Mesh security model. Other bottlenecks may result from particular local services, such as a signature server that must sign requests for numerous room reservation servers, but are not a concern for this discussion.

5.9 Summary

In this chapter, we have seen an implementation of the Mesh security model. It made several assumptions about its environment to simplify some problems. Most notably, it trusted all servers to send requests to a path server. We used the implementation to develop an example room reservation service.

Within this environment, we were able to develop several servers based on a generic server. These included the authentication server, path server, domain servers, a client, and two servers needed for the room reservation service: a signature server and a room reservation server. In order to implement these servers, we needed to develop a representation for security policies.

Finally, we looked at a room reservation system in action. We developed a small world of servers around MIT that could sign requests and reserve rooms. This example was a proof of concept and allowed us to run some tests to help us determine where bottlenecks may occur in a real implementation. Finally, we looked at possible means of opening up these bottlenecks.

Chapter 6

Conclusions

We conclude this paper with a brief summary of the ideas that have been presented and a review of further research that must be done before the Mesh security model is feasible.

We started by discussing the Information Mesh project and its goals to give us an understanding of the environment for which we needed a security model. Our model needed to support the Information Mesh's goals, including flexibility and evolvability, while providing a uniform model for all servers. The Information Mesh also uses an object model, with which the security model had to comply.

We then discussed two previous works which we could use as a foundation for the Mesh security model. The first was cascaded authentication. Cascaded authentication provided us with the protocols needed to allow a server to authenticate the entire chain of a cascaded request. It brought up the need for an access certificate which every server that handles a request must sign. Cascaded authentication provided the authentication for the security model.

The second work proposed a model where every server controlled its own security policy, although its default policy may be to query a central policy server for advice. While this model conforms to the object model, it has the problem that it requires all the servers in a policy domain to be trusted to implement the correct security policy. It also has the potential for inconsistencies to develop when updating a policy that involves many servers.

We wanted to expand this server-based model so that it provided the option for centralized policy control, while allowing objects to control their own policies. This was accomplished by allowing groups of servers to behave as a single object, while still retaining their identity as an object. This domain object would intercept all messages destined for servers under its control, apply its own security policy, then pass the request on to its destination if it passes the security check.

This model requires three types of trusted servers. An authentication server is necessary to verify the access certificates needed for cascaded authentication. It also must provide conversation keys so servers may perform a pairwise authentication with each other as part of the cascaded authentication protocol. A domain server is necessary to enforce a domain's security policy. Finally, the lack of a protocol to allow domains to intercept incoming requests made it necessary to develop a path server. The path server directs messages to the appropriate domain servers before they are passed on to their destinations.

Using this model, we were able to develop a generic server that could perform all the actions required of a server. It communicated with other servers using the cascaded authentication protocol and with the authentication server using a modified version of the protocol. The generic server also provided two forms of generic services: providing new access certificates for services and allowing authorized principals to update its security policies. One problem that was encountered when developing the generic server was finding an appropriate representation for the security policy. A solution was given, but further work must be done to find an appropriate representation.

At this point, we were ready to build an example implementation of the model. The example provided services for reserving conference rooms around MIT. In order to simplify the implementation of the example, we assumed that all servers could be trusted to send requests to the path server instead of to the destination of the request.

The generic server was used to implement the path and domain servers. Some minor modifications to the generic server were necessary to use it to implement the authentication server. The generic server was also used as a foundation for the signature and room reservation servers that were necessary to provide the room reservation services in the example. Finally, a client to invoke all the services was implemented using the generic server. The implementation of these servers uncovered a bootstrapping problem that is inherent in cascaded authentication – the need to have some initial access certificates to get the process started.

Finally, we were able to run the example and run some tests that allowed us to recognize some potential bottlenecks in the model. They basically occurred at each of the three required servers. Each bottleneck could potentially be solved using either replication or distribution, though they will have different consequences in different situations.

6.1 Further Research

In developing the model and the example implementation, we discussed two ideas that require further research. First, protocols need to be developed that will allow messages to be redirected to the appropriate domain servers. Second, a flexible, evolvable representation for a security policy must be designed.

A protocol allowing domains to intercept requests is necessary before this security model can truly be implemented. The protocol must be secure against attackers, such as unauthorized domains, trying to redirect requests. It must be able to guarantee that the messages will always be redirected appropriately. The protocol must also be efficient. It will not be acceptable if it adds much overhead to a request, especially a request that does not require any authentication. Additionally, any effects the protocol may have on the model presented here must be explored. This research must be completed before the model will be valid.

