# Directory Enabled Distributed Packet Filtration System:

# A Scalable and High Performance Security Architecture

**by**

**Siddhartha Gavirneni**

B.Tech., Jawaharlal Nehru Technological University, Hyderabad, India 2001

Submitted to the Department of Electrical Engineering and Computer Science and the Faculty of the Graduate School of the University of Kansas in partial fulfillment of the requirements for the degree of Master of Science

______________________

Professor in Charge

______________________

______________________

Committee Members (2)

______________________

Date of Acceptance

# Acknowledgements

The work for this thesis has been done at the Networking and Telecommunications Services (NTS), University of Kansas.

I must firstly thank my supervisor at NTS, George F. Willard, who deserves the greatest of thanks, since he has provided me with incredible support, encouragement, and guidance in both the writing of this thesis and the work that preceded it. I would also like to thank the people at NTS for the excellent work environment, which has helped me concentrate on this project.

I thank my advisor, Prof. Joseph Evans, for taking his time to supervise this Master's thesis despite his sabbatical leave. He has furthermore been an excellent advisor during my two years of study at the University of Kansas. I thank Prof. Gary Minden for many insightful conversations during the development of the ideas in this thesis, and for helpful comments on the text. I also thank Prof. Victor Frost for being a part of my committee and evaluating this project.

Away from work, special thanks go to all my friends and roommates, both near and far, who helped me achieve the balance between recreation and studies, and also for their much-appreciated advice.

Most importantly, I would like to thank my family, especially my parents, for their absolute confidence in me. To them I dedicate this thesis.

## Abstract

Monolithic firewall implementations in an enterprise or university environment are impractical for many reasons. Each department or subdivision has unique needs for their hosts and servers. This brings in the need for distributed firewall architectures, which allow for a border firewall and localized firewalls, leading to a consistent filtering implementation in a distributed way. They also provide defense in depth, resulting in more secure networks. In large organizations, managing and synchronizing the security policies of all the firewalls requires an efficient network management tool. An open source distributed management framework that can be easily customized and expanded can address the distributed firewall management needs. A directory server is an effective backbone for this framework. It will facilitate the distributed delegation, policy-based modeling, and cooperative management necessary to enable the next generation of secure and intelligent enterprise and campus networks.

This report presents a distributed firewall architecture for large networks. It also describes the implementation of a Directory Enabled policy management system, which is a generic model for various types of firewalls (both commercial and non-commercial). This report can be used as a prototype for designing the security architecture of an organization's network. This claim is substantiated by applying the scalable, high performance directory enabled distributed firewall architecture to the network of the University of Kansas.

# Contents

## List of figures

## List of Tables

# Chapter 1

# Introduction: Motivation and Goals

## 1.1 Motivation

Our increasing use of and reliance on computers and networks has resulted in a greater sensitivity to various security related issues. As a consequence we are also more concerned about taking measures to protect these systems against compromise and threats, especially since the Internet has significantly increased exposure to those who might wish to do them harm. One technology that has generated much interest in this regard is the firewall. This security tool is now widely in use by most organizations.

Monolithic, or single firewall implementations in an enterprise or university environment are impractical for many reasons: The enterprise may have several connections to the outside world; each department or subdivision has unique needs for their hosts and servers; and departments within the enterprise have to protected from each other. This brings in the need for a distributed firewall architecture. Distributed firewall architectures allow for multiple border firewalls and local department firewalls that allow for a consistent filtration model in a distributed way. They also provide defense in depth, resulting in secure networks.

But, in large organizations, managing and synchronizing the security policies of all the firewalls requires an efficient monolithic network management tool. An open source distributed management framework, that can be easily customized and expanded, can address the distributed firewall management needs. A directory server is an effective backbone for this framework. The directory service has excelled at providing an abstraction layer for applications to lookup information, authenticate, and authorize users while sharing information for wide-scale distributed deployment and information sharing for scientific applications.

This will facilitate the distributed delegation, policy-based modeling, and cooperative management necessary to enable the next generation of secure and intelligent enterprise and campus networks. Using proven middleware design techniques and principles for managing the delegation of authority, reliable dynamic configuration of the network topology and naming services become practical through a single interface. The details of the implementation of the network infrastructure are modeled into modular components that handle the system specific controls across heterogeneous equipment. The complexities are then hidden from the network administrators, allowing them to focus on departmental resource management and enabling collaborative efforts of their constituents and peer institutions. The resulting middleware enhanced network infrastructure maintains transparency to the host operating systems and users. Seamless management and policy enforcement of similar network devices from a variety of vendors can be accomplished by abstracting the capabilities from the device while keeping the management logic in a central location. Attempts at directory enabled networking (DEN) initiatives have had little success [1] because these middleware issues were not addressed, only the method and schema by which the network devices are modeled in a directory service were considered. A combination of a directory enabled network modeling with additional middleware services to facilitate distributed management and device integration logic will define the modular core of the open network middleware management system architecture.

## 1.2 Project Goals

The main goal of the project is to substantiate the claim that the use of the models shown in Figure 1.1 and Figure 1.2 will lead to an efficient and effective way of implementing security policies for a low cost distributed security architecture, and for network traffic accounting for the organization, with host-centric delegated configuration and security management.

### *1.2.1 Distributed security architecture:*

The first goal is to study the concept of a distributed security architecture for large enterprise networks. Parts of this study will be the evaluation of the performance of low cost (non-commercial) packet filters, and the effect of load balancing firewalls.

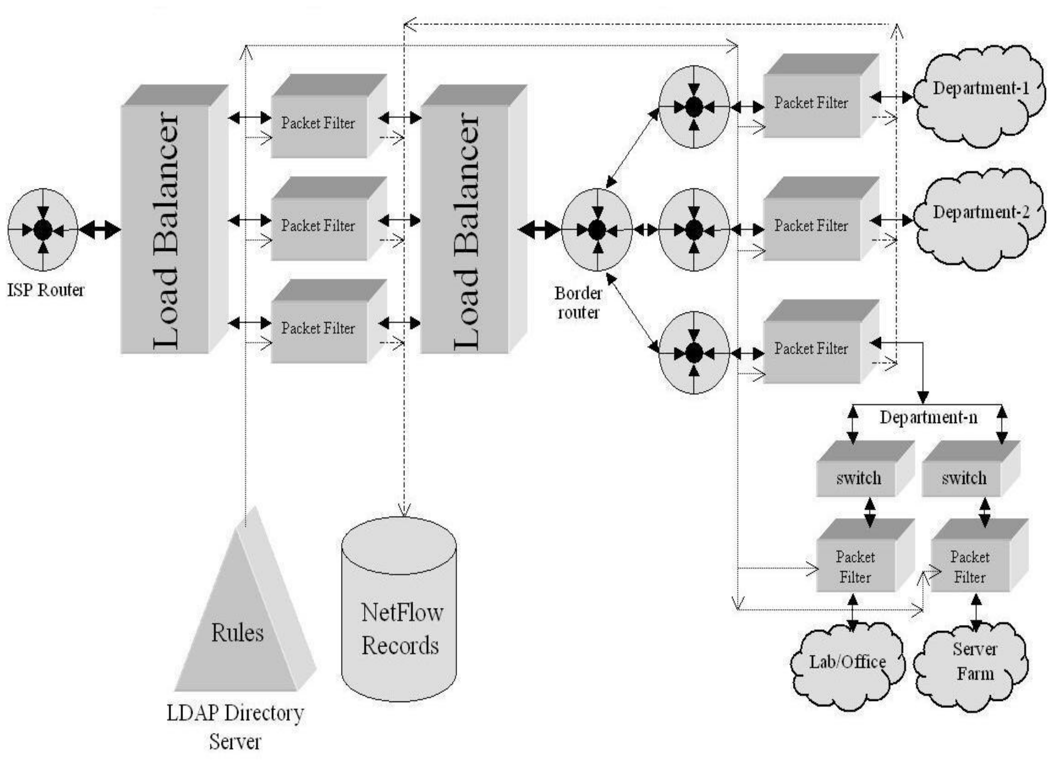


***Figure 1.1***: The proposed distributed security architecture

### *1.2.2 Directory Enabled firewall policy management system:*

The second goal is to develop a framework for managing security policies for the distributed security architecture, using an LDAP compliant directory server. We will also develop a framework for creating rules from the policies specified in the directory server, for all the firewalls in the organization.

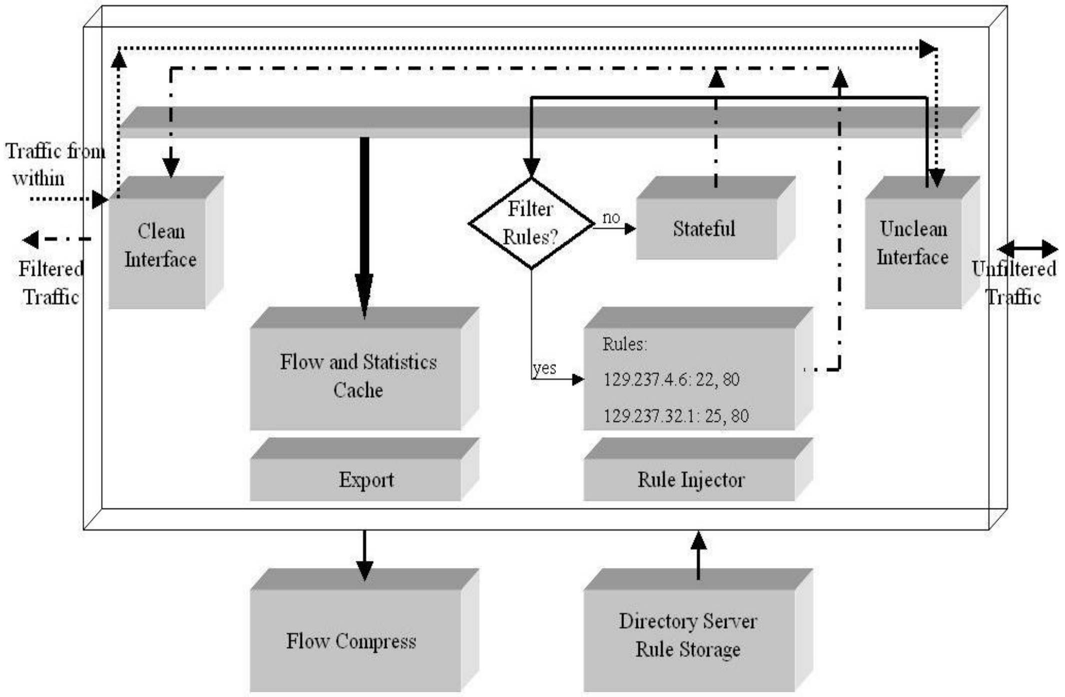


***Figure 1.2***: The Directory Enabled packet filter architecture

### *1.2.3 Validation of the above mentioned goals:*

Finally, we will apply the low-cost distributed security architecture, and the directory enabled policy management system to the network of the University of Kansas.

## 1.3 Document Layout

This document is organized into the following sections:

Chapter 2 gives some background information needed to understand this project. The next chapter discusses the need for a distributed security architecture. The chapter titled “distributed firewall architecture”, analyses the concepts that need to be considered before deploying a distributed security architecture in a large network. The issues discussed are the firewall location, the firewall deployment architecture, and the performance issues of different types of firewalls.

Chapter 5, which is “distributed firewall policy management”, presents a brief discussion about the issues involved with managing the policies for a distributed firewall architecture. The next chapter “Directory enabled policy management” provides a description of the framework developed for managing the security policies, and for migrating those policies into actual firewall rules.

Chapter 6, “Load Balancing for firewalls”, is a discussion of the load balancing issues related to firewalls, and some test results for non-commercial firewall load balancing.

The next chapter is a case study for the network of the University of Kansas. The aim is to relate the above topics to this network, and derive a suitable, low cost distributed security architecture. The last chapter is “Conclusion and Future work”, which discusses the lessons learned, and future work that can be done on the system.

# Chapter 2

# Background:

## 2.1 The Firewall

In a building, a firewall is designed to keep a fire from spreading from one part of the building to another. A network firewall serves a similar purpose: it prevents the threats of one network from spreading to another network. In practice, an Internet firewall is more like a gate of a medieval castle. It serves multiple purposes: It restricts people to entering at a carefully controlled point, it prevents attackers from getting close to your other defenses, and it restricts people to leaving at a carefully controlled point.

An Internet firewall is most often installed at the point where your internal network connects to the Internet, as shown in Figure 2.1 [2].

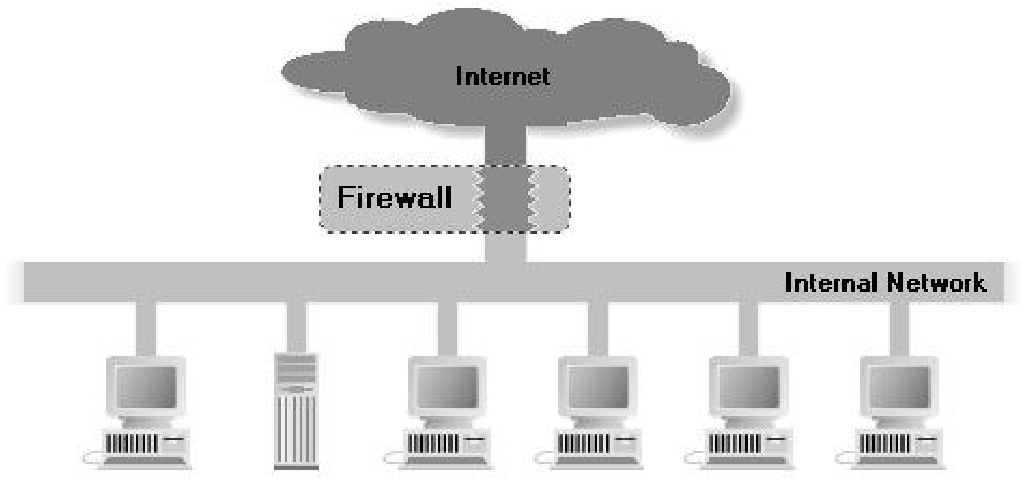


***Figure 2.1***: A firewall usually separates an internal network from the Internet

Logically, a firewall is a separator, restrictor, and an analyzer. The traffic coming from the Internet or going out from your internal network passes through the firewall. Because the traffic passes through it, the firewall has the opportunity to make sure that this traffic is acceptable. And this "acceptable traffic" is decided by the security policy of the site implementing the firewall.

### 2.1.1 *Firewall functions:*

Firewalls can do a lot for your site's security. In fact, some advantages of using firewalls extend even beyond security.

- *A firewall is a focus for security decisions*

  Think of a firewall as a choke point. All traffic in and out must pass through this single, narrow choke point, thus giving you an enormous amount of leverage for network security. Focusing your security at this single point is far more efficient than spreading security decisions and technologies around, trying to cover all the bases in a piecemeal fashion.

- *A firewall  enforces a security policy*

  Many of the services that people want from the Internet are inherently insecure. The firewall is the traffic cop for these services. It enforces the site's security policy, allowing only "approved" services to pass through. Also, depending on technologies you choose to implement your firewall, a firewall may have a greater or lesser ability to enforce either simple or complex security policies.

- *A firewall can log Internet activity efficiently*

  Because all traffic passes through the firewall, the firewall provides a good place to collect information about system and network use – and misuse. As a single point of access, the firewall can record what occurs between the protected network and the external network.

- *A firewall limits your exposure*

  Sometimes, a firewall will be used to keep one section of your site's network separate from another section. By doing this, you prevent the problems that impact one section of your network from spreading through the entire network, i.e., you are limiting the damage to only a part of your network.

### 2.1.2 *Firewall limitations:*

Firewalls offer excellent protection against network threats, but they aren't a complete security solution. Certain threats are outside the control of the firewall.

- *A firewall can't protect you against malicious insiders*

  If the attacker is already inside the firewall - if the fox is inside the henhouse - a firewall can do virtually nothing for you. Inside users can steal data and damage resources without ever coming near the firewall. Insider threats require internal security measures, which are beyond the scope of a firewall.

- *A firewall can't protect you against connections that don't go through it*

A firewall can effectively control the traffic that passes through it; however, there is nothing it can do about traffic that doesn't pass through it. For example, what if the site allows dial-in access to internal systems behind the firewall? The firewall has absolutely no way of preventing an intruder from getting in through such a modem.

- *A firewall can't protect against completely new threats*

  A firewall is designed to protect against known threats. A well-designed one may also protect against some new threats. However, no firewall can automatically defend against every new threat that arises. You can't set up a firewall once and expect it to protect you forever.

- *A firewall can't fully protect against viruses*

  Firewalls can't keep computer viruses out of a network. It's true that all firewalls scan incoming traffic to some degree, but detecting a virus in a random packet of data is very difficult. Moreover, what can a firewall do against other sources of viruses, like floppies? The most practical way to address the virus problem is through host-based virus protection software.

## 2.2 Firewall Types

Firewalls can filter at a number of levels in the network protocol stack. There are three main categories: packet filters, circuit gateways, and application gateways. These work at different layers of the protocol stack, but, they often get blurred.

Another category is the dynamic packet filter, which is a combination of the packet filter and circuit-level gateway. [3]

### 2.2.1 Packet Filters

Packet filters work by verifying packets based on their source or destination addresses or port numbers. Little or no context information is kept. Filtering may be done at the incoming interface, the outgoing interface, or both. It is easy to permit or deny access at the host or network level with a packet filter.

### 2.2.2 Circuit-level gateways

Circuit-level gateways work at the TCP level. TCP connections are relayed through a system that essentially acts as a wire. The relay system runs a program that copies bytes between two connections. When a client wishes to connect to a server, it connects to a relay host and possibly supplies connection information through a simple protocol. The relay host, in turn, connects to the server. The name and IP address of the client is usually not available to the server.

### 2.2.3 Application gateways

Application-level filters deal with the application level of the protocol stack. They check the details of the particular service they are responsible for. Rather than using a general-purpose mechanism to allow many different types of traffic to flow, special-purpose code can be used for each desired application.

### 2.2.4 Dynamic packet filters or Stateful packet filters

These are most commonly used type of firewalls. Dynamic packet filters are packet filters with the ability to capture connection semantics. The transit packets are examined for the usual criteria, like the addresses and port numbers. In addition to this, note is made of the identity of outgoing packets, and the incoming packets for the same connection are allowed to pass through. It is thus possible to handle UDP as well as TCP, despite UDP's lack of an ACK bit. Since these filters keep track of the state information, they are also called stateful packet filters, and simple packet filters are called stateless packet filters.

## 2.3 Directory Enabled Network (DEN) initiative

The Internet2 Middleware Initiative web page [17] identifies four core components: Identity, Authentication, Directories, and Authorization. It states:

> Directories are the operational lynchpin of almost all middleware services. They can contain critical customization information for people, processes, resources and groups. By placing such information in a common storage area, diverse applications from diverse locations can access a consistent and comprehensive source for current values of key data. In future information technology environments, directories will be among the most critical services offered.

The Directory Enabled Networking (DEN) initiative involves the industry-standard specification for constructing and storing information related to a network's users, applications, resources, and data in a central directory.

Objectives of a directory include providing central storage for information about people, groups, and resources; access by multiple processes, for multiple purposes; and the usual advantages of eliminating redundancy. These properties will help us define the policy management system for the distributed security architecture.

## 2.4 LDAP

With the growth of the number of different networks and applications, the number of specialized directories of information has also grown. This has resulted in islands of information that cannot be shared and are difficult to maintain.

The Lightweight Directory Access Protocol (LDAP) is an open industry standard that has evolved in an effort to maintain and access this dispersed information in a consistent and controlled manner. LDAP is gaining wide acceptance as the directory access method of the Internet and is therefore also becoming strategic within corporate intranets.

## 2.5 JNDI

The Java Naming and Directory Interface (JNDI) is an extension to the Java platform, provided by Sun Microsystems. According to Sun Microsystems, [28]

> The **Java Naming and Directory Interface** (**JNDI**) is a standard extension to the Java platform, providing Java technology-enabled applications with a unified interface to multiple naming and directory services in the enterprise. As part of the Java Enterprise API set, JNDI enables seamless connectivity to heterogeneous enterprise naming and directory services.

The JNDI is included in the Java 2 SDK, v1.3 and later releases. It is also available as a Java Standard Extension for use with the JDK 1.1 and the Java 2

SDK, v1.2. It extends the v1.1 and v1.2 platforms to provide naming and directory functionality.

To use the JNDI, one must have the JNDI classes and one or more service providers. The Java 2 SDK, v1.3 includes three service providers for the following naming/directory services: [5]

- Lightweight Directory Access Protocol (LDAP)
- Common Object Request Broker Architecture (CORBA), Common Object Services (COS) name service
- Java Remote Method Invocation (RMI) Registry

We used Java 2 SDK 1.4 for the rule generator applications, and it has been used with LDAP.

# Chapter 3

# The evolving security model

Conventional security architectures make use of a single point of protection for their network, such as the one shown in Figure 3.1.

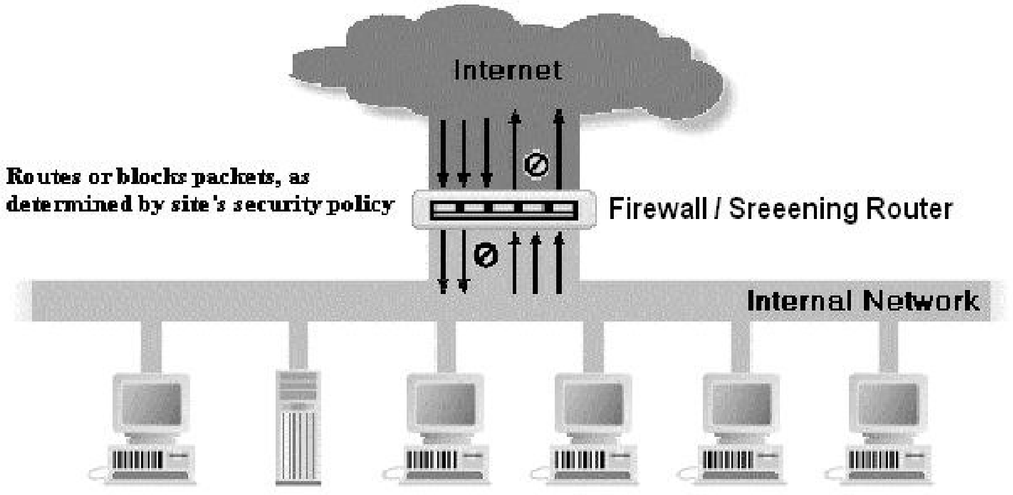

Internet
Routes or blocks packets, as determined by site's security policy
Firewall / Sreeening Router
Internal Network

***Figure 3.1***: Single firewall architecture [2]

The firewall is implemented at the boundary between the external network and the internal network. Such an architecture works for a small network, with a few hosts to be protected from the Internet, but it fails in large networks. Some of the problems faced by single-firewall architectures are:

a. **Insider threats:** In a large enterprise network, threats exist from within the internal network. The biggest threat to an organization's assets come from within.

Figures 3.2, 3.3 and 3.4 show statistical data from the CSI/FBI 2002 Computer Crime Survey. [6] The risk and loss from insider threats are extremely high. Perimeter firewalls, as in Figure 3.1, are not designed to solve the insider problem.

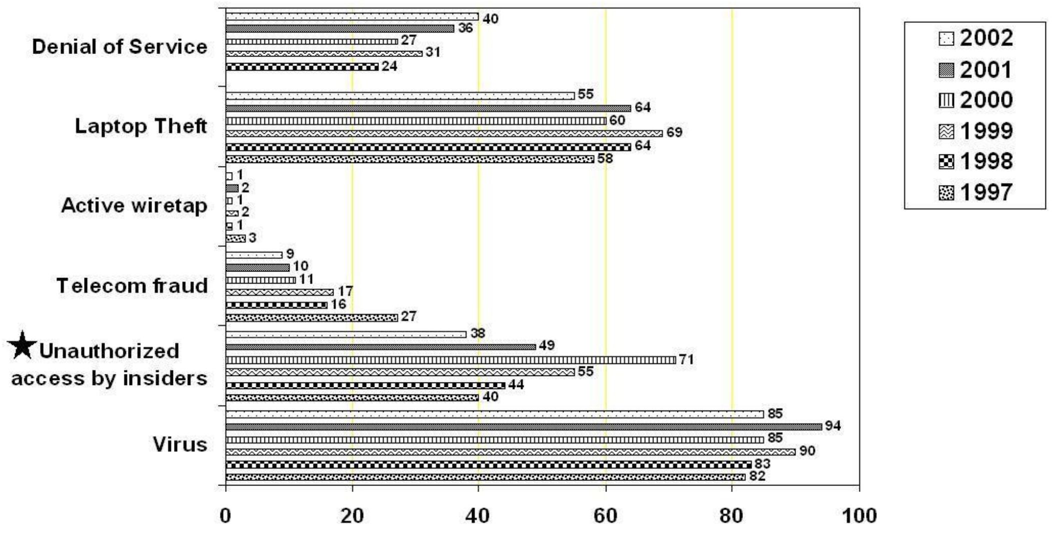


*Figure 3.2*: Observed Misuse and Attacks, part-1 [6]

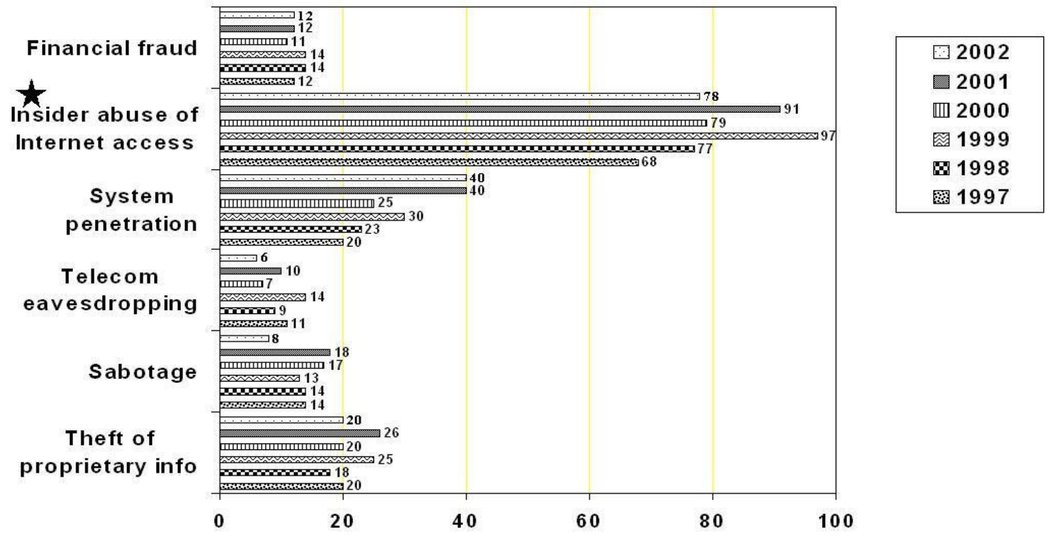


*Figure 3.3*: Observed Misuse and Attacks, part-2 [6]

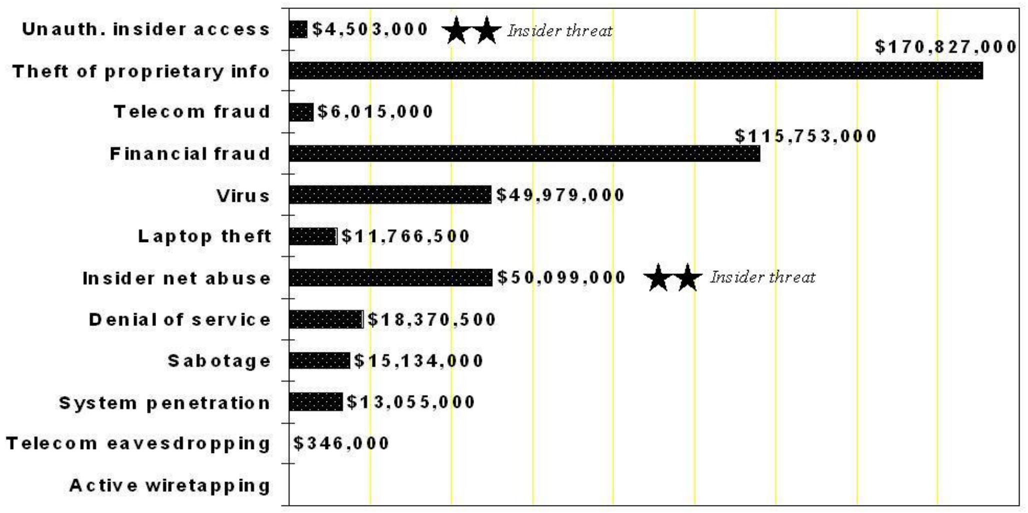


***Figure 3.4***: Reported Losses [6]

b. **Bandwidth bottleneck:** By installing a firewall as a network gateway, an access control policy can be enforced for every computer system that resides on the protected network. This means a huge rule set, and hence a low performance. The firewall will be a bottleneck to the network bandwidth.

c. **Low trust level:** It is possible to configure a firewall's rules to permit or restrict access to a single host, but these rules are applied at the network boundary. If malicious traffic is able to pass through the firewall, then all hosts on the network are vulnerable to this malicious traffic. Every organization survives on the trust that people have in its resources, and hence, low security results in a low trust level in the organization's network.

These problems can be overcome by using a distributed security architecture, wherein security can be provided at different levels in the network. A typical distributed security model is shown in Figure 3.5.

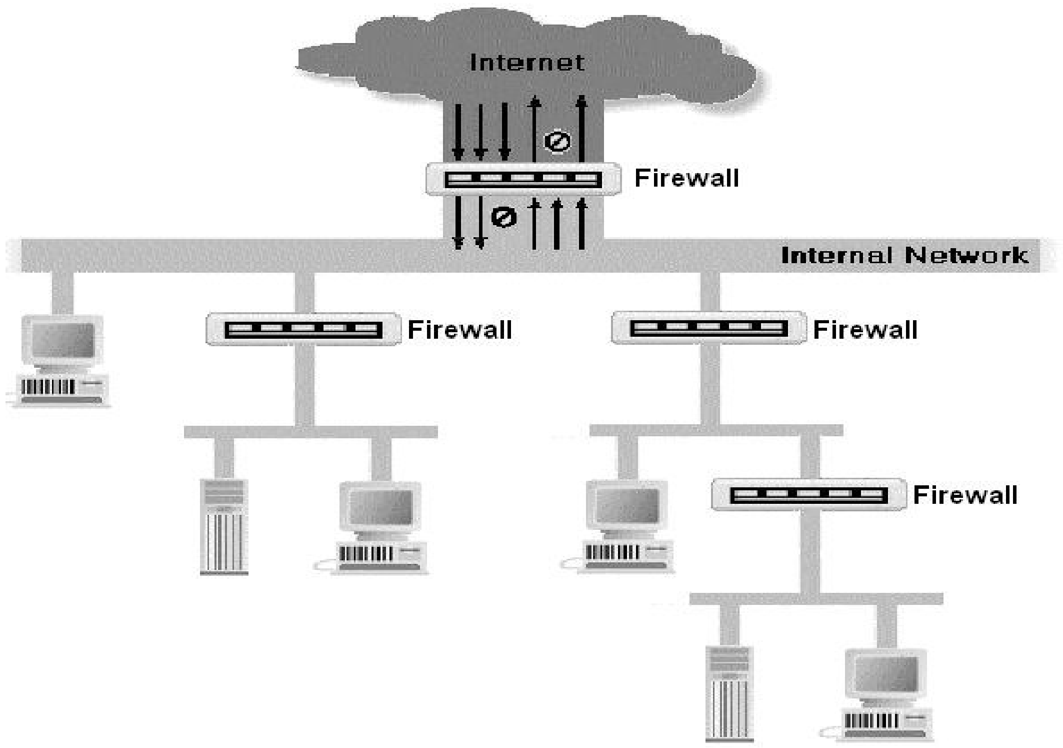


***Figure 3.5***: The distributed security architecture.

This model eliminates the drawbacks of the single-firewall architecture:

a. **Little or No Insider threats:** The aim of the distributed architecture is to divide the complete network into smaller networks (subnets), and secure each of those with a firewall. This way, every subnet is protected from threats arising in the other subnets. If every subnet is a single host, then the insider threats can be completely eliminated, as long as insecure ports are not open to attacks.

b. **Mitigation of the bandwidth bottleneck:** Since every subnet has its own firewall, the access control policies for that subnet are configured in that firewall, rather than the border firewall. This keeps the rules in the border firewall to the minimum. It is a widely accepted fact that the performance of a firewall is inversely proportional to the number of rules configured in the firewall, in the worst case where a packet is matched against all the rules. Hence, lesser number of rules in the border firewall will have little effect to its performance and traffic throughput.

c. **Higher trust level:** “Its easier to secure a studio apartment than a mansion”. [2] In terms of complexity, this also applies to a computer network. The smaller the protected network, the better the security to that network. This goes well with the concept of dividing the organization’s network into smaller subnets for a better security architecture, which ultimately leads to a higher trust in the network. The effect of complexity on security will be discussed in the next chapter.

# Chapter 4

# The Distributed Firewall Architecture:

One widely accepted property of a firewall is that it acts as a bandwidth bottleneck - all traffic to be controlled must pass through the firewall. Modern networks are also getting faster - on all levels, the bandwidth available has been increasing. As a result, the performance demanded from firewalls is increasing.

A distributed security architecture follows the tenet: *"It is easier to secure a studio apartment than a mansion"*. This is with respect to a single person taking care of the security for the apartment or mansion. [2] In a large organization, like a University, the whole network can be treated as a mansion. And the distributed architecture attempts to divide this mansion into studio apartments, namely subnets, each of them protected by a firewall. Figure 4.1 illustrates this architecture.

This architecture aims at providing security at different levels in the network, and at the same time, increasing the performance of the firewalls.

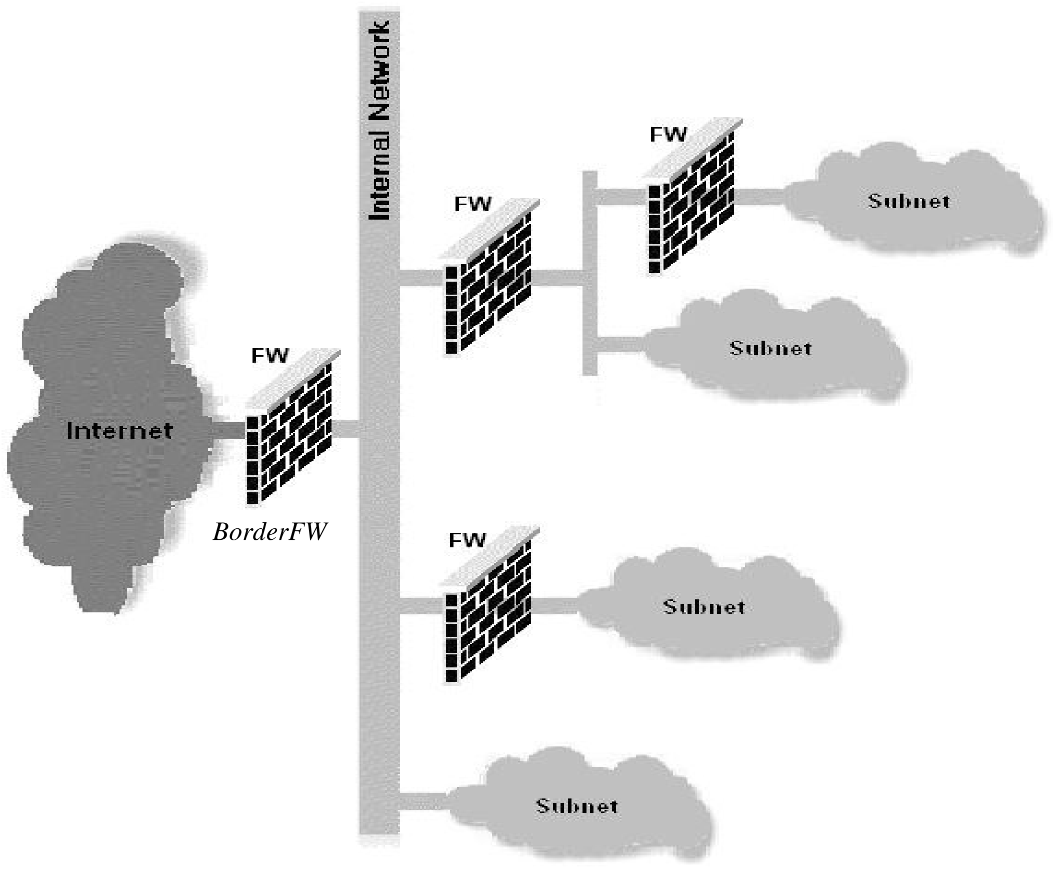


***Figure 4.1***: The distributed firewall architecture.

## 4.1 Security strategies provided by the distributed architecture:

Let us analyze the distributed security architecture with respect to some of the security strategies suggested by Elizabeth D. Zwicky et. al., in “Building Internet Firewalls” [2]:

a. Defense in Depth:

The trick to security is to make an attack attempt too risky or too expensive for the attackers. This can be done by adopting multiple mechanisms that provide backup and redundancy to each other. The distributed architecture caters to this strategy by placing firewalls at various locations in the network. For example, in Figure 4.1, even though the *BorderFW* fails, every subnet is protected by its respective firewall, assuming that the policies have been properly configured in the firewalls.

b. Choke Point:

A choke point forces attackers to use a narrow channel, which can be effectively monitored and controlled. In network security, every firewall acts as a choke point. The distributed architecture incorporates numerous choke points in the network. Each of these can be monitored and controlled, resulting in an effective and efficient security strategy.

c. Weakest Link:

A fundamental tenet of security is that a chain is only as strong as its weakest link and a wall is only as strong as its weakest point. Smart attackers are going to seek out that weak point and concentrate their attentions there. Consider a case where the weakest point is a host X. If there is only a single firewall (at the border), a small mistake in configuring the firewall will allow access to host X, and thus, to all the hosts in the network. On the other hand, in a distributed architecture, the hosts in other subnets are protected by their respective firewalls, thus reducing the risk of being affected by the weakest point.

d. Diversity of Defense:

This is the idea that a network needs not only multiple layers of defense, but different kinds of defense. A popular theory is to use different types of systems – for instance, using systems from different vendors. After all, if all the systems are the same, somebody who knows how to break into one of them probably knows how to break into all of them. The strategy, *diversity of defense*, can be easily applied in a distributed security architecture, since every firewall is independent of every other firewall. Figure 4.2 illustrates the application of this strategy to a large network. This strategy is one of the important considerations in this project.

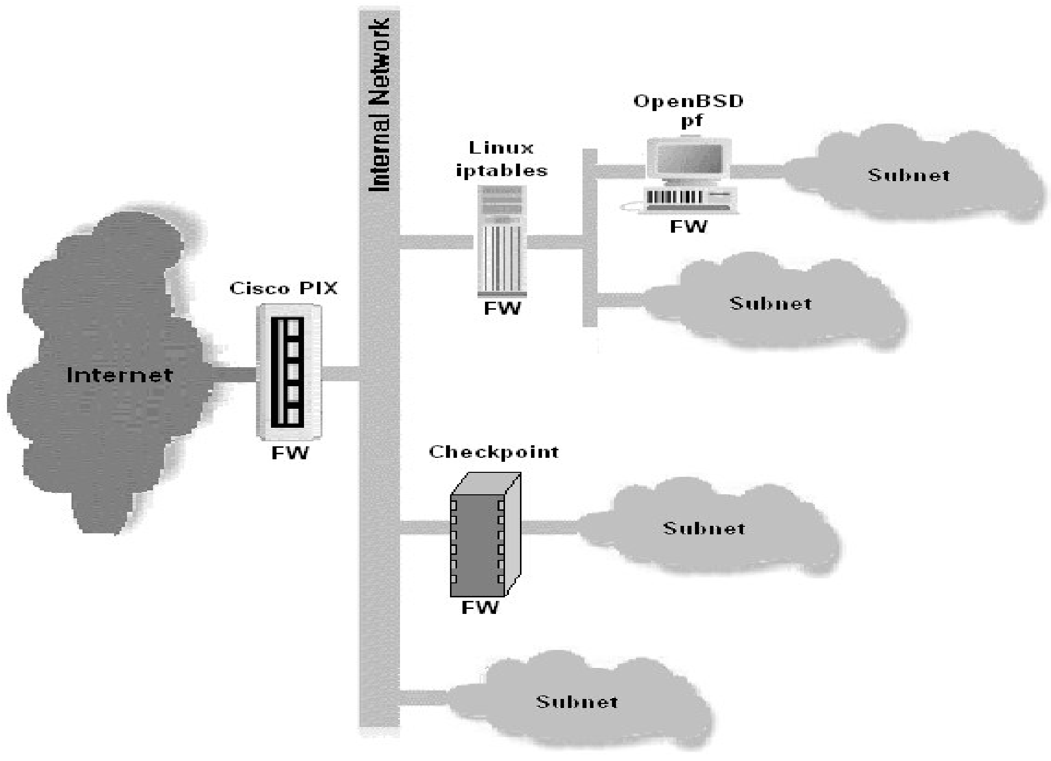


***Figure 4.2***: Diversity of defense.

e. Simplicity:

Simplicity is a security strategy for two reasons: First, keeping things simple makes them easier to understand; if you don't understand something, you can't really know whether or not it's secure. Second, complexity provides nooks and crannies for all sorts of things to hide in. Having a single firewall protect thousands of systems makes the firewall configurations complex, risking the network's security. The distributed architecture attempts to break up this firewall into smaller and simpler firewalls, putting this security strategy into effect. On the other hand, though every firewall's configuration is now simple, the large number of firewalls makes the distributed system complex. We will look at managing this complexity, without effecting security, in the next chapter "Distributed firewall policy management".

The other major advantages of the distributed security architecture are:

a. Scalability:

The distributed architecture scales well, with respect to the number of hosts and the number of rules, to the changing requirements of a network. The addition or removal of any host in the network would require changes only in the firewall protecting that host. In cases where more than one firewall needs to be configured, the directory enabled framework proposed in this report, will help us maintain consistency without affecting scalability. In single firewall architectures, all the changes need to be done in that single firewall, which becomes complex with network growth.

b. High Performance:

The performance of a firewall depends on the average number of rules each packet transits. But, in our discussion, we will consider the worst-case scenario for the firewalls. This scenario is one in which all packets traverse all rules. For example, at a given moment, there might be several connections going through the firewall. If one of these connections generated heavy traffic, and if this connection has a matching rule at the end of the rule list, then all the packets of this connection will be matched against all the rules, thus decreasing the performance of the firewall. Hence, large number of rules implies low performance in the worst-case scenario. With a single firewall, rules for all hosts need to be specified in that firewall, thereby drastically reducing its performance. The firewall becomes a bandwidth bottleneck. With a distributed architecture, the rules are distributed among all the firewalls. Every firewall has a limited number of rules to process, with little effect to both their performance and the network traffic bandwidth.

## 4.2 Firewall location

Now that we have decided upon a distributed security architecture, we need to think about the locations for the firewalls. A secure architecture is that, wherein security can be provided to the lowest level in the network hierarchy. This can be best achieved by pushing network security to the inner edge of the network.[7] The concept of the "edge" of the network could be classified as:

- A single host
- A group of hosts

It can be easily argued that pushing network security all the way to a host would be the most secure solution. This has also been proved by Markham et. al. **[7]** But, in a large network, like that of a University with thousands of hosts, such a security architecture would be practically impossible. Each host needs to be configured with its security policies, requiring extensive work by the network/system administrators.

The other option is to assume a small group of hosts as the network edge. In this case, the same set of policies can be easily applied to that group of hosts, thus simplifying the management, compared to the previous case. For example, consider a University with 50 departments. It is easier to have a firewall for each department. To increase the depth of defense, each department can be divided into sections like the administrative office, the laboratories, and conference rooms. Each of these sections can have an independent firewall (these sections are now considered the network edge).

## 4.3 Firewall deployment in an existing network

Modern networks have two sets of people working for the network: the network engineers and the security administrators. These two teams attempt to work together, but actually constrain each other. Network engineers provide routing of traffic, high availability and high bandwidth. They do this in part by deciding where within the topology to create links and install routers, firewalls, etc. Security administrators decide where within the network topology to install firewalls/filtering routers, and what filtering rules should be implemented in each of these intermediate network devices. Any changes in one area affects the other area. [7] One solution is to divide the complete network into two topologies:

- The network topology: which consists of hosts, servers, routers, etc., without any firewalls.
- The security topology: which denotes the firewall deployment architecture, i.e., the actual location of the firewalls with respect to the network topology.

In an existing network, the security topology should be implemented with minimal affect to the network topology. Let us analyze three different types of firewall deployment architectures, and select the one that satisfies our requirement of ***independent network and security topologies***.

### 4.3.1 The traditional architecture:

The traditional firewall deployment architecture consists of a dual-homed firewall device, a public subnet that is routable to the rest of the network and/or the network Internet uplinks. An internal private subnet uses a non-routed subnet to encapsulate the firewalled hosts. Traffic flow originates on the public subnet as a request to a firewalled host. The traffic is then passed through the firewall device's public interface, filtered, and is then passed to a private distribution switch. The distribution switch connects to a separate patch panel and ultimately to the firewalled host. Figure 4.3 illustrates this architecture.

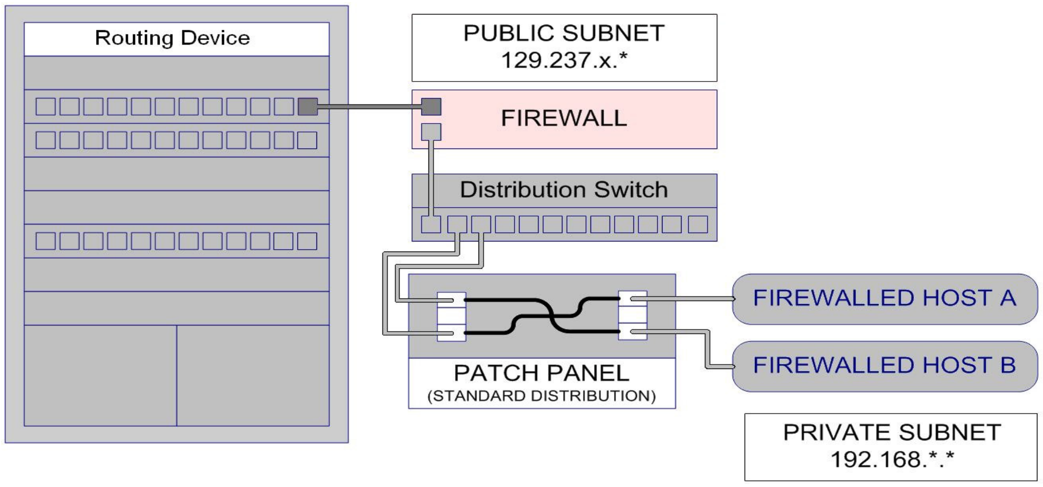


***Figure 4.3***: Traditional firewall deployment architecture.

The advantages of the traditional firewall deployment architecture are: Only a single connection to the network layer 3 (routing) device is needed to connect the firewall; and firewall management is simpler if the firewall is dedicated to a single set of servers with a single firewall manager. This however, does require additional firewalls.

On the other hand, the firewall is limited to the bandwidth capacity of the public and private interfaces on the firewall. It contains multiple single points of failure, including the distribution switches. Also, it does not scale beyond distribution switches without physical wiring. This deployment architecture requires dedicated switches for each firewall deployed, and the management of these distribution switches requires access through the firewall. The implementation of this architecture often requires an additional patch panel. Distribution switch and patch panel population may be sparse and underutilized.

### 4.3.2 The blade architecture:

The blade firewall deployment architecture consists of a router firewall module or "firewall blade", a public VLAN that is routable to the rest of the network and/or the network Internet uplinks. An internal private VLAN uses a non-routed subnet to encapsulate the firewalled hosts.

Traffic flow originates on the public VLAN as a request to a firewalled host. The traffic is then passed through the firewall blade, filtered, and is then passed to the private VLAN on the routing device for connection to the firewalled hosts.

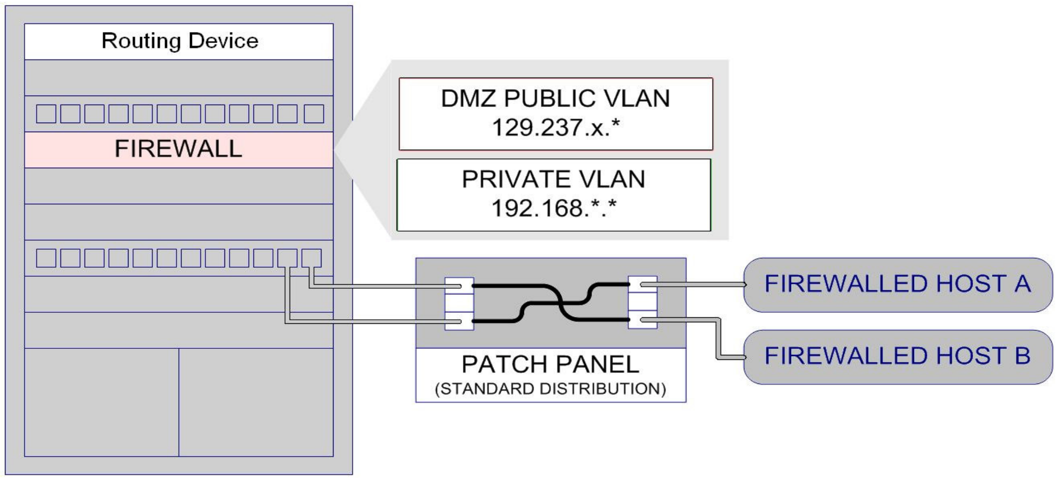


***Figure 4.4***: Blade firewall deployment architecture

The advantages of the blade firewall deployment architecture are: The firewall is located inside the router chassis and benefits from the redundancy of the router as well as the performance and high bandwidth of the router backplane. Also, existing network patch panels handle distribution to firewalled hosts that may reside anywhere on the network. Another advantage is that the utilization of public and private VLANs allows for additional router-level security mechanisms to be applied.

The disadvantages of this architecture are that: It requires a network layer 3 (routing) capable device that can host the firewall blade and VLANs. This architecture introduces management complexity when combining the firewall rules for multiple firewalled host groups. Also, it effects the physical perception: the presence of the firewall is not as physically prominent.

### 4.3.3 The loop-back architecture:

The loop-back firewall deployment architecture consists of a dual-homed firewall host, a public VLAN that is routable to the rest of the network and/or the network Internet uplinks. An internal private VLAN uses a non-routed subnet to encapsulate the firewalled hosts.

Traffic flow originates on the public VLAN as a request to a firewalled host. The traffic is then passed through the firewall device's public interface, filtered, and is then returned to the private VLAN on the routing device for connection to the firewalled hosts.

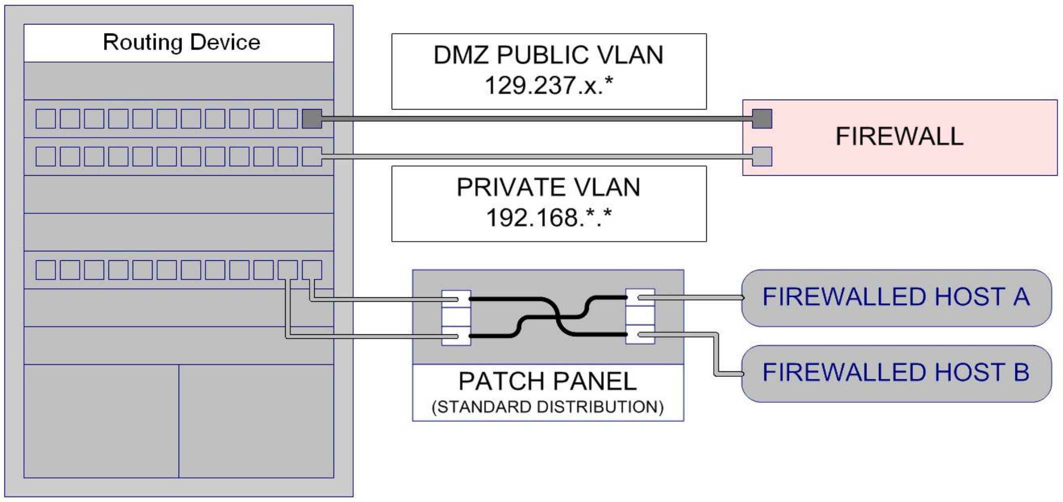


***Figure 4.5***: Loop-back firewall deployment architecture.

The advantages of the loop-back firewall deployment architecture are: The firewall can be physically located anywhere in the network: a secure location, network closet, or server rack. The existing network patch panels handle distribution to firewalled hosts that may reside anywhere on the network. Also, utilization of public and private VLANs allows for additional router-level security mechanisms to be applied.

The disadvantages of this architecture are: The firewall is limited to the bandwidth capacity of the public and private interfaces on the firewall. The firewall contains multiple single points of failure, but this drawback may be mitigated by the introduction of multiple network interfaces and/or a backup firewall unit. This architecture requires a network layer 3 (routing) capable device to handle packet routing between the public and private VLANs.

All the above firewall deployment architectures have been described using a public and a private subnet. But, the private subnet can be replaced by a public subnet. With such a setup, the loop-back firewall deployment architecture would be best suited to separate the network topology from the security topology. But this is not always the optimal solution at every point in the network. Every department might have its own limitations and could choose to implement any of the three architectures discussed above. Our suggestion is to use the loop-back architecture, or the blade architecture. The disadvantages of the loop-back architecture, specially the bandwidth constraints can be overcome by load balancing, as will be discussed in Chapter 7.

## 4.4 Firewalls that can be used

The next important decision that needs to be made is the type of firewall to be used in the security architecture. The major factors that affect this decision are:

- Performance / throughput
- Cost

The two broad categories to be considered are:

- Commercial firewalls: high throughput, high cost
- Non-commercial firewalls: low throughput, low cost

The decision needs to be made about what type of firewall is to be used at what level in the distributed security architecture.

For example, in a University network: The firewall at the border (between the Internet and the University network) needs to process large amounts of traffic, and a commercial firewall would be a good choice. In the same network, a firewall for the History department would not have to process huge amounts of traffic. For such implementations, a simple non-commercial firewall would be sufficient.

But one way of compensating for the low performance of non-commercial firewalls is to load balance them. This is discussed in detail in Chapter 7.

Also note that not all commercial firewalls have greater throughputs than non-commercial firewalls. For example, tests conducted by Samuel Patton, et. al. [8] show that Linux Kernel 2.2 ipchains had a better performance than the Cisco IOS firewall feature set.

### 4.4.1 The best Non-commercial firewall/packet filter

The “best” firewall, in this section, indicates the one with highest throughput. A commercial firewall is best suited where high bandwidth is needed. But let us look at the bandwidth that can be made available using non-commercial firewalls. The firewalls that will be considered are:

- Drawbridge, FreeBSD
- *ipchains*, Linux Kernels 2.2.x
- *iptables*, Linux Kernels 2.4.x
- OpenBSD packetfilter

#### 4.4.1.1 Drawbridge:

Drawbridge is a firewall package that was developed at Texas A&M University and was designed with a large academic environment in mind. It is a copyrighted, but freely distributable, bridging IP packet filter with a powerful filter language. It uses a constant-time table lookup algorithm so it can provide the same level of packet throughput regardless of the number of filters defined. Drawbridge is composed of three components: the Drawbridge filter code, the Drawbridge Manager, and the Drawbridge Filter Compiler. These three components run on a FreeBSD system where the filter code is compiled into the kernel and the manager and compiler are user level applications.

Drawbridge is different from any of the current standard firewall configurations. Using the categorization of firewalls developed by Ranum [9], drawbridge compares best to a filtering router firewall configuration as shown in Figure 4.6.

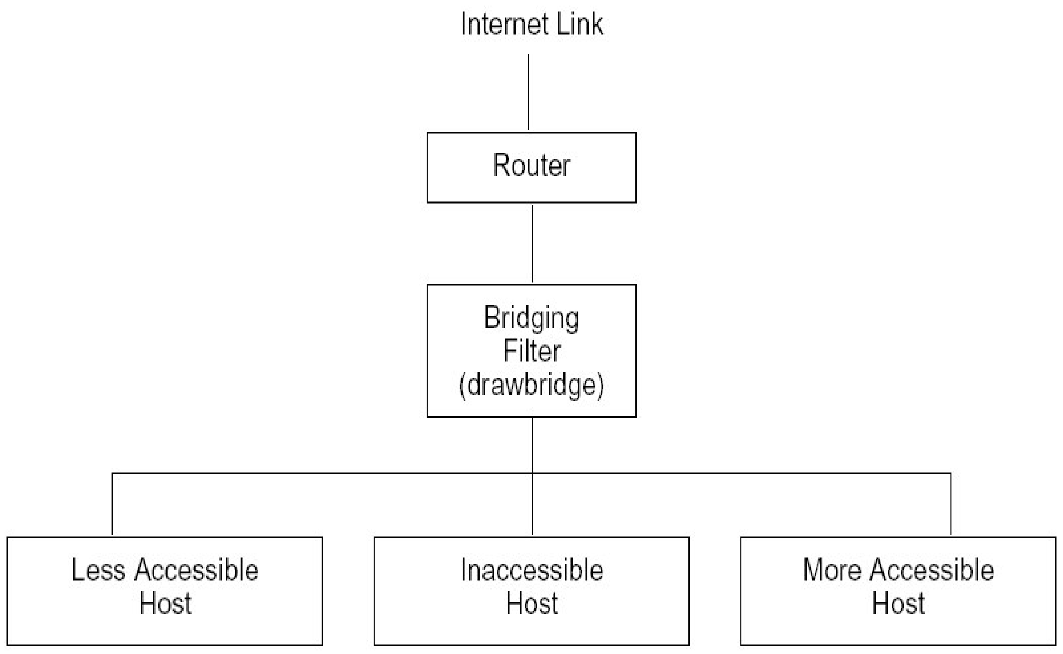


***Figure 4.6***: The Drawbridge configuration [10]

In a filtering router firewall, a router which has packet filtering support is used to filter packets to and from hosts on the “inside” of the router. This is used to establish a policy where hosts are provided more or less access depending on the decisions of the network managers.

A typical drawbridge firewall configuration is related to a filtering router firewall. The difference is that instead of using a filtering router as the firewall, the filtering function is moved from the router into drawbridge which acts as a bridging filter.

**Advantages of using drawbridge:**

- In conventional filtering routers, as filters are added, the performance begins to quickly drop due to how they implement the filtering rules. In drawbridge, arbitrary numbers of complex filters can be set up and the performance remains almost constant since simple look ups are performed and only connection establishment packets are filtered for TCP.
- A second problem with most filtering implementations is that testing filter configurations is difficult. Drawbridge remedies this by allowing the administrator to check the results of a compiled configuration file to see if the correct filtering rules have been applied. Since drawbridge is less algorithmic than current filtering implementations, it is sufficient to investigate the compiler output. The administrator can look at the class that a host has been assigned and at the filtering lists defined for each subtable in that class.
- Unlike other filtering implementations like *iptables*, the order in which the rules are specified in the configuration file does not matter. This simplifies the user's task of generating and maintaining the rules.

**Disadvantages of using drawbridge:**

- Low throughput: Drawbridge can provide a maximum throughput of 5.5Mbps, which is not sufficient for most of the modern networks.
- Drawbridge specifically defines an entire subtable to support TCP source port filtering, but UDP source port filtering is not currently supported.

- Only connection establishment packets are filtered for TCP.
- No further development on the drawbridge project.

**Performance of Drawbridge:**

Drawbridge was tested with a few filtering rules, but varying packet sizes. The test systems used: two Linux Kernel 2.4.20 boxes, and one FreeBSD 4.0 for filtration.

The performance of drawbridge for various packet sizes is as shown below (Figure 4.7):

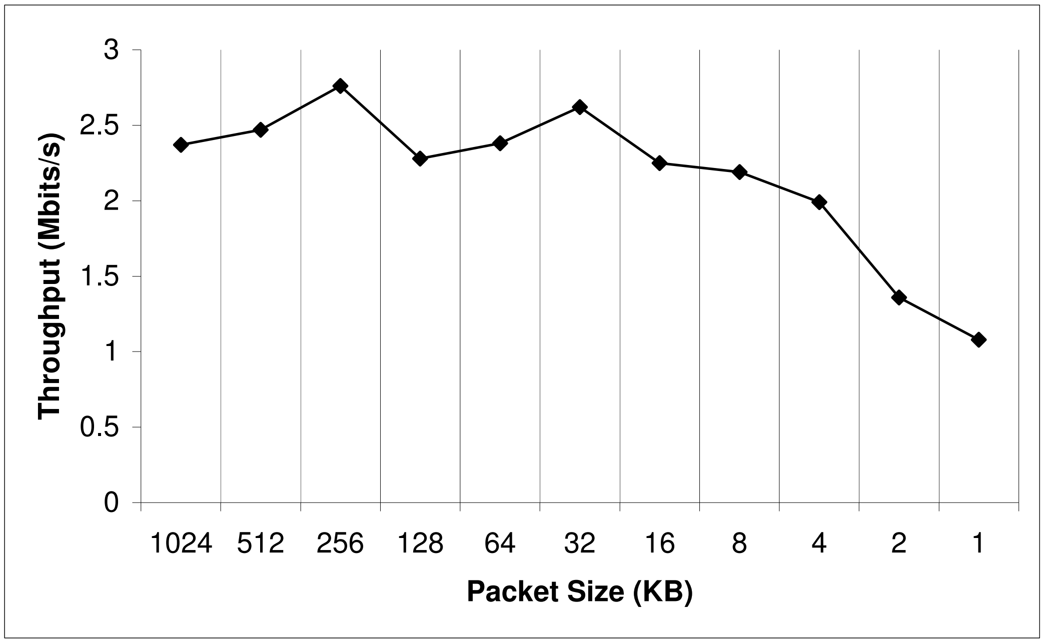


***Figure 4.7***: Performance of Drawbridge for different packet sizes.

#### 4.4.1.2 Linux and OpenBSD filtering packages

These were not tested because of already existing reports on their performance. One such report is "Packet Filters and their Behavior under high Network Load" [11]. An overview of the maximal TCP throughput test is shown in Figure 4.8.

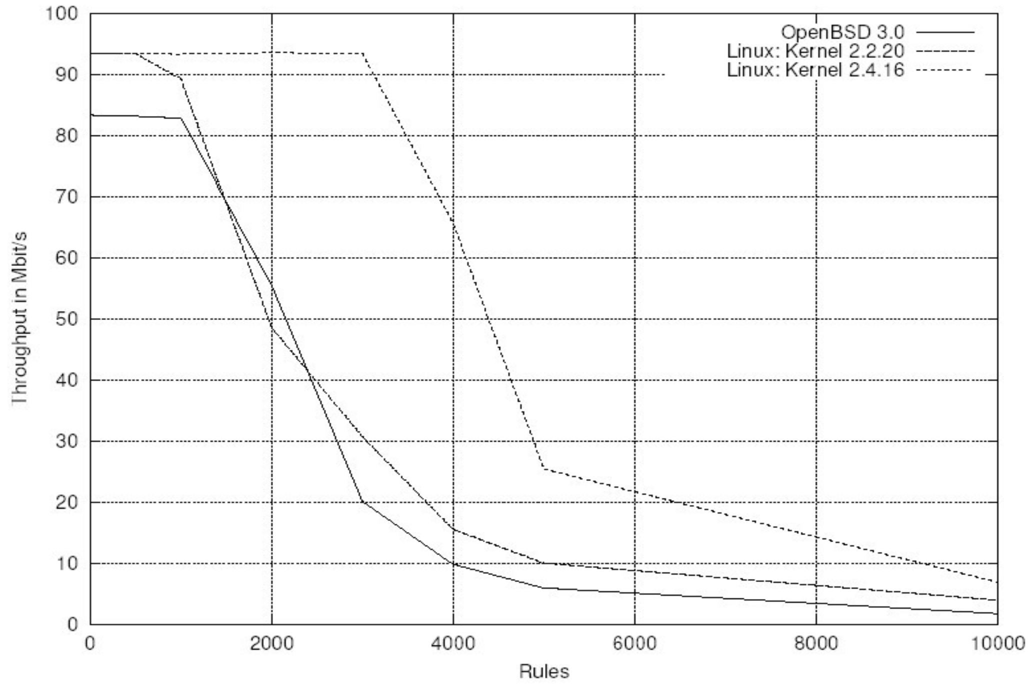


***Figure 4.8***: Performance of other non-commercial packet filters. [11]

The report also shows that the packet loss with Linux Kernel 2.4.16 was the least as compared to the other packet filters.

Looking at these performance results, *iptables* in Linux Kernels 2.4.x can be considered the "best" packet filter among the tested non-commercial firewalls.

# Chapter 5

# Distributed firewall policy management

Specifying and managing security policies for a firewall is not a trivial task. It requires extensive study of the needs of the entire network. Every aspect of the network needs to be considered, and worse than that, every need of every user needs to be considered. The best way to do this is to ask: to ask managers, system administrators, and users; then look at the actual computers and see what is going on. Let us look at the possible ways of defining security policies in a large enterprise network.

## 5.1 Who creates/manages the policies?

### 5.1.1 A central policy management committee

Consider a University network: ~60 departments, ~120 buildings, ~20000 hosts, ~35000 users. Deciding the security policies for such a network, including specific needs of each department and user, is not a task that can be easily accomplished by a small committee. The policies set for the History department will not work for the Computer Science department. Similarly, the policies for the office staff might not be what a computer lab needs. Framing such a vast and varied set of policies is not feasible for a small committee. Also, every department would like to be independent: it might not be happy with what the committee imposes upon them. Keeping everyone happy is an important tenet of security. People don't like learning or working in a hostile environment; and because they won't do it, the security administrators will either lose the security or lose the organization.

### 5.1.2 Individual Network Administrators

On the other hand, if every department has a network administrator, the task of creating policies becomes easier. The network administrator needs to worry only about a small group of systems. This is under the assumption that every network administrator is well versed with the concepts of security and firewalls. This decentralized model makes the framing and management of security policies easy, with every department getting what it needs. Also, one of the security strategies is *simplicity*. Managing all the firewalls is a complex task. This complex task can be divided into simpler tasks by allowing every network administrator to maintain the firewall(s) for that particular department.

In short, a central committee will not be able to properly carry out the tasks of creating security policies: because it cannot ask the 35000 users and managers about their needs. On the other hand, a network administrator who is in charge of a small department, can frame effective security policies by asking the few 100 or 200 users.

With the above factors, we suggest that individual network/security administrators decide and implement their security policies, whereas a central committee, who are believed to be security experts, help/advise the network/security administrators in deciding and configuring the security policies for their department. The above discussions are based on the assumption that ever department has a network/security administrator who is capable of maintaining and configuring firewalls.

## 5.2 How are the policies managed?

Now that we have each network administrator framing the security policies for the department, how will these policies be managed? Should this be left to the network administrator's discrepancy, or should there be a centralized policy management system?

We propose a centralized policy management system because of the following reasons:

i. In a distributed security architecture, the firewalls are interdependent in a hierarchical manner. For example, the border firewall should not block packets that are needed by a host in department X. In such a case, the traffic allowed by the firewall of department X will be blocked by the border firewall, thus displeasing the users in that department.

ii. A centralized management system creates a framework for maintaining the firewalls. Information like the old and new policies, the last time the firewall was updated, etc., can be maintained for every firewall in the network. Maintaining firewalls is an important security strategy, and the policy management system serves this purpose.

The policy management system: we propose a Directory Enabled policy management system, which is discussed in the next chapter.

# Chapter 6

# Directory Enabled Policy Management System

## 6.1 What is Directory Enabled, and why is it needed?

Directory Enabled describes a class of software drawing on a common data repository to unite the previously disparate functions of an enterprise information technology presence into a coherent, cooperating mirror of the business processes it is designed to aid. The heart and soul of this network design is the LDAP directory. It serves as the hippocampus of the system, maintaining a "memory" of business transactions by storing and cataloging successive transaction results for ready access on demand.

As with all high-quality products, the mechanisms which provide the de facto standard for network management have stood the test of time. These include such packages as the Internet Software Consortium's BIND and DHCP packages for name resolution and host configuration, and the *iptables* implementation famous from Linux and BSD. Over time, these solutions have been perfected to a point where deciding ***not*** to use them seems foolhardy. However, with age also comes a loss of flexibility. In the case of these software giants, their ancient designs have survived essentially unchanged in the face of a world full of evolving methodologies. In most cases the networks in which they operate are fundamentally different than the ones in which they matured. This inexorable march toward obsolescence has in some cases been slowed through the implementation of awkward, hacked-in appendages which do nothing to make the maintainer's life any easier, and often expose vulnerabilities, bugs, or degraded performance.

We intend to bridge the gap between tomorrow's service requirements and today's rock-solid but aging systems. Our approach is to embed all of the operational data required for network management into a single data source - the directory. After all, the separation of operational data for each problem domain is artificial, and in most cases, explains the difference between desired and actual system performance. The reason is simple: separation and overlap of operational data inevitably means synchronization of that data between any two systems which participate in the same service offering. Simply stated, synchronization consumes bandwidth unnecessarily, sapping your businesses ability to transact business.

This is where our architecture parts company with the background of mediocre solutions. The directory enabled nature of our architecture enables low-overhead integration of services without the need for a bandwidth-hungry synchronization bus. It provides centralized control of services into a single user interface for configuration and management. Finally, delegation of enterprise management to regional or departmental system administrators is made possible by the access control features of the directory itself.

In short, directory enabled software allows an enterprise to do everything it did before, only ***smarter***.

For a distributed security architecture, synchronization and coordination is utmost important. Directory enabled policy management serves this purpose.

## 6.2 LDAP schema for policy management:

The security policy management is carried out in two steps:

*Step 1:* Networked device registration:

In order to facilitate policy decisions for networked devices, the devices must first be registered within the system. The minimal registration requires a device owner and a MAC address. Security policies are then applied to the registered hosts or group of hosts within the system.

*Step 2:* Distributed firewall support:

This provides for specifying various security policies for a host or a group of hosts. Policies like which services a registered host is allowed to initiate or receive, what addresses the host can communicate with, etc., can be specified, and securely propagated to the appropriate firewall.

The first step is essential in any directory enabled network middleware management framework, and will not be discussed in this report.

We define two object classes specifically targeted at the distributed firewall support with LDAP:

- IPPacketFilterHost: describing the firewall/packet filter.
- IPPacketFilter: specifying the policies/rules for the protected network.

In our discussion, we use the term “protector” to denote a firewall or packet filter, simply because it protects the internal network.

***IPPacketFilterHost:***

This object class represents and describes the protector. The attributes are:

1. cn

   The commonName attribute.

2. insideInterfaceMACAddress

   The hardware address of the interface connected to the internal network (protected network).

3. outsideInterfaceMACAddress

   The hardware address of the interface connected to the external network.

4. protectedNetworkDN

   The DN of the network being protected by this protector.

5. sysadmin

   The system/network administrator who is responsible for this protector and the protected network.

6. insideInterfaceName

   The name of the interface connected to the internal network (protected network). This is used for creating rules using the interface name, e.g., eth0, eri1, de0, etc.

7. outsideInterfaceName

   The name of the interface connected to the external network. This is used for creating rules using the interface name, e.g., eth0, eri1, de0, etc.

The attributes described above are the *required* attributes. These are needed to describe and identify any type of protector, and for the management of that protector. The other attributes, which are not required, but add to the description of the protector, are:

8. insideInterfaceIPAddress

   The IP address of the interface connected to the internal network (protected network).

9. outsideInterfaceIPAddress

   The IP address of the interface connected to the external network.

10. typeBridging

    *true* indicates that the protector is a transparent bridging firewall, e.g., Drawbridge, OpenBSD pf. *false* indicates that it is a forwarding type of firewall.

11. typeForwarding

    *true* indicates that the protector is a forwarding firewall, e.g., Linux *iptables*. *false* indicates that it is a transparent bridging type of firewall.

If the type is not defined (neither typeBridging, nor typeForwarding), then the protector is assumed to be a forwarding type of firewall.

12. statefulFiltration

    *true* indicates that this is a stateful packet filter. *false* indicates that it is a stateless packet filter. The default, if this attribute is not defined, is stateless filtration.

***IPPacketFilter:***

The IPPacketFilter object class consists of attributes for specifying the policies for the protected networks. These attributes specify the TCP/UDP services to be allowed or denied, the ICMP types to be allowed or rejected, the trusted internal and external network addresses. These attributes are:

1. packetFiltrationLevel

   This attribute denotes the level of the firewall in the distributed firewall architecture. The architecture is considered to be hierarchical, and hence, the attribute specifies the placement of the firewall in the hierarchy. This will be discussed in detail in section 6.3.

2. packetFilterProtectedInternalIP

   This specifies the IP addresses within in the protected network.

3. packetFilterDefaultAllowAll

   *true* indicates that the default policy is to allow all packets.

4. packetFilterDefaultBlockAll

*true* indicates that the default policy is to drop all packets.

5. packetFilterAllowInboundICMPType

   This is a list of all the ICMP types that should be allowed to enter the internal network.

6. packetFilterAllowOutboundICMPType

   This is a list of all the ICMP types that should be allowed to leave the internal network.

7. packetFilterDenyInboundICMPType

   This is a list of all the ICMP types that should be denied entry into the internal network.

8. packetFilterDenyOutboundICMPType

   This is a list of all the ICMP types that should be denied exit from the internal network to the external network.

9. packetFilterAllowInternalTCPServices

   The list of TCP port numbers that should be allowed access to from the external network.

10. packetFilterAllowExternalTCPServices

    The list of TCP port numbers that should be allowed access to from the internal network.

11. packetFilterDenyInternalTCPServices

    The list of TCP port numbers that should be denied access to from the external network.

12. packetFilterDenyExternalTCPServices

    The list of TCP port numbers that should be denied access to from the internal network.

13. packetFilterAllowInternalUDPServices

    The list of UDP port numbers that should be allowed access to from the external network.

14. packetFilterAllowExternalUDPServices

    The list of UDP port numbers that should be allowed access to from the internal network.

15. packetFilterDenyInternalUDPServices

    The list of UDP port numbers that should be denied access to from the external network.

16. packetFilterDenyExternalUDPServices

    The list of UDP port numbers that should be denied access to from the internal network.

17. packetFilterAcceptInternalIP

    The list of acceptable IP addresses in the internal network. All outbound packets with the source as any of these IP addresses will be accepted.

18. packetFilterRejectInternalIP

    The list of unacceptable IP addresses in the internal network. All outbound packets with the source as any of these IP addresses will be rejected.

19. packetFilterAcceptExternalIP

    The list of acceptable IP addresses in the external network. All inbound packets with the source as any of these IP addresses will be accepted.

20. packetFilterRejectExternalIP

    The list of unacceptable IP addresses in the external network. All inbound packets with the source as any of these IP addresses will be rejected

The LDIF files for these object classes are shown in Appendix I: IPPacketFilterHost.ldif and IPPacketFilter.ldif. The attributes described are only some of the various features that are possible using the directory enabled system. More of these can be added when required. Some such attributes can be the specifications for log files, compatibility with various protocols, etc.

## 6.3 Applying the LDAP schema to the distributed firewall architecture

This section describes the application of the packet filtration schema, i.e., the IPPacketFilterHost and IPPacketFilter to the distributed firewall architecture. Figure 6.1 shows a sample network. The LDAP schema will be applied to this network.

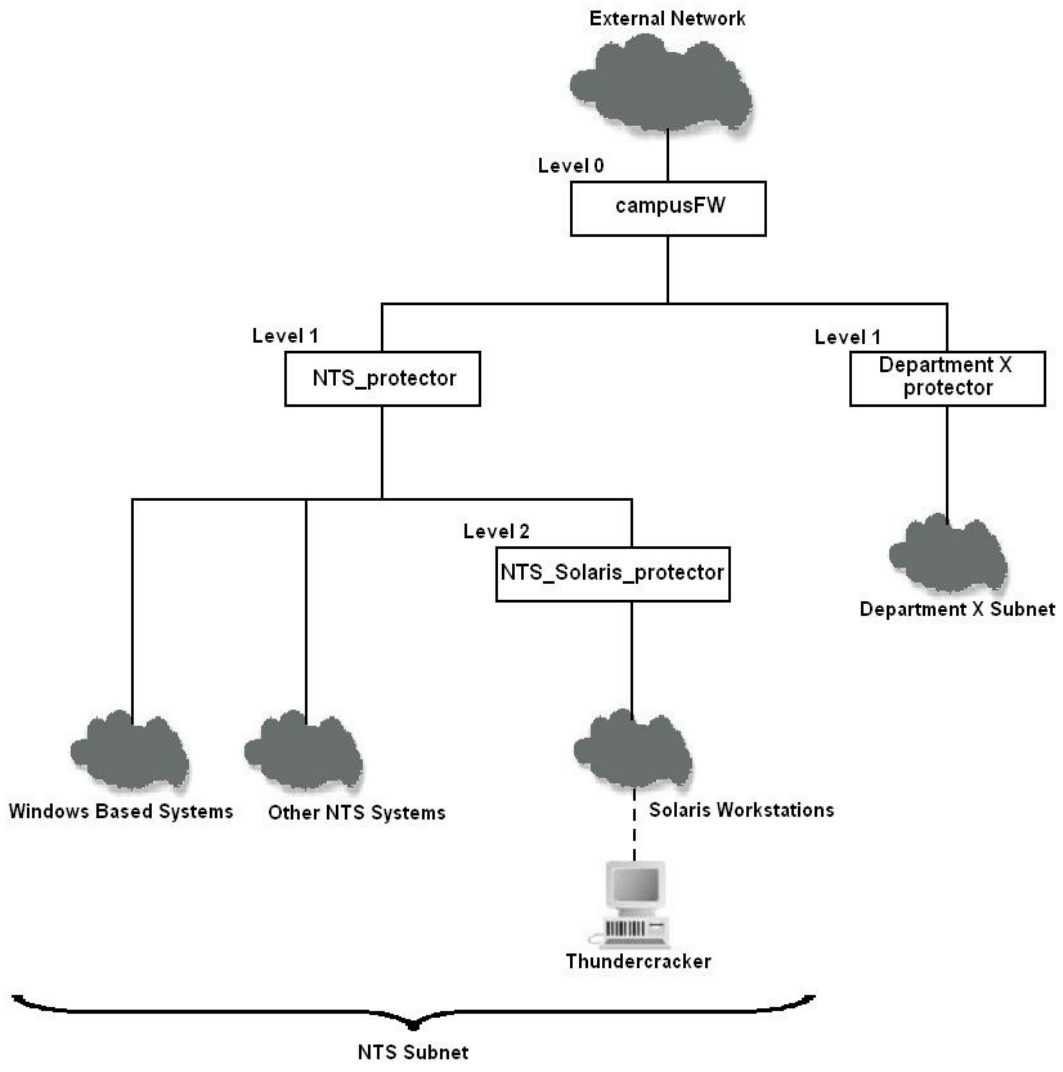


***Figure 6.1***: A network with a distributed firewall architecture

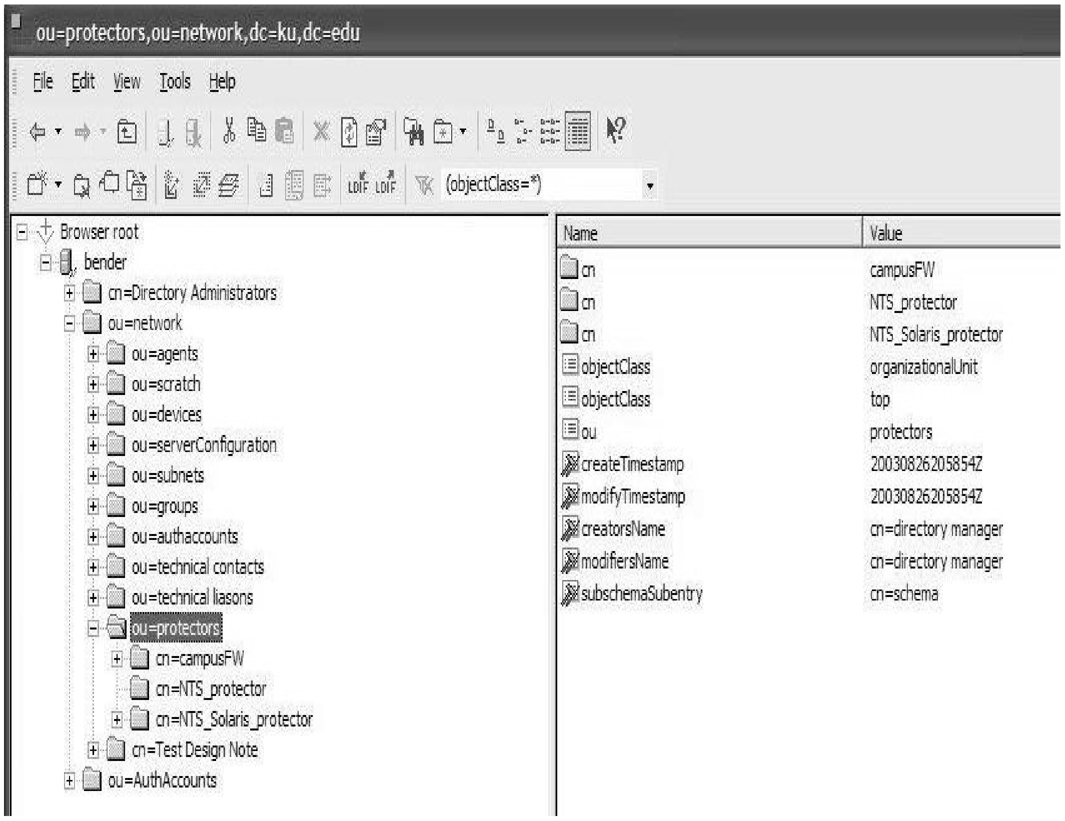
The four firewalls are:

- campusFW: the firewall for the campus, at the border.
- NTS_protector: the firewall for the NTS department.
- NTS_Solaris_protector: the firewall for the Solaris workstations in NTS.
- Department X protector: the firewall for Department X.

Each of these firewalls has a "level" in the hierarchy. The protector campusFW is at level 0, NTS_protector and Department X protector are at level 1, and NTS_Solaris_protector is at level 2. These levels are denoted by the *packetFiltrationLevel* attribute in the directory.

The NTS subnet is divided into three parts: the "Windows based systems", the "Solaris workstations", and "Other NTS systems".

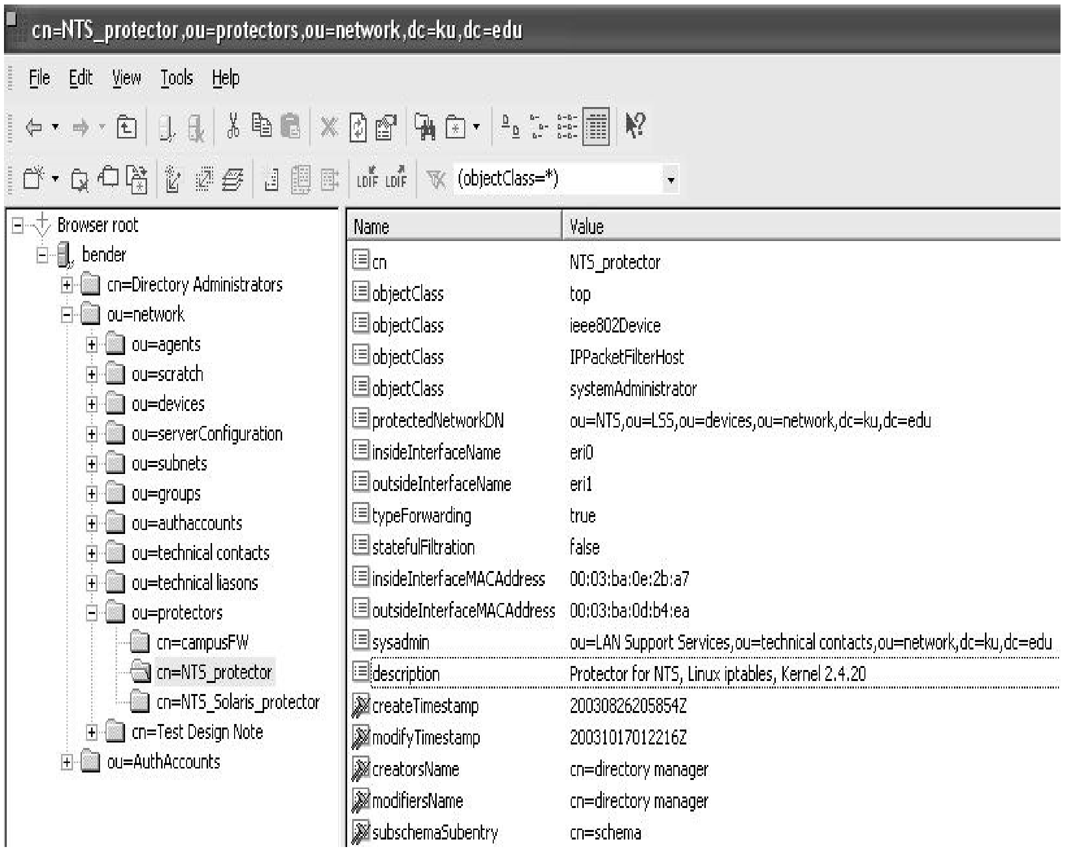
We create an entry for maintaining the information about all the protectors, as in Figure 6.2.



| Name | Value |
|---|---|
| cn | campusFW |
| cn | NTS_protector |
| cn | NTS_Solaris_protector |
| objectClass | organizationalUnit |
| objectClass | top |
| ou | protectors |
| createTimestamp | 20030826205854Z |
| modifyTimestamp | 20030826205854Z |
| creatorsName | cn=directory manager |
| modifiersName | cn=directory manager |
| subschemaSubentry | cn=schema |

***Figure 6.2***: directory entry for *ou=protectors*

Each of these protectors is described using the attributes of the IPPacketFilterHost object class. A description of NTS_protector is shown in Figure 6.3.



cn=NTS_protector,ou=protectors,ou=network,dc=ku,dc=edu

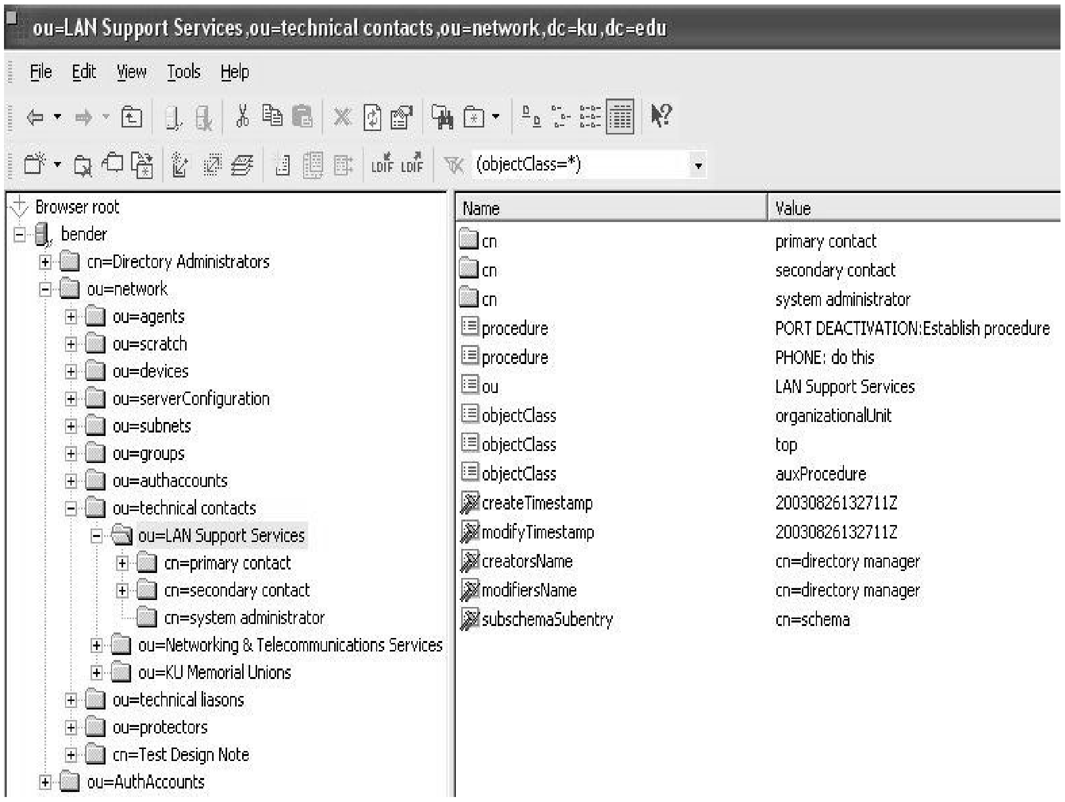
(objectClass=*)

```
Browser root
  bender
    cn=Directory Administrators
    ou=network
      ou=agents
      ou=scratch
      ou=devices
      ou=serverConfiguration
      ou=subnets
      ou=groups
      ou=authaccounts
      ou=technical contacts
      ou=technical liasons
      ou=protectors
        cn=campusFW
        cn=NTS_protector
        cn=NTS_Solaris_protector
      cn=Test Design Note
    ou=AuthAccounts
```

| Name | Value |
|---|---|
| cn | NTS_protector |
| objectClass | top |
| objectClass | ieee802Device |
| objectClass | IPPacketFilterHost |
| objectClass | systemAdministrator |
| protectedNetworkDN | ou=NTS,ou=LSS,ou=devices,ou=network,dc=ku,dc=edu |
| insideInterfaceName | eri0 |
| outsideInterfaceName | eri1 |
| typeForwarding | true |
| statefulFiltration | false |
| insideInterfaceMACAddress | 00:03:ba:0e:2b:a7 |
| outsideInterfaceMACAddress | 00:03:ba:0d:b4:ea |
| sysadmin | ou=LAN Support Services,ou=technical contacts,ou=network,dc=ku,dc=edu |
| description | Protector for NTS, Linux iptables, Kernel 2.4.20 |
| createTimestamp | 20030826205854Z |
| modifyTimestamp | 20031017012216Z |
| creatorsName | cn=directory manager |
| modifiersName | cn=directory manager |
| subschemaSubentry | cn=schema |

***Figure 6.3***: directory entry for *cn=NTS_protector*

The *sysadmin* attribute points to another entry in the directory, containing the information about the system/network administrator for the firewall and the protected network (Figure 6.4). In this case, the system administrator is the LAN Support Services group. The contacts for the system administrator are maintained in this entry.



ou=LAN Support Services,ou=technical contacts,ou=network,dc=ku,dc=edu

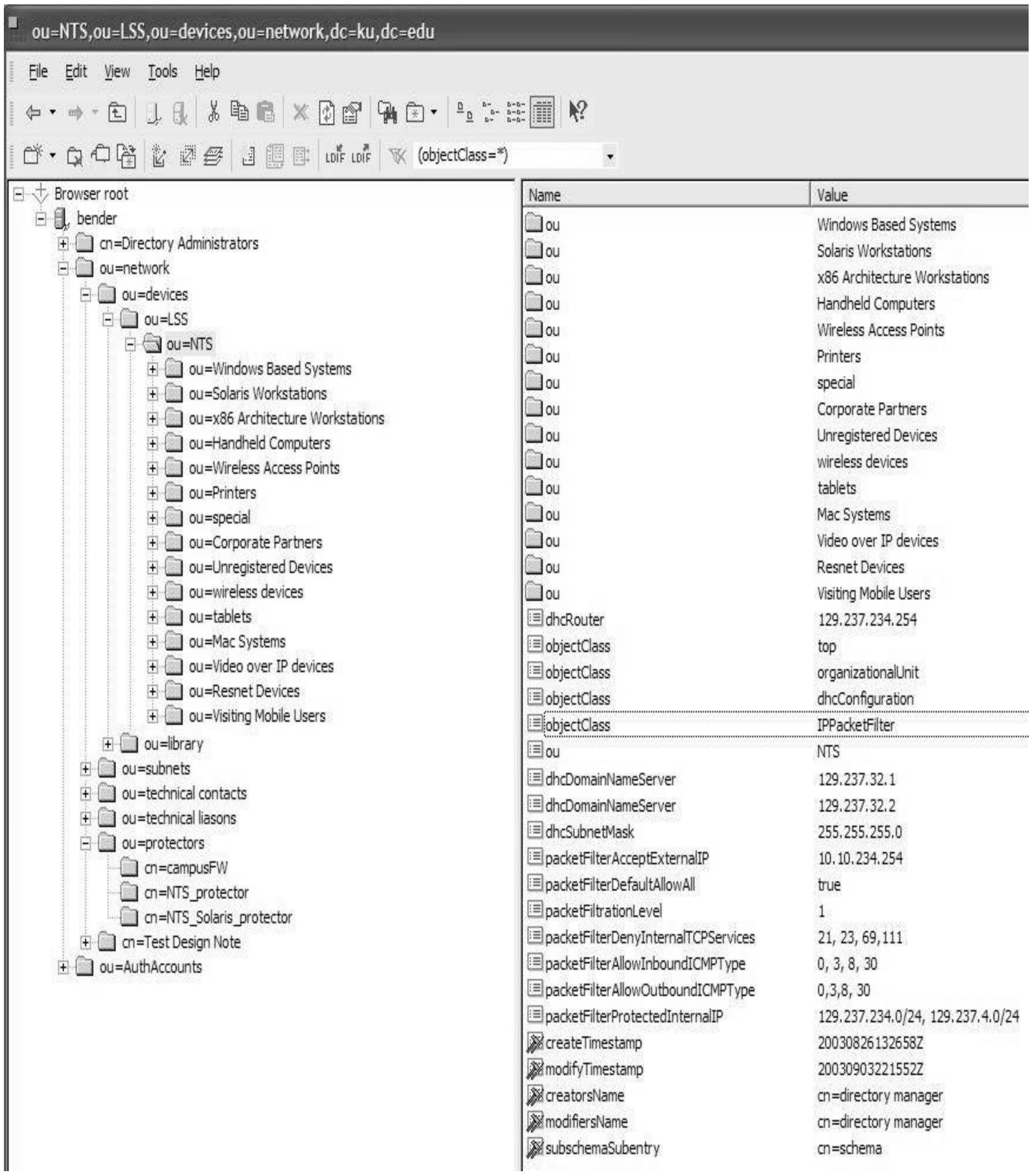
(objectClass=*)

```
Browser root
  bender
    cn=Directory Administrators
    ou=network
      ou=agents
      ou=scratch
      ou=devices
      ou=serverConfiguration
      ou=subnets
      ou=groups
      ou=authaccounts
      ou=technical contacts
        ou=LAN Support Services
          cn=primary contact
          cn=secondary contact
          cn=system administrator
        ou=Networking & Telecommunications Services
        ou=KU Memorial Unions
      ou=technical liasons
      ou=protectors
      cn=Test Design Note
    ou=AuthAccounts
```

| Name | Value |
|---|---|
| cn | primary contact |
| cn | secondary contact |
| cn | system administrator |
| procedure | PORT DEACTIVATION:Establish procedure |
| procedure | PHONE: do this |
| ou | LAN Support Services |
| objectClass | organizationalUnit |
| objectClass | top |
| objectClass | auxProcedure |
| createTimestamp | 20030826132711Z |
| modifyTimestamp | 20030826132711Z |
| creatorsName | cn=directory manager |
| modifiersName | cn=directory manager |
| subschemaSubentry | cn=schema |

***Figure 6.4***: directory entry for *ou=LAN Support Services*

Similarly, in Figure 6.3, *protectedNetworkDN* points to the protected network. This is yet another entry in the directory (Figure 6.5). This is where the policies for the firewall are specified. The object class IPPacketFilter is used in the description of the policies.



ou=NTS,ou=LSS,ou=devices,ou=network,dc=ku,dc=edu

```
Browser root
  bender
    cn=Directory Administrators
    ou=network
      ou=devices
        ou=LSS
          ou=NTS
            ou=Windows Based Systems
            ou=Solaris Workstations
            ou=x86 Architecture Workstations
            ou=Handheld Computers
            ou=Wireless Access Points
            ou=Printers
            ou=special
            ou=Corporate Partners
            ou=Unregistered Devices
            ou=wireless devices
            ou=tablets
            ou=Mac Systems
            ou=Video over IP devices
            ou=Resnet Devices
            ou=Visiting Mobile Users
        ou=library
      ou=subnets
      ou=technical contacts
      ou=technical liasons
      ou=protectors
        cn=campusFW
        cn=NTS_protector
        cn=NTS_Solaris_protector
      cn=Test Design Note
    ou=AuthAccounts
```

| Name | Value |
|---|---|
| ou | Windows Based Systems |
| ou | Solaris Workstations |
| ou | x86 Architecture Workstations |
| ou | Handheld Computers |
| ou | Wireless Access Points |
| ou | Printers |
| ou | special |
| ou | Corporate Partners |
| ou | Unregistered Devices |
| ou | wireless devices |
| ou | tablets |
| ou | Mac Systems |
| ou | Video over IP devices |
| ou | Resnet Devices |
| ou | Visiting Mobile Users |
| dhcRouter | 129.237.234.254 |
| objectClass | top |
| objectClass | organizationalUnit |
| objectClass | dhcConfiguration |
| objectClass | IPPacketFilter |
| ou | NTS |
| dhcDomainNameServer | 129.237.32.1 |
| dhcDomainNameServer | 129.237.32.2 |
| dhcSubnetMask | 255.255.255.0 |
| packetFilterAcceptExternalIP | 10.10.234.254 |
| packetFilterDefaultAllowAll | true |
| packetFiltrationLevel | 1 |
| packetFilterDenyInternalTCPServices | 21, 23, 69,111 |
| packetFilterAllowInboundICMPType | 0, 3, 8, 30 |
| packetFilterAllowOutboundICMPType | 0,3,8, 30 |
| packetFilterProtectedInternalIP | 129.237.234.0/24, 129.237.4.0/24 |
| createTimestamp | 20030826132658Z |
| modifyTimestamp | 20030903221552Z |
| creatorsName | cn=directory manager |
| modifiersName | cn=directory manager |
| subschemaSubentry | cn=schema |

***Figure 6.5***: directory entry for *ou=NTS*, which is protected by a firewall

As seen in Figure 6.5, the protector for this network is at level 1 in the firewall hierarchy (*packetFiltrationLevel*). The internal IP addresses protected by this firewall are the 129.237.234.0/24 and 129.237.4.0/24 subnets (*packetFilterProtectedInternalIP*). The default policy for this firewall is to allow all packets (*packetFilterAllowAll*). The IP address 10.10.234.254 is an acceptable external IP address (*packetFilterAcceptExternalIP*). Similarly, various other policies can be specified using the attributes of the object class IPPacketFilter.

Now let us look at how the policy specification can be extended to other systems in the hierarchy. The NTS subnet shown in Figure 6.1 will be the focus of discussion. This subnet is depicted in Figure 6.6.

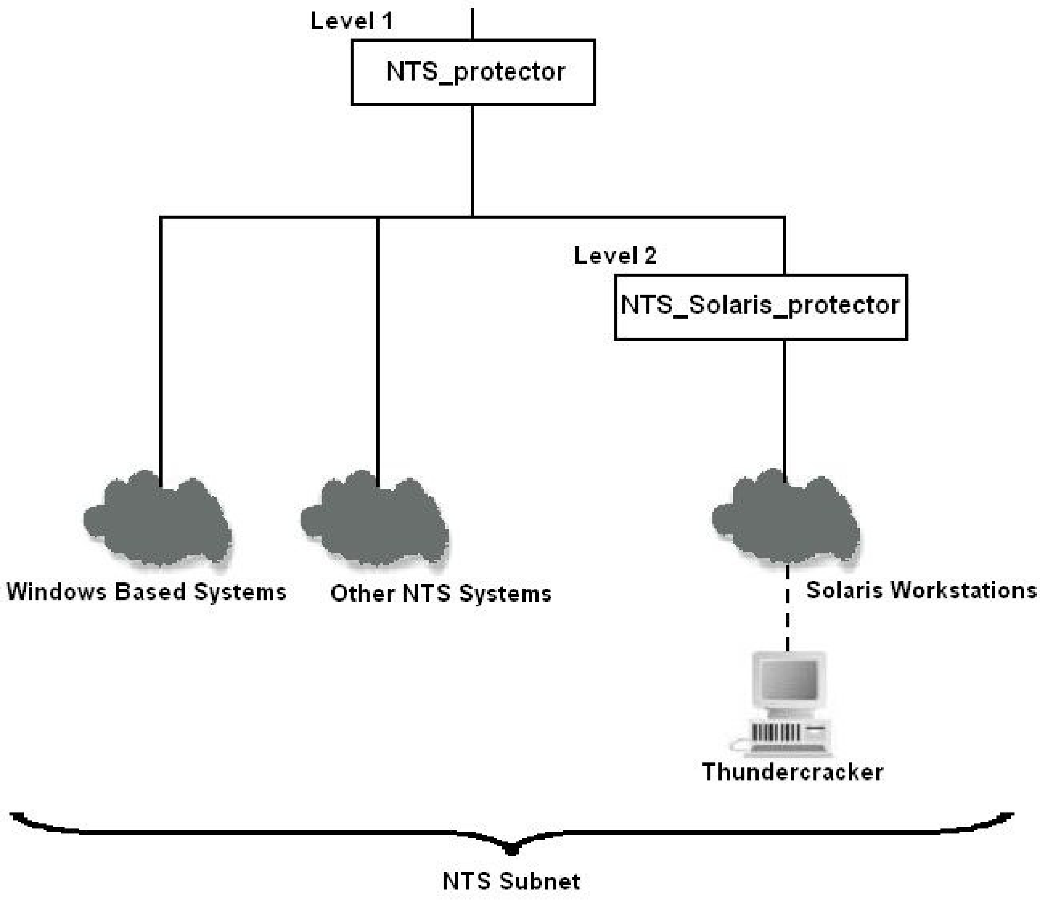


***Figure 6.6***: distributed firewall architecture for the NTS subnet.

The security policies follow a hierarchy as described below:

1. The policies in NTS_protector apply to all the systems in the NTS subnet.
2. In case a group of systems in the subnet require policies other than those specified for the NTS subnet, then the policies for that group of systems are specified as attributes for the entry corresponding to that group.
3. In case a host requires policies other than those in the higher levels of the hierarchy, then the policies for that system are specified as attributes for the entry corresponding to that system.
4. If a group of systems in that subnet have a separate firewall, then the policies for that group need not be specified in the firewall above it. But, the firewall above it must have policies such that the functioning of the firewall for this group is not hindered.
5. The policies specified at level = X in the directory override the policies specified at levels < X, as long as these entries are under the same protector.

In Figure 6.6, the policies specified for the host *Thundercracker*, override the policies specified for the group *Solaris Workstations*. The policies, if specified, for the groups *Windows Based Systems* and *Other NTS Systems*, will override the policies specified for *NTS Subnet*.

Consider the following scenario:

1. *Windows Based Systems* require policies other than those specified for *NTS Subnet*, whereas, *Other NTS Systems* do not have any special requirements.
2. *Solaris Workstations* need to be protected with an additional layer of security, and hence, another firewall *NTS_Solaris_protector*. But, one of those workstations, *Thundercracker*, requires additional or different access policies.

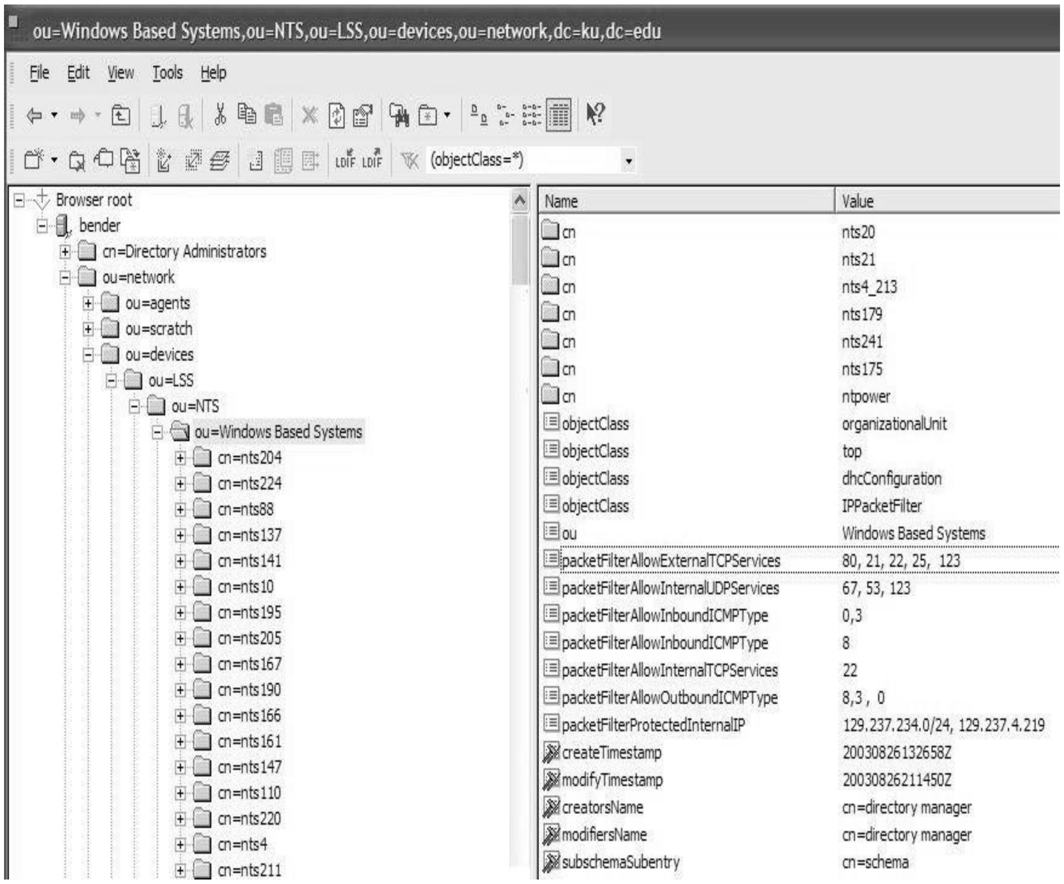
These requirements can be easily implemented and managed using the LDAP directory, as shown in the following figures:

We have already seen the policies for the *NTS Subnet* in Figure 6.5.

Figure 6.7 shows the specifications for *Windows Based Systems*, fulfilling our first requirement.

All the packets to or from the *Windows Based Systems* will first be verified using the rules specified here (Figure 6.7). In case none of these rules matches the packet, then the verification will continue with the rules specified for *NTS Subnet* (Figure 6.5). The attribute *packetFilterProtectedInternalIP* is used to identify the

IP addresses that will be subject to the policies of *Windows Based Systems*.



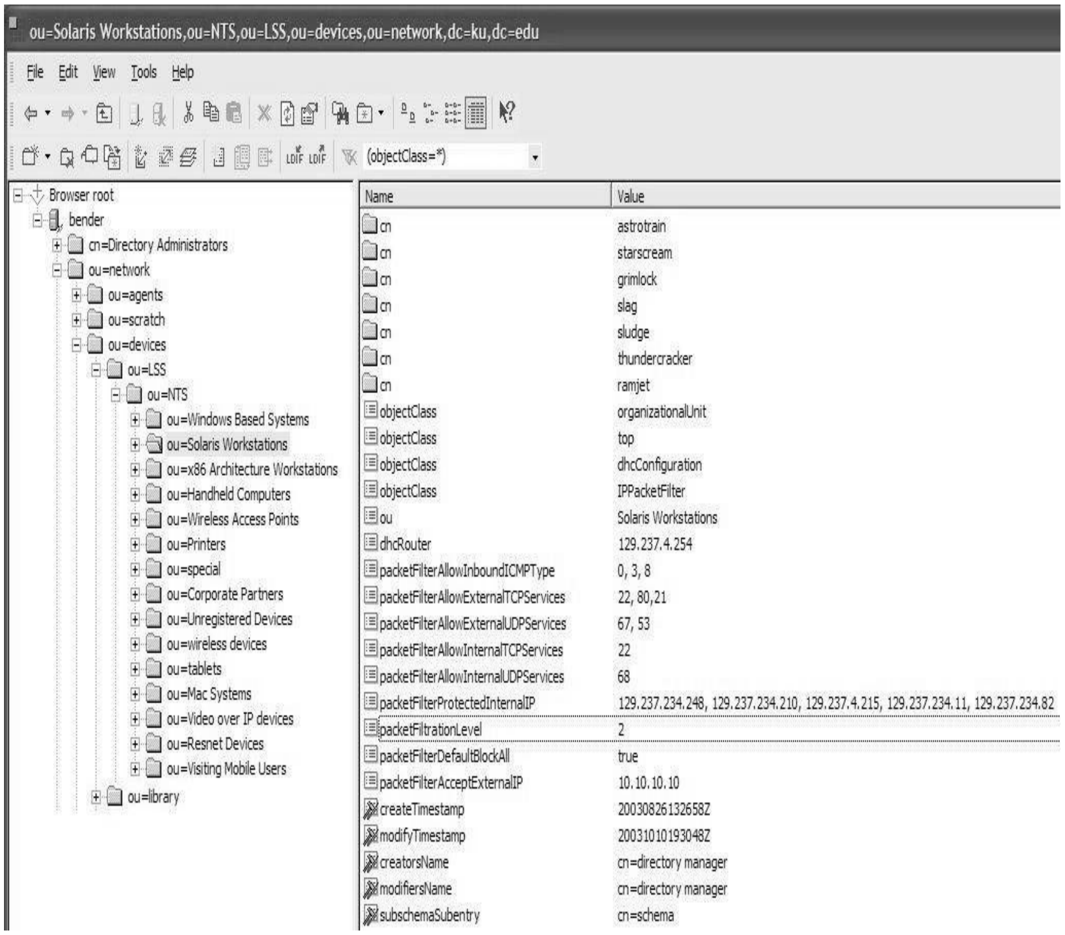
ou=Windows Based Systems,ou=NTS,ou=LSS,ou=devices,ou=network,dc=ku,dc=edu

```
Browser root
  bender
    cn=Directory Administrators
    ou=network
      ou=agents
      ou=scratch
      ou=devices
        ou=LSS
          ou=NTS
            ou=Windows Based Systems
              cn=nts204
              cn=nts224
              cn=nts88
              cn=nts137
              cn=nts141
              cn=nts10
              cn=nts195
              cn=nts205
              cn=nts167
              cn=nts190
              cn=nts166
              cn=nts161
              cn=nts147
              cn=nts110
              cn=nts220
              cn=nts4
              cn=nts211
```

| Name | Value |
|---|---|
| cn | nts20 |
| cn | nts21 |
| cn | nts4_213 |
| cn | nts179 |
| cn | nts241 |
| cn | nts175 |
| cn | ntpower |
| objectClass | organizationalUnit |
| objectClass | top |
| objectClass | dhcConfiguration |
| objectClass | IPPacketFilter |
| ou | Windows Based Systems |
| packetFilterAllowExternalTCPServices | 80, 21, 22, 25, 123 |
| packetFilterAllowInternalUDPServices | 67, 53, 123 |
| packetFilterAllowInboundICMPType | 0,3 |
| packetFilterAllowInboundICMPType | 8 |
| packetFilterAllowInternalTCPServices | 22 |
| packetFilterAllowOutboundICMPType | 8,3 , 0 |
| packetFilterProtectedInternalIP | 129.237.234.0/24, 129.237.4.219 |
| createTimestamp | 20030826132658Z |
| modifyTimestamp | 20030826211450Z |
| creatorsName | cn=directory manager |
| modifiersName | cn=directory manager |
| subschemaSubentry | cn=schema |

***Figure 6.7***: directory entry for *ou=Windows Based Systems* in the NTS subnet.

The *Solaris Workstations* are protected by another firewall *NTS_Solaris_protector*. The policies for the *Solaris Workstations* are shown in Figure 6.8.



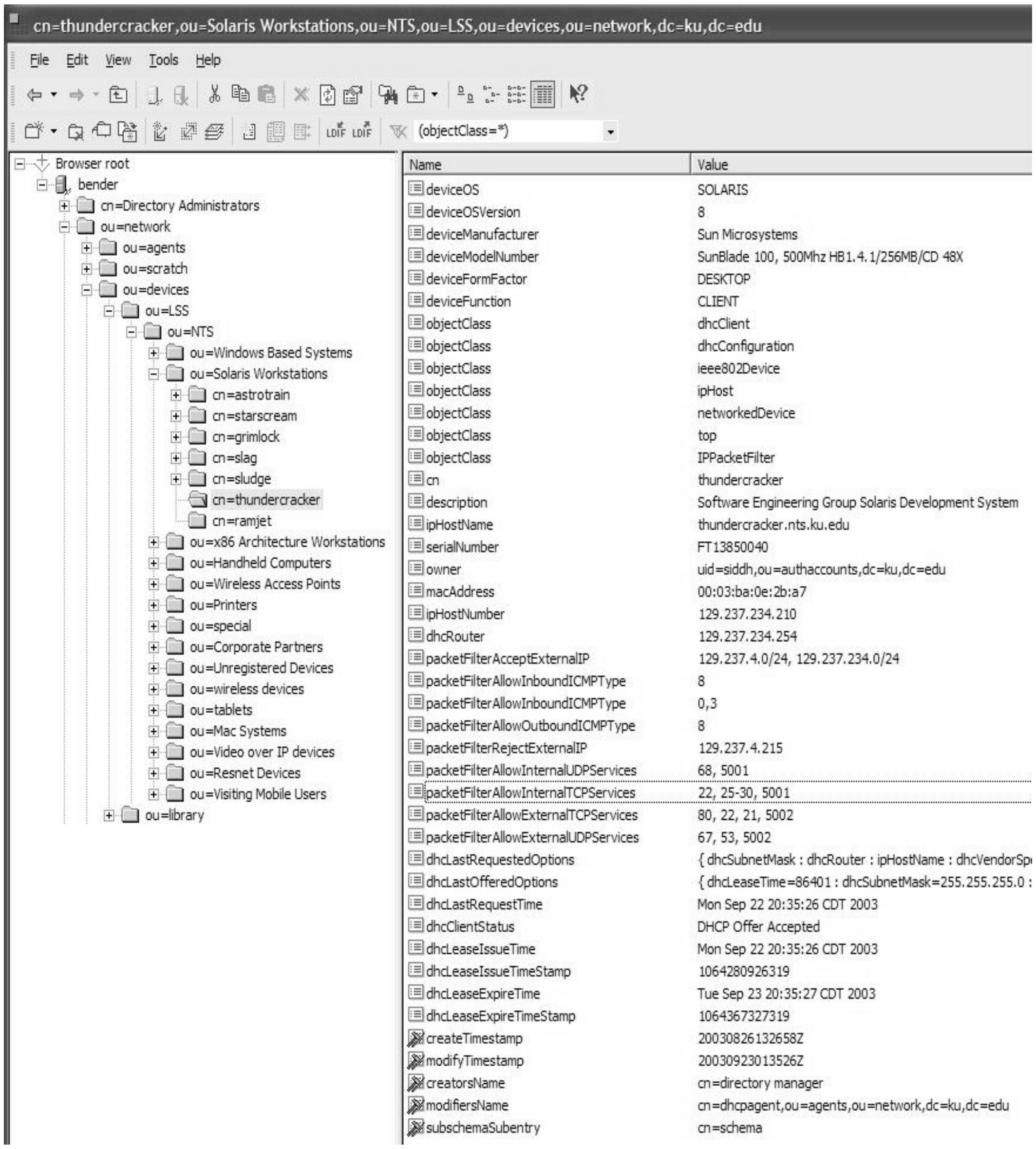
ou=Solaris Workstations,ou=NTS,ou=LSS,ou=devices,ou=network,dc=ku,dc=edu

```
Browser root
  bender
    cn=Directory Administrators
    ou=network
      ou=agents
      ou=scratch
      ou=devices
        ou=LSS
          ou=NTS
            ou=Windows Based Systems
            ou=Solaris Workstations
            ou=x86 Architecture Workstations
            ou=Handheld Computers
            ou=Wireless Access Points
            ou=Printers
            ou=special
            ou=Corporate Partners
            ou=Unregistered Devices
            ou=wireless devices
            ou=tablets
            ou=Mac Systems
            ou=Video over IP devices
            ou=Resnet Devices
            ou=Visiting Mobile Users
        ou=library
```

| Name | Value |
|---|---|
| cn | astrotrain |
| cn | starscream |
| cn | grimlock |
| cn | slag |
| cn | sludge |
| cn | thundercracker |
| cn | ramjet |
| objectClass | organizationalUnit |
| objectClass | top |
| objectClass | dhcConfiguration |
| objectClass | IPPacketFilter |
| ou | Solaris Workstations |
| dhcRouter | 129.237.4.254 |
| packetFilterAllowInboundICMPType | 0, 3, 8 |
| packetFilterAllowExternalTCPServices | 22, 80,21 |
| packetFilterAllowExternalUDPServices | 67, 53 |
| packetFilterAllowInternalTCPServices | 22 |
| packetFilterAllowInternalUDPServices | 68 |
| packetFilterProtectedInternalIP | 129.237.234.248, 129.237.234.210, 129.237.4.215, 129.237.234.11, 129.237.234.82 |
| packetFiltrationLevel | 2 |
| packetFilterDefaultBlockAll | true |
| packetFilterAcceptExternalIP | 10.10.10.10 |
| createTimestamp | 20030826132658Z |
| modifyTimestamp | 20031010193048Z |
| creatorsName | cn=directory manager |
| modifiersName | cn=directory manager |
| subschemaSubentry | cn=schema |

***Figure 6.8***: directory entry for *ou=Solaris Workstations*, in the NTS subnet.

Once again, the value of 2 for the attribute *packetFiltrationLevel* says that the firewall for the *Solaris Workstations* is at a level 2 in the firewall hierarchy. To satisfy the second requirement, i.e., a single host having special access policies, let us look at the specifications for the host *Thundercracker*, in Figure 6.9.



cn=thundercracker,ou=Solaris Workstations,ou=NTS,ou=LSS,ou=devices,ou=network,dc=ku,dc=edu

```
Browser root
  bender
    cn=Directory Administrators
    ou=network
      ou=agents
      ou=scratch
      ou=devices
        ou=LSS
          ou=NTS
            ou=Windows Based Systems
            ou=Solaris Workstations
              cn=astrotrain
              cn=starscream
              cn=grimlock
              cn=slag
              cn=sludge
              cn=thundercracker
              cn=ramjet
            ou=x86 Architecture Workstations
            ou=Handheld Computers
            ou=Wireless Access Points
            ou=Printers
            ou=special
            ou=Corporate Partners
            ou=Unregistered Devices
            ou=wireless devices
            ou=tablets
            ou=Mac Systems
            ou=Video over IP devices
            ou=Resnet Devices
            ou=Visiting Mobile Users
        ou=library
```

| Name | Value |
|---|---|
| deviceOS | SOLARIS |
| deviceOSVersion | 8 |
| deviceManufacturer | Sun Microsystems |
| deviceModelNumber | SunBlade 100, 500Mhz HB1.4.1/256MB/CD 48X |
| deviceFormFactor | DESKTOP |
| deviceFunction | CLIENT |
| objectClass | dhcClient |
| objectClass | dhcConfiguration |
| objectClass | ieee802Device |
| objectClass | ipHost |
| objectClass | networkedDevice |
| objectClass | top |
| objectClass | IPPacketFilter |
| cn | thundercracker |
| description | Software Engineering Group Solaris Development System |
| ipHostName | thundercracker.nts.ku.edu |
| serialNumber | FT13850040 |
| owner | uid=siddh,ou=authaccounts,dc=ku,dc=edu |
| macAddress | 00:03:ba:0e:2b:a7 |
| ipHostNumber | 129.237.234.210 |
| dhcRouter | 129.237.234.254 |
| packetFilterAcceptExternalIP | 129.237.4.0/24, 129.237.234.0/24 |
| packetFilterAllowInboundICMPType | 8 |
| packetFilterAllowInboundICMPType | 0,3 |
| packetFilterAllowOutboundICMPType | 8 |
| packetFilterRejectExternalIP | 129.237.4.215 |
| packetFilterAllowInternalUDPServices | 68, 5001 |
| packetFilterAllowInternalTCPServices | 22, 25-30, 5001 |
| packetFilterAllowExternalTCPServices | 80, 22, 21, 5002 |
| packetFilterAllowExternalUDPServices | 67, 53, 5002 |
| dhcLastRequestedOptions | { dhcSubnetMask : dhcRouter : ipHostName : dhcVendorSp |
| dhcLastOfferedOptions | { dhcLeaseTime=86401 : dhcSubnetMask=255.255.255.0 : |
| dhcLastRequestTime | Mon Sep 22 20:35:26 CDT 2003 |
| dhcClientStatus | DHCP Offer Accepted |
| dhcLeaseIssueTime | Mon Sep 22 20:35:26 CDT 2003 |
| dhcLeaseIssueTimeStamp | 1064280926319 |
| dhcLeaseExpireTime | Tue Sep 23 20:35:27 CDT 2003 |
| dhcLeaseExpireTimeStamp | 1064367327319 |
| createTimestamp | 20030826132658Z |
| modifyTimestamp | 20030923013526Z |
| creatorsName | cn=directory manager |
| modifiersName | cn=dhcpagent,ou=agents,ou=network,dc=ku,dc=edu |
| subschemaSubentry | cn=schema |

***Figure 6.9***: directory entry for *cn=thundercracker*, a host in the Solaris Workstations group of the NTS subnet.

The attribute *ipHostNumber*, in conjunction with the packet filter attributes, is used to verify the packets to or from *Thundercracker* in the firewall *NTS_Solaris_protector*.

## 6.4 Use of the directory by system/network administrators

As discussed earlier, every network administrator will be responsible for the department's firewall(s). Hence, the network administrator should be able to access the directory and define the security policies of the network. The directory access can be carefully controlled by using the authentication/authorization features available in directory servers. Also, access control lists can help in determining what part of the tree in the directory can be accessed by the administrator. There are many LDAP administration tools available for accessing the directory server.

Figure 6.10 shows an administrative hierarchy which allows for delegation of privileges within the security management and tracking information for the hosts in the network.

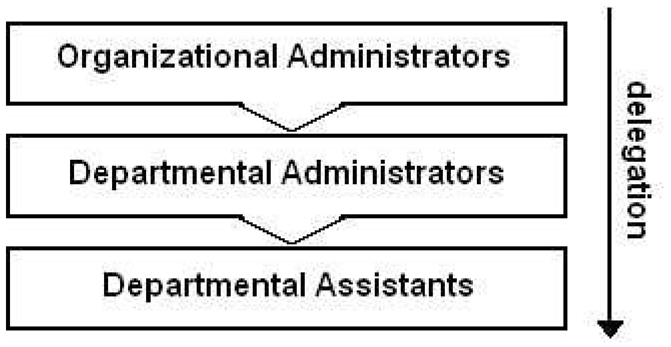


***Figure 6.10***: Delegation of privileges within the organization.

As the above examples show, the Directory Enabled system is simple and useful for firewall policy management.

**Summary of the reasons for using LDAP for policy management:**

From the previous sections, we can say that the use of LDAP and directories for firewall policy management eases this task. It provides for delegated management of the policies, mainly due to the flexible hierarchical model. Also, we obtain a high granularity of the security system, including host-level security. The policies can be easily synchronized and coordinated. Another major advantage is the scalable nature of this framework: it is scalable in terms of both the number of hosts and the number of policies. The directory framework and its firewall policy management schema act as a common language for different types of firewalls, both commercial and non-commercial firewalls.

The directory provides high speed search and security audit capabilities. As for directory access, the identification, authentication, and authorization take place before changes can be made. LDAPS provides for encrypted communication on the network. The encrypted user credentials are stored in the directory and on the underlying file system. There exist various flexible LDAP administration client tools for use by the system/network administrators. LDAP provides a protocol-oriented communication with external systems, i.e., ModPerl, or Java JNDI, or OpenLDAP APIs. Also, replication agreements with peer directory servers is possible. The directory servers are easy to load-balance, and easy to make backups via LDIF export.

## 6.5 Rule generation

We now have all the policies specified in the LDAP directory. This representation is not what a firewall needs. The firewall needs to be configured with rules in its own elaborate rule specification format. Therefore, the policies specified in the directory need to be converted to a format understandable by the firewall. This brings in the need for a ***rule generator***.

The rule generator can be considered as the front-end to the firewall policy management system. This rule generator depends on the firewall being used. Linux *iptables* requires a rule generator completely different from OpenBSD pf. There are many such firewalls, and every firewall in the distributed architecture might be a different type of firewall.

### 6.5.1 Requirements of the rule generator

1. The rule generator must be able to identify the entries for which rules need to be created.
2. The rules generated should conform to the hierarchical architecture.
3. The rule generator must be platform independent.

### 6.5.2 Algorithm for identifying the entries

In this section, we give an overview of the algorithm for identifying the entries for which rules need to be generated. The result is a list of entries, in the order in which rules are configured in the firewall. This ordering is based on the following assumption.

**Assumption:** In every firewall, packets are matched against the rules in a linear fashion. The packet traverses the rule list only until it reaches a matching rule.

**The Algorithm:**

1. Start traversing the tree from the DN specified in the *protectedNetworkDN* attribute of the protector. Let this DN be denoted by "RootDN".
2. Algorithm for obtaining all the nodes for which rules need to be generated:
    a. Let X = RootDN, index = 1, last = 1
    b. Put X in Table(last)
    c. Get children of X
    d. For each child "C" of X:

        last++ , and put C at Table(last) if and only if:

        - C does not have the *packetFiltrationLevel* attribute,

          AND
        - C is *not* "leaf without the attribute *objectClass=IPPacketFilter*"

    e. index++, X = Table(index)
    f. if X is not a leaf node, GOTO step c, else GOTO step e.
    g. In a Last-In-First-Out manner, move the values in Table to another table EntryList, only if that node has the attribute *objectClass=IPPacketFilter*.
    h. ***EntryList*** now has the entries in the order in which rules are to be created.

This algorithm only gives an idea about the correct way to obtain the list of entries, but not the best and efficient way.

### 6.5.3 Rule generator for Linux *iptables*

In our effort to apply the directory enabled policy management system to a real world situation, we created an application to generate *iptables* rules from the policies specified in the directory.

The application can be divided into two parts:

1. A firewall independent directory support system

   The firewall independent directory support system is used for:

   i. Connecting and authenticating to the LDAP server.

   ii. Providing modules for search and retrieval operations from the directory.

   iii. Obtaining the list of entries for which rules need to be created (section 6.5.2).

2. A firewall dependent rule creator

   The firewall dependent rule creator is a module that creates rules for that particular type of firewall. Hence, there will be separate modules for *iptables*, Drawbridge, Cisco PIX, etc.

The Java Naming and Directory Interface (JNDI) was used to provide access to the enterprise LDAP server. Te Java Standard Edition (J2SE) was used for the rest of the application. The reason for using Java was to make the modules platform independent, so that the system administrators are not bound to a specific platform. The basic classes used in developing the application are described in Appendix II.

# Chapter 7

# Load Balancing for firewalls

Figure 7.1 shows a setup with a single firewall separating two IP networks. All traffic from one network to the other network must pass through this firewall. The firewall applies the packet filtering rules to every packet, and makes a decision whether to forward the packet or reject the packet. These rules are applied sequentially to the packets, i.e., in the same order that they were configured. Therefore, the load on the firewall depends not only on the number of packets traversing the firewall, but also on the number and order of the configured filter rules associated with each network interface.

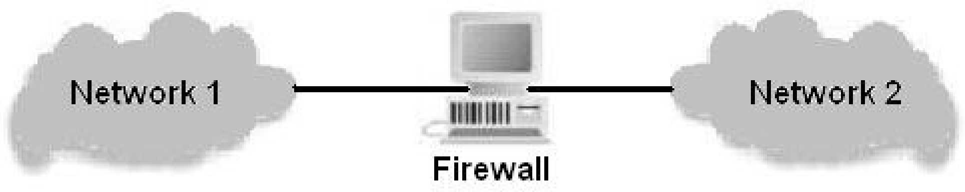


***Figure 7.1***: A single firewall between two networks

In high speed networks, a single firewall becomes a bottleneck to the network bandwidth. There are two possible ways to compensate for the bottleneck. One way to increase the performance is to get a better processor, but this solution does not scale for fast scaling high speed networks. The other solution is to use parallel processing, thus distributing the packet load evenly to a set of firewalls working in parallel.

## 7.1 Performance of a single firewall

The test systems were out-of-the-box Linux systems, without any performance tuning. The OS on every system is Linux Kernel 2.4.20.

The test setup consisted of the following systems: Figure 7.2.

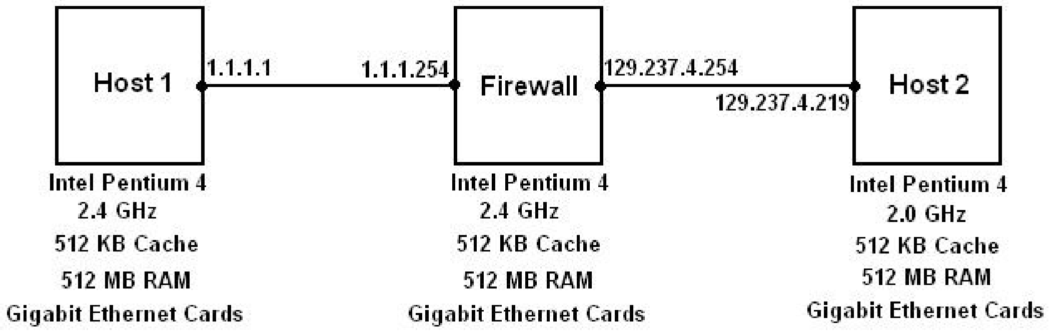


***Figure 7.2***: Setup for testing the performance of one firewall.

The tests were conducted with varying packet sizes, using iperf for TCP connections. Note that these are bi-directional, which involve requests and replies. The firewall/packet filter was configured with no rules, ~160 rules, and ~2180 rules.

### 7.1.1 No rules:

Figure 7.3 shows the performance of the firewall for varying packet sizes. IP forwarding was enabled and no *iptables* rules were configured.

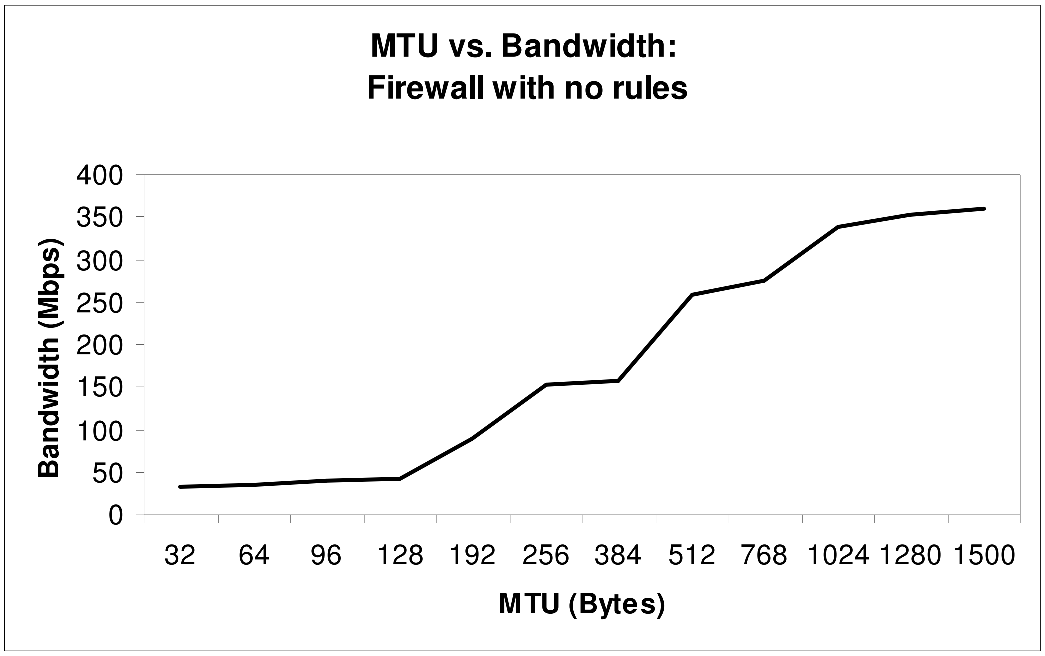


***Figure 7.3***: Performance of *iptables* under varying packet sizes: No Rules.

### 7.1.2 Around 160 rules:

Figure 7.4 shows the performance of the firewall for varying packet sizes. IP forwarding was enabled, and the packets were made to traverse around 160 rules before being accepted in the FORWARD chain. The test was conducted for both stateless and stateful packet filter rules.

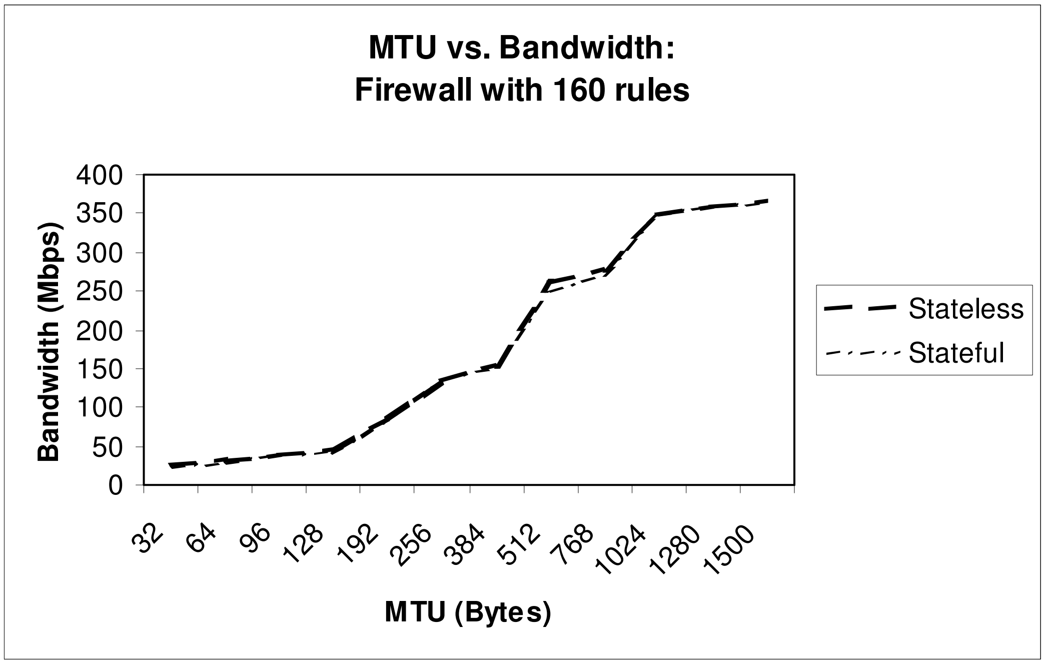


***Figure 7.4***: Performance of *iptables* under varying packet sizes: 160 Rules.

### 7.1.3 Around 2180 rules:

Figure 7.5 shows the performance of the firewall for varying packet sizes. IP forwarding was enabled, and the packets were made to traverse around 2180 rules before being accepted in the FORWARD chain. The test was conducted for both stateless and stateful packet filter rules.

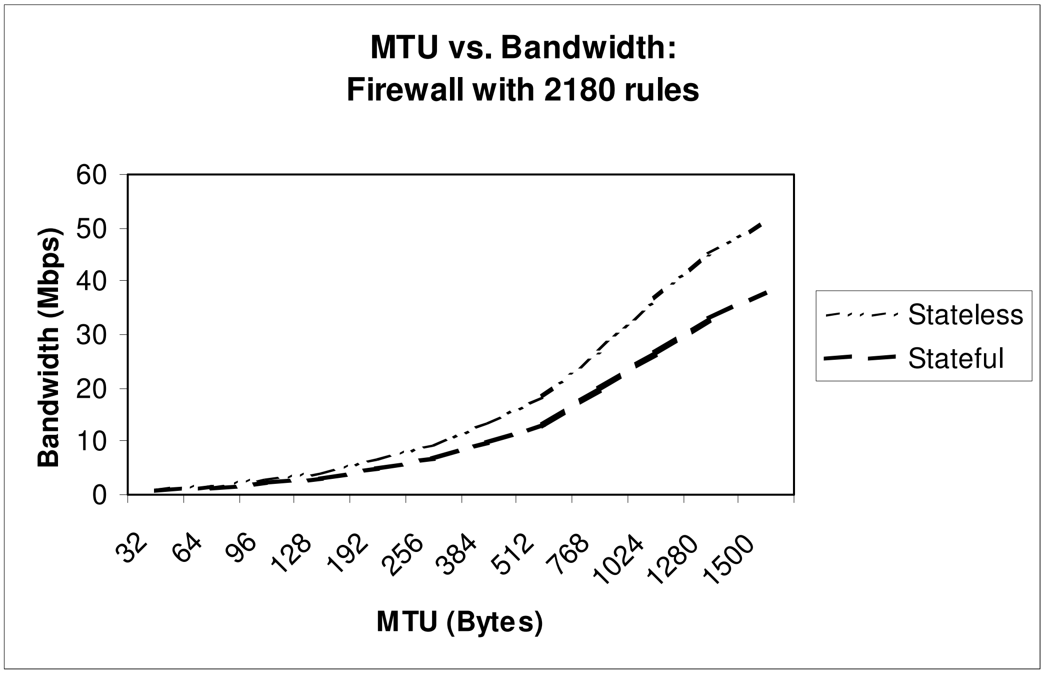


***Figure 7.5***: Performance of *iptables* under varying packet sizes: 2180 Rules.

As the above figures show, the same throughput is maintained for the case with no rules and the case with 160 rules. The real crunch is seen when the firewall is configured with around 2180 rules. For large packets, the bandwidth drops from 365Mbps to 51Mbps (for stateless filtration).

## 7.2 Parallel packet filtration

The model in Figure 7.1 can be extended to use more than one firewall between the two networks, as shown in Figure 7.6.

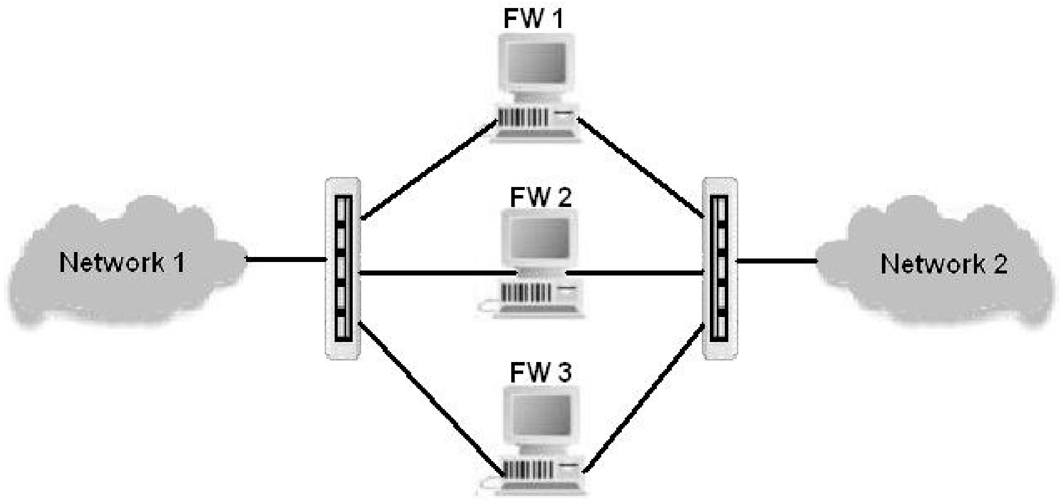


***Figure 7.6***: A parallel packet filter setup.

This brings up the issue of load balancing. The traffic between the two networks needs to be load balanced among the firewalls. This aspect can be integrated into the system in two ways:

- The firewall "selects" the packets that it will process.
- The firewall "gets" the packets that it will process.

### 7.2.1 Firewall *selects* the packets to be processed

This is the case wherein both the filtration and load balancing functions are combined in a single system. All the packets to be filtered are forwarded to all the firewalls. The firewalls then use a packet selection algorithm to select the packets that will be filtered using the packet filtration rules. This is a hub-based approach, as shown in Figure 7.7.

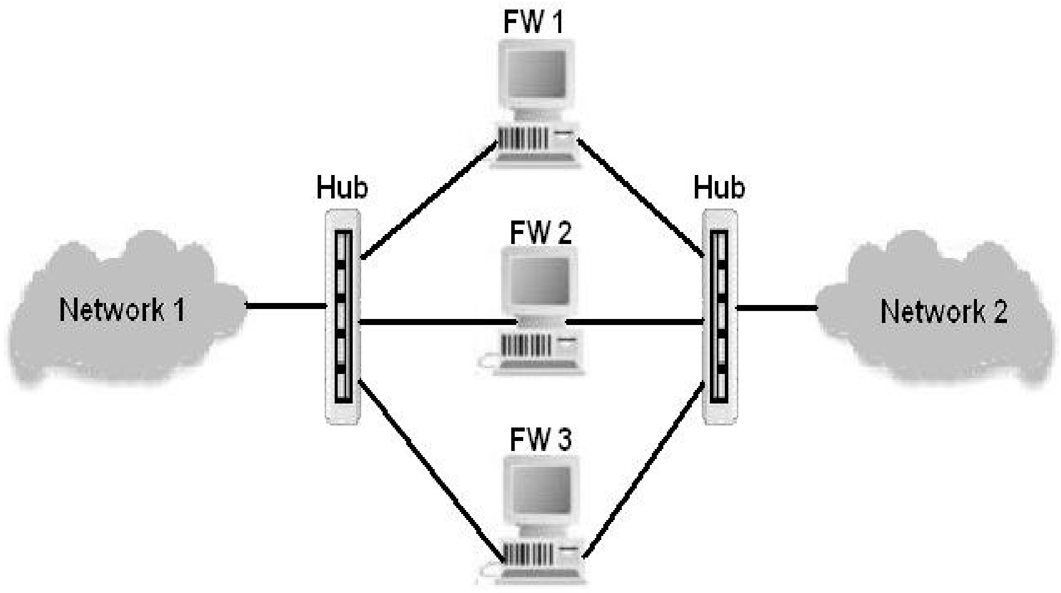


***Figure 7.7***: A parallel packet filter setup, using hubs.

This approach was tested for its performance by Carsten Benecke in [12].

Table 7.1 shows the speedups obtained by such a setup.

| Processors | Speedup |
|---|---|
| 1 | 1 |
| 2 | 1.82 |
| 3 | 2.3989 |
| 4 | 2.9557 |

***Table 7.1***: Speedup obtained for hub-based load balancing. [12]

The drawbacks of this hub-based architecture are:

- The major drawback is the half duplex mode of the hubs employed. Hubs implement a shared segment and all connected workstations compete with each other for the available bandwidth.
- In conditions of heavy network load, the number of collisions on the Ethernet will increase dramatically.
- If two fully switched fast Ethernets are joined by parallel packet filters based on hubs, the total bandwidth of 200Mbits/s (full duplex) is reduced to at most 100Mbits/s.

### 7.2.2 Firewall *gets* the packets to be processed

The following is an excerpt from an article "Firewall and Load-Balancer: Perfect Union?". [13]

> At The Internet Security Conference recently, we participated in a panel discussion on the merits of load-balancing in the enterprise and the security issues involved.
> Not surprisingly, many audience members asked the panel about integrating firewall services with load-balancers. The panel's response was decisive: Integrating a firewall appliance into a load-balancer or vice versa creates more problems than it solves. Combining the specialized functionality of both products--not to mention trying to create a single point of failure--is difficult. Segmenting the processes allows for additional fine-tuning of performance parameters and configuration problems.

This section describes an attempt to separate packet filtration from load balancing. The system is as shown in Figure 7.8.

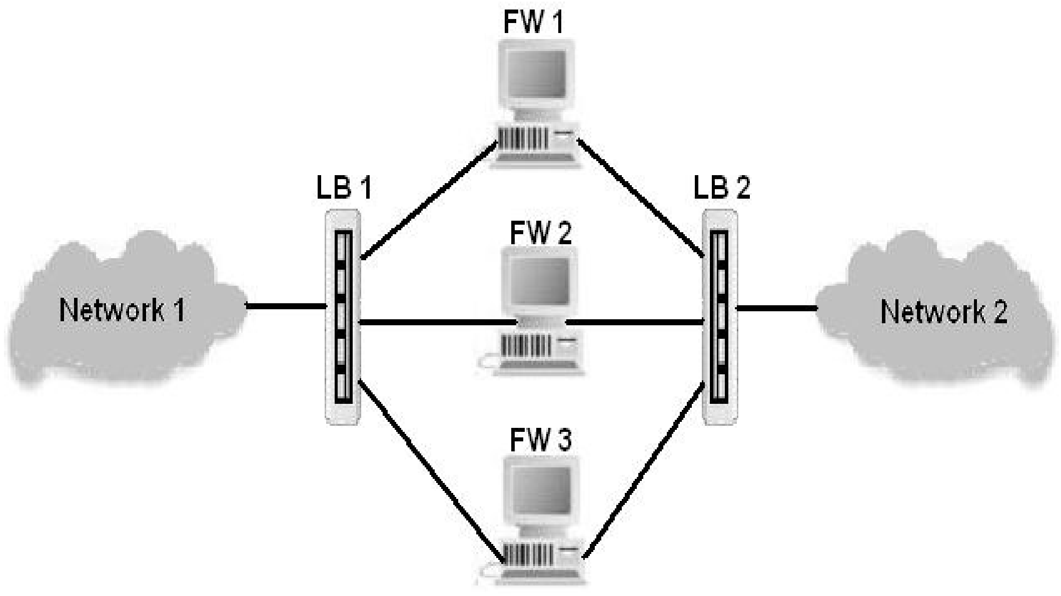


***Figure 7.8***: A parallel packet filter setup, using load balancers.

The hubs in Figure 7.7 are now replaced by load balancers (LB1 and LB2), which can be considered as yet another level of packet filtration. Each of the firewalls (FW1, FW2, FW3) is connected to an Ethernet interface on the load balancers. The packets are forwarded to the firewalls based on a packet selection algorithm in the load balancers.

The advantages of this architecture, compared to the hub-based architecture are:

- The firewalls do what they are intended to do: they need not worry about the selection of packets to be filtered. This removes the additional load, and the firewalls can perform to their best.
- This overcomes the half duplex limitations of the hub-based architecture. The firewalls do not compete for the available bandwidth. The bandwidth available to the firewalls now depends on the load balancers.

- In conditions of heavy load, the rate of collisions is not as high as in the hub-based architecture.

#### 7.2.2.1 Performance of this architecture

This architecture was tested for its performance, using two firewalls. The setup is as shown in Figure 7.9.

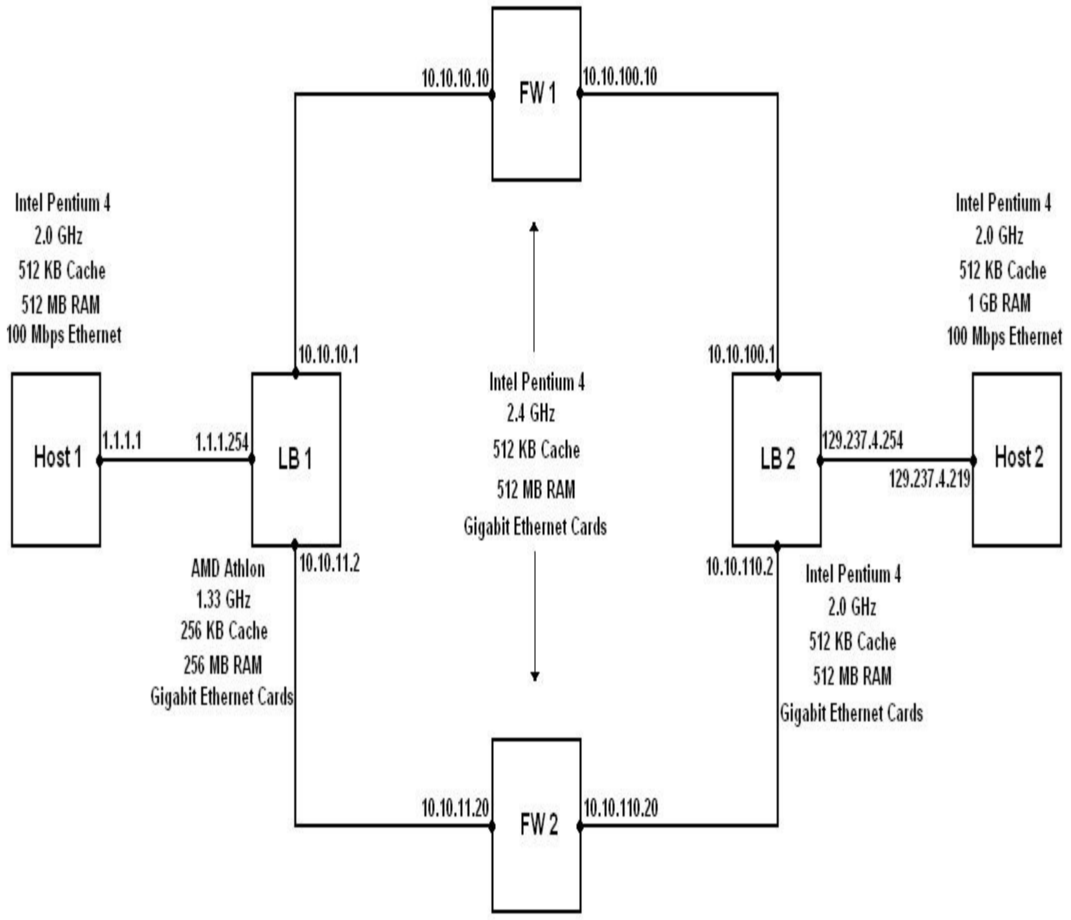


***Figure 7.9***: Test setup for parallel packet filters, using load balancers.

The test setup consisted of the systems shown in the figure, with the following environment:

- All systems had the RedHat Linux 9.0 Kernel 2.4.20 operating system.
- The firewalls FW1 and FW2 were each configured with 2188 rules.
- The load balancers LB1 and LB2 used Linux iproute2 for load balancing the packets between the two firewalls. Load balancing with iproute2 follows a route based algorithm.

The limitations to the test setup were Host1 and Host2 that had 100Mbits/s Ethernet cards. This limited the ability of the load balancers and firewalls (having gigabit Ethernet cards).

As seen in Figure 7.5 and Figure 7.4, the bandwidth possible with 2188 rules is around 51Mbits/s for stateless filtration, whereas, 162 rules would give us 365Mbits/s. To test the load balancing between the two firewalls, and with the limitations of the two end systems, the firewalls were configured with 2188 rules. This setup was used to calculate the speedup achievable by load balancing.

Iperf was used for testing the throughput. The test parameters were:

- Window size: 128KB
- Maximum Segment Size (MSS): the default for iperf: 1460 (MTU = 1500)
- Read/Write buffer length: 8KB for TCP and 1470 bytes for UDP

Though these parameters do not represent the real world traffic, they can be used, without loss of generality, to calculate the speedup achievable by load balancing the firewalls.

Unlike the tests conducted by Carsten Benecke [12], discussed in section 7.2.1, the TCP and UDP connections were 2-way connections: including both requests (client to server) and replies (server to client). This is a more accurate representation of the communication protocols. Note that the number of reply packets is less than the number of request packets.

The tests performed included the following:

1. TCP: one connection
2. TCP: two parallel connections, one in each direction
3. TCP: two parallel connections in the same direction

The results of the tests:

1. TCP: one connection

   This test is represented in Figure 7.10.

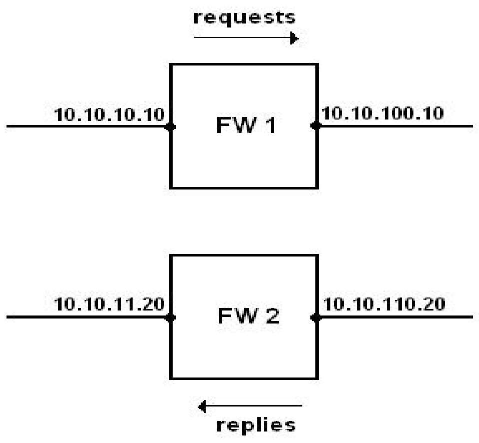


***Figure 7.10***: Direction of traffic flows, one TCP connection.

The requests were forwarded through FW1, and the replies through FW2. With a single firewall, the throughput obtained was 46Mbits/s, whereas the two firewalls together were capable of providing 64.5Mbits/s.

2. TCP: two parallel connections, one in each direction

   This test is represented in Figure 7.11.

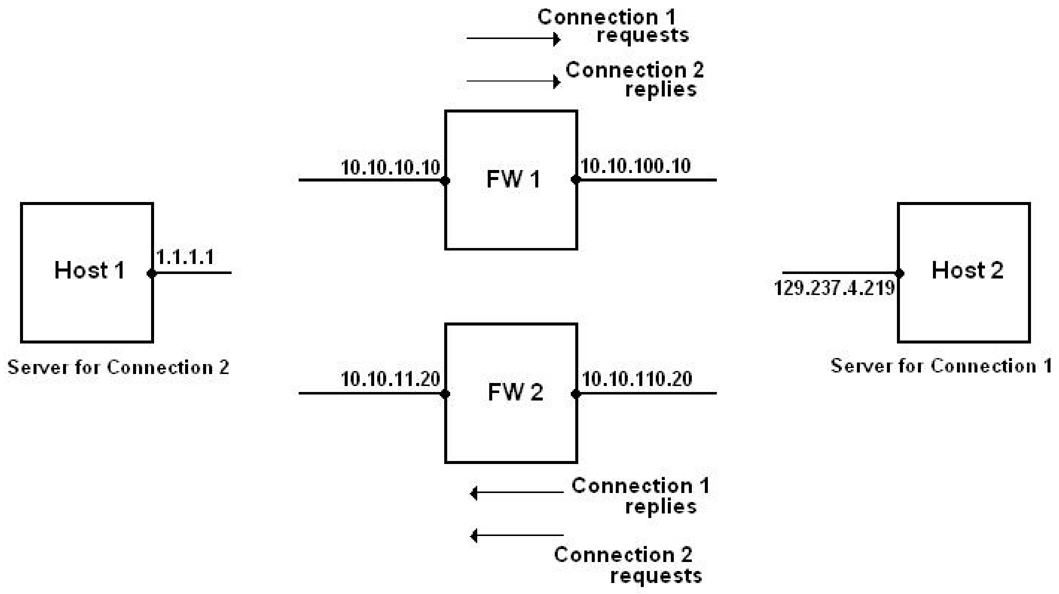


***Figure 7.11***: Direction of traffic flows, two TCP connections in different directions.

FW1 filtered the requests for Connection1 and replies for Connection2. FW2 filtered the requests for Connection2 and replies for Connection1. With a single firewall, the throughputs obtained were 23.6Mbits/s and 26.0Mbits/s, whereas the two firewalls together were capable of providing 54.0Mbits/s and 55.7Mbits/s.

3. TCP: two parallel connections in the same direction

   This test is represented in Figure 7.12.

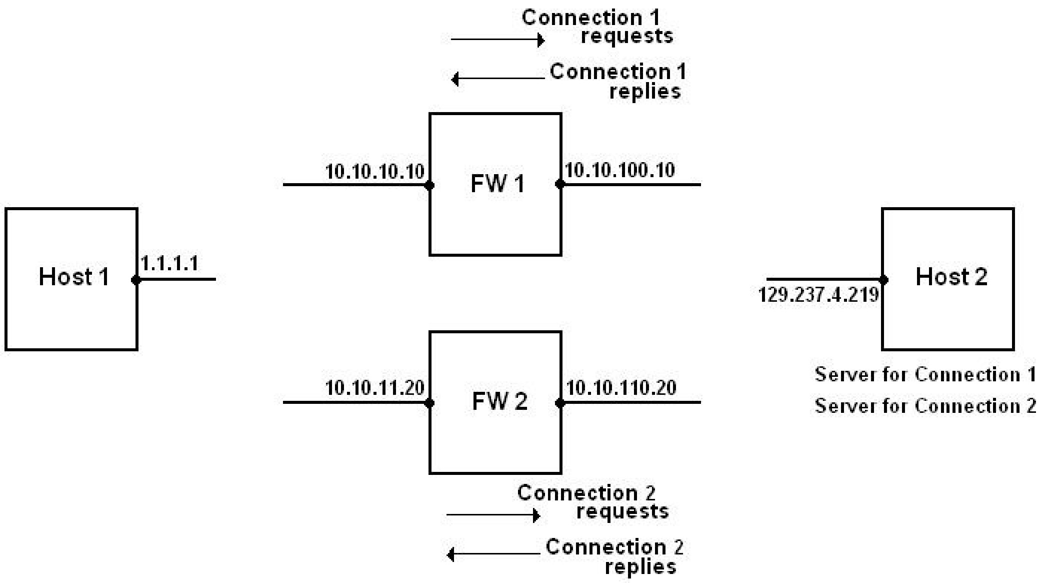


***Figure 7.12***: Direction of traffic flows, two TCP connections in same direction.

   FW1 filtered all the packets for Connection1. FW2 filtered all the packets for Connection2. With a single firewall, the throughputs obtained were 18.8Mbits/s and 19.6Mbits/s (a sum of 37.4Mbits/s), whereas the two firewalls together were capable of providing 47.5Mbits/s and 46.4Mbits/s (a sum of 91.9Mbits/s).

Table 7.2 shows the speedup obtained for each of these tests.

| Test | Bandwidth with 1 firewall (Mbits/s) | Bandwidth with 2 firewalls (Mbits/s) | Speedup |
|---|---|---|---|
| ***One connection*** | | | |
| Connection 1 | 46.0 | 64.5 | 1.40217 |
| ***Two parallel connections: one in each direction*** | | | |
| Connection 1 | 23.6 | 54.0 | 2.28814 |
| Connection 2 | 26.0 | 55.7 | 2.14231 |
| ***Two parallel connections: both in same direction*** | | | |
| Connection 1 | 18.8 | 47.5 | 2.52660 |
| Connection 2 | 19.6 | 46.4 | 2.36735 |
| Aggregate | 37.4 | 91.9 | 2.45722 |

***Table 7.2***: Speedup obtained for load balancing, using load balancers.

The speedups (more than 2 for 2 processors) obtained with two connections can be explained by Little's law. With a high packet arrival rate, and as the queue gets overloaded, the system performance decreases rapidly. This explains the low performance of a single firewall, and the high speedup obtained with multiple firewalls.

The load balancing was route based. This was one of the limitations for the performance of the parallel firewall architecture. The route based algorithm does not distribute the packets evenly between the firewalls. This explains the low speedup for the single connection, as compared to the speedup for parallel connections. In case of the parallel connections in the same direction, the load was almost evenly split up between the two firewalls, because each connection followed a separate route.

This test was performed with stateless filtering, because of the unavailability of a load balancer for stateful filtering, which requires that the replies follow the same route as the requests (flow based load balancing).

Another important aspect to be considered is the CPU utilization. This gives an idea about the bottlenecks in the setup. Table 7.3 shows the CPU usage for the firewalls and the load balancers for each of the tests.

| **Test** | **CPU Usage** | | | |
|---|---|---|---|---|
| | **LB 1** | **LB 2** | **FW 1** | **FW 2** |
| One connection | 20% | 20% | 100% | 35% |
| Two parallel connections: one in each direction | 25% | 25% | 100% | 100% |
| Two parallel connections: both in same direction | 25% | 25% | 100% | 100% |

***Table 7.3***: CPU usage for the systems involved in the test.

Table 7.3 indicates that the bottlenecks are the firewalls. The load balancers can handle much more traffic than the two firewalls, even though the firewalls have better processors.

## 7.3 Packet Selection Algorithm

The requirements of the packet selection algorithm in the load balancers are:

1. The algorithm must be fast enough to prevent the load balancer from being the bottleneck.
2. The algorithm should map each packet to exactly one firewall.
3. The algorithm should allow the system to be scalable, easily adapting to increasing number of firewalls.
4. The algorithm should be able to handle flow based load balancing for stateful packet filters.

Another key aspect to load balancing is the selection of the node to process the packet. The scheduling algorithm decides this aspect. Some of the scheduling strategies that can be used for load balancing firewalls are:

- Weighted Round Robin

  Round robin scheduling consists of allocating a packet to each firewall in turn. Weighted round robin gives each firewall a number of packets based on their relative processing power. This is not useful for stateful filtration, unless the stateful filters are synchronized with the state information.

- Minimal Node Load

  In this approach, the load balancer maintains load statistics for every firewall. A new request is assigned to the node with the lowest current workload, based on some internal benchmark - CPU idle time, free memory, ping time, or some compound measure. This does not work for stateful filtration.

- Hash Functions based on packet information

  A hash-function can be used to map the incoming packets to the firewalls. The various types of packet information that could be used for the hash calculation are: [12]

  - IP Address
  - IP Address and Port Number
  - IP header checksum
  - Frame checksum
  - Combinations of the above

  The packet information like IP addresses and port numbers can be used for load balancing stateful packet filters.

## 7.4 Available Load Balancers

Most of the available load balancers support load balancing for a cluster of servers. Load balancing firewalls is completely different from load balancing servers. According to T.W.Verwoerd, [14]

> In order to load balance all traffic directed into a clustered gateway while allowing the use of packet filtering or transparent proxies, a distributed load-balancing gateway would be required. Such a system would have to compensate for NAT effects on the gateways, and correctly channel related packets into the same node on all interfaces. In addition, such a system would have to maintain internal synchronization between load-balancing devices, without forming a new performance bottleneck or single point of failure. To the best of our knowledge, no such system currently exists.

Though there are no non-commercial products, there are some commercial products that have recently made their way into load balancing for firewalls. Some such products are:

- F5 Networks: BigIP Application Switches
- Nortel Networks: Alteon Application Switches
- Cisco Systems: CSS Services Switches

These products follow the architecture described in section 7.2.2.

# Chapter 8

# Case Study: University of Kansas

This chapter focuses on applying the distributed security architecture, and the directory enabled policy management system described in the previous chapters to the network of the University of Kansas.

## 8.1 The University network

The University of Kansas operates enterprise class voice, video, and data transport systems in support of the needs of its faculty, staff, and students.
The data network provides connectivity to some 17,000 workstations, many dozens of servers, high performance computing platforms, several remote sites, regional networks, the commodity Internet, and Internet 2.

10/100BaseT and IEEE 802 Ethernet protocols provide basic LAN support. The backbone of this network consists of five Cisco Catalyst 6509s running native IOS in a layer 3 "routed" environment. These devices are currently fully interconnected in a mesh topology via single-mode fiber optic media operating at 1000 Mbps full-duplex.

KU's data network is assigned its own Autonomous System (AS) number, and controls its own class B IP address range. The IPX and Appletalk protocols are no longer routed at layer 3, and KU currently employs an IP-only environment. Multicast is also enabled for selected locations on campus.

KU connects to the Kansas Research and Education Network (KANREN) for communicating with other State educational institutions and the commodity Internet (70 Mbps of Internet bandwidth) and also peers with the Great Plains Network (GPN) for regional communications with other state networks and access to Internet 2. All of this connectivity is currently aggregated using a Cisco Lightstream 1010 ATM switch and PVCs over an ATM OC3c link operating at 155 Mbps.

**Some statistics**

- Number of students, faculty and staff: ~35000
- Number of buildings: ~100
- Number of hosts: ~20000
- Internet 1 link: 70Mbps rate limited on 100Mbps connection

## 8.2 Distributed firewall implementation for KU

The University environment is a heterogeneous environment with different users and different requirements. It has different facets: it is an educational and research entity, it is a Corporation, and it is an ISP. The major challenges faced by security administrators in a University network are: [15] Lack of control over users, a loose confederation of autonomous entities, an academic culture and tradition of open access to information, the complex trust relationships between departments at various Universities for research, and the availability of excellent platforms for launching attacks (high bandwidth Internet, sophisticated computing capacity, insecure systems in dorms).

Securing such a vast and diverse environment is not easy. This requires a delegated security management system.

The distributed architecture, described in Chapter 4, fits well into the University environment. Every department can have its own firewall, with its own specifications. For example, the History department's security policies can be independent from those of the Computer Science department. Also, different sections in the same department may have different security policies.

The distributed security architecture would drastically decrease the number of insider threats in the network.  If the network only had a single border firewall, and a host was compromised behind the firewall then used to attack internal hosts, or an attack originated from the internal network, a hacker could target any of the 20,000 systems in the network. On the other hand, with distributed firewall architecture in place, the hacker's access is limited to only a few systems in that section. The rest of the systems would still maintain layers of protection from the compromised section.

Another issue that requires special attention is the need for higher bandwidth connectivity. In a large network with thousands of users, the bandwidth cannot be compromised. But it is a property of firewalls to become a bandwidth bottleneck. In case of a single border firewall, with thousands of rules for the complete network, the firewall is bound to become a bottleneck. On the other hand, in the distributed architecture, the campus border firewall will have only a few rules, allowing it to have minimal impact on network throughput. The other rules will be distributed among all the other firewalls, with little effect to their performance.

## 8.3 Firewall deployment

We had seen three possible firewall deployment architectures in section 4.3. In a University, where network connectivity is utmost important, an extended disruption to the network would be catastrophic. This brings in the need to separate the network topology from the security topology, especially because the network topology is already well established. Hence, the ***loop-back*** deployment architecture would be best suited to the existing network in a University environment.

## 8.4 Firewalls that can be used: Cost effective solution for KU

### 8.4.1 At the border

In this section, we make the assumption that all packets are matched against all the rules in the firewall. This is the worst case scenario.

The University is currently using a rate limited 70Mbps out of a 100Mbps connection to Internet 1. The rate limiting will likely be removed in the near future as commodity Internet bandwidth costs decline, hence, at this stage, we need a firewall that can provide a throughput of at least 100Mbits/s. The major factors affecting the throughput of a firewall are:

- The number of rules.

- The type of filtration: stateless or stateful.
- The size of packets.
- The number of flows (connections) passing through the firewall.

Currently, the number of packet filtering access control lists in the border router, pertaining to the KU campus, is approximately 50 rules. The average packet size is 200 to 500 bytes. Stateless filtration is used.

The firewall at the border of the network need not be configured with the policies for individual hosts or subnets within departments. These will be configured in the respective firewall of the department. Also, the border firewall need not be a stateful filter. Stateful filtration can be used in the inner firewalls.

The following assumptions will be used for determining the firewall architecture at the border:

- Number of rules: ~100
- Packet size: ~200 to 500 bytes
- Type of filtration: Stateless
- Number of flows: not an issue with stateless filtration.

With these parameters, and from Figure 7.4, we can say that a non-commercial firewall/packet filter can be used.

Note that the results in Figure 7.4 were obtained using simple, low cost systems, and freely available packet filter (*iptables*). With a packet size of 192 bytes, the throughput achieved was 85.5Mbits/s. Packets larger than this were processed at speeds greater than 100Mbits/s when gigabit network interfaces on the firewall were utilized.

The bandwidth of 85.5Mbits/s is not sufficient for the border firewall. Hence, we suggest the use of load balancers, as described in section 7.2.2. With the parameters specified above, the load balancer need not keep track of state. Hence, the traffic can be evenly distributed among all the firewalls.

Summary of the proposed firewall architecture at the border:

- Non-commercial firewalls with stateless filtration.
- Load balancing for the firewalls, need not keep track of state.
- Number of firewalls required: 2 (from Figure 7.4 and Table 7.2). The number might vary for other firewalls (OpenBSD pf, Drawbridge, etc.)
- Scalability issues: If the bandwidth requirement increases, adding firewalls to the load balanced architecture is simple.
- Need for high-availability protocols in the load balancers to prevent failures.

### 8.4.2 For a department

In general, the following observations have been made for departments in the University:

- Number of rules: ~200
- Packet size: ~200 to 500 bytes, on an average.
- Number of unique flows: ~100 per minute (from netflow record analysis).
- Type of firewall: stateless or stateful.

For a department, let us make the following assumptions:

- Number of rules: ~500
- Packet size: ~200 to 500 bytes
- Number of flows: ~250 per minute
- Type of firewall: stateful

With these parameters, and from Figure 7.4, we see that a non-commercial firewall can support the security requirements of the department. As for the stateful filtration, the maximum number of connections the state table can contain depends on the physical memory available. For example, for *iptables*, with 512Mb of physical memory, the default maximum is 32760. The states for UDP have a timeout of 19 seconds, and the maximum for TCP is two minutes, except for the "established" state, which has a timeout of 5 days (but the entry will be removed as soon as the connection is closed). Hence, the 500 flows can be handled by the firewalls without any overflows.

The number of parallel packet filters needed depends on the bandwidth required by the department. The network usage of the departments varies with the work being done. Hence, the decision about the number of firewalls is to be made by the department, and cannot be generalized.

Summary of the proposed firewall architecture for the departments:

- Non-commercial firewalls with stateless/stateful filtration.

- Load balancing for the firewalls, depending on the requirements of the department. For stateful filtration, the load balancer needs to keep track of state.
- Scalability issues: If the bandwidth requirement increases, adding firewalls to the load balanced architecture is simple.
- If more than one firewall, need for high-availability protocols in the load balancers to prevent failures.

**Note:** The above discussions are based on the test results obtained in section 7.1 and section 7.2.2.1. The test systems were not tuned for better performance. With performance tuning, and using systems with better processors, the bandwidth achieved can be improved.

## 8.5 Policy management for KU

In section 6.3, we discussed the use of LDAP directory servers for firewall policy management. The figures and snapshots in that section pertain to the University of Kansas, and to one of the departments.

Every network administrator will have access to the central LDAP directory, but only for that part of the network the administrator is responsible for. The rule generator makes the administrators' task easy. The administrator will have to specify the policies in the directory, generate the rules using the rule generator, and configure the firewall with the rules generated.

## 8.6 An example for the use of the above described architecture

This section describes the application of the distributed security architecture to a real world example: The recent W32.Nachi worm attack.

The Nachi worm spreads by exploiting a vulnerability in Microsoft Windows. It scans the local class-b subnet (port 135) for target machines. It sends an ICMP ping to potential victim machines, and upon a reply, sends the exploit data. A remote shell is created on the target system, which connects to the infected machine on a TCP port in the range 666-765. Victim machines are instructed to download the worm via TFTP.

With the current architecture, i.e., a single firewall architecture, the network and security administrators faced the following problems:

A few infected hosts in the internal network were trying to infect other hosts, both inside and outside the campus network. The whole network was flooded with ICMP ping packets, and the routers were overloaded with the excessively high number of flows.

The steps taken to overcome this problem:

- The packet filter in the border router was configured to block packets destined to TCP or UDP port 135.
- The infected systems were identified and repaired.

The first step could only stop the spread of the infection from the internal network to the external network, and vice versa, but it could not stop the infection from spreading to other systems within the internal network. Also, this could not prevent the flooding of the backbone network. Each infected system attempted to generate approximately 100,000 flows per minute. The border router was still overloaded: because it had to filter millions of packets using the Access Control List (ACL) mechanism. Another problem was the time taken to identify and repair the infected systems. This process takes time, and in the meanwhile, the system would have infected many more systems. Identifying and repairing many hundreds of infected hosts required a big task force, and hence incurs high expenses.

With a distributed architecture, the management "nightmare" can be transformed into a simple but efficient task.

- Every firewall in the network can be immediately configured to block all packets addressed to TCP or UDP port 135. This prevents the worm from spreading to areas outside the firewall. Hence, a system in department X will not be able to infect a system in department Y. The threat remains at the network edge.
- The traffic generated by the affected system remains within the subnet of that department. This avoids the exponential traffic growth at the border router, thus preventing the router's overloading and failure risks.

- Every firewall deals with the situation, thus distributing the heavy traffic load among them.
- The firewalls can be configured before identifying and repairing the infected systems. This removes the “extra” time given to the system for infecting other hosts in the campus.
- The directory has a complete list of the firewalls and the hosts they protect. This helps in identifying the firewall for an infected host. The mechanism of targeting the firewall to be configured will speed up the process.
- The complete process can be divided into four steps:
    - i. Enter the policy in the directory, for every firewall.
    - ii. Generate the rules for the firewalls.
    - iii. Inject the rules into the firewalls.
    - iv. Identify and repair the infected systems.
- The management of the security incident does not require a huge task force, as required by the current architecture. Not many systems will be infected, and hence, not many people are required to track and repair them.

## 8.7 Summary

The above discussions and example indicate that a distributed security architecture, in conjunction with directory enabled policy management will work best for the University of Kansas. The important features of this system are that it provides defense in depth, including host-level security, the insider threats remain at the network edge, and do not spread throughout the campus network. Synchronized and coordinated policy management can be achieved. This system facilitates quick response to security incidents and emergencies. The system can be a low cost system with the use of non-commercial firewalls/packet filters.

# Chapter 9

# Conclusion & Future Work

## 9.1 Conclusion

The discussions, analyses, and test results presented in this report have shown that

- A distributed security architecture is required for large networks.
- The distributed architecture can be implemented by using low cost, non-commercial firewalls/packet filters, and satisfy the bandwidth requirements of the network.
- The directory enabled policy management system greatly simplifies the task of managing the distributed security architecture.

In effect, the directory enabled system helps us maintain a distributed security architecture while retaining the ability of the departmental system administrators to make fine-grained decisions about what may pass through the distributed firewall to their managed systems.

## 9.2 Challenges

The major challenge in this project was the design of the LDAP schema and the hierarchical organization of the policies. The other network management components which are being developed along with the distributed firewall support module, are device policy management, IP address management, DNS, DHCP and SLP management, Usage-based charging system, VLAN administration, and wireless access point administration. Hence, the distributed firewall support modules had to be designed without disrupting the existing hierarchy in the directory. Also, the object classes were developed with an idea of simplifying the task of the network administrator. These were created after studying the rule formats and features of Linux *iptables*, Drawbridge and Cisco PIX firewalls. The resulting specifications are generic, and can be used for creating rules for any type of packet filter.

## 9.3 Future Work

The additional work that needs to be done to complete and enhance this project:

1. More features for firewall maintenance

   Other attributes, related to firewall maintenance need to be added. Some of those are the time the firewall was last updated and the policies were last modified, the firewall access policies, for example, policies for the input and output chains, and an application that can inject rules into all the firewalls, during security incidents.

2. Rule generators for different types of firewalls

   A rule generator for *iptables* has been developed. Rule generators for other types of firewalls/packet filters need to be developed to support the various types of firewalls in the organization.

3. Managing firewall auditing

Firewall auditing capabilities have to be included in the framework. These include logging facilities, packet counters (netflow), and usage based metering/ charging.

4. Integrating IDS into the firewall architecture

   It is always a good idea to use Intrusion Detection Systems, in conjunction with firewalls. The directory enabled framework can be enhanced to include IDS support.

# Bibliography

**[1]** Enterasys Networks, *Directory Enabled Networking, A Technology Guide*. http://www.enterasys.com/products/whitepapers/den

**[2]** Elizabeth D. Zwicky, Simon Cooper and D. Brent Chapman. *Building Internet firewalls, 2nd edition*. O'Reilly, June 2000.

**[3]** William R. Cheswick, Steven M. Bellovin, Aviel D. Rubin. *Firewalls and Internet Security, Second Edition: Repelling the Wily Hacker*. Addison-Wesley Professional Computing Series, © 2003.

**[4]** Heinz Johner, Larry Brown, Franz-Stefan Hinner, Wolfgang Reis, Johan Westman. *Understanding LDAP*. IBM Corporation, International Technical Support Organization, © 1998.

**[5]** Rosanna Lee. *The JNDI Tutorial: Building Directory-Enabled Java Applications*. Sun Microsystems, Inc. © 1998-2000. http://java.sun.com/products/jndi/tutorial/

**[6]** CSI/FBI National Computer Crime Survey, 2002. Oregon State Controller's Division.

**[7]** Tom Markham, Charlie Payne. *Security at the Network Edge: A Distributed Firewall Architecture*. Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEXII'01), © 2001 IEEE.

**[8]** Samuel Patton, David Doss, William Yurcik. *Open Source Versus Commercial Firewalls: Functional Comparison*. Proceedings of the 25th Annual IEEE Conference on Local Computer Networks (LCN 2000), Tampa FL. USA, November 2000.

**[9]** Marcus J. Ranum. *Thinking about Firewalls*. Presented at SANSII in Washington, DC, 1993. http://www.linuxsecurity.com/resource_files/firewalls/ThinkingFirewalls/ThinkingFirewalls.html

**[10]** David R. Safford, Douglas Lee Schales, and David K. Hess. *The TAMU Security Package: An Ongoing Response to Internet Intruders in an Academic Environment*. Proceedings of the Fourth USENIX Security Symposium.

**[11]** Roberto Nibali, ratz and Jonathan Heusser. *Packet Filters and their Behavior under high Network Load: The impact of extreme (real) conditions analyzed on Linux, Solaris and OpenBSD*. March 20, 2002.

**[12]** Carsten Benecke. *A Parallel Packet Screen for High Speed Networks*. 15th Annual Computer Security Applications Conference, December 1999.

**[13]** Gregory Yerxa. *Firewall and Load-Balancer: Perfect Union?* Network Computing, February 7, 2000.
http://www.networkcomputing.com/1102/1102ws1.html

**[14]** T. W. Verwoerd. *Stateful Distributed Firewalls*. University of Canterbury, 2001.

**[15]** Paul Howell. *Security Interchange*. MAIS / Technical Infrastructure Operations, June 2002.

**[16]** Nortel networks. *Firewall load balancing: Application switching to optimize firewall performance*. © 2002.

**[17]** Internet2 middleware and directory initiatives web page.
http://middleware.internet2.edu/core/directories.html

**[18]** James Grant, Philip Attfield, Ken Armstrong. *Distributed firewall Technology*. EWA Canada, March 2001.

**[19]** Documentation for iPlanet Directory Server, Sun ONE Directory Server, and LDAP available at http://docs.sun.com/db/prod/sunone.

**[20]** Rusty Russell. *Linux iptables HOWTO*. September 1999. http://www.linuxguruz.com/iptables/howto/iptables-HOWTO.html

**[21]** Bill Klein. *System/Network Security: Firewall Issues*. http://www.nts.ku.edu/information/whitepapers/firewalls.jsp

**[22]** Sun's website on JNDI: http://java.sun.com/products/jndi/

# Appendix I

# LDAP schema for policy management

As discussed earlier, the two objectclasses defined for the distributed firewall policy management system are the *ipPacketFilter* and *ipPacketFilterHost*.

## I.1. *ipPacketFilter* objectclass:

```
#!/bin/sh

. ./credentials

eval $LDAPMODIFY -c $BINDDN $BINDPW << "EOF"
#
# University of Kansas Proposed IP Packet Filter Schema
#
# University of Kansas, Networking & Telecommunications Services
#
# This schema defines the object class and attributes that facilitate
# packet filtering hosts, with host centric rules sets.
#
# SCHEMA_FORMAT = iPlanet Directory Server 5.1 schema
#
# OBJECTCLASSES
# --------------------------------------------------------------------------
# IPPacketFilter          attributes for packet filtering
#
# AUTHORS + DESIGNERS
# --------------------------------------------------------------------------
# Siddhartha Gavirneni          siddh@ku.edu
#
# UPDATE_ON    UPDATE_BY    VER    DESCRIPTION
# --------------------------------------------------------------------------
# 06/03/2002   siddh@ku.edu  0.1   Architecture Planning
# 08/12/2002   siddh@ku.edu  0.2   Initial Incarnation
# 01/14/2003   siddh@ku.edu  0.3   Added protocol filtration attributes
# 03/17/2003   siddh@ku.edu  0.4   IP address filtration attributes
# 06/30/2003   siddh@ku.edu  0.5   Service based filtration
# 07/02/2003   siddh@ku.edu  0.6   Removed unnecessary attributes
# 07/17/2003   siddh@ku.edu  0.7   Added attribute for identifying internal IPs
#
```

```
dn: cn=schema
changetype: modify
#
# IPPacketFilter
#
# delete the existing attributeTypes
#
#delete: attributeTypes
#attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.1.1 NAME 'packetFilterAllowInboundICMPType' )
#attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.1.2 NAME
'packetFilterAllowOutboundICMPType' )
#attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.1.3 NAME 'packetFilterDenyInboundICMPType' )
#attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.1.4 NAME 'packetFilterDenyOutboundICMPType'
)
#attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.1.5 NAME 'packetFilterAcceptInternalIP' )
#attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.1.6 NAME 'packetFilterRejectInternalIP' )
#attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.1.7 NAME 'packetFilterAcceptExternalIP' )
#attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.1.8 NAME 'packetFilterRejectExternalIP' )
#attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.1.9 NAME 'packetFiltrationLevel' )
#attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.1.10 NAME 'packetFilterDefaultBlockAll' )
#attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.1.11 NAME 'packetFilterDefaultAllowAll' )
#attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.1.12 NAME
'packetFilterAllowInternalTCPServices' )
#attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.1.13 NAME
'packetFilterDenyInternalTCPServices' )
#attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.1.14 NAME
'packetFilterAllowInternalUDPServices' )
#attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.1.15 NAME
'packetFilterDenyInternalUDPServices' )
#attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.1.16 NAME
'packetFilterAllowExternalTCPServices' )
#attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.1.17 NAME
'packetFilterDenyExternalTCPServices' )
#attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.1.18 NAME
'packetFilterAllowExternalUDPServices' )
#attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.1.19 NAME
'packetFilterDenyExternalUDPServices' )
#attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.1.20 NAME 'packetFilterProtectedInternalIP' )
#-
#
# re-add the attributes -- in case there is a change of definition
#
add: attributeTypes
attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.1.1
	NAME 'packetFilterAllowInboundICMPType'
	DESC 'allows the specified type of inbound ICMP traffic'
	SYNTAX 1.3.6.1.4.1.1466.115.121.1.26
	X-ORIGIN 'IP Packet Filter Schema' )
```

```
attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.1.2
        NAME 'packetFilterAllowOutboundICMPType'
        DESC 'allows the specified type of outbound ICMP traffic'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26
        X-ORIGIN 'IP Packet Filter Schema' )
attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.1.3
        NAME 'packetFilterDenyInboundICMPType'
        DESC 'denies the specified type of inbound ICMP traffic'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26
        X-ORIGIN 'IP Packet Filter Schema' )
attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.1.4
        NAME 'packetFilterDenyOutboundICMPType'
        DESC 'denies the specified outbound ICMP traffic'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26
        X-ORIGIN 'IP Packet Filter Schema' )
attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.1.5
        NAME 'packetFilterAcceptInternalIP'
        DESC 'the acceptable IP address within the internal network'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26
        X-ORIGIN 'IP Packet Filter Schema' )
attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.1.6
        NAME 'packetFilterRejectInternalIP'
        DESC 'the rejectable IP address within the internal network'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26
        X-ORIGIN 'IP Packet Filter Schema' )
attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.1.7
        NAME 'packetFilterAcceptExternalIP'
        DESC 'the acceptable external IP addresses'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26
        X-ORIGIN 'IP Packet Filter Schema' )
attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.1.8
        NAME 'packetFilterRejectExternalIP'
        DESC 'the rejectable external IP addresses'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26
        X-ORIGIN 'IP Packet Filter Schema' )
attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.1.9
        NAME 'packetFiltrationLevel'
        DESC 'the level at which the packet filter is placed'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
        X-ORIGIN 'IP Packet Filter Schema' )
attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.1.10
        NAME 'packetFilterDefaultBlockAll'
        DESC 'block all traffic to and from this system/subnet'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.7
        X-ORIGIN 'IP Packet Filter Schema' )
attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.1.11
        NAME 'packetFilterDefaultAllowAll'
        DESC 'Allow all traffic to or from this system/subnet'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.7
        X-ORIGIN 'IP Packet Filter Schema' )
attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.1.12
        NAME 'packetFilterAllowInternalTCPServices'
        DESC 'The accessible TCP services on the internal network'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26
        X-ORIGIN 'IP Packet Filter Schema' )
attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.1.13
        NAME 'packetFilterDenyInternalTCPServices'
        DESC 'The inaccessible TCP services on the internal network'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26
        X-ORIGIN 'IP Packet Filter Schema' )
attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.1.14
        NAME 'packetFilterAllowInternalUDPServices'
        DESC 'The accessible UDP services on the internal network'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26
        X-ORIGIN 'IP Packet Filter Schema' )
attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.1.15
        NAME 'packetFilterDenyInternalUDPServices'
        DESC 'The inaccessible UDP services on the internal network'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26
        X-ORIGIN 'IP Packet Filter Schema' )
attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.1.16
        NAME 'packetFilterAllowExternalTCPServices'
        DESC 'The accessible TCP services on the external network'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26
        X-ORIGIN 'IP Packet Filter Schema' )
attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.1.17
        NAME 'packetFilterDenyExternalTCPServices'
        DESC 'The inaccessible TCP services on the external network'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26
        X-ORIGIN 'IP Packet Filter Schema' )
attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.1.18
        NAME 'packetFilterAllowExternalUDPServices'
        DESC 'The accessible UDP services on the external network'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26
        X-ORIGIN 'IP Packet Filter Schema' )
attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.1.19
        NAME 'packetFilterDenyExternalUDPServices'
        DESC 'The inaccessible UDP services on the external network'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26
        X-ORIGIN 'IP Packet Filter Schema' )
attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.1.20
        NAME 'packetFilterProtectedInternalIP'
        DESC 'The internal IP addresses for which the rules apply'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26
        X-ORIGIN 'IP Packet Filter Schema' )
-
```

```
#
# delete the existing objectClass
#
#delete: objectClasses
#objectClasses: ( 1.3.6.1.4.1.11314.2.2.4.2.1 NAME 'IPPacketFilter' )
#-
#
# now re-add the objectClass properly defined
#
add: objectClasses
objectClasses: ( 1.3.6.1.4.1.11314.2.2.4.2.1
        NAME 'IPPacketFilter'
        DESC 'attributes for packet filtering'
        SUP top
        MAY (  packetFilterAllowInboundICMPType
            $ packetFilterAllowOutboundICMPType
            $ packetFilterDenyInboundICMPType
            $ packetFilterDenyOutboundICMPType
                $ packetFilterAcceptInternalIP
                $ packetFilterRejectInternalIP
                $ packetFilterAcceptExternalIP
                $ packetFilterRejectExternalIP
                $ packetFiltrationLevel
                $ packetFilterDefaultAllowAll
                $ packetFilterDefaultBlockAll
            $ packetFilterAllowInternalTCPServices
            $ packetFilterAllowExternalTCPServices
            $ packetFilterAllowInternalUDPServices
            $ packetFilterAllowExternalUDPServices
            $ packetFilterDenyInternalTCPServices
            $ packetFilterDenyExternalTCPServices
            $ packetFilterDenyInternalUDPServices
            $ packetFilterDenyExternalUDPServices
            $ packetFilterProtectedInternalIP )
        X-ORIGIN 'IP Packet Filter Schema' )
-
#
# end of LDIF
#
EOF
```

## I.2. *ipPacketFilterHost* objectclass:

```
#!/bin/sh
. ./credentials
eval $LDAPMODIFY -c $BINDDN $BINDPW << "EOF"
#
# University of Kansas Proposed IP Packet Filtering Host Schema
#
# University of Kansas, Networking & Telecommunications Services
#
# This schema defines the object class and attributes that describe
# packet filtering hosts.
#
# SCHEMA_FORMAT = iPlanet Directory Server 5.1 schema
#
# OBJECTCLASSES
# ---------------------------------------------------------------------------
# IPPacketFilterHost       attributes describing the packet filtering host
#
# AUTHORS + DESIGNERS
# ---------------------------------------------------------------------------
# Siddhartha Gavirneni               siddh@ku.edu
#
# UPDATE_ON     UPDATE_BY     VER       DESCRIPTION
# ---------------------------------------------------------------------------
# 04/12/2003        siddh@ku.edu      0.1       Initial Incarnation
# 07/01/2003        siddh@ku.edu      0.2       Changing the attribute requirements
#                                    (protectedNetworkDN and sysadmin are made a MUST)
# 07/14/2003        siddh@ku.edu      0.3       Changing the attribute requirements
#                        (insideInterfaceName and outsideInterfaceName are made a MUST)
# 07/29/2003        siddh@ku.edu      0.4       Attributes for stateful/stateless filtration
#
dn: cn=schema
changetype: modify
#
#IPPacketFilterHost
#
# delete the existing attributeTypes
#
#delete: attributeTypes
#attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.3.1 NAME 'insideInterfaceName' )
#attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.3.2 NAME 'outsideInterfaceName' )
#attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.3.3 NAME 'insideInterfaceMACAddress' )
#attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.3.4 NAME 'outsideInterfaceMACAddress' )
#attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.3.5 NAME 'insideInterfaceIPAddress' )
#attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.3.6 NAME 'outsideInterfaceIPAddress' )
#attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.3.7 NAME 'typeBridging' )
#attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.3.8 NAME 'typeForwarding' )
#attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.3.9 NAME 'protectedNetworkDN' )
#attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.3.10 NAME 'statefulFiltration' )
```

```
#-
#
# re-add the attributes -- in case there is a change of definition
#
add:attributeTypes
attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.3.1
        NAME 'insideInterfaceName'
        DESC 'name of the interface on the internal network'
        EQUALITY caseExactIA5Match
        SUBSTR caseExactIA5SubstringMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26
        X-ORIGIN 'IP Packet Filter Schema')
attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.3.2
        NAME 'outsideInterfaceName'
        DESC 'name of the interface on the external network'
        EQUALITY caseExactIA5Match
        SUBSTR caseExactIA5SubstringMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26
        X-ORIGIN 'IP Packet Filter Schema')
attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.3.3
        NAME 'insideInterfaceMACAddress'
        DESC 'mac address of the interface on the internal network'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26
        X-ORIGIN 'IP Packet Filter Schema')
attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.3.4
        NAME 'outsideInterfaceMACAddress'
        DESC 'mac address of the interface on the external network'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26
        X-ORIGIN 'IP Packet Filter Schema')
attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.3.5
        NAME 'insideInterfaceIPAddress'
        DESC 'IP address of the interface on the internal network'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26
        X-ORIGIN 'IP Packet Filter Schema')
attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.3.6
        NAME 'outsideInterfaceIPAddress'
        DESC 'IP address of the interface on the external network'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26
        X-ORIGIN 'IP Packet Filter Schema')
attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.3.7
        NAME 'typeBridging'
        DESC 'indicates if bridging is used'
        EQUALITY booleanMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.7
        X-ORIGIN 'IP Packet Filter Schema')
attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.3.8
        NAME 'typeForwarding'
        DESC 'indicates if forwarding is used'
        EQUALITY booleanMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.7
        X-ORIGIN 'IP Packet Filter Schema')
attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.3.9
        NAME 'protectedNetworkDN'
        DESC 'the department/LAN protected by this host'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.12
        X-ORIGIN 'IP Packet Filter Schema' )
attributeTypes: ( 1.3.6.1.4.1.11314.2.2.4.3.10
        NAME 'statefulFiltration'
        DESC 'indicates whether this is a stateful or stateless packet filter'
        EQUALITY booleanMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.7
        X-ORIGIN 'IP Packet Filter Schema' )
-
#
# delete the existing objectClass
#
#delete: objectClasses
#objectClasses: ( 1.3.6.1.4.1.11314.2.2.4.4.1 NAME 'IPPacketFilterHost' )
#-
#
# now re-add the objectClass properly defined
#
add: objectClasses
objectClasses: ( 1.3.6.1.4.1.11314.2.2.4.4.1
        NAME 'IPPacketFilterHost'
        DESC 'attributes describing the packet filtering Host'
        SUP top
        MUST ( insideInterfaceMACAddress
             $ outsideInterfaceMACAddress
             $ protectedNetworkDN
             $ sysadmin
             $ insideInterfaceName
             $ outsideInterfaceName )
        MAY ( insideInterfaceIPAddress
            $ outsideInterfaceIPAddress
            $ typeBridging
            $ typeForwarding
            $ statefulFiltration )
        X-ORIGIN 'IP Packet Filter Schema' )
-
#
# end of LDIF
#
EOF
```

# Appendix II

# Directory Support and *iptables* Rule Generator

This appendix will outline the basic classes designed and used for the firewall directory support, and the *iptables* rule generator.

## II.1. Directory Access:

**Package *edu.ku.net.den.util***

| **Class Summary** | |
|---|---|
| **DirectoryServer** | The DirectoryServer class provides an abstraction to the back-end directory server to provide a mechanism to decouple and change the directory server. |
| **DirectoryServerConfigFile** | The DirectoryServerConfigFile class processes a directory server configuration file, converting the XML syntax into a collection of DirectoryServerInfo objects. |
| **DirectoryServerInfo** | The DirectoryServerInfo class contains the information necessary to connect to a directory server. |

## II.2. Directory support for the packet filter

**Package *pfDirSupport***

| **Class Summary** | |
|---|---|
| **PacketFilterDirectorySupport** | The PacketFilterDirectorySupport class provides the directory access support for the packet filtration system. |
| **PacketFilterProtectedNodes** | The PacketFilterProtectedNodes class provides functions for obtaining the directory entries (subnets or systems, or group of systems), for which rules need to be created. |

## II.3. *iptables* Rule Generator

### Package *iptables*

| **Class Summary** | |
|---|---|
| **CreateIptablesRules** | The CreateIptablesRules provides methods for creating the *iptables* rules for the nodes being protected by a packet filter. |