An appropriate representation for security policies must also be developed. In order to meet the Information Mesh's goals of an infrastructure that is flexible and evolvable, it is necessary to design a policy representation that also adheres to these goals. A possible solution, as suggested in Section 5.2.4, is to make the security policy a first-class object and use the Information Mesh's object model to specify different types of policies. By allowing different types of policies to be specified, the security policies can evolve as needed. This solution and others should be investigated to find an appropriate representation for security policies.

6.2 Conclusion

The security model presented in this paper has much potential as a security model for the Information Mesh and other systems requiring a model that conforms to an object-based infrastructure. Its flexible mix of centralized and local policy control give it the power to implement complex security policies. Future work on security policy representations will allow these security policies to be modified for different applications and evolve as required.

Currently, its major flaw is the lack of a protocol to redirect messages to the domain servers. Once a solution is devised for this flaw, the model will be valid. This will eliminate the need for both the path servers and the trust in all servers to send their request to the path servers. The overhead incurred by contacting the path server will also be eliminated.

In conclusion, once this protocol problem has been resolved, a full scale implementation is needed to demonstrate the model and the ideas presented in this paper. It will have to consider the trade-offs between using public and private keys and determine the appropriate ways to relieve bottlenecks for the environment which it is built. The answers to these questions will vary between different systems and different domains.

Bibliography

[1] Architecture Project Management, Ltd., "ANSA Security Framework", AR.009.00, May 1993.

[2] Berners-Lee, T., Masinter, L., McCahill, M., "Uniform Resource Locators (URL)", Network Working Group RFC 1738, December, 1994.

[3] Bull, J. A., Gong, L., Sollins, K., "Towards Security in an Open Systems Federation", *Computer Security - ESORICS 92*, pp. 3-20 Springer-Verlag LNCS Series, Nov. 1992.

[4] Dean, D., Felten, E., and Wallach, D., "Java Security: From HotJava to Netscape and Beyond", *IEEE Symp. on Security and Privacy*, 1996.

[5] ISO/IEC DIS 10181-1 "Security Frameworks for Open Systems: Overview"

[6] ISO/IEC DIS 10746-1,2,3,4 "Basic Reference Model for Open Distributed Processing"

[7] Kohl, J. and Neuman, C. "The Kerberos Network Authentication Service". RFC 1510, September 1993.

[8] Kunze, J., "Functional Recommendations for Internet Resource Locators", Network Working Group RFC 1736, February 1995.

[9] Mockapetris, P., "Domain names – concepts and facilities", Network Working Group RFC 1034, Nov. 1, 1987.

[10] Mockapetris, P., "Domain names – implementation and specification", Network Working Group RFC 1035, Nov. 1, 1987.

[11] Needham, R. M., and Schroeder M. D., "Using encryption for authentication in large networks of computers". *CACM*, Vol. 21, No. 12, December 1978, pp. 993-998.

[12] Rivest, R. "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.

[13] Sollins, K., "Cascaded Authentication", *Proc. of the IEEE Symp. on Security and Privacy*, Oakland, CS, April 1988, pp. 156-163.

[14] Sollins, K., Masinter, L., "Functional Requirements for Uniform Resource Names" Network Working Group RFC 1737, December 1994.

[15] Sollins K., Van Dyke J. R., "Linking in a Global Information Infrastructure", *WWW Journal*, Conf. Proc. - 4th International WWW Conf., Boston, MA, Dec. 1995, pp. 493-508.

[16] Van Dyke, J. R., "Link Architecture for a Global Information Infrastructure", MIT/LCS/TR-659, June 1995.

[17] Veléz-Rivera, B. "Information Mesh Objects", Working Document.

[18] Veléz-Rivera, B., and Bawden, A., "The Information Mesh Kernel", Working Document.

[19] Voydock, V. L. and Kent, S. T. "Security Mechanisms in High-Level Network Protocols", *ACM Computing Surveys*, Vol. 15, No. 2, June 1983, pp.135-171.

[20] Wulf, W. A., Levin, R., and Harbison, S. P., *HYDRA/C.mmp: An Experimental Computer System*. McGraw-Hill, New York, 1981